SPERRY UNIVAC
COMPUTER SYSTEMS

PUBLICATIONS REVISION

Operating System/3 (OS/3)

Sort/Merge

User Guide

(Series 90)

For System 80 see {UP-8819} {UP-9072}

UP-8342 Rev. 3

This Library Memo announces the release and availability of "SPERRY UNIVAC® Operating System/3 (OS/3) Sort/Merge User Guide", UP-8342 Rev. 3.

This revision documents the following new sort/merge features for the 8.0 release:

- Data management environment considerations have been included indicating the file types supported.
- UOS parameter of OUTFIL control statement for indpendent sort/merge now defaults to 100% rather than 0%.
- All messages for the sort/merge job are displayed at the initiating workstation.
- SORT3 is now compatible with IBM System/32 and 34 sorts, as well as with the IBM System/3 sort.
- The record type specification for SORT3 now allows the specification of UDATE, UDAY, UMONTH, and UYEAR for Factor 2.
- The field specification for SORT3 now allows you to specify up to a maximum of 256 bytes for the overflow field length.

All other changes are corrections or expanded descriptions applicable to features present in sort/merge prior to the 8.0 release.

Destruction Notice: If you are going to OS/3 release 8.0, use this revision and destroy all previous copies. If you are not going to OS/3 release 8.0, retain the copy you are now using and store this revision for future use.

Copies of UP-8342 Rev. 2, UP-8342 Rev. 2-A and UP-8342 Rev. 2-B will be available for 6 months after the release of 8.0. Should you need additional copies of this edition, you should order them within 90 days of the release of 8.0. 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 Univac representative.

LIBRARY MEMO ONLY	LIBRARY MEMO AND ATTACHMENTS	THIS SHEET IS:
Mailing Lists BZ, CZ and MZ	Mailing Lists A00, A02, A03, A04, 18, 18U, 19, 19U, 20, 20U, 21, 21U, 75, 75U, 76, and 76U (Cover and 336 pages)	Library Memo for UP-8342 Rev. 3
		RELEASE DATE:
		September, 1982

Sort/Merge

OS/3

User Guide

Environment: 90/25, 30, 30B, 40 Systems

H

SPERRY UNIVAC

UP-8342 Rev. 3

This document contains the latest information available at the time of preparation. Therefore, it may contain descriptions of functions not implemented at manual distribution time. To ensure that you have the latest information regarding levels of implementation and functional availability, please consult the appropriate release documentation or contact your local Sperry Univac representative.

Sperry Univac reserves the right to modify or revise the content of this document. No contractual obligation by Sperry Univac regarding level, scope, or timing of functional implementation is either expressed or implied in this document. It is further understood that in consideration of the receipt or purchase of this document, the recipient or purchaser agrees not to reproduce or copy it by any means whatsoever, nor to permit such action by others, for any purpose without prior written permission from Sperry Univac.

Sperry Univac is a division of the Sperry Corporation.

This document was prepared by Systems Publications using the SPERRY UNIVAC UTS 400 Text Editor. It was printed and distributed by the Customer Information Distribution Center (CIDC), 555 Henderson Rd., King of Prussia, Pa., 19406.

PRINTED IN U.S.A.

PAGE STATUS SUMMARY

ISSUE: UP-8342 Rev. 3
RELEASE LEVEL: 8.0 Forward

Part/Section	Page Number	Update Level
Cover/Disclaimer		
PSS	1	
Preface	1, 2	
Contents	1 thru 9	
PART 1		
Title Page		
1	1 thru 14	
PART 2		
Title Page		
2	1 thru 12	
3	1 thru 65	
4	1 thru 19	
PART 3		
Title Page		
5	1 thru 11	
6	1 thru 54	
7	1 thru 6	
8	1 thru 16	
9	1 thru 18	
PART 4		
Title Page		
10	1 thru 13	
11	1 thru 37	
12	1 thru 7	
PART 5		
Title Page		
Appendix A	1 thru 8	
Appendix B	1 thru 4	
Appendix C	1 thru 4	
Appendix D	1 thru 11	
Index	1 thru 18	
User Comment Sheet		

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 UNIVAC Operating System/3 (OS/3). It specifically describes the function and effective use of the sort/merge programs available to the user of OS/3. The intended audience is the novice programmer with a basic knowledge of data processing but with little programming experience, and the programmer whose experience is on systems other than Sperry Univac.

An introductory manual, the introduction to sort/merge, UP-8073 (current version), is also available. It briefly describes the general characteristics and facilities offered by the sort/merge programs for OS/3. For the more experienced programmer, the sort/merge programmer reference, UP-8054 (current version) is available.

In this manual, sort/merge subject matter is divided into the following parts:

- Part 1. Survey of Sort/Merge
- Part 2. Independent Sort/Merge
- Part 3. Subroutine Sort/Merge
- Part 4. System/3, 32, and 34 Compatible Sort (SORT3)
- Part 5. Appendixes

We suggest that you read this manual from beginning to end. Although each part is self-contained, you can profit by comparing the three sort/merge programs in order to select the one most suited for your application.

Part 1. Survey of Sort/Merge

Introduces you to the process of record sorting in relationship to the three sort programs offered by OS/3. The information is presented in terms of what each program does, how you use them, and what you should consider before using them.

Part 2. Independent Sort/Merge

Describes this sort/merge program from the system viewpoint and accents the software framework and the operational phase structure. Part 2 emphasizes independent sort/merge from the user viewpoint, including independent sort/merge requirements you must supply:

- Job control stream
- Sort/merge control statements
- User exits to own-code routines

This part also discusses the independent sort/merge application to a typical disk sort problem and ends with program and control stream examples.

Part 3. Subroutine Sort/Merge

Describes this sort/merge program from the user viewpoint by using the same disk sort problem from Part 2 to emphasize:

- User program interface with subroutine sort/merge
- Assembling, link editing, and executing a typical disk sort program
- Special job control stream applications
- Subroutine sort/merge user own-code routines.

Part 3 also supplies subroutine sort/merge program examples.

Part 4. System/3, 32, and 34 Compatible Sort

Describes this sort/merge program from the system viewpoint to emphasize the purpose, application, software framework, and operational structure of the program. In addition, Part 4 describes the IBM System/3, 32, and 34 compatible sort from the user viewpoint to include the supportive requirements you must supply to run the sort. This includes:

- Job control stream
- Sort control statements

Part 5. Appendixes

Contain:

- Statement conventions
- Sort parameter table contents
- Subroutine sort/merge interface requirements for the COBOL programmer
- OS/3 and standard EBCDIC and ASCII collating sequences

Contents

PART 2. INDEPENDENT SORT/MERGE

2. INDEPENDENT SORT/MERGE BASIC CONCEPTS

3. INDEPENDENT SORT/MERGE REQUIREMENTS YOU SUPPLY

PART 4. SYSTEM/3, 32, AND 34 COMPATIBLE SORT

FIGURES

TABLES

PART 1. SURVEY OF OS/3 SORT/MERGE PROGRAMS

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. SORT PROGRAMS AVAILABLE TO THE OS/3 USER

The SPERRY UNIVAC Operating System/3 (OS/3) offers you three alternative methods of sorting your data files by providing you with a sort package containing the coding for three sort programs: independent sort/merge, subroutine sort/merge, and IBM System/3, 32, and 34 compatible sort (SORT3). All three programs are modular in structure and are capable of operating in a minimum system configuration. Although they produce the same results, the programs are implemented at different levels of programming, require different degrees of user program intervention, and permit a varying degree of user control over the actual sort operation. Each has its distinct advantages and disadvantages.

1.2.1. Independent Sort/Merge

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 the user of indexed sequential access method files (ISAM) and 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 assembly 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 Part 2.

Independent sort/merge programs can be executed either in a batch environment (on cards) or in an interactive environment (from a workstation). Although the sample job control streams for independent sort/merge are shown on cards, they can be keyed in from a workstation. The rules for preparing your sort control statements and specifications on cards also apply to workstation keyins. The procedure for executing your sort program interactively is described in Section 3.

1.2.2. Subroutine Sort/Merge

Subroutine sort/merge, unlike independent sort/merge, is not a *canned* service program. It simply provides you with sort modules capable of performing various sorting and merging functions and allows you to write the sort program you want, using these modules. This affords you the benefit of exercising greater control over the sort/merge process and the advantage of flexibility in specifying:

- External input record formats
- Sources of input records
- External output record formats
- Disposition of final output records

However, greater user control means an increase in user programming. In fact, you will have to assemble, link, and execute subroutine sort/merge from your job control stream in addition to programming many of the other activities that are automatically performed for the user of independent sort/merge. This means you get directly involved with job control, data management, the assembler, etc. You provide the routines for inputting and outputting data, and you establish the necessary communication links between your program and subroutine sort/merge. Your responsibilities will also include the programming needed for defining files, reserving buffer areas, manipulating register addresses, and delivering data to and retrieving data from the sort, as well as initiating and terminating the sort process.

By design, subroutine sort/merge can be incorporated as part of a much larger program or, if combined with input and output routines, it can be part of a more conventional run where sorting is the primary objective. Both basic assembly language (BAL) and COBOL can serve as the medium through which you establish the communications link to subroutine sort/merge. When BAL is employed, sort/merge and data management macros are used for defining sort requirements. (See Part 3.) When COBOL is used, you must use COBOL SORT statements to define the sort. (For more information, see Appendix C.)

1.2.3. System/3, 32, and 34 Compatible Sort (SORT3)

SORT3, like the independent sort/merge, is also considered an easy-to-use, *canned* sort program because it is modular in design and requires a minimum of user programming and does not need to be assembled or linked to your program. It increases the versatility of the OS/3 sort package by providing you with a program that is compatible with the IBM System/3, 32, or 34 sort. That is, SORT3 accepts, with minor differences, all System/3, 32, or 34 sort specifications and offers all of the features of these sorts that are feasible within the OS/3 operating system. In addition to disk and tape input files, the SORT3 program is capable of processing input data from card files. It also provides you with added control over the record sequencing, data reduction, and data disposition without the necessity of reverting to user own-code routines.

SORT3 is designed to operate under control of the OS/3 supervisor and data management systems. However, it can be initiated through either OS/3 job control language (JCL) or the operation control language (OCL) processor. Running SORT3 under the OCL processor does not require you to make any changes to your existing System/3, 32, or 34 sort job stream; the OCL statements and sequence specification remain the same as though you were running in a System/3, 32, or 34 environment. Instructions for running SORT3 under OS/3 JCL are provided in Part 4.

Like independent sort/merge, SORT3 gives you a choice of operating either in a batch environment or interactively from a workstation. The procedure for executing SORT3 from a workstation is the same as described for independent sort/merge, except that the // EXEC statement will specify SORT3 instead of SORT. (See 3.5.)

1.3. CONCEPT OF MODULAR SORT STRUCTURE

In the process of describing the OS/3 sort package, we have referred to the sort programs 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 provided to you as load modules residing in the system load library ($Y$LOD). Most load modules are common to all three OS/3 sort programs. 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 sort programs to the requirements of your installation by increasing programming flexibility.

In addition to the sort modules, the OS/3 sort package provides call modules to interface each sort program with the system. The call modules for the two *canned* programs, independent sort/merge and the System/3, 32, or 34 compatible sort, are the SORT and SORT3 system driver programs, respectively. They both reside in $Y$LOD. The call module for the subroutine sort is the sort common module (SG$ORT) that resides in the system object library file ($Y$OBJ). All three call modules perform similar introductory functions, but each contains elements peculiar to the sort program that it calls.

Figure 1-1 shows the modular structure of the OS/3 sort package. (Note that the load modules are common to all three sort programs.)

If you wish to copy the OS/3 sort package onto your own user library file, you can do so by means of the librarian as described in the system service program (SSP) user guide, UP-8062 (current version). Be sure to include all of the following modules:

- System load library file ($Y$LOD)
  - Sort load modules beginning with SM$
  - Independent sort/merge system driver program SORT
  - System/3 compatible sort system driver program SORT3
- System object library file ($Y$OBJ)
  - Subroutine sort/merge sort common module (SG$ORT)
- System macro library file ($Y$MAC)
  - Five subroutine sort/merge macros beginning with MR$
  - Two subroutine merge-only macros beginning with MG$

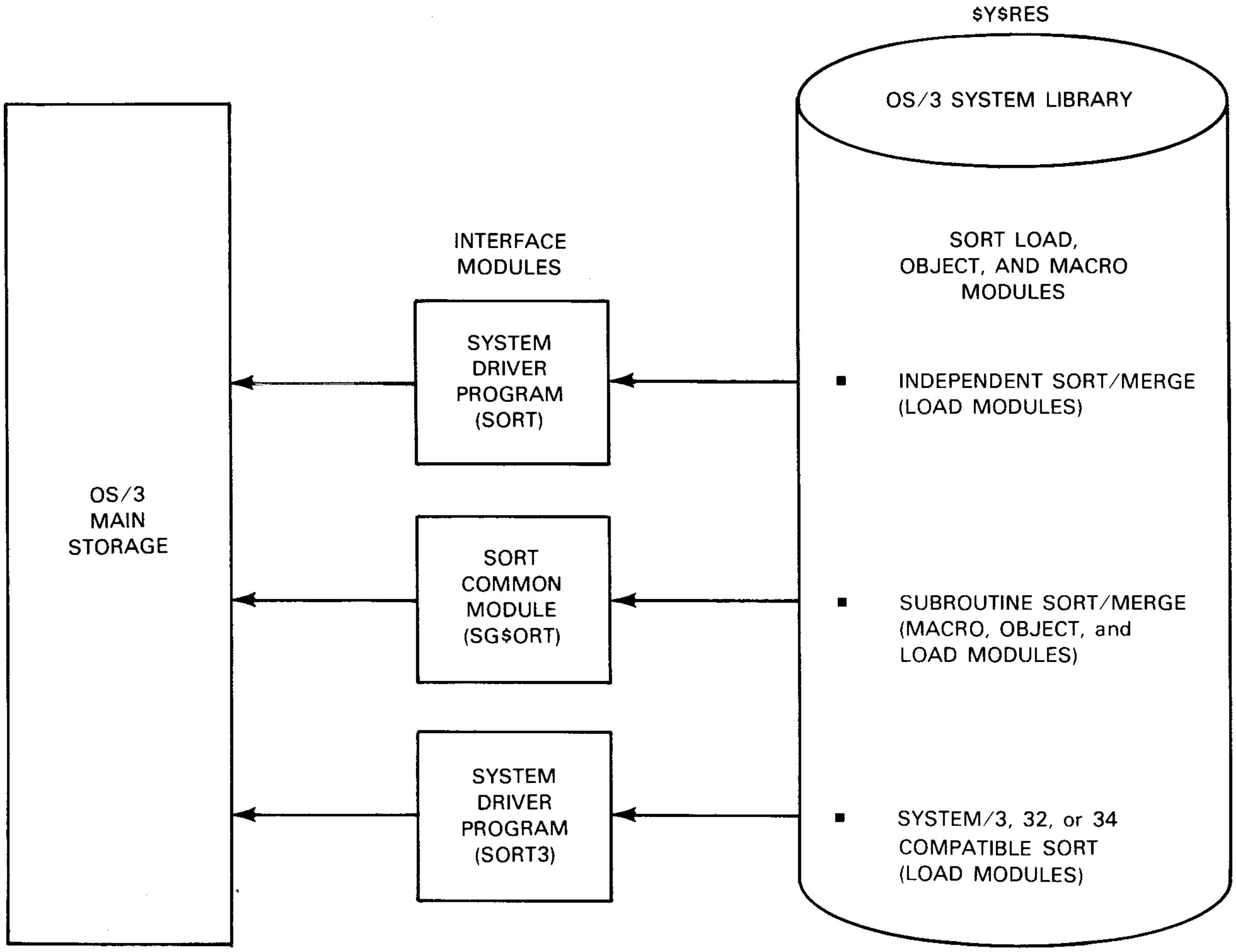

$Y$RES
OS/3 SYSTEM LIBRARY
SORT LOAD, OBJECT, AND MACRO MODULES
■ INDEPENDENT SORT/MERGE (LOAD MODULES)
■ SUBROUTINE SORT/MERGE (MACRO, OBJECT, and LOAD MODULES)
■ SYSTEM/3, 32, or 34 COMPATIBLE SORT (LOAD MODULES)
INTERFACE MODULES
SYSTEM DRIVER PROGRAM (SORT)
SORT COMMON MODULE (SG$ORT)
SYSTEM DRIVER PROGRAM (SORT3)
OS/3 MAIN STORAGE

Figure 1—1. Modular Structure of OS/3 Sort Programs

1.4. WHAT OS/3 SORT PROGRAMS CAN DO FOR YOU

In general, the sort programs available in OS/3 assist you in producing a tailored output file from your existing input data files. Through the sorting and merging techniques employed in these programs, 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. All three sort programs 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 or control fields;

- recognize key and control fields in the following formats:
  - character
  - binary (signed and unsigned)
  - decimal (signed zoned and unsigned zoned)
  - packed decimal
  - 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-8069 (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.

1.5. DATA MANAGEMENT CONSIDERATIONS

The data management environment you use governs the types of files that can be sorted by the sort/merge programs. Table 1-1 lists the file types and shows the data management environments that support them.

Table 1—1. File Types Supported in the Data Management Environments

File Type	Data Management Environment		
	Consolidated Data Management Only	Mixed	Basic Data Management Only
MIRAM disk	Yes	Yes	No
IRAM disk	Yes	Yes	Yes
Nonindexed disk	No	Yes	Yes
SAM disk	No	Yes	Yes
Dam disk	No	No	No
ISAM disk	No	No	No
Card*	Yes	Yes	Yes
Tape	Yes	Yes	Yes
Data set diskette*	Yes	Yes	Yes

*SORT3 only

For further information on the various file types, refer to the basic data management user guide, UP-8068 (current version) and the consolidated data management concepts and facilities, UP-8825 (current version).

1.6. PROGRAM RESTRICTIONS

Variations in a program design, capability, and implementation sometimes restrict the use of a sort program for specific applications or for specific system configurations. The restrictions that apply to the sort programs are as follows:

- All Sort Programs
  - All sorting is limited to disk-only and tape-only sorts or storage-only sorts.
  - The disk work files that you assign to your program must be of the same type, that is, all type 8430 or all type 8418, and so on.
  - If you use interactive services (consolidated data management environment only), your disk files must be multiple indexed random access method (MIRAM) files.
  - Volume of data sorted and merged is limited by the type and physical capacity of the tape or disk space assigned as auxiliary work storage.
  - Indexed random access method (IRAM) files are supported. Input files must have been created by OS/3. The records must be fixed length. The output files are nonindexed, and all volumes of the output file must be mounted.

- Independent Sort/Merge and Subroutine Sort/Merge
  - Input files must be disk only or tape only.
  - IRAM files are not supported when the independent or subroutine sort/merge is used for a merge-only application.
  - Auxiliary storage work areas can be disk or tape, but not both. The independent or subroutine sort/merge is limited to eight disk files or six tape files.
  - User own-code routines can be substituted for those provided in the independent or subroutine sort/merge if they satisfy the requirements of the program and OS/3 programming conventions.
- SORT3
  - Input files can be card, tape, disk or diskette.
  - A merge-only application cannot be performed by the SORT3 program.
  - SORT3 does not support data reduction or record sequencing and checking through the use of user own-code exit routines.
  - Auxiliary storage work areas can be disk or tape, but not both. SORT3 is limited to six disk files or six tape files.

1.7. 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.

1.7.1. 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. The minimum main storage requirements for your sort depend upon which method you use:

- 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.

- Subroutine sort/merge requires 12,400 bytes. If the record length is greater than 100 bytes, you should allow 12,400 bytes plus five times the input record length. (These figures do not include the requirements for your program, its preamble, or your own-code routines.)

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

- SORT3 requires a minimum of 16,000 bytes. However, the more sort specifications you include in your program the less main storage available for the actual sort. Each sort specification processed requires 12 bytes of main storage. Also bear in mind that additional main storage is required when using an alternate collating sequence (280 bytes), field specifications for packed data (40 bytes each), and include or omit, or both, specifications for packed data (100 bytes).

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

1.7.2. 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, i.e., sectorized or nonsectorized. 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 the independent sort/merge and subroutine sort/merge programs. The SORT3 program is limited to a maximum of six disk files. 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 1-2 contains a comparison of the direct access storage devices used by the sort programs. Sort execution time tables that indicate specific performance times of an average sort application with disk work files are available in the sort/merge programmer reference, UP-8054 (current version).

Table 1—2. Comparison of Data Capacities and Access Speeds for Direct Access Devices

Characteristics	Disk Subsystem Type								
	8411	8414	8415	8416	8418–92/93	8418–94/95	8424	8425	8430
Maximum data capacity (8-bit bytes per disk pack)	7,250,000	29,170,000	33,200,000	28,958,720	28,958,720	57,917,440	58,300,000	58,352,000	100,018,280
Maximum track capacity (bytes)	3625	7294	13,030	10,240	10,240	10,240	7294	7294	13,030
Minimum cylinder access time (ms)	25	20	10	10	10	10	10	7.5	7
Average cylinder access time (ms)	75	60	33	30	27	33	30	41.5	27
Maximum cylinder access time (ms)	135	130	60	60	45	60	55	80	50

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 from different tape units varies according to the tape density (number of bits recorded across the width of the tape) and the number of recording tracks (7 or 9). Refer to Table 1-3.

Table 1—3. Comparison of Transfer Rates for Magnetic Tape Devices

	Magnetic Tape Subsystem Type					
Data Transfer Rate	**UNISERVO 10**	**UNISERVO 12**	**UNISERVO 14**	**UNISERVO 16**	**UNISERVO 20**	**UNISERVO VI-C**
9-track (phase encoded)						
1600 bpi*	40,000 bps	68,320bps**	96,000 bps	192,000 bps	320,000 bps	–
9-track (NRZI)						
800 bpi	20,000 bps	34,160 bps	48,000 bps	96,000 bps	–	34,160 bps
7-track (NRZI)						
200 bpi	5,000 bps	8,540 bps	12,000 bps	24,000 bps	–	8,540 bps
556 bpi	13,900 bps	23,740 bps	33,400 bps	66,700 bps	–	23,740 bps
800 bpi	20,000 bps	34,160 bps	40,000 bps	96,000 bps	–	34,160 bps

* bpi = bits per inch

**bps = bits per second

1.7.3. 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 keys and more keys per record increase sort time because lengthier comparisons are needed.

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.

1.7.4. Sort Options

When using either independent sort/merge or subroutine sort/merge, there are two optional parameters that affect performance:

- NOCKSM=D or NOCKSM=T

- USEQ

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 USEQ optional parameter to indicate a special collation sequence other than EBCDIC or ASCII, you increase sort/merge performance time.

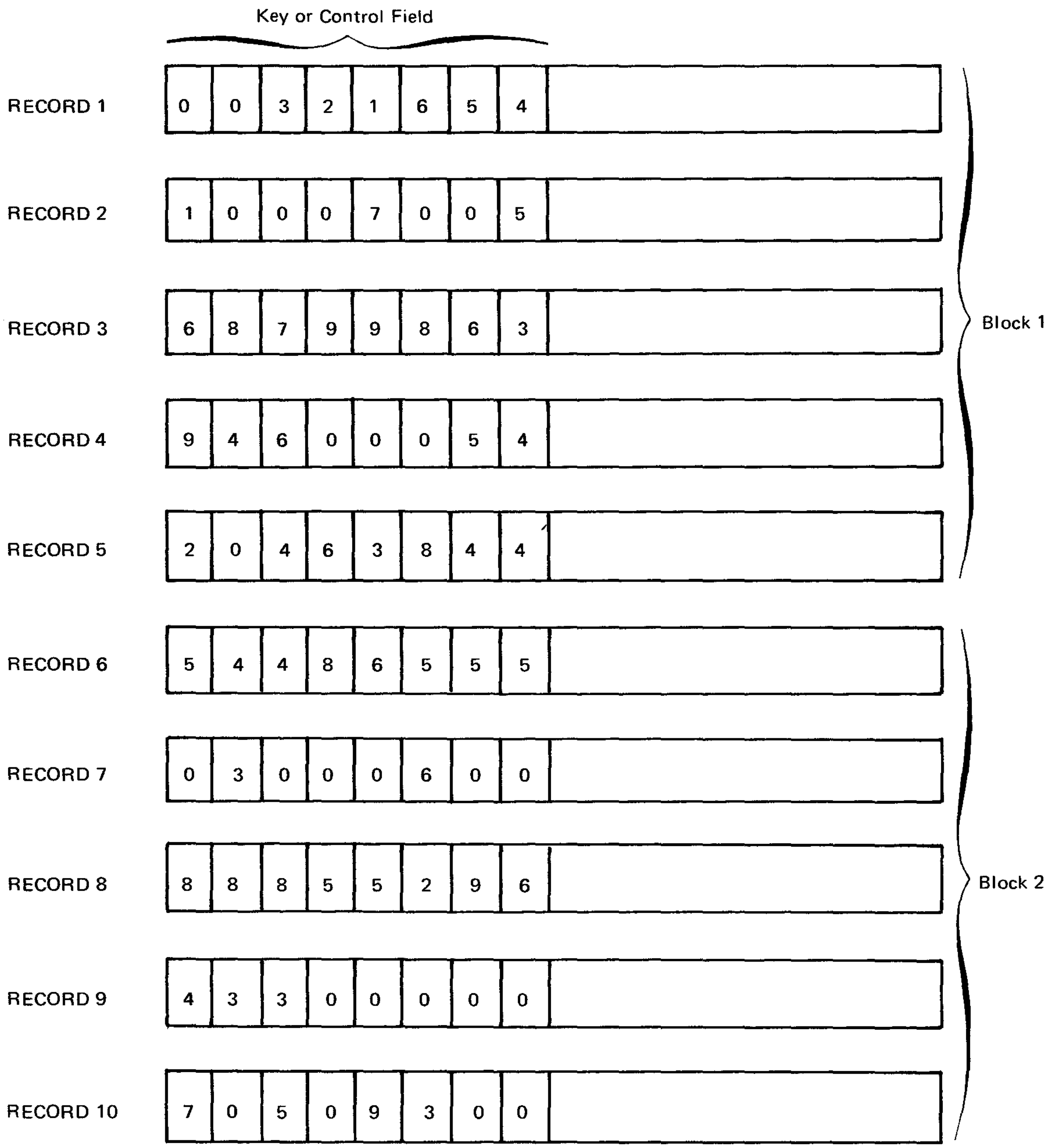
The SORT3 counterparts to the aforementioned options are the VERIFY OPTION and the ALT COLLATING SEQ entries in the header specification.

1.8. 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* in the independent sort/merge and subroutine sort merge program and *control fields* in the SORT3 program, 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.

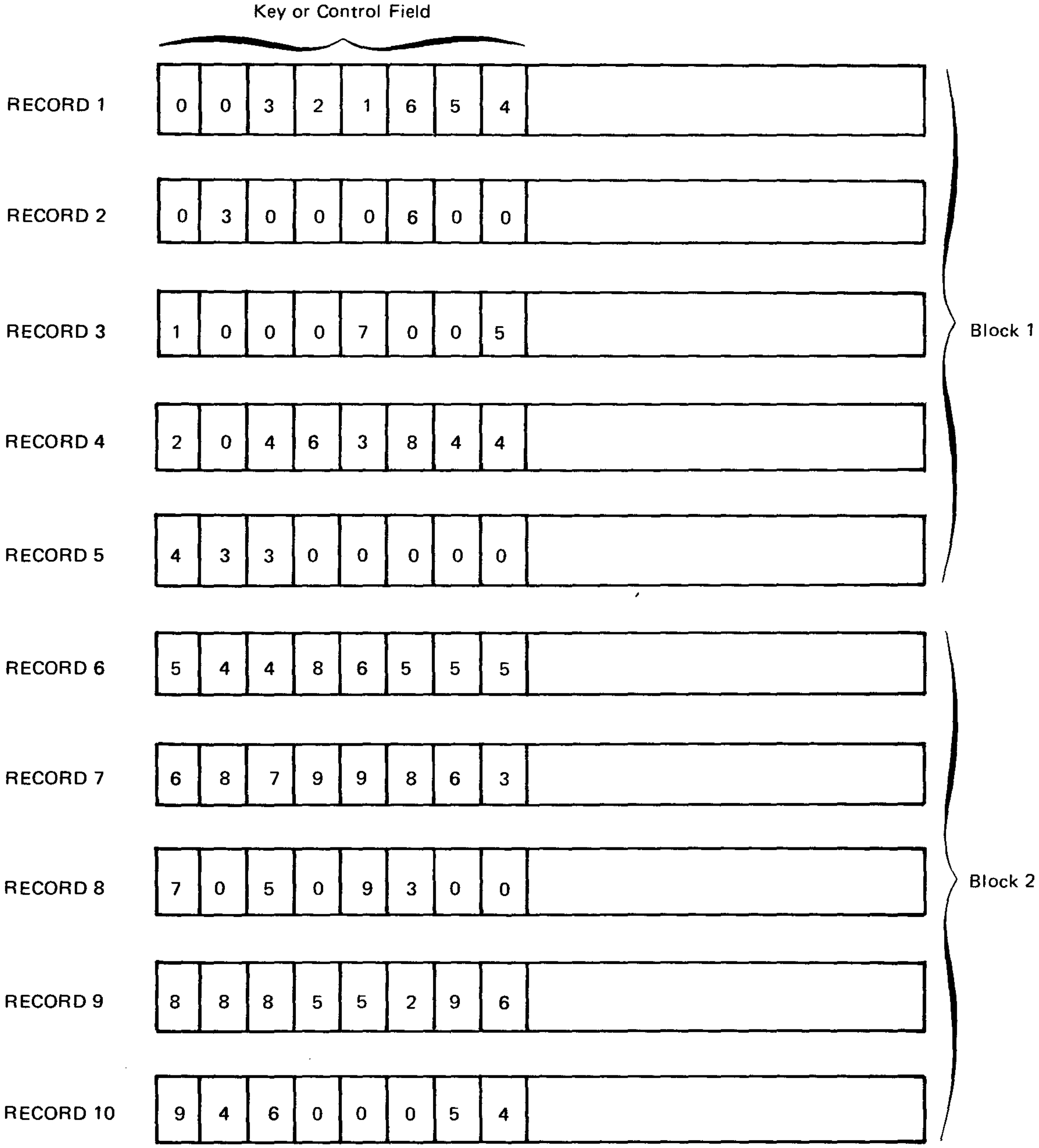
Figure 1—2 illustrates what the data contained in key fields of the first two input data record blocks might look like before the sort:



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

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

PART 2. INDEPENDENT SORT/MERGE

2. Independent Sort/Merge Basic Concepts

2.1. GENERAL

The independent 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, independent 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. There is only one exception to this capacity: the OS/3 sort does not support indexed sequential files (ISAM). Any ISAM files must be read or written via user own-code; thus, if you have indexed sequential files and use the independent sort/merge, you would enter your own-code routine via an exit code. (See 3.3.)

In addition to simplifying your sort/merge job execution, independent 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 the independent 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 the independent sort/merge handle one file while your own-code routine handles the other. Thus, you can have both convenience and flexibility when you use the independent sort/merge.

The main advantage in using the independent sort/merge is that you have no assembly or linkage steps to perform. You simply submit job control and sort/merge control statements, then execute them. This is a tremendous time and storage saver. To help you understand the activities performed by the independent sort/merge and the information it requires from you, we now examine a disk sort problem application.

2.2. SORT PROBLEM: A SOLUTION

A brief narrative outline of the independent sort/merge disk sort program follows. For details about input and output files before and after the sort/merge operation, refer to 1.4.

SYSTEM: OS/3

PROGRAM: Independent Sort/Merge Disk Sort

FUNCTION:

1. This program sorts and merges an unordered file of employee records.
2. It is a disk sort.
3. It uses a sort key to sort and merge records.
4. The sort key is the employee number.
5. Employee number is located in the first eight byte positions of each record.
6. Records are sorted in ascending order.

INFORMATION:

1. This program needs one additional work file to perform the sort/merge ($SCR1).
2. Because data volume for the independent sort/merge example (Figures 3—2 and 3—4) is low, the work file is assigned to disk device 50. Usual job situations, however, call for a larger volume of data. For optimum sort efficiency, a volume should never hold more than one work file, and input and output files should be on a separate volume.

INPUT AND OUTPUT:

1. The program produces an output file of records sorted in ascending order.
2. Both input and output files use fixed-length, blocked records.
3. Each record contains 80 bytes.
4. Each block contains 5 records.

OUTPUT:

The program produces an output file of records sorted in ascending order.

Figure 2—1 summarizes these specifications.

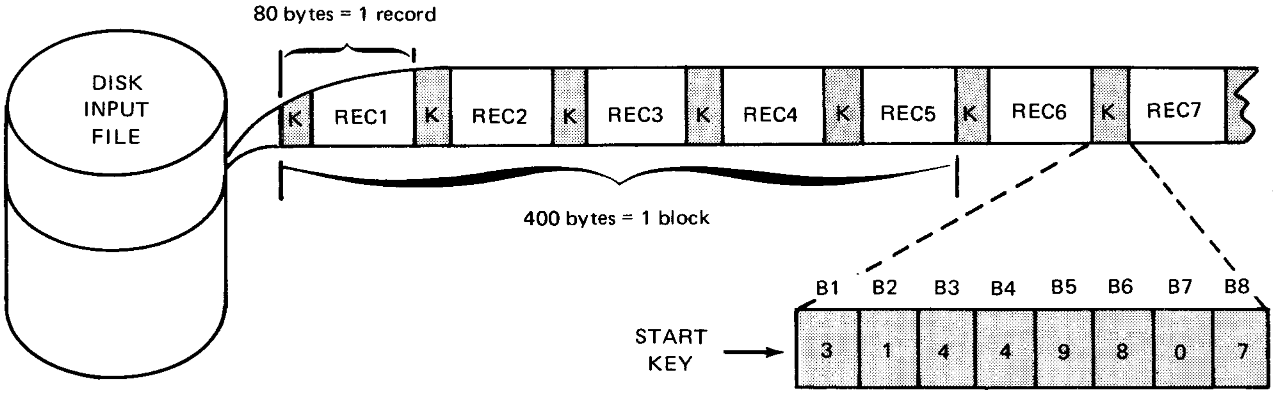


Figure 2—1. Key, Record, and Block Interrelationship

2.3. WHAT SORT/MERGE DOES FOR YOU

Independent sort/merge is a modular canned program. Each module performs a specific function when needed by a particular operational phase. After the needed modules are called by the system driver program, they are loaded into main storage and executed. Not all called modules are independent sort/merge modules. Some may be your own modules. To determine which modules to use during certain phases, independent sort/merge examines your specifications in the sort parameter table that you indicate via sort/merge control statements and sometimes via job control statements in the job control stream. Entries in the sort parameter table provide information concerning the type of sort, the peculiarities of the input files, the final disposition and sequence of the output file, characteristics of the records to be processed, and control functions. See Appendix B for the format of the sort parameter table. Figure 2—2 illustrates how the independent sort/merge operation executes.

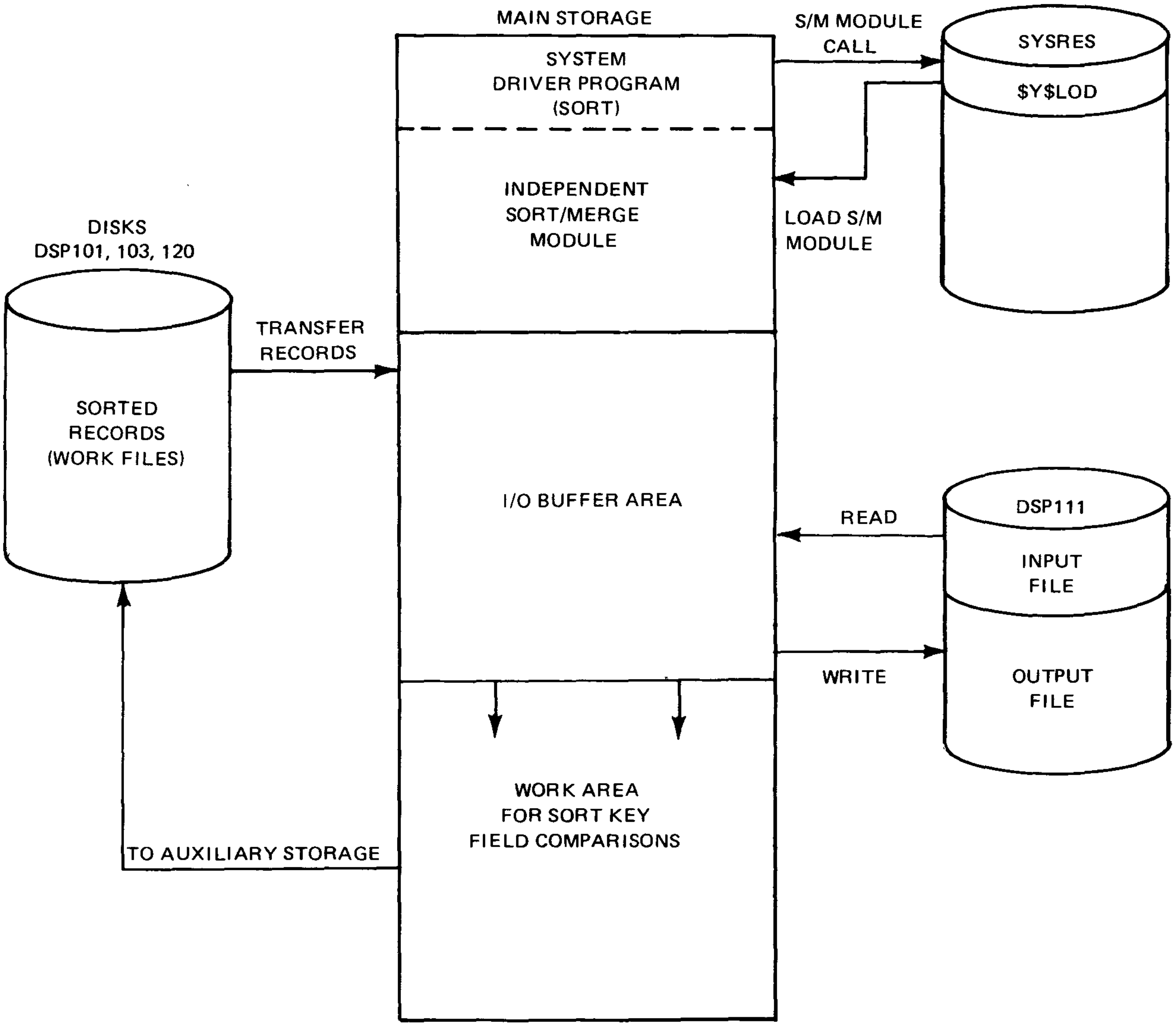


Figure 2—2. Execution of Independent Sort/Merge

When you submit your job control stream including the sort/merge control statements to the system and the EXEC statement is read, the system driver program (SORT) is loaded, and it calls the first module of the independent sort/merge into main storage from the system load library file ($Y$LOD). Your previous job control statements include device assignment sets that tell independent sort/merge where to find the input files, where to perform the sorting in work files, and where to write the output file. The EXEC statement signals independent sort/merge to accept your sort parameter table specifications that provide information concerning sort procedures peculiar to your problem.

Notice there is an interplay of activities between your job control stream and the independent sort/merge canned program. The entire sort/merge operation centers around elements you supply and elements that independent sort/merge supplies.

2.3.1. Software Framework

Independent sort/merge and subroutine sort/merge modular organization is basically the same. Both types of sort/merge consist of two to four operational phases normally executed in this sequence:

- Sort initialization and assignment (phase 0)
- Data input and internal sort (phase 1)
- Preliminary merge (phase 2)
- Final merge and output (phase 3)

In cases where the input file is biased (partially sequenced) or is small enough so that one final merge produces the required output file, independent sort/merge bypasses the preliminary merge (phase 2) and proceeds to the final merge and output (phase 3). In a merge-only operation, phases 1 and 2 are both bypassed. Independent sort/merge proceeds directly from initialization and assignment to phase 3, where records are read into main storage, merged, and written to the output file. Figure 2—3 shows the independent sort/merge operational phases.

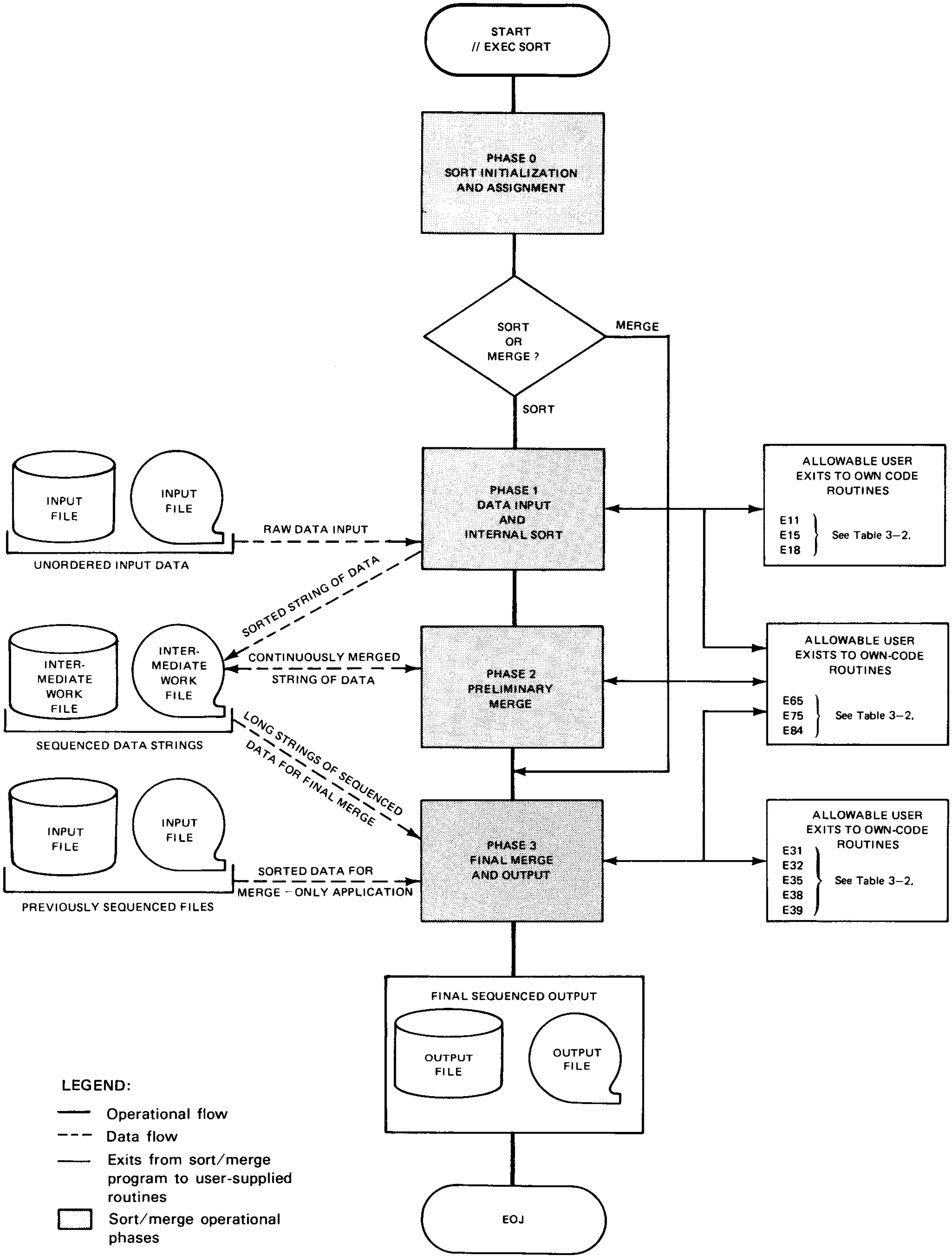


Figure 2—3. Independent Sort/Merge Operational Phases

2.3.2. Sort/Merge Operation Phases

2.3.2.1. Phase 0: Sort Initialization and Assignment

Phase 0 initializes the sort process by reading sort control statements from the job control stream (Figure 2—4). It validates both the content and syntax of these statements and then passes control to an assignment segment of the phase. By examining your parameters, the assignment segment determines the type of sort function to be performed (tape, disk, internal sort, or merge-only) and calculates and structures the sort/merge processor to perform only the sort functions you specify. When the assignment segment completes its task, phase 0 passes control to phase 1 or, in a merge-only procedure, to phase 3.

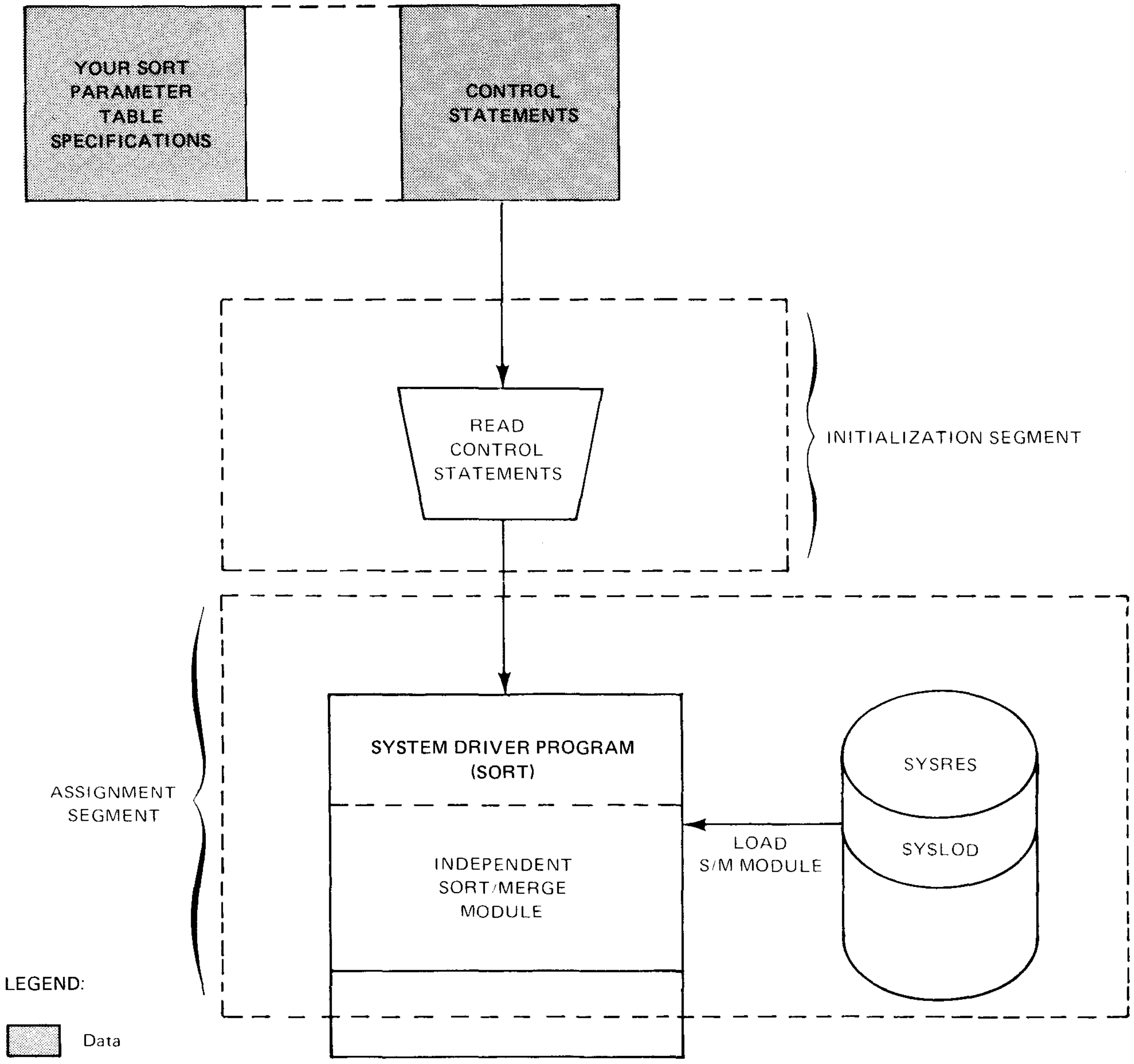


Figure 2—4. Data to Independent Sort/Merge Program (Phase 0)

2.3.2.2. Phase 1: Data Input and Internal Sort

When phase 1 receives control at the- end of phase 0, it initiates the input routine and performs internal sort operations. The input routine can be supplied by independent sort/merge or by you. If you use your own input routine, you must identify it to the independent sort/merge via a user exit code (3.3). The input routine opens input files, validates file labels, and reads data records one at a time before giving them, in successive order, to the internal sort routine for initial sorting. Internal sorting in main storage produces strings of sequenced data that are written as intermediate files to auxiliary storage devices (tape or disk). If the number of data strings produced during the internal sort are few enough to be merged in one final merge, phase 2, the preliminary merge, is bypassed and control passes to phase 3 for final merging and output to the output file. Otherwise, strings of sequenced data must be continuously merged into larger and larger data strings until only one final merge operation is required to produce an output file sequenced in the order you specified. Figure 2—5 illustrates data flow from the input file through internal sort processing.

When the internal sort is completed, control passes to either phase 2 or phase 3.

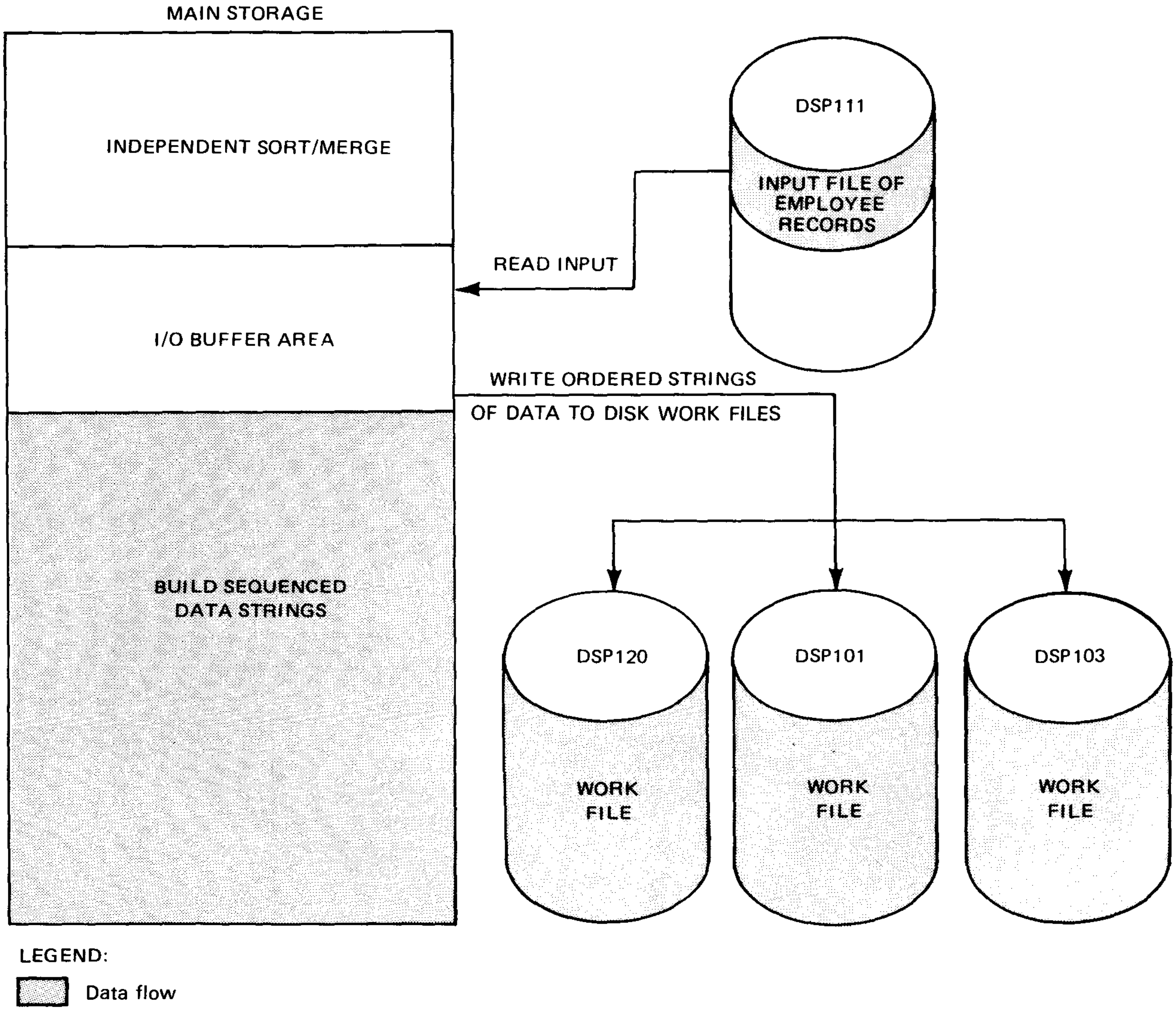


Figure 2—5. Data Route (Phase 1)

2.3.2.3. Phase 2: Preliminary Merge

When phase 2 receives control, the module executed for it continuously merges data strings produced in phase 1. These merge passes occur between auxiliary storage devices, each successive merge producing longer and longer sequenced data strings. When only one final merge pass is needed to create a single sequenced string (final output string), phase 2 passes control to phase 3. Figure 2—6 shows long sequenced data strings ready to be given to the final merge phase as a result of phase 2 operations.

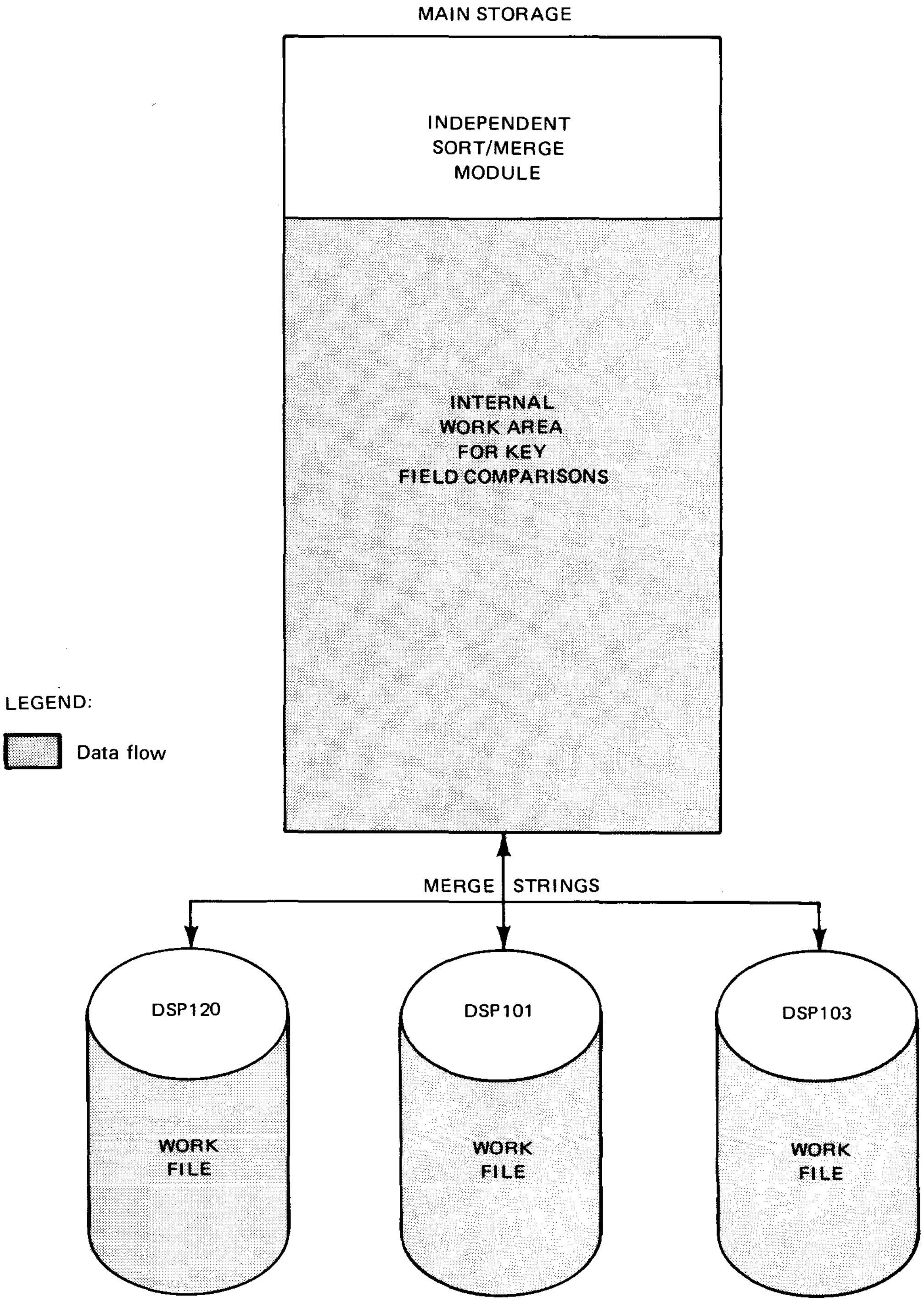


Figure 2—6. Data Route (Phase 2)

When you use the input data records shown in Figure 1—2, the record strings produced by phase 2 are:

FIRST STRING:

Key Field

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

SECOND STRING:

Key Field

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

THIRD STRING:

⋮

2.3.2.4. Phase 3: Final Merge

The final merge phase merges all strings written to the intermediate files into one sequenced string and passes it either to the independent sort/merge output routine or to your own output routine. If you provide the output routine, you must identify the exit code required to transfer control to your routine. User exit codes allowed during this phase are described in 3.3.2.4, 3.3.2.5, 3.3.2.6, 3.4.2, and 3.4.3. When you use your own output routine, the final output data string passes to your routine for final disposition. Otherwise, control passes to the independent sort/merge output routine which opens the output file, writes the output sequenced data, closes the output file, and returns control to the supervisor. Figure 2—7 depicts data flow from auxiliary storage devices to the output file via phase 3.

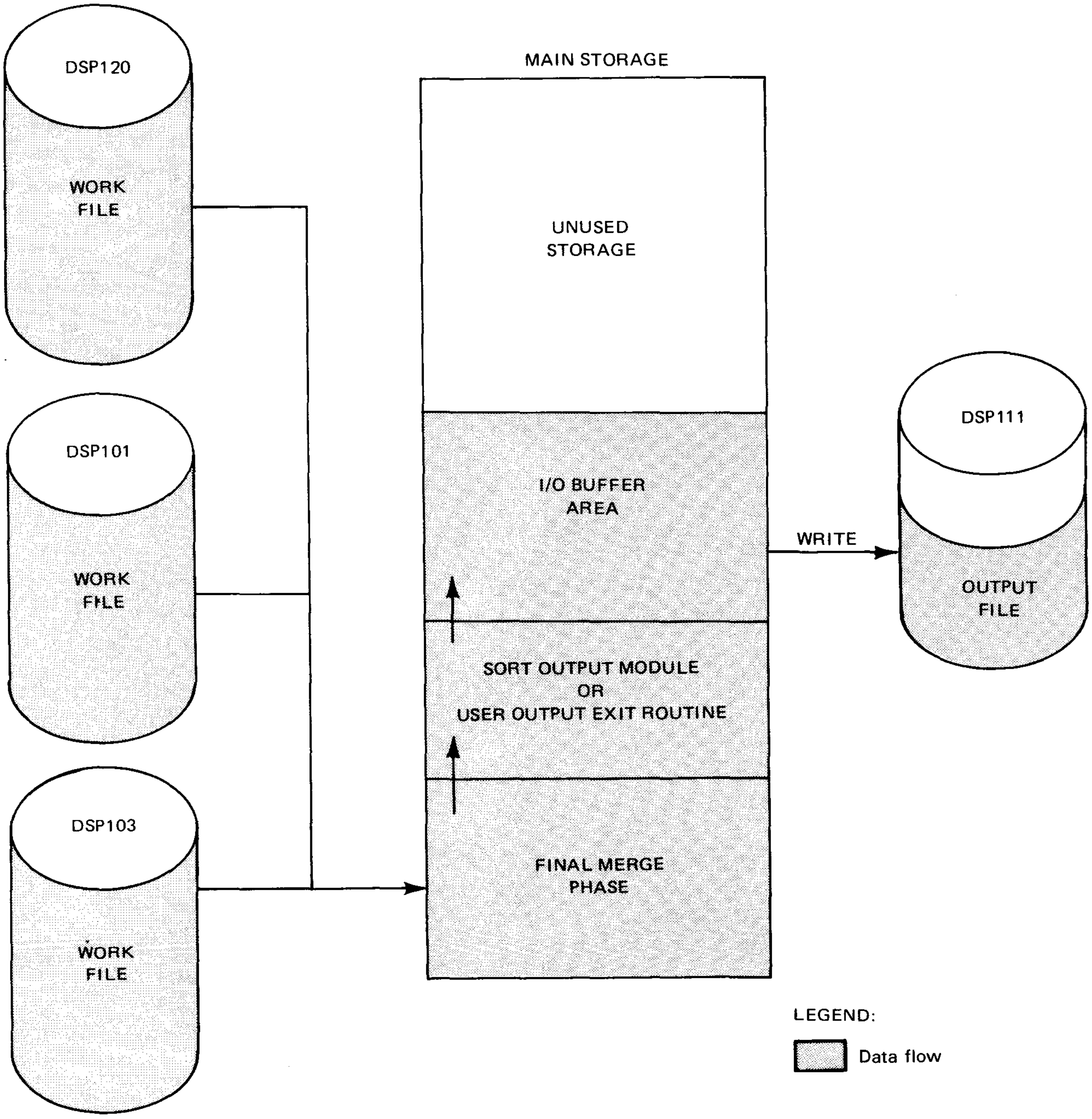


Figure 2—7. Data Route (Phase 3)

Contents of a file after phase 3 final merge processing would appear as follows:

	Key Field								
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	

To compare file contents before and after the sort, refer to Figures 1—2 and 1—3.

General-purpose registers and a branch table provide the interface required to link either independent sort/merge or your own modules with a particular phase of operation. Refer to 3.3.4 and 3.3.5 for more information about the role general-purpose registers and the branch table play in independent sort/merge.

3. Independent Sort/Merge Requirements You Supply

3.1. GENERAL

The independent 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 independent sort/merge modules to suit the requirements of a particular sort/merge operation (disk, tape, default, etc).

By answering these three questions concerning the independent 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 independent sort/merge 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).

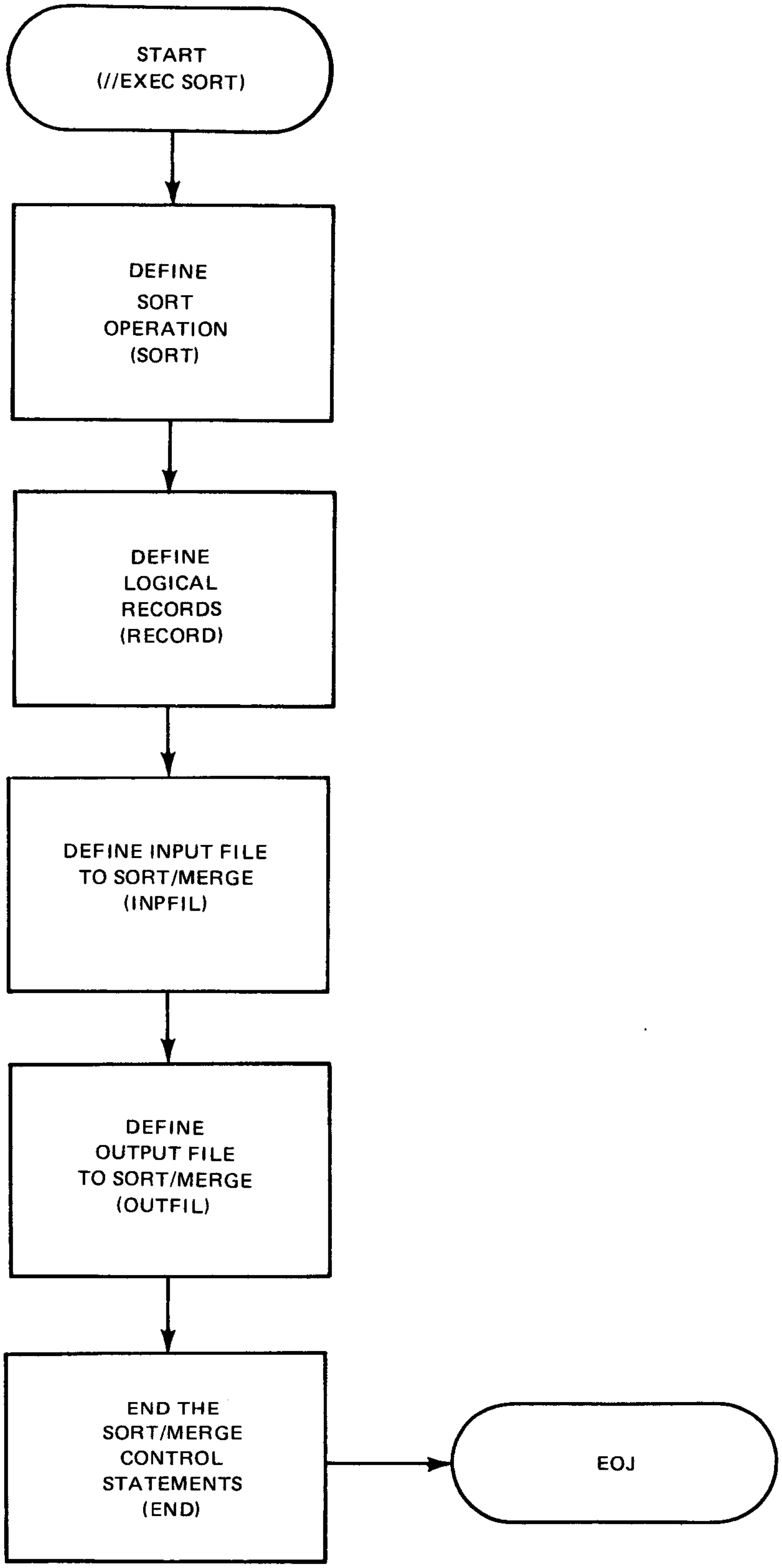


Figure 3—1. Disk Sort Program Flowchart Using Independent Sort/Merge

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, including the input data before the sort and the output data after the sort.

```
1        10        20
// JOB SRTEXMPL,,7ØØØ,9ØØØ,2
// DVC 2Ø // LFD PRNTR
// DVC 5Ø // VOL DSPØ28 // LBL INFILE  // LFD SORTIN1
// DVC 5Ø // VOL DSPØ28 // LBL OUTFILE // LFD SORTOUT,,INIT
// DVC 5Ø // VOL DSPØ28 // EXT ST,C,,CYL,5
// LBL $SCR1 // LFD DMØ1
// EXEC SORT
/$
 SORT FIELDS=(1,8,CH),DISC=1
 RECORD LENGTH=(8Ø),TYPE=F
 INPFIL BLKSIZE=4ØØ
 OUTFIL BLKSIZE=4ØØ
 END
/*
/&
// FIN
```

Figure 3—2. Independent Disk Sort Coding (Part 1 of 7)

DATA FILE BEFORE SORT

```
BLK# REC# BKSZ RCSZ  ........10 ........20 ........30 ........40 ........50 ........60 ........70 ........80

   1    1   80   80  00321654
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     0032165400 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   2    1   80   80  10007005
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     1000700500 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   3    1   80   80  68799863
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     6879986300 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   4    1   80   80  94600054
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     9460005400 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   5    1   80   80  20463844
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     2046384400 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   6    1   30   80  54486555
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     5448655500 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   7    1   80   80  03000600
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     0300060000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   8    1   80   80  88855296
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     8885529600 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   9    1   80   80  43300000
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     4330000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  10    1   80   80  70509300
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     7050930000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  11    1   80   80  DAYTONOH
                     CCEEDDDC44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     4183656800 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000
```

Figure 3—2. Independent Disk Sort Coding (Part 2 of 7)

DATA FILE BEFORE SORT (cont)

```
BLK# REC# BKSZ RCSZ  .........10 .........20 .........30 .........40 .........50 .........60 .........70 .........80
  12    1   80   80  STLOUISM
                     EEDDECED44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     2336492400 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  13    1   80   80  YORKPEN
                     EDDDDCD444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     8692755000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  14    1   80   80  NEWARKNJ
                     DCECDDDD44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     5561925100 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  15    1   80   80  MIAMIFLA
                     DCCDCCDC44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     4914963100 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  16    1   80   80  OH00001
                     DCFFFFF444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     6800001000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  17    1   80   80  M0000004
                     DDFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     4600000400 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  18    1   80   80  PA000007
                     DCFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     7100000700 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  19    1   80   80  NJ000012
                     DDFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     5100001200 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  20    1   80   80  FL000006
                     CDFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     6300000600 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  21    1   80   80  33655307
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     3365530700 0000000000 000000000C 0000000000 0000000000 0000000000 0000000000 0000000000

  22    1   80   80  10985469
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     1098546900 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000
```

Figure 3—2. Independent Disk Sort Coding (Part 3 of 7)

DATA FILE BEFORE SORT (cont)

```
BLK# REC# BKSZ RCSZ  ........10 ........20 ........30 ........40 ........50 ........60 ........70 ........80

  23    1   80   80  98654777
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     9865477700 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  24    1   80   80  68548833
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     6854883300 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  25    1   80   80  40675987
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     4067598700 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  26    1   80   80  77330659
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     7733065900 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  27    1   80   80  90675004
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     9067500400 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  28    1   80   80  09436750
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     0943675000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  29    1   80   80  11766325
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     1176632500 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  30    1   80   80  50964097
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     5096409700 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000
```

Figure 3—2. Independent Disk Sort Coding (Part 4 of 7)

DATA FILE AFTER SORT

```
BLK# REC# BKSZ RCSZ  .........10 .........20 .........30 .........40 .........50 .........60 .........70 .........80

   1    1  400   80  DAYTONOH
                     CCEEDDDC44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     4183656800 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   1    2  400   80  FL000006
                     CDFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     6300000600 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   1    3  400   80  MIAMIFLA
                     DCCDCCDC44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     4914963100 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   1    4  400   80  MO000004
                     DDFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     4600000400 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   1    5  400   80  NEWARKNJ
                     DCECDDDD44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     5561925100 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   2    1  400   80  NJ000012
                     DDFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     5100001200 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   2    2  400   80  OH00001
                     DCFFFFF444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     6800001000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   2    3  400   80  PA000007
                     DCFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     7100000700 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   2    4  400   80  STLOUISM
                     EEDDECED44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     2336492400 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   2    5  400   80  YORKPEN
                     EDDDDCD444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     8692755000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   3    1  400   80  00321654
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     0032165400 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000
```

Figure 3—2. Independent Disk Sort Coding (Part 5 of 7)

DATA FILE AFTER SORT (cont)

```
BLK# REC# BKSZ RCSZ  .........10 .........20 .........30 .........40 .........50 .........60 .........70 .........80

  3    2   400   80  03000600
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     0300060000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  3    3   400   80  09436750
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     0943675000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  3    4   400   80  10007005
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     1000700500 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  3    5   400   80  10985469
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     1098546900 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  4    1   400   80  11766325
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     1176632500 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  4    2   400   80  20463844
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     2046384400 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  4    3   400   80  33655307
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     3365530700 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  4    4   400   80  40675987
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     4067598700 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  4    5   400   80  43300000
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     4330000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  5    1   400   80  50964097
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     5096409700 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  5    2   400   80  54486555
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     5448655500 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000
```

Figure 3—2. Independent Disk Sort Coding (Part 6 of 7)

DATA FILE AFTER SORT (cont)

```
BLK# REC# BKSZ RCSZ  .........10 .........20 .........30 .........40 .........50 .........60 .........70 .........80

   5    3  400   80  68548833
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     6854883300 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   5    4  400   80  68799863
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     6879986300 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   5    5  400   80  70509300
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     7050930000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   6    1  400   80  77330659
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     7733065900 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   6    2  400   80  88855296
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     8885529600 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   6    3  400   80  90675004
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     9067500400 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   6    4  400   80  94600054
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     9460005400 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   6    5  400   80  98654777
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     9865477700 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000
```

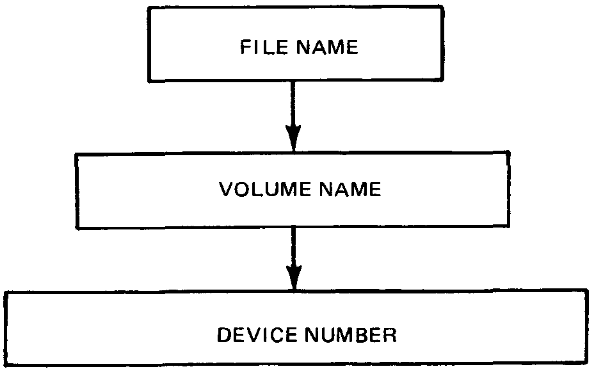
Figure 3—2. Independent Disk Sort Coding (Part 7 of 7)

According to the example job control stream (Figure 3—2), SRTEXMPL is the 8-character alphanumeric name of your job (line 1). 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. The number 2 indicates that no more than two tasks can be active at the same time in any job step. A task is a unit of work that the supervisor schedules.

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-8065 (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 (Figure 3—2, line 2).

Your next series of job control statements (lines 3 through 10) 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.



In this example (Figure 3—2), 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).

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 in Figure 3—2. 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-8065 (current version).

There are three important features to consider when building an independent sort/merge control stream.

1. No device assignment or execute statements are needed for the assembler because independent sort/merge does not need to be assembled.

2. To link file descriptions, LFD job control statements for independent sort/merge programs must use the standard names SORTINn and SORTOUT.

3. No linkage editor job control statements are required because independent sort/merge does not need to be linked. Thus, when you execute (EXEC) the sort program using independent sort/merge, the independent sort/merge interface module comes from the system load library file ($Y$LOD).

In part 1 of Figure 3—2, beginning with line 8, /$ indicates the start-of-data to the independent sort. Lines 9 through 13 are your parameters for the sort parameter table being structured for your disk sort program. Lines 14 through 16 indicate the end-of-data to the independent sort/merge, the end of job stream, and the end of the card reader operation. For details about job control and its language, see the job control user guide, UP-8065 (current version).

Figure 3—3 shows the job control stream required to execute your disk sort program using independent sort/merge. Figure 3—4 illustrates the results of your device assignment set specifications. This same job control stream can also be created and executed from a workstation. (See 3.5.)

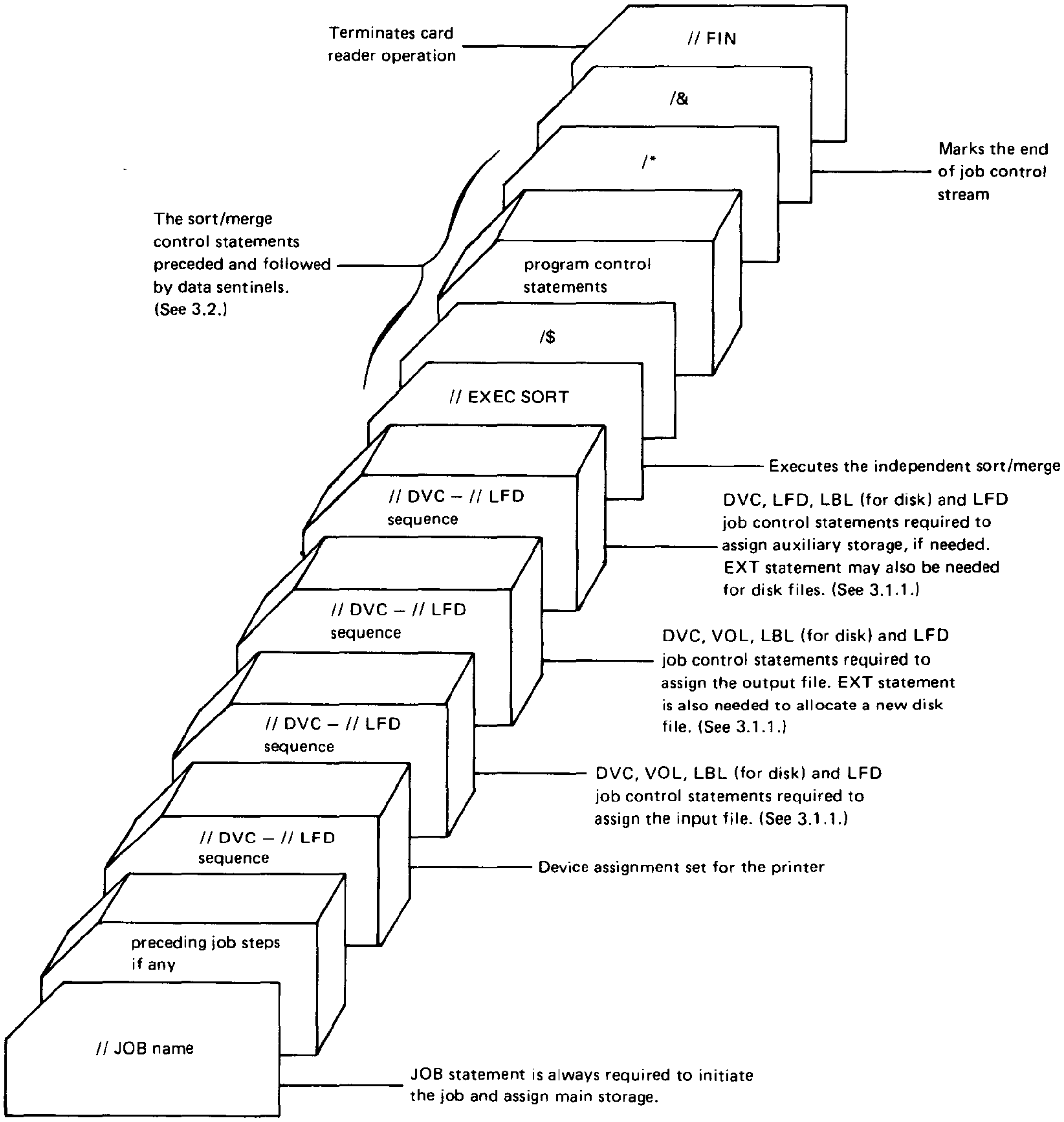

Terminates card reader operation
// FIN
/&
/*
Marks the end of job control stream
The sort/merge control statements preceded and followed by data sentinels. (See 3.2.)
program control statements
/$
// EXEC SORT
Executes the independent sort/merge
// DVC – // LFD sequence
DVC, LFD, LBL (for disk) and 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 name
JOB statement is always required to initiate the job and assign main storage.

Figure 3—3. Typical Job Control Stream for an Independent Sort/Merge Application

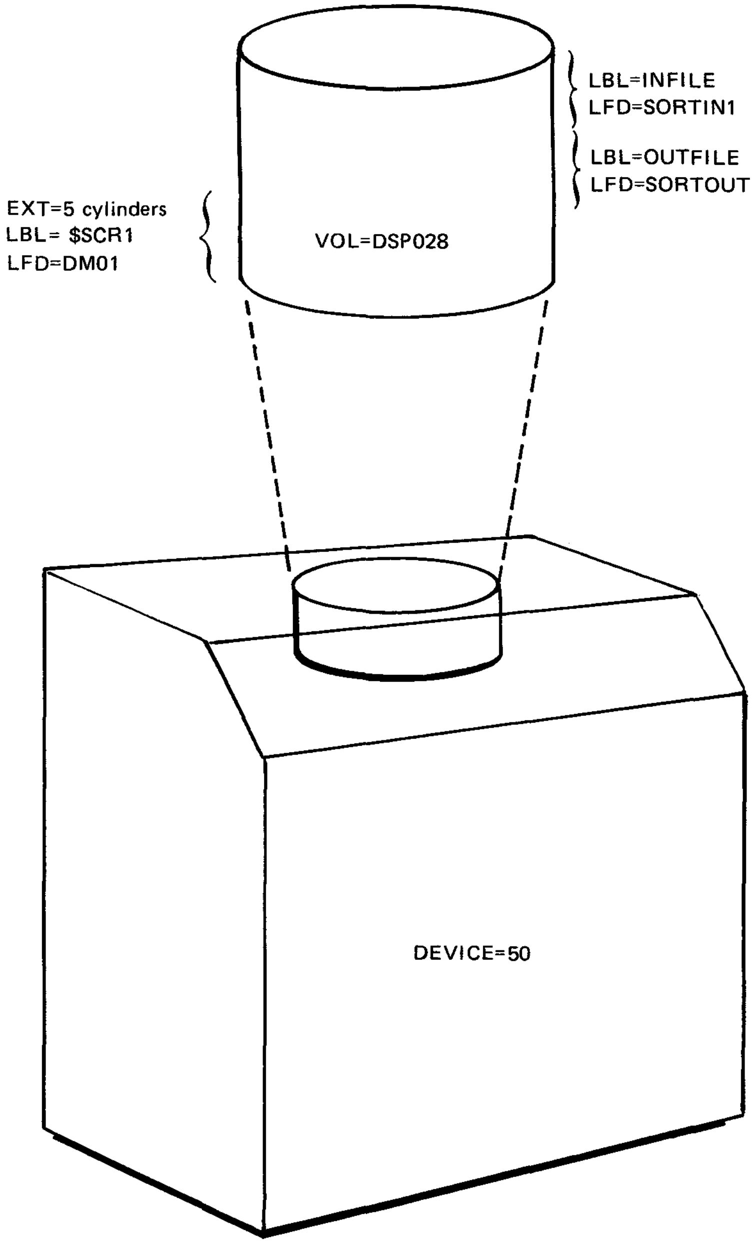


Figure 3—4. Device Assignment Results

3.2. SORT/MERGE CONTROL STATEMENTS

Sort/merge control statements are your way of providing the independent sort/merge with the information needed to sort and merge records of your input files. These statements, issued from the control stream, define the functional structure of the independent sort/merge by:

- defining the sort/merge to be performed;
- describing your records, the input and output files, and the sort key fields; and
- specifying any own-code routine that you may use during program execution.

The actual structure of the independent sort/merge is based on the entries contained in the sort parameter table. The initialization phase of independent sort/merge interprets and uses the parameters you specify in the control statements to write entries into the sort parameter table. (See Appendix B.)

Independent sort/merge uses eight 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.

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 output 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.

To understand the functions of the sort/merge control statements, we will examine each of them and relate them to the coding for our disk sort program.

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.

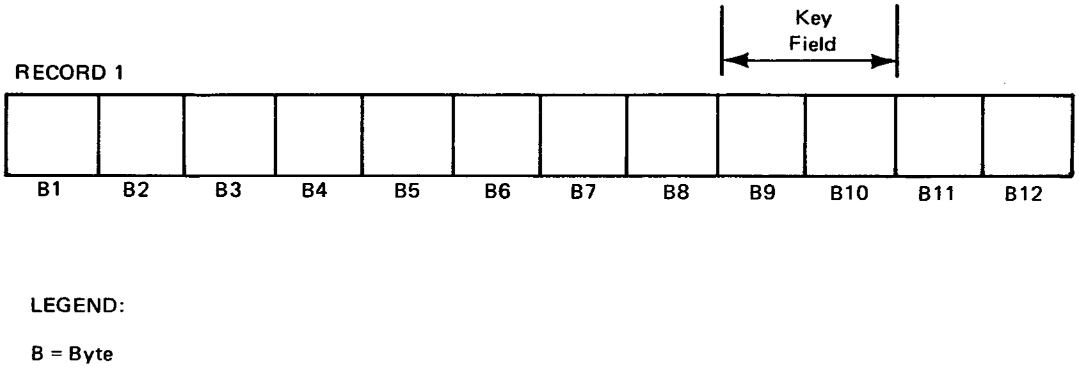
The format of the SORT control statement is:

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 (input-file-partition-number·output-file-partition-number [,...,input-file-partition-number·output-file-partition-number]) }]
		[, { DISC TAPE WORK } =number]
		[,FILE=number { number 1 }]
		[,NOCKSM= { D T }]
		[,SIZE=number]
		[,SORTP=(output-file-partition-number,input-file-1-partition-number [,...,input-file-9-partition-number])]
		[, { CHPT CHKPT }] Provided and accepted for compatibility with other systems; however, no action is performed by OS/3 sort/merge.

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 independent 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. The key field starting positions for independent sort/merge differ from the subroutine sort/merge and SORT3 programs in that starting positions are indicated by byte numbers instead of position numbers. For example, specifying 9 as *strt-pos* indicates that the most significant byte of the key field begins at byte 9 of the record as illustrated by the following diagram:



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.

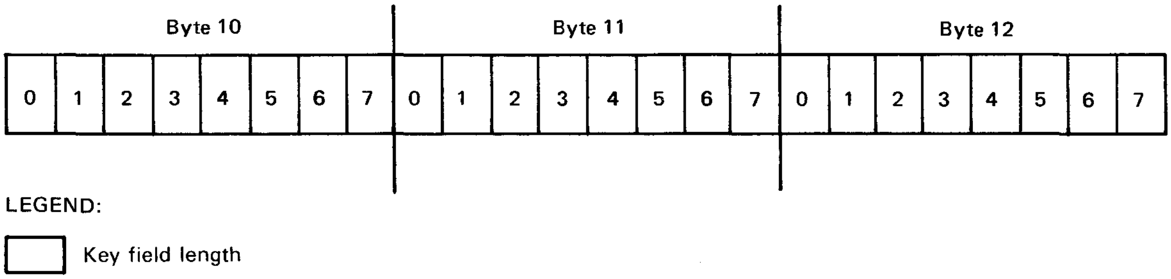
The *lgth* subparameter is a decimal number specifying the key field length in full bytes following any of these formats:

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.



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 (Part 1 of 2)

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

Table 3—1. Data Format Codes (Part 2 of 2)

Format Code	Description	Allowable Field Length
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. This parameter is optional.

```
     LABEL      ΔOPERATIONΔ          OPERAND
     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
```

Occasionally you may want to use multipartitioned disk files. Partitioning files simplifies sorting where only certain data are involved in the sort. Suppose you perform a sort on data records in one partition. At the end of the sort, you still want to write out the other partitions to the output partitioned file. The COPY keyword parameter is then required to copy disk input files directly into the output file without becoming involved in the sort. This keyword parameter may be used only when one input file is specified.

There are two ways to specify the COPY parameter. COPY=ALL specifies that data records contained in all input file partitions not involved in the sort are to be copied into the corresponding partitions of the output file.

```
LABEL    ΔOPERATIONΔ        OPERAND
1          10         16
--------------------------------------------------------------
           SORT       COPY=ALL
```

Figure 3—5 illustrates an action caused by this parameter.

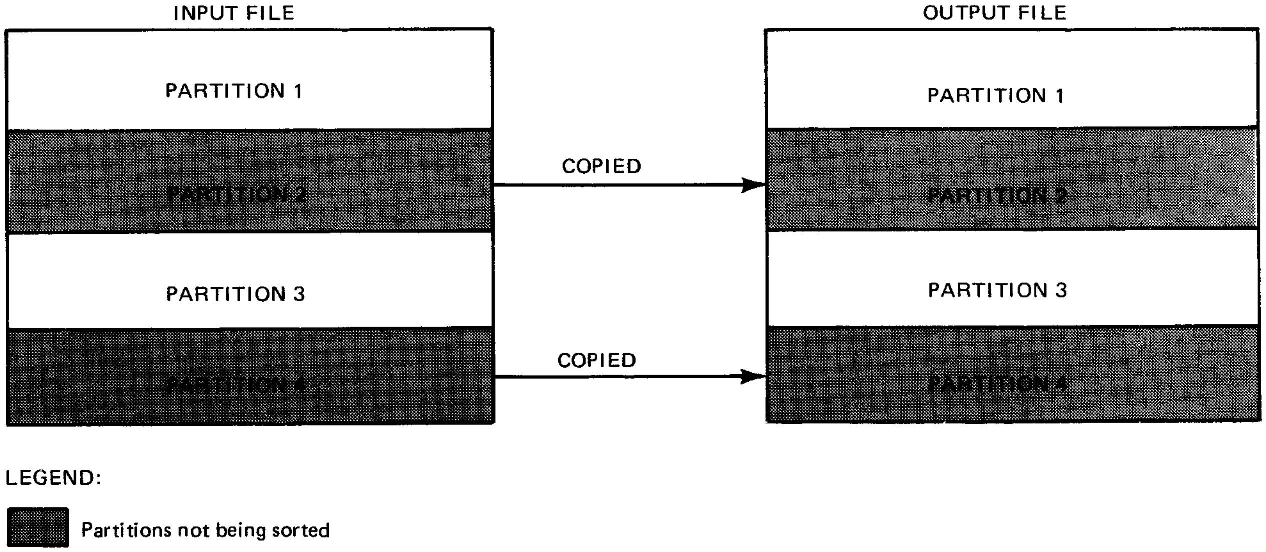


Figure 3—5. Copying Corresponding Partitions

Using the alternate specification, you would indicate the *input-file-partition-number* and *output-file-partition-number*. Each specification is a decimal number from 1 to 7. The first number identifies the input file partition from which data records are to be copied into the output file. The second number specifies the output file partition into which the data records of the previously specified input file partition is to be written. The following coding and Figure 3—6 illustrate the specifications needed to copy partitions 2 and 4 from the input file to partitions 1 and 2 of the output file.

```
           SORT       COPY=(2.1,4.2)
```

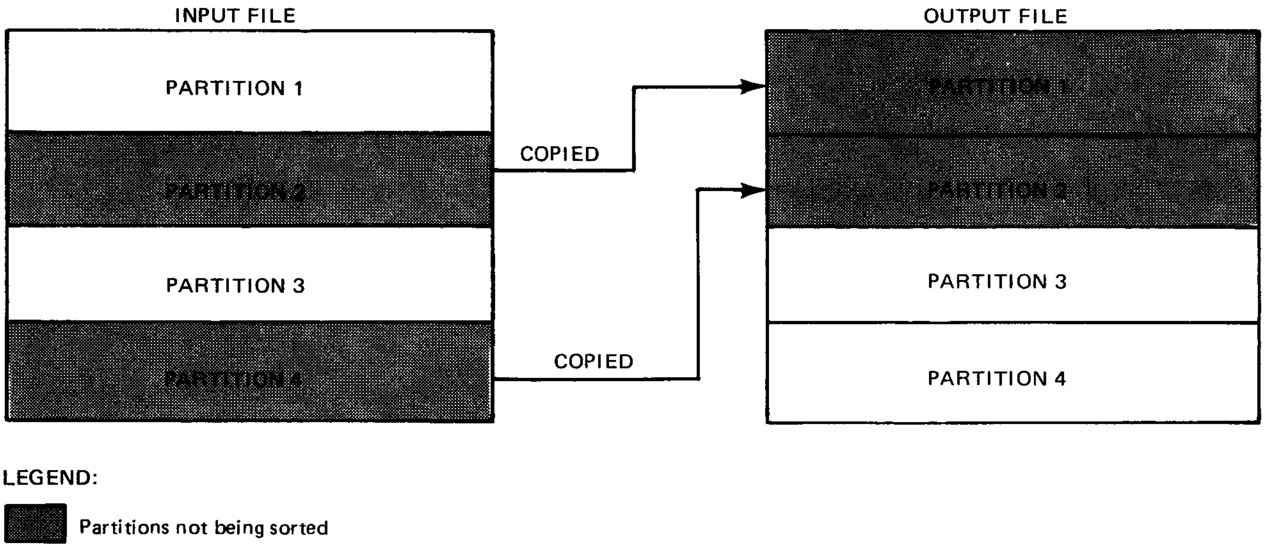


Figure 3—6. Copying Specific Partitions

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 independent sort/merge as working storage.

After designating the medium, you must specify a decimal number indicating the maximum number of files available to independent 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, independent 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 how this parameter could be added to the previously described FIELDS parameters. In line 1, the WORK parameter indicates three work files comprising tape or disk. In line 2, the DISC parameter indicates that three disks are to be used for work files.

```
      LABEL    ΔOPERATIONΔ        OPERAND
      1        10          16
     ____________________________________________________________
1.  |          SORT        FIELDS=(1,4,CH,A,1Ø,12,BI,A),WORK=3
2.  |          SORT        FIELDS=(1,4,,1Ø,12),FORMAT=CH,DISC=3
```

Independent 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 on LFD statements in the job control stream. If you have more than one input file and forget to code the FILE parameter, independent sort/merge will process only your first input file. The following coding indicates that two input files are to be entered into the sort.

```
LABEL    ΔOPERATIONΔ      OPERAND
1        10    16
------------------------------------------------------------------------
         SORT       FILE=2
```

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.

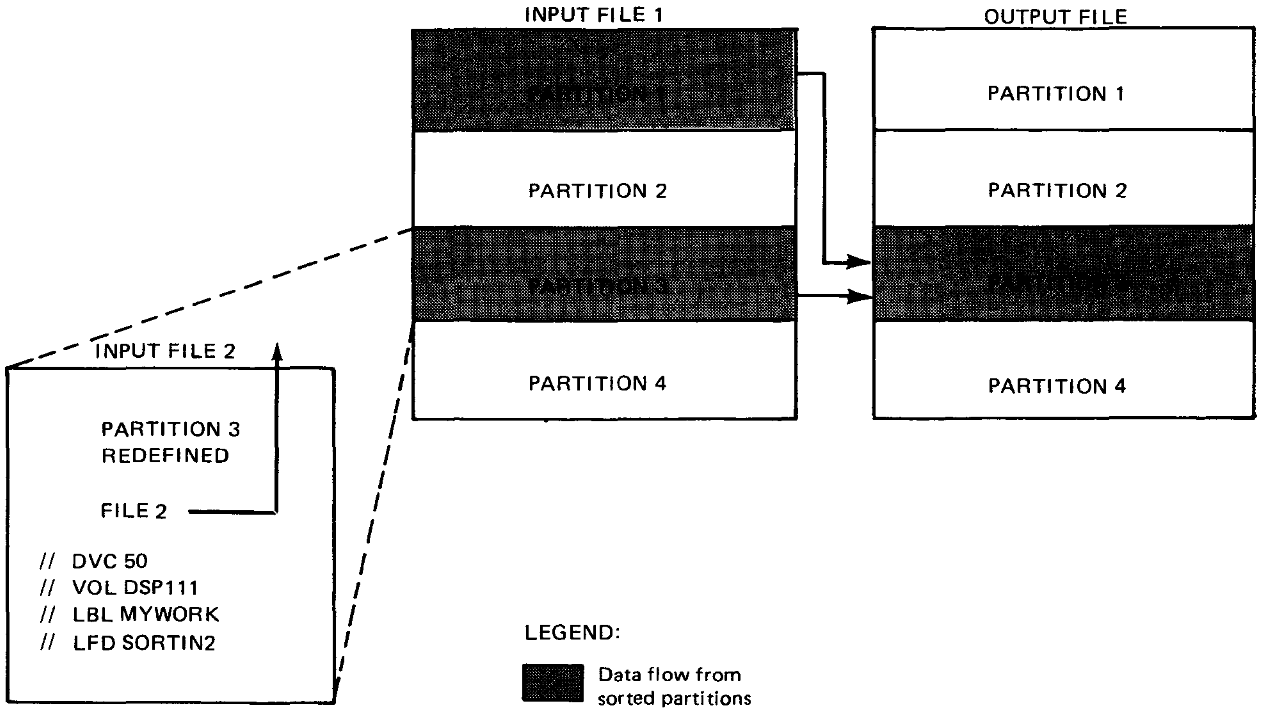
Another parameter, SIZE, specifies the approximate number of records in the input file. If you use the CALCAREA parameter in the OPTION sort control statement, the SIZE parameter is required for an accurate calculation of optimum sort time and disk work space. The following coding indicates that 3500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

```
         SORT       SIZE=35ØØ
```

Like the COPY parameter, which handles the transfer of disk input file partitions not involved in the sort to disk output file partitions, the SORTP parameter transfers those sorted partitions from the disk input file or files to a disk output file partition. The SORTP keyword parameter is required when the sort/merge operation involves data records read from or written to multipartitioned disk files. The first keyword subparameter of the SORTP specifies the *output-file-partition-number.* This is a decimal number from 1 to 7 that identifies the specific partition of an output file to which sorted data is to be written.

You may specify up to nine partitioned input files to be read to the sort. The input file specification format is *input-file-1-partition-number [,...,input-file-9-partition-number]*. The FILE keyword parameter in the SORT control statement specifies the exact number of input files involved in the sort. Each input file may be subdivided into a maximum of seven partitions. If you are sorting two or more partitioned input files, you can sort only one partition from each file. There is a way to make independent sort/merge sort more than one partition of an input file by causing it to treat the additional partition as a partition of a new file. To do this you redefine as a new input file each additional partition you want sorted from the original input file by writing an additional job control device assignment set in your job control stream. The following coding and Figure 3—7 illustrate the use of SORTP parameter.

```
LABEL    ΔOPERATIONΔ        OPERAND
1         10          16
------------------------------------------------------------------
          SORT        SORTP=(3,1,3)
```



```
// DVC 50
// VOL DSP111
// LBL MYWORK
// LFD SORTIN2
```

Figure 3—7. Moving Sorted Partitions

The three subparameter numbers on the SORTP parameter indicate that output file partition 3 receives the sorted data from partitions 1 and 3 of the input file. Although on the SORTP parameter you may specify more than one input partition from which data is received for sorting, the SORTP parameter is not put into effect until you redefine additional input files in your job control device assignment sets.

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, IRAM files, and when input processing is handled by a user exit routine (3.3). The RECORD sort control statement format is:

LABEL	ΔOPERATIONΔ	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)]

If input is from tape or IRAM disk files, you must include either a LENGTH or RCSZ parameter.

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).

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, independent 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.

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

The TYPE parameter specifies the type (D, F, or V) of records to be processed by the independent 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
```

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 independent 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 sizes, and a number specifying either the percentage or estimated number of the most frequently occurring records. If the number is less than 100, independent 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 independent 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.

```
      LABEL   ΔOPERATIONΔ         OPERAND
      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, independent sort/merge calculates bin size from the *lgth* specifications of the FIELDS parameter.

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 sort/merge control statement defines your input file to independent sort/merge and specifies open and close procedures for tape files. It is not required if input files are on disk, except for IRAM files. The BLKSIZE parameter must be specified for IRAM files; all other parameters are optional.

LABEL	ΔOPERATIONΔ	OPERAND
	INPFIL	$\left[\text{BLKSIZE=}\left\{\begin{matrix}\text{bytes}\\ \text{(bytes-1[,...,bytes-8])}\end{matrix}\right\}\right]$ [,BUFOFF=n] [,BYPASS] $\left[\text{,CLOSE=}\left\{\begin{matrix}\text{NORWD}\\ \text{RWD}\\ \text{RWI}\\ \text{UNLD}\end{matrix}\right\}\right]$ $\left[\text{,DATA=}\left\{\begin{matrix}\text{A}\\ \text{E}\end{matrix}\right\}\right]$ [,EXIT] $\left[\text{,INTERLACE=}\left\{\begin{matrix}\text{factor}\\ \text{(factor-file-1[,...,factor-file-n])}\end{matrix}\right\}\right]$ $\left[\text{,OPEN=}\left\{\begin{matrix}\text{NORWD}\\ \text{RWD}\end{matrix}\right\}\right]$ [,SKIPBYTE=n] $\left[\text{,VOLUME=}\left\{\begin{matrix}\text{vol}\\ \text{(vol-1[,...,vol-8])}\end{matrix}\right\}\right]$ Provided and accepted for compatibility with other systems; however, no action is performed by OS/3 sort/merge.

Figure 3—8 shows coding examples referenced in the following INPFIL control statement discussion.

```
      LABEL    ΔOPERATIONΔ      OPERAND
      1          10          16
    ┌──────────────────────────────────────────────────────
 1. │           INPFIL      BLKSIZE=8ØØ
 2. │           INPFIL      BLKSIZE=(8ØØ,12ØØ,16ØØ)
 3. │           INPFIL      BUFOFF=3Ø,DATA=A
 4. │           INPFIL      BLKSIZE=8ØØ,BYPASS,CLOSE=NORWD
 5. │           INPFIL      EXIT
```

Figure 3—8. INPFIL Control Statement Coding Examples

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. Line 1 of Figure 3—8 illustrates the first format. 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. Line 2 of Figure 3—8 illustrates three input files, each of a different block size. If you omit the BLKSIZE parameter and also the RCSZ keyword parameter on the RECORD control statement, independent sort/merge assumes that all input blocks are the size of the first block processed. You must include the BLKSIZE parameter if you have IRAM files.

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. Figure 3—8, line 3 shows this parameter as well as the data format parameter specifying ASCII code. These two parameters are usually coded together.

Another optional INPFIL parameter is the BYPASS parameter. It has no associated values but when you specify BYPASS, you direct the independent sort/merge input phase to ignore all unreadable blocks of data on the input file. Independent sort/merge does not keep a record of the blocks ignored. Figure 3—8, line 4 shows an 800-block input file for which all unreadable data blocks are to be ignored by the sort/merge input phase.

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 which 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. Figure 3—8, line 4 shows that no rewinding is performed upon closing the input file.

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 coding examples of Figure 3—8, since no data format is specified, the system assumes a normal default condition of E (EBCDIC) except on line 3, where ASCII data format is specified.

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. Figure 3—8, line 5 shows this coding by indicating that you want to read the input file via your own input routine.

Data management has a special feature called the interlace feature. Interlacing is a method of recording data records on a disk file so that more than one physical record may be written or accessed during each disk rotation. To properly access data records from such an input file, your program must provide the independent sort/merge with the interlace factor used during file creation. The INTERLACE keyword parameter has two formats. The first specifies the interlace factor for a single input file, the second for multiple input files. The following coding illustrates the two formats:

```
LABEL    ΔOPERATIONΔ      OPERAND
1        10       16
-------------------------------------------------------
         INPFIL   INTERLACE=4
         INPFIL   INTERLACE=(4,3,4)
```

The interlace fator is based upon many variables: time frame (internal between data records); block size (number of physical records per track); disk rotational speed; and I/O time (time required to read or write a record). The data management user guide, UP-8068 (current version) provides a detailed explanation of interlace.

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

1. 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 pre-positional location.

2. 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. Thus, for each of the coding lines of Figure 3—8, input tape files are rewound to load point on opening.

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:

```
LABEL    ΔOPERATIONΔ       OPERAND
1         10         16
----------------------------------------------------------------
          INPFIL      SKIPBYTE=1Ø
```

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 coding example, the first data record starts at byte 11.

3.2.4. Defining the Output File

The OUTFIL control statement defines output procedures to independent 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      ΔOPERATIONΔ   OPERAND

           OUTFIL        ┌          ( bytes                                          ) ┐
                         │BLKSIZE= <  (bytes-partition-1,bytes-partition-2          >  │
                         └          (    [,...,bytes-partition-7] )                  ) ┘
                         [,BUFOFF=n]
                         ┌        ( NORWD ) ┐
                         │,CLOSE=<  RWD    >│
                         │        ( RWI   ) │
                         └        ( UNLD  ) ┘
                         [,EXIT]
                         ┌           ( IRAM ) ┐
                         │,FILTYPE= <  NI    >│
                         └           ( SAM  ) ┘
                         ┌             ( factor                                          ) ┐
                         │,INTERLACE= <  (factor-partition-1,factor-partition-2         >  │
                         └             (    [,...,factor-partition-7] )                  ) ┘
                         [,NOTPMK]
                         ┌        ( number-of-partitions ) ┐
                         └,NPTN= (           1          ) ┘
                         ┌        ( NORWD ) ┐
                         └,OPEN= ( RWD   ) ┘
                         ┌,RCSZ=(max-bytes-partition-1,max-bytes-partition-2 ┐
                         └          [,...,max-bytes-partition-7] )           ┘
                         ┌       ( percentage                                            ) ┐
                         │,SIZE= < (percentage-partition-1,percentage-partition-2         > │
                         └       (    [,...,percentage-partition-7] )                    ) ┘
                         ,TYPE=(partition-1-type,partition-2-type[,...,partition-7-type] )
                         ┌,UOS=(ext-percent-partition-1,ext-percent-partition-2 ┐
                         └          [,...,ext-percent-partition-7] )            ┘
```

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, the output file is to have the same block size and record size as the input files, and output is to a single-partition file or a previously-defined multipartitioned file. 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.

The BLKSIZE parameter can specify:

- the number of bytes in the output data block written to a tape or single-partition disk output file (format 1); or
- the number of bytes in the output data blocks written to specific partitions in a multipartitioned disk output file (format 2).

If block size is needed and you do not specify the BLKSIZE parameter or the RCSZ parameter in any sort control statement, independent sort/merge assumes a block size equal to the input block. In the following coding, line 1 indicates that you are writing 400-byte data blocks to the output file. Line 2 indicates that you are writing output data blocks to three partitions on a disk output file. The first partition receives 400-byte output data blocks; the second partition receives 150-byte data blocks; and the third partition receives 640-byte data blocks.

```
       LABEL   ΔOPERATIONΔ        OPERAND
       1         10        16
     ------------------------------------------------------------
1.   |           OUTFIL    BLKSIZE=4ØØ
2.   |           OUTFIL    BLKSIZE=(4ØØ,15Ø,64Ø)
3.   |           OUTFIL    BUFOFF=2Ø
```

The BUFOFF parameter is used in the OUTFIL control statement, for specifying 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 in line 3 indicates an adjustment of 20 bytes for an ASCII format file.

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.

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.

By using the FILTYPE parameter, you can create an indexed random access method, nonindexed, or sequential access method output file. This parameter may be omitted if the output file is to be the same type as the input file. IRAM output files are always nonindexed, even if the input file has an index. If you are operating in the mixed data management environment with tape input and you want disk output, the disk output file will always be a nonindexed MIRAM file.

The INTERLACE parameter is required if you want to use the data management interlace feature to create the data output file for the independent sort/merge. On output processing, interlacing is a method of recording more than one physical record on a disk for each rotation. In order to write an interlaced output file, your program must give independent sort/merge the interlace factor required to create the file. You specify this factor number in the INTERLACE parameter. For a more detailed explanation of the variables used to determine the interlace factor, see the data management user guide, UP-8068 (current version). Line 1 of the following coding example specifies an interlace factor of 4 to create a single-partition output file. Line 2 specifies interlace factors for four partitions of a multipartitioned output disk file. A maximum of seven partition factors may be specified.

```
      LABEL   ΔOPERATIONΔ        OPERAND
      1        10        16
    ┌─────────────────────────────────────────────
1.  │           OUTFIL     INTERLACE=4
2.  │           OUTFIL     INTERLACE=(3,7,4,5)
```

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.

When you plan to use a multipartitioned disk output file, you must specify the number of partitions to be created in that file via the NPTN parameter. The decimal number you specify must be from 1 to 7. For example, the following coding indicates that the output disk file contains four partitions.

```
              OUTFIL     NPTN=4
```

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.

In addition to the block size, you can also indicate the maximum number of bytes in records written to each partition of a multipartitioned disk output file. The RCSZ parameter does this. You may specify up to seven maximum record sizes — one for each partition. The following coding example indicates that output records are written to three disk partitions. Each number within the parentheses represents the maximum number of bytes for records written to partitions 1, 2, and 3, respectively.

```
LABEL    ΔOPERATIONΔ       OPERAND
1         10        16
------------------------------------------------------
          OUTFIL        RCSZ=(8Ø,12Ø,2Ø5)
```

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

To indicate to independent sort/merge the size of each partition on the output disk, you use the SIZE parameter. The size specifies a *percentage* (a 1- or 2-digit decimal number). There are two formats for the SIZE parameter. The first applies to a single-partition output disk file occupying less than 100% of the available file space. Suppose you coded:

```
          OUTFIL        SIZE=5Ø
```

Your disk partition size would be 50% of the file space available (Figure 3—9).

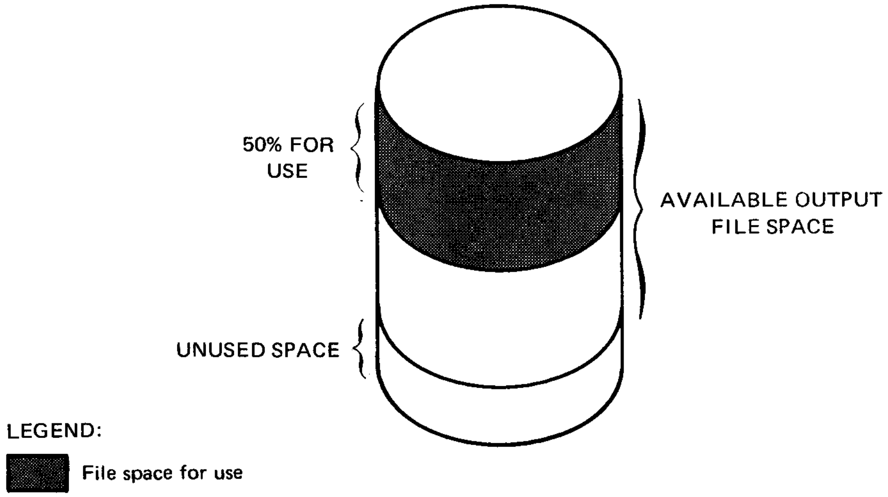


Figure 3—9. Partition Sizing for Single-Partition Output Disk Files

If you do not specify the SIZE parameter, independent sort/merge assumes a single-partition output disk file occupying 100% of the available file space.

The second SIZE parameter *(percentage-partition-1, percentage-partition-2[,...,percentage-partition-7])* format can specify file space assigned to specific partitions of a multipartitioned disk output file. Since a maximum of seven partitions are allowed, you may specify up to seven percentages on this parameter. The following coding specifies a disk output file with three partitions. The first partition would use 30% of the output disk file; the second partition, 45%; and the third partition, 15%. Figure 3—10 shows how your output disk file space would be partitioned.

```
LABEL    ΔOPERATIONΔ      OPERAND
1        10       16
_________________________________________________________
         OUTFIL       SIZE=(3Ø,45,15)
```

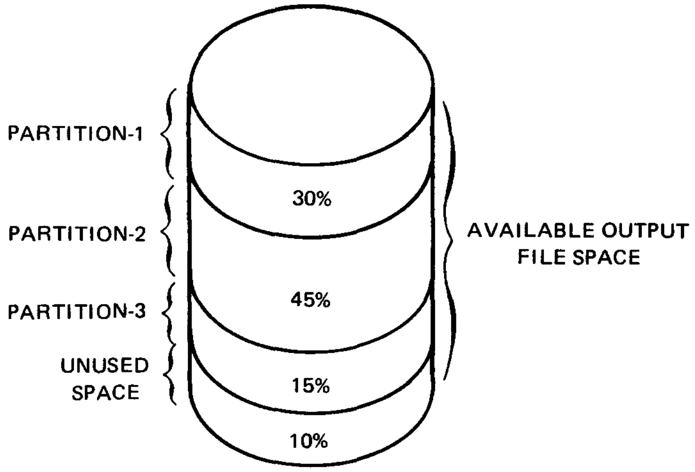


Figure 3—10. Partition Sizing for Multipartitioned Output Disk Files

You can tell independent sort/merge the type of record you write to each partition of a multipartitioned file via the TYPE parameter. Indicate a record type for each partition by choosing one of the following specifications:

- D ASCII variable-length records
- F Fixed-length records
- V Variable-length records

If you omit this parameter, the fixed type is assumed by default.

Suppose each of the three partitions you described in the previous SIZE parameter example has a different record type: partition-1 contains ASCII variable-length records; partition-2, fixed-length records; and partition-3, variable-length records. Your TYPE parameter would have to include at least the D and V specifications. If you coded a comma for partition-2 type record as follows, independent sort/merge would supply the default condition of fixed-length records.

```
         OUTFIL       TYPE=(D,,V)
```

After you've assigned percentages of disk output file space to specific partitions, your number of records might increase and you might find that you exceed the amount of output file space allocated for certain partitions. The UOS parameter solves this problem by allowing the partitions to be dynamically extended by data management when they become 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 allocated. In the UOS parameter, you indicate what percentage of that amount you want each partition extended by when it requires more space. You can specify up to 100% for each partition. If you want to extend a partition by 100%, you can specify 100 for that partition or you can omit the specification because the default is 100%. If you want to extend a partition by less than 100, you must specify a percentage for that partition in the UOS parameter. Suppose, for example, you specified five cylinders for the third parameter of the EXT statement. If you have a 3-partition file, you might want to specify 100% for the first and second partition and 20% for the third partition.

```
LABEL    ΔOPERATIONΔ        OPERAND
1         1Ø          16
-------------------------------------------------------------
          OUTFIL      UOS=(1ØØ,,2Ø)
```

If the first or second partition becomes filled, data management will extend the size of that partition by five cylinders. If five additional cylinders aren't enough, data management will extend the partition by another five cylinders. If the third partition is used up, data management will extend it by one cylinder at a time.

3.2.5. Ending Input to Sort/Merge

The END control statement is optionally used to notify independent 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:

```
LABEL    ΔOPERATIONΔ        OPERAND
1         1Ø          16
-------------------------------------------------------------
          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 Independent Sort/Merge Specifications

The OPTION control statement consists of optional parameters that supply independent 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	$\left[\text{ADDROUT=}\begin{Bmatrix}\text{A}\\\text{D}\end{Bmatrix}\right]$
		$\left[,\begin{Bmatrix}\text{CALCAREA}\\\text{CALCAREA=}\begin{Bmatrix}\text{NO}\\\text{YES}\end{Bmatrix}\end{Bmatrix}\right]$
		$\left[\text{,CSPRAM=}\begin{Bmatrix}\text{NO}\\\text{OPTION}\\\text{YES}\end{Bmatrix}\right]$
		[,KEYLEN=bytes]
		[,LABEL=(output,input-1 [,...,input-n] ,work)]
		$\left[\text{,PRINT=}\begin{Bmatrix}\text{ALL}\\\text{CRITICAL}\\\text{NONE}\end{Bmatrix}\right]$
		$\left[\text{,RESERV=}\begin{Bmatrix}\text{work-file-name}\\\text{(work-file-name[,output-file-name])}\end{Bmatrix}\right]$
		[,RESTART]
		$\left[\text{,SHARE=}\begin{Bmatrix}\text{work-file-name}\\\text{(work-file-name[,input-file-name])}\end{Bmatrix}\right]$
		[,STORAGE=bytes]
		[,VERIFY]
		$\left.\begin{matrix}\text{[,ALTWK]}\\\text{[,DUMP]}\\\text{[,ERASE]}\\\text{[,ROUTE]}\\\text{[,SORTIN]}\\\text{[,SORTOUT]}\\\text{[,SORTWK]}\end{matrix}\right\}$ Provided and accepted for compatibility with other systems; however, no action is performed by OS/3 sort/merge.

For compatibility with other systems, OS/3 independent sort/merge accepts but does not act upon seven parameters: ALTWK, DUMP, ERASE, ROUTE, SORTIN, SORTOUT, and SORTWK.

You may specify the other 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). Figure 3—11 contains coding examples pertaining to the following parameter discussion.

```
      LABEL    ΔOPERATIONΔ         OPERAND
      1        10        16                                               72
1.             OPTION    ADDROUT=D,CALCAREA=YES,CSPRAM=YES
2.             OPTION    STORAGE=18ØØØ,KEYLEN=1Ø,PRINT=NONE,VERIFY
3.             OPTION    STORAGE=18ØØ,LABEL=(S,S,S),RESERV=(SMØ6),            C
4.                       SHARE=(SMØ1)
```

Figure 3—11. OPTION Control Statement Coding Examples

The ADDROUT parameter is required when independent sort/merge must perform a *tag sort*. The tag sort performed by independent sort/merge 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 or IRAM file. Multiple input files cannot be tag sorted. If input is from an IRAM file, the output of the tag sort will be an IRAM file without an index.

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. Figure 3—11, line 1, illustrates the ADDROUT parameter. Figures 3—12, 3—13, and 3—14 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 Figures 3—12, 3—13, and 3—14 but to illustrate the concept of record sorting by key fields and the outputs produced by a tag sort operation.

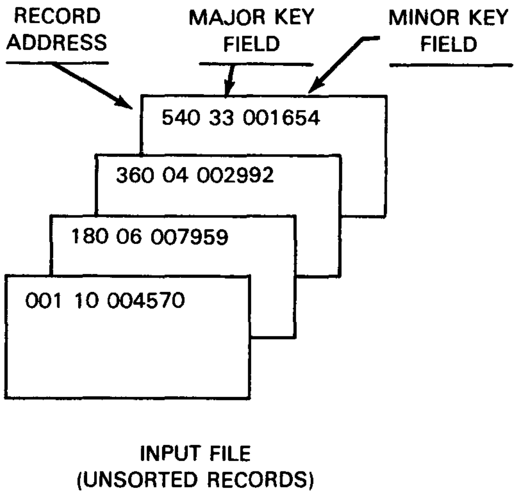


Figure 3—12. Input File, Unsorted Records (Additional Data Fields Not Shown)

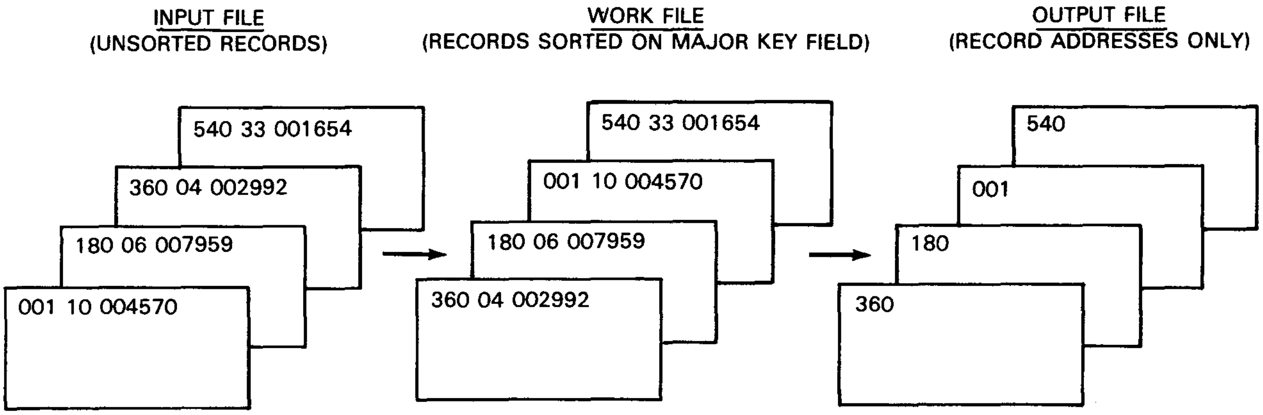


Figure 3—13. Tag-Sorted Output File when ADDROUT=A

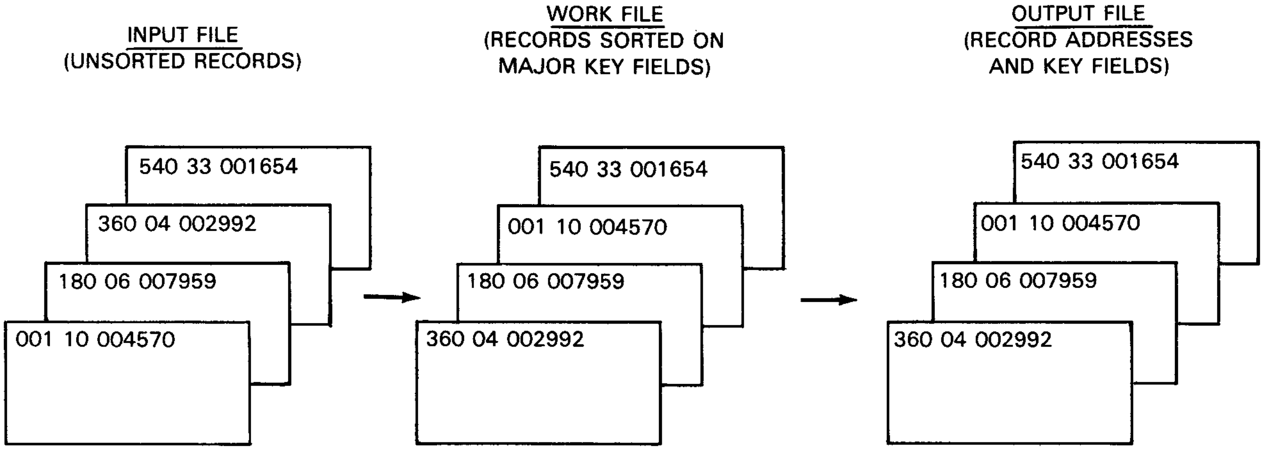


Figure 3—14. Tag-Sorted Output File when ADDROUT=D

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

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. Figure 3—11, line 2 shows a KEYLEN parameter coding example. If you do not specify the KEYLEN parameter, it is assumed that blocks do not have keys.

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 exits E11 and E31. 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. Figure 3—11, line 3 illustrates the coding of a LABEL parameter, indicating standard labels for output, input, and work files.

By coding the RESERV parameter, you can reserve for your output data file a tape unit assigned to the independent 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. Figure 3—15 illustrates this.

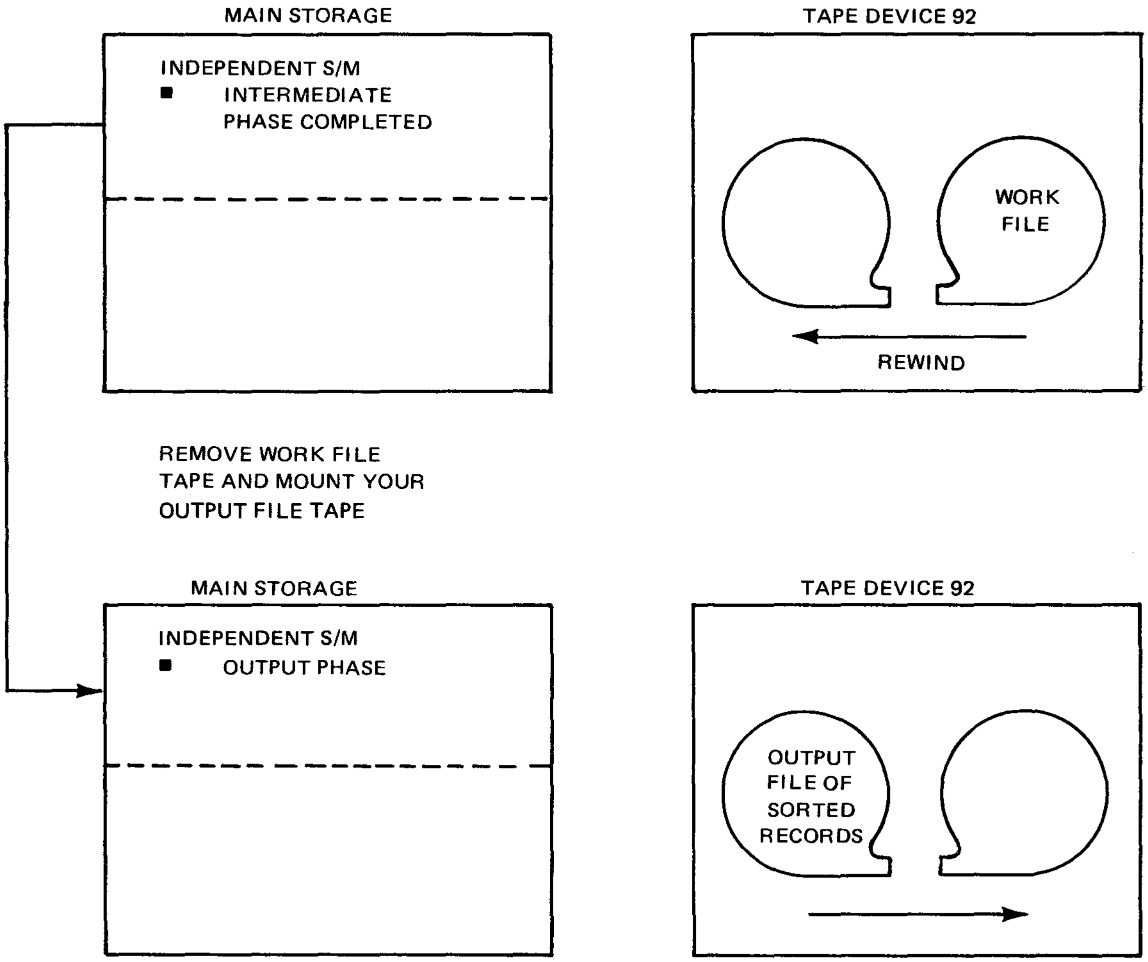

MAIN STORAGE
TAPE DEVICE 92
INDEPENDENT S/M
■ INTERMEDIATE PHASE COMPLETED
WORK FILE
REWIND
REMOVE WORK FILE TAPE AND MOUNT YOUR OUTPUT FILE TAPE
MAIN STORAGE
TAPE DEVICE 92
INDEPENDENT S/M
■ OUTPUT PHASE
OUTPUT FILE OF SORTED RECORDS

Figure 3—15. Same Work File Device Reserved for Output File Processing

Independent 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.

In like manner, the SHARE parameter specifies the double use of one tape device by input and work files. It allows a tape unit assigned to the independent sort/merge to be used as the input device during the input phase of sort/merge operation and as a sort work file during the intermediate and output phases (Figure 3—16).

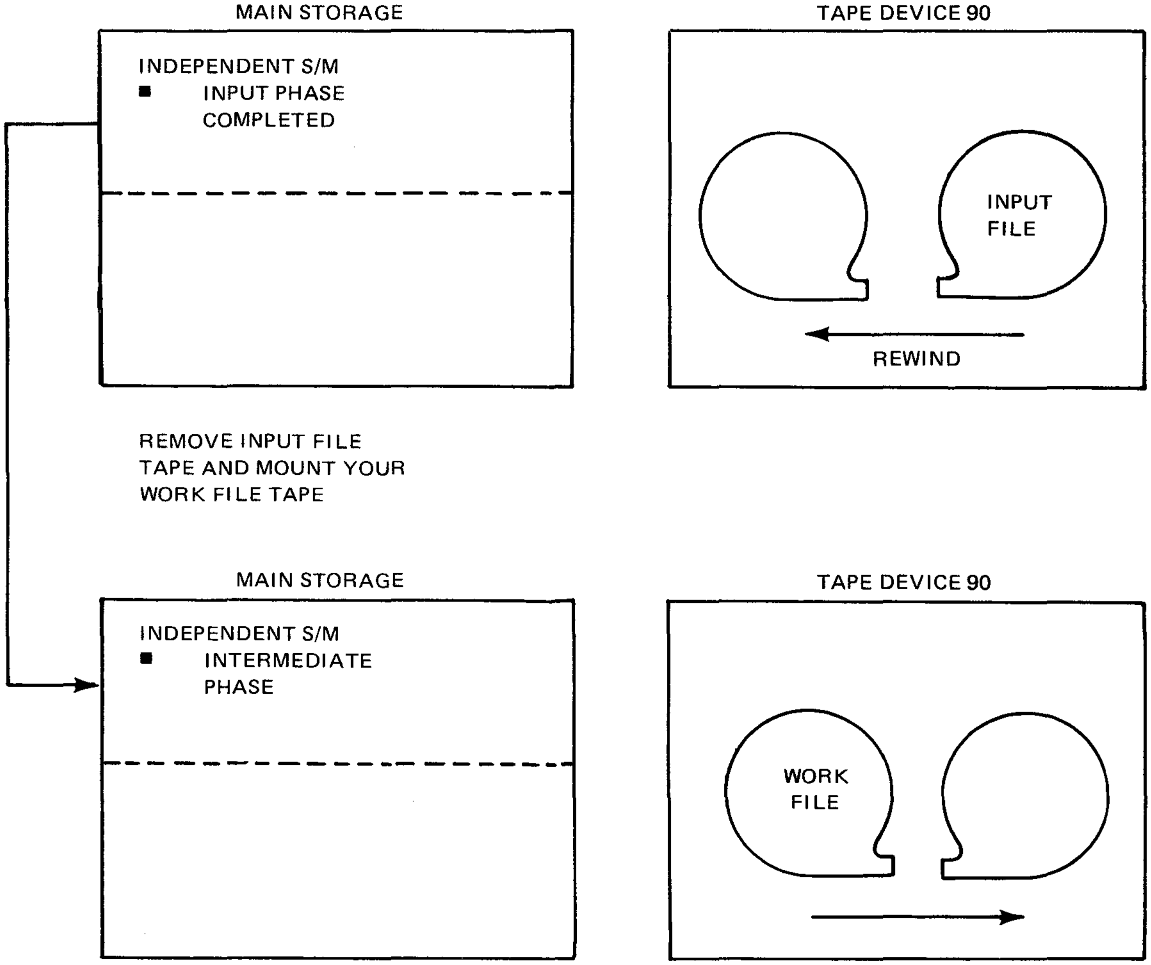


Figure 3—16. Same Input Device Shared between Input File and Sort Work File

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.

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. Refer to Figure 3—11, line 2 for an illustration of the VERIFY parameter.

In a disk sort, independent 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 sort/merge 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 independent 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.

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, independent sort/merge obtains this information from your job control statements. Lines 2 and 3 of Figure 3—11 show this parameter coded.

Occasionally, you may need to include certain parameters from the job control stream at execution time. To tell independent sort/merge you are submitting parameters in this way, you must use the CSPRAM parameter. The keyword parameters that independent sort/merge can accept via the control stream at run time are BIN, DISC, NOCKSM, RESERV, RESUME, SHARE, and TAPE. You enter these keyword parameters via PARAM job control statements. There are three values to choose from on the CSPARM parameter: NO, OPTION, 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 OPTION and YES keywords operate identically. The control stream is tested for the presence of //PARAM statements. If they are present, they are read. See Figure 3—11, line 1 for an example of this parameter.

When independent 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 be written to the log. Figure 3—11, line 2 illustrates this specification. Error messages that are written to the job log also are displayed on the operator console.

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 independent sort/merge by displaying messages concerning sort/merge execution status, fatal errors, possible recovery information, and directions for mounting, dismounting, 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.

```
       LABEL    ΔOPERATIONΔ        OPERAND
       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

Independent sort/merge allows you to pass control during certain phases of its operation to your own-code routines which you write in BAL or to the system-supplied DELETE data reduction routine. These points where control passes to your routines are called exits.

You use them when you want your own routines to handle input or output file processing, record sequencing, data reduction, or special collation sequencing or you want independent sort/merge to perform automatic data reduction. The MODS control statement (3.3.1) allows you to specify:

- the phase number where your own-code routine or the DELETE routine enters the independent sort/merge or merge-only operation; and
- the load module name of the routine, its approximate size, and the applicable exit-code number.

Because each exit allows definite functions to be performed and these functions are contained only in specific operational phases, your must choose exits and assign them to the operational phase to which they are associated. Table 3—2 lists exit codes, the functions each exit code allows you to perform, and the phase associated with each exit-code number.

Table 3—2. Exit Codes: Their Allowable Functions and Associated Phases

Phase	Exit Code	Function
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
3	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
1–3	E65	Record sequencing
	E75	Data reduction
	E84	User-defined collation sequencing

3.3.1. Defining Exits

3.3.1.1. Exiting to Your 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 is:

LABEL	ΔOPERATIONΔ	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-internal sort)
3	Phase 3 (final merge-output)
6	All phases (record sequencing routine)
7	All phases (data reduction routine)
8	All phases (user-defined collation sequencing)

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, independent 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 required for input label (E11) and input file (E15) processing (line 1).

```
      LABEL   ΔOPERATIONΔ        OPERAND
      1       10       16                                                   72
1.            MODS     PH1=(PHASE1,385Ø,E11,E15),                            C
2.                     PH3=(PHASE3,27ØØ,E31,E35)
```

Since both exits pertain to data input phase 1 of sort/merge, 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 independent sort/merge execution via exits 31 and 35. These exits process output file labels and read output files.

3.3.1.2. Exiting to System-Supplied DELETE Routine

A special format of the MODS control statement is provided to allow you to exit to a system-supplied data reduction routine called DELETE. The DELETE routine is supplied as a load module in the system load library ($Y$LOD) and, when activated, automatically deletes duplicate records from your files. (See 3.3.2.8.) Defining the exit for the DELETE routine is the same as that for exiting to your own-code routines (3.3.1.1), except that you must use the system name DELETE when specifying the name of the load module.

The MODS control statement format for automatic data reduction is:

LABEL	ΔOPERATIONΔ	OPERAND
	MODS	**PH7=(DELETE [,length] ,E75)**

3.3.2. Using Exit Codes

Independent sort/merge can exit to your own-code routines only during the phases you specify in the MODS control statement. Independent sort/merge passes control 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 independent sort/merge, you must save certain register contents. (See 3.3.3.) 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.

3.3.2.1. 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 independent 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. However, you must specify the input files as unlabeled (LABEL=U) in the OPTION control statement. Exit code E11 enables the input files to interface with your own-code routine. This exit is essentially the same as the LABADDR keyword parameter routine in DTF mode, or the ULABEL keyword parameter routine in consolidated data management mode. For more information on the use of these routines, basic data management users should consult the basic data management user guide, UP-8068 (current version); consolidated data management users should consult the consolidated data management concepts and facilities, UP-8825 (current version).

In the use of register 15, there is a conflict between sort/merge exit conventions and data management. The sort/merge conventions require that register 15 be used as the base register for each exit module, whereas data management requires that register 15 contain the DTF address for the label processing routine in DTF mode. For exit 11, register 1 will contain the address of the DTF or, if consolidated data management is being used, the CDIB. In DTF mode, after the user has established a new base register other than 15, the following instructions must be issued:

```
LR  15,1          Puts DTF address in 15

L   1,176(0,15)   Puts buffer address in 1
```

This will set up the registers so that the LBRET macros can be issued. The consolidated data management user label routine must be executed without adding this code.

3.3.2.2. Input File Processing

Input file processing (exit code E15) enables independent sort/merge to enter your own-code routine to perform any of the following functions during 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 phase 1. 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 independent 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 independent sort/merge to specify location of records and information affecting record processing (3.3.6).

3.3.2.3. Input File Read Error Processing

By specifying exit code E18 on the MODS control statement, you enable independent sort/merge to enter your own-code read error processing routine for the input file. Independent sort/merge takes the address of your error routine and places it in the sort input DTF. This supplies to data management the ERROR keyword parameter that names your error handling routine, and you return to the sort via the BR 14 instruction. For more information about the ERROR instruction, refer to the data management user guide, UP-8068 (current version). You write only the BR 14 instruction to return to the sort. Independent sort/merge dynamically activates the ERROR keyword parameter. 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.

3.3.2.4. 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 LABADDR data management DTF keyword parameter and the LBRET imperative macro instruction. As in exit 11, there is a conflict with the use of register 15. (See 3.3.2.1.) The user must establish a new base register and then load registers 15 and 1 as required for exit 11.

In consolidated data management, there is no register conflict. Register 1 will contain the address of the CDIB as required by the consolidated data management user label routine. Refer to the consolidated data management concepts and facilities, UP-8825 (current version).

3.3.2.5. Output File Processing

Exit code E35 enables independent sort/merge to enter your own-code routine for output file processing during 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 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 independent sort/merge to specify the location of records and information affecting record processing. Additional details concerning parameter list format are given in 3.3.6.) 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.

3.3.2.6. 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. To interface with the output file, your own-code routine uses the same approach as data management error handling via the ERROR keyword parameter of the DTF declarative macro and the BR 14 instruction. The ERROR keyword parameter specifies the name of your error processing routine, and the BR 14 instruction returns control from your error processing routine to the independent sort/merge. Refer to the data management user guide, UP-8068 (current version) for more information about the ERROR keyword parameter. You write only the BR 14 instruction. Independent sort/merge dynamically activates the ERROR keyword parameter by taking your error processing routine address and placing it in the sort input DTF.

3.3.2.7. Record Sequencing

Exit code E65 is used during phases 1, 2, and 3 for entering your own-code record sequencing routine from independent sort/merge. Independent 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 register 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 independent 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 independent sort/merge via a branch to register 14.

3.3.2.8. Data Reduction

When independent 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 independent 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. Independent 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 independent 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 independent sort/merge at the address specified in register 14.

3.3.2.9. User-Defined Collation Sequencing

Exit code E84 is used whenever you want to specify an alternate collating sequence to the one supplied by independent 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 independent 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 D for OS/3 EBCDIC and ASCII standard collating sequences.)

3.3.3. An Example of Exit-Code Use

Figure 3—17 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 independent 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 101). This example modifies the record contents (line 57).

Notice the DTF keyword parameters, ERROR (line 19) and EOFADDR (line 20), which give data management the name of the error routine (line 68) and the end-of- data routine (line 60). When errors occur or the end-of-data condition is reached, data management enters the error handling routine or end-of-data routine, respectively. Data management requires I/O buffers to be half-word aligned (line 21). If your input files are on 8416 or 8418 fixed-sector disks, or 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 22 and 23). The 18-word (72 byte) data management save area must be full-word aligned (line 24). There are two ways of providing the save area address to data management: loading the address into general register 13 before entering the data management imperative macro (see line 34, Figure 3-17), or specifying the label of the area via the SAVAREA keyword parameter in your DTF. For more details about the SAVAREA keyword parameter, refer to the data management user guide, UP-8068 (current version). Using the SAVAREA keyword frees register 13 for other use by your program.

```
// 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 AND DTF
*
INPUT    DTFSD BLKSIZE=4ØØ,RECSIZE=8Ø,IOAREA1=BUFF1,IOAREA2=BUFF2,       C
               IOREG=(2),RECFORM=FIXBLK,ERROR=IOERROR,OPTION=YES,        C
               EOFADDR=EOF,TYPEFLE=INPUT
         DS    OH
BUFF1    DS    CL4ØØ                    IOAREA1
BUFF2    DS    CL4ØØ                    IOAREA2
SAVEAREA DS    18F                      DATA MANAGEMENT SAVE AREA
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
         LA    13,SAVEAREA              SET DATA MANAGEMENT SAVE AREA
TAG      BC    Ø,NEXT                   FALL THROUGH ON FIRST TIME
         OPEN  INPUT                    OPEN THE INPUT FILE
         MVI   TAG+1,X'FØ'              ALTER BRANCH FOR NEXT ENTRY
NEXT     EQU   *
         GET   INPUT                    GET A RECORD
         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
*
*
MOD      EQU   *
*
*
```

Figure 3—17. Coding Example for Using Exit Code E15 (Part 1 of 2)

```
*              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    Ø(4,3),EOD               SET ACTION CODE 8 FOR END OF
*                                        EXIT ACTIVITY
         CLOSE  INPUT                    CLOSE INPUT ROUTINE
         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 5Ø
// VOL DSPØ28
// LBL MYFILE1
// LFD INPUT
// DVC 5Ø
// VOL DSPØ28
// EXT ,,,CYL,4
// LBL MYFILE2
// LFD SORTOUT,,INIT
// DVC RES                 )
// EXT ST,C,,CYL,5          } = //DMØ1 WORK1 BLK=2ØØØØ
// LBL $SCR1                }
// LFD DMØ1                )
// EXEC SORT,$Y$RUN
/$
 SORT FIELDS=(1,8,CH)
 RECORD RCSZ=8Ø,TYPE=F
 INPFIL EXIT
 OUTFIL BLKSIZE=8Ø
 MODS PH1=(PHASE1,,E15)
 END
/*
/&
// FIN
```

Figure 3—17. Coding Example for Using Exit Code E15 (Part 2 of 2)

Since your routines are associated with phases of independent 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, belongs to 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 independent sort/merge that:

- your key field starts in byte 1, extends eight bytes, and has a character data format (line 97);
- your records are fixed type and 80 bytes long (line 98);
- you intend to use a phase 1 exit code (E15) for your own-code input processing routine (line 101);
- your output block size is 80 bytes (line 100);
- the load module name for your input routine is PHASE1 (line 101); and
- you are supplying a routine to modify input records (line 51—57 and line 99).

The first step you must take in your own-code routine is to save those registers used by independent sort/merge (line 30) and load the address of the data management save area into register 13 (line 34). You set up a new base register for your own-code routine (line 33), open the input file, and read records (lines 36—39). To modify records, you branch out to your record modification routine (line 40) 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 41 and 42). This record modification information is needed by independent 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 independent sort/merge in the first full word addressed by register 1. You must then tell independent 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 44) 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 71), into the parameter list at the second full-word position (line 45). 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 62) and setting the action code to 8 (line 63). You close the input file (line 65). The final step in coding is to restore the registers used by independent sort/merge (line 47) and return to the independent sort/merge (line 48). In keeping with the data management DTFSD macro specification for the EOFADDR keyword parameter, the end-of-file routine (line 60) 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 the independent 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—17 to understand how registers help implement the linkage from independent sort/merge to your own-code routines.

3.3.4. General Purpose Registers

Four general purpose registers play important roles in enabling independent 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, independent 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 independent sort/merge receives this action code, but first it needs the address of the parameter list in order to locate the action code. Independent sort/merge places the address of the parameter list 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 the independent sort/merge are described in 3.3.2.2. The format for your own-code parameter list is discussed in 3.3.6.

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 independent 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. Independent sort/merge places the address of a save area in register 13.

Before independent 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. Independent sort/merge places its return address in register 14. At its conclusion, your own-code routine must then branch back to the independent sort/merge via register 14.

When exiting to a user routine, independent sort/merge loads register 15 with the address of the exit routine. The appropriate exit routine is then entered via the branch table (3.3.5) which is required at the beginning of each exit load module.

If you use exit codes E11, E18, E38, or E39, you must enter and leave them by using data management DTF keyword parameters and imperative macros explained in the discussion of each exit code (3.3.2). You can find more detailed information concerning data management user exits in the data management user guide, UP-8068 (current version).

3.3.5. Providing a Branch for User Own-Code Exits

Independent 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—17 for an illustration of this coding.)

Table 3—3. Branch Table Format

Applicable Phase of Sort/Merge Operation	Typical Table Format
1	entry B E11 B E15 B E18
3	entry B E31 B E32 B E35 B E38 B E39

When independent 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 independent 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)
```

3.3.6. Formatting the Exit Parameter List

Independent 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 independent 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 keywords and imperative macros to locate own-code routines and return to the sort. The data management user guide, UP-8068 (current version) discusses these keywords and 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 LABADDR routines.*
E15	1	Address of record in the input buffer
	2	Address of action word
E32	1	Address of record in input buffer
E35	1	Address of record next scheduled for the output buffer
	2	Address of last record in the output buffer
	3	Address of action word
	4	Address of sequence check word
E18, E38	None	See data management conventions for ERROR routines.*
E39	None	See conventions for data management ERROR routine.*
E65	None	See E65 description (3.3.2.7).
E75	None	See E75 description (3.3.2.8).
E84	None	No executable code at this exit

*Refer to the data management user guide, UP-8068 (current version).

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.

3.3.7. Job Control for the Own-Code Routine

After you have written your own-code routine, you must assemble and link it (see Figure 3—17, lines 3 to 5 and 76 through 81) before you can use the routine in your independent 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—17. In this case, independent 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 which you can use over and over 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 independent 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        20
 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 independent sort/merge, you use the job control stream in example 2.

Example 2:

```
1        10        20
// JOB SRTEXMPL,,6ØØØ,8ØØØ
// DVC 2Ø  // LFD PRNTR
// DVC 5Ø  // VOL DSPØØ1
// LBL MYFILE1  // LFD INPUT
// DVC 5Ø  // VOL DSPØØ1
// EXT ,,,CYL,4
// LBL MYFILE2
// LFD SORTOUT
//DMØ1 WORK1
//DMØ2 WORK2
// EXEC SORT
/$
 SORT FIELDS=(1,8)
 INPFIL EXIT
 MODS PH1=(PHASE1,3588,E15)
 END
/*
/&
// 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 DTF when you wrote the program. (See Figure 3—17, line 18.) 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 phase 1 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:

```
     1        10        20
 1.  // JOB OWNCODE
 2.  // DVC 2Ø  // LFD PRNTR
 3.  // ASM
 4.  /$
 5.  *
 6.  *              YOUR PROGRAM CODING
 7.  *
 8.  /*
 9.  // DVC 5Ø  // VOL DSPØØ2
10.  // EXIT ST,,,CYL,1
11.  // LBL OWNCODE  // LFD OWNCODE
12.  // WORK1
13.  // EXEC LNKEDT
14.  /$
15.   LINKOP OUT=OWNCODE
16.   LOADM PHASE1
17.   INCLUDE PHASE1
18.  /*
19.  /&
20.  // 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 which already 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         20
     ---------------------------------------------------------
10a. | // DVC 51    // VOL DSPØØ2
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 two to eight similarly ordered files into one final output file arranged in the same sequence as the input files. When independent 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 phase 3 of the sort. Thus, when independent sort/merge performs a merge-only operation, it begins with phase 0, skips phases 1 and 2, and ends with phase 3, where it enters your own-code routine via exit codes E32 or E38 if you specified them on your MODS sort control statement. See 2.3.1 and Figure 2—3 for a better understanding of independent sort/merge phase operation during merge-only processing.

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 ORDER}={number 16}]
		[,MERGEP=(output-file-partition-number, input-file-1-partition-number, input-file-2-partition-number [,...,input-file-number-8-partition-number])]

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.

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, independent 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.

The MERGEP option keyword parameter is used when data records involved in a merge-only operation are read from and written to partitioned disk files. A partitioned disk file (input or output) may contain up to seven partitions; however, you are limited to merging data in only one partition from each input file read to the merge. Independent sort/merge can merge data from a maximum of eight previously sequenced input files. (All input files must be on the same device type.) Thus, you can specify up to eight partitions, one for each input file, from which data records are read to the merge. (The exact number of input files involved in the merge must agree with that specified in the FILES or ORDER keyword parameters.) Since only one output file can be assigned to a merge-only operation, your MERGEP keyword parameter must identify the partition of that file into which merged data records are written. The position of the partition number specification in that MERGEP parameter identifies the files with which it is associated. Notice that at least two *input-file-partition-numbers* are required if you decide to use the MERGEP parameter.

Line 1 in the following coding example specifies that the key field begins at byte 9 of the record being merged, is one byte long, has character data format, and is to be merged in ascending order. It also specifies three input files to be merged. Line 2 shows the same specification except independent sort/merge assumes the character data format code and ascending merge sequence by default. In line 3, we add the optional keyword parameter, MERGEP. MERGEP specifies that the disk output file is to receive the merged data records in partition 3 from partition 2 of the first input file and partition 6 of the second input file.

```
      LABEL   ΔOPERATIONΔ      OPERAND
      1        10       16
    ----------------------------------------------------------
1.  |          MERGE    FIELDS=(9,1,CH,A),FILES=3
2.  |          MERGE    FIELDS=(9,1),FILES=3
3.  |          MERGE    FIELDS=(9,1),FILES=3,MERGEP=(3,2,6)
```

Figure 3—18 illustrates the action of the MERGEP parameter specifications from line 3.

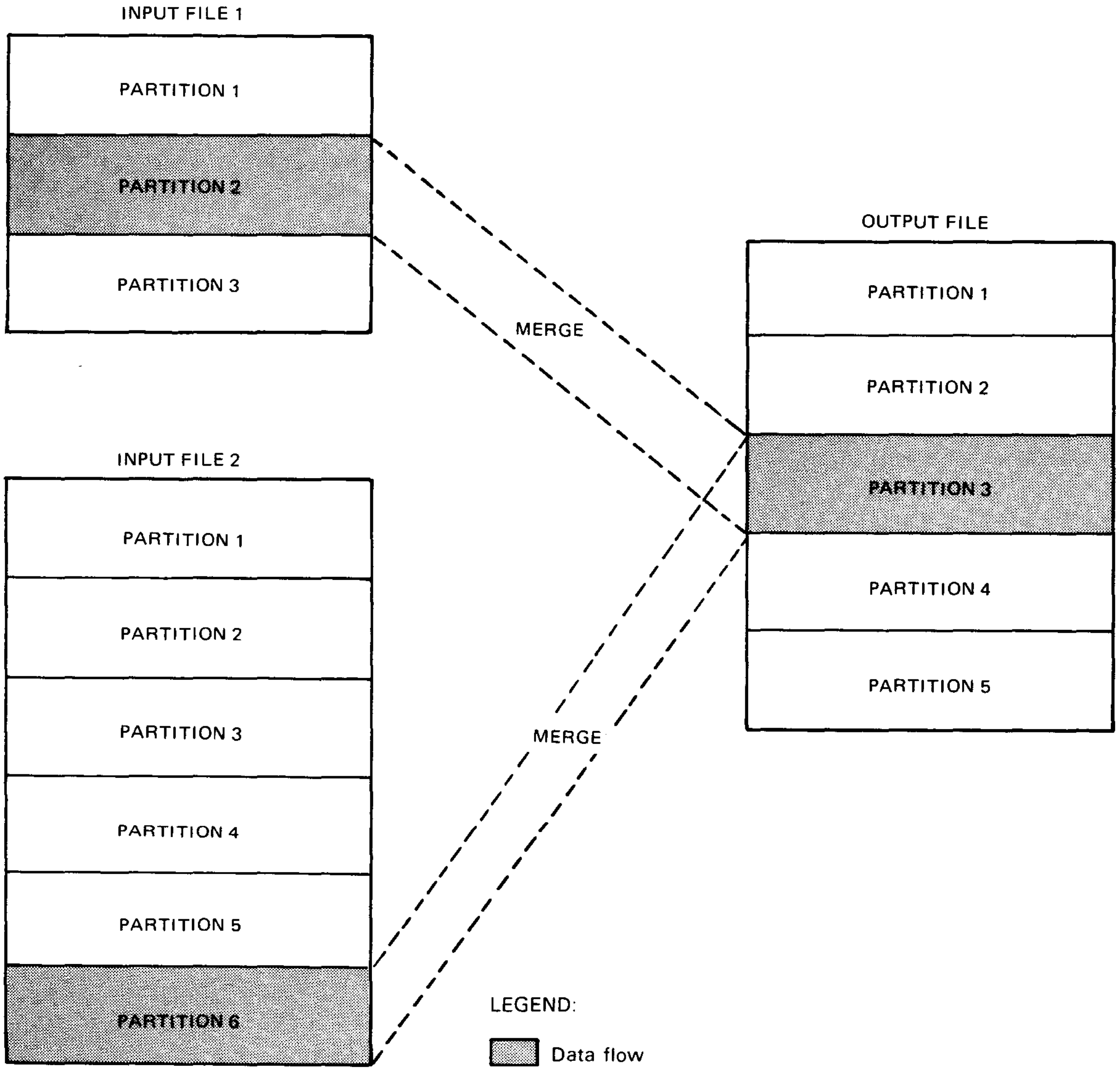


Figure 3—18. Writing Merge-Only Records from Two Partitioned Input Files to a Partitioned Output File

3.4.2. Merge-Only Exit Code for Input File Processing

You can use exit code E32 to enter your own-code routine from phase 3 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—19 illustrates a typical job control stream required for an independent merge-only operation. In the following discussion, we step through these statements to describe the processing involved.

```
// JOB MRGEXMP2,,7ØØØ,9ØØØ,2
// OPR 'MERGE EXAMPLE 2'
// OPR 'VARYING BLOCK SIZE'
// DVC 5Ø
// VOL DSPØ28
// LBL MYLIB1
// LFD SORTIN1
// DVC 5Ø
// VOL DSPØ28
// LBL MYLIB2
// LFD SORTIN2
// DVC 5Ø
// VOL DSPØ28
// LBL MYLIB3
// LFD SORTIN3
// DVC 5Ø
// VOL DSPØ28
// LBL MYLIB4
// LFD SORTOUT,,INIT
// EXEC SORT
/$
 MERGE FIELDS=(1,8,PD),FILES=3
 RECORD TYPE=F,RCSZ=8Ø
 INPFIL BLKSIZE=(8ØØ,4ØØ,16ØØ)
 OUTFIL BLKSIZE=8ØØ
 END
/*
/&
// FIN
```

Figure 3—19. Typical Job Control Stream for an Independent 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 records 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) tells the independent merge-only 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-8065 (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. The data management keyword parameter ERROR on the DTF statement and the BR 14 instruction provide the interface between merge-only and your routines. The data management user guide, UP-8068 (current version) supplies more detailed information concerning this keyword and its interfacing functions. Remember the sort dynamically activates the ERROR parameter by taking your read error processing routine address and placing it in the sort input DTF. You write only the BR 14 instruction to return to the sort.

3.5. RUNNING YOUR SORT JOB FROM A WORKSTATION

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

- you can build a control stream to execute independent sort/merge at a workstation, as opposed to punching it on cards or writing it to a diskette; and

- 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 control stream from a workstation is by using the job control dialog. The control dialog is an interactive facility of OS/3 that allows you to describe your job requirements 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. From here, you can 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 using another interactive facility of OS/3 called the general editor.

The procedures for activating the job control dialog, initializing the running of a job, and activating the general editor are described in detail in the OS/3 workstation user guide, UP-8845 (current version).

More detailed descriptions of the job control dialog and the general editor are presented in the job control user guide, UP-8065 (current version) and the general editor user guide, UP-8828 (current version), respectively.

Note that if a job has been initiated from a workstation, all messages will be displayed on the workstation rather than the system console. This includes those messages that you have specified to be printed on the system printer. (See 3.2.6.)

4. Independent Sort/Merge 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 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:

- Independent disk sorts
- Independent tape sorts
- An independent default sort

4.2. INDEPENDENT SORT/MERGE CONTROL STATEMENT EXAMPLES

The following six examples illustrate the sort/merge control statements needed to supply information to independent 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 to the independent sort/merge or merge-only.

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

Example 1:

```
/$
 SORT FIELDS=(I,4,CH,A,1Ø,12,BI,A),WORK=3,SIZE=35ØØ
 RECORD TYPE=F,RCSZ=82
 INPFIL BLKSIZE=82Ø,OPEN=RWD,CLOSE=UNLD,DATA=E
 OUTFIL BLKSIZE=82Ø,OPEN=RWD,CLOSE=UNLD
 OPTION PRINT=ALL,STORAGE=2ØØØØ,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=(6,10,CH,A,12,10,CH,D),WORK=3,SIZE=3500
 RECORD TYPE=V,LENGTH=(400,30,,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 exits.

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 independent 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 INDEPENDENT DISK SORTS

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

- disk input files and disk work files are used to create a disk output file;

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

- a multipartitioned disk input file is used to create a multipartitioned disk output file;

- a multipartitioned disk input file is used to copy sorted records to a multipartitioned output file showing the use of keyword parameter COPY=ALL; or

- a multipartitioned disk input file is used to copy a sorted record to a multipartitioned output file showing the use of the selective COPY feature.

Example 1 illustrates a typical job control stream required for performing an independent sort using disk input file, disk work files, and a disk output file. The job named SRTEXMP1 performs an independent sort of the data records contained in disk input file SORTIN1. The three disk work files (DM01 through DM03) assigned to the sort are SAT (system access technique) files. 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:

```
// JOB SRTEXMP1,,7ØØØ,9ØØØ,2
// DVC 5Ø
// VOL DSPØØ1                  INPUT FILE DEVICE
// LBL MYLIB1                  ASSIGNMENT SET
// LFD SORTIN1
// DVC 5Ø
// VOL DSPØØ1                  OUTPUT FILE DEVICE
// LBL MYLIB2                  ASSIGNMENT SET
// LFD SORTOUT,,INIT
// DVC 51
// VOL DSP111                  WORK FILE1 DEVICE
// LBL SRTWK1                  ASSIGNMENT SET
// LFD DMØ1,,INIT
// DVC 52
// VOL DSP112                  WORK FILE2 DEVICE
// LBL SRTWK2                  ASSIGNMENT SET
// LFD DMØ2,,INIT
// DVC 53
// VOL DSP113
// LBL SRTWK3                  WORK FILE3 DEVICE
// LFD DMØ3,,INIT              ASSIGNMENT SET
// EXEC SORT
/$
 SORT FIELDS=(9,1,,,1,8),WORK=3,SIZE=1ØØØØ
 RECORD LENGTH=(8Ø),TYPE=F
 INPFIL BLKSIZE=8ØØ                          SORT PROGRAM
 OUTFIL BLKSIZE=8ØØ
 END
/*
/&
// FIN
```

Line Number	Explanation
1	The JOB statement defines the job named SRTEXMP1 to the system, minimum and maximum main storage required for the job, and number of tasks active simultaneously during the job execution.
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 through they were new files.
22	Initiates the execution of the independent sort/merge.
23—29	The data set containing the independent 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.

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:

```
// JOB SRTEXMP2,,7ØØØ,9ØØØ                                         1
// OPR ' SORT EXAMPLE 7'                                           2
// OPR 'MULTIPLE INPUT FILES'                                      3
// DVC 5Ø                  )                                       4
// VOL DSP1ØØ              { INPUT FILE1 DEVICE                    5
// LBL SORTIN1             { ASSIGNMENT SET                        6
// LFD SORTIN1             )                                       7
// DVC 5Ø                  )                                       8
// VOL DSP1ØØ              { INPUT FILE2 DEVICE                    9
// LBL INPUTØ2             { ASSIGNMENT SET                        10
// LFD SORTIN2             )                                       11
// DVC 5Ø                  )                                       12
// VOL DSP1ØØ              { INPUT FILE3 DEVICE                    13
// LBL INPUTØ3             { ASSIGNMENT SET                        14
// LFD SORTIN3             )                                       15
// DVC 5Ø                  )                                       16
// VOL DSP1ØØ              { OUTPUT FILE DEVICE                    17
// LBL OUTPUT              { ASSIGNMENT SET                        18
// LFD SORTOUT,,INIT       )                                       19
// DVC 51                  )                                       20
// VOL DSP1Ø1              {                                       21
// LBL WORK                { WORK FILE DEVICE                      22
// LFD DMØ1,,INIT          { ASSIGNMENT SET                        23
// EXEC SORT               )                                       24
/$                                                                 25
 SORT FIELDS=(4,8,PD),FILE=3,SIZE=5ØØØØ  )                         26
 INPFIL BLKSIZE=(8ØØ,4ØØ,8ØØ)            {   SORT PROGRAM          27
 OUTFIL BLKSIZE=(8ØØ)                    {                         28
 END                                     )                         29
/*                                                                 30
/&                                                                 31
// FIN                                                             32
```

Line Number	Explanation
1	The JOB statement defines the job named SRTEXMP2 to the system and the minimum and maximum main storage bytes, in hexadecimal, 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 the independent sort/merge.
25—30	The data set containing the independent 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.

Example 3 illustrates the job control stream required to perform an independent disk sort using a multipartitioned disk input file and single-partition disk output file. The job named SRTEXMP3 sorts the data records contained in partition 3 of the input file SORTIN1 and then writes the sorted records to output file SORTOUT. The key fields of the data records are four bytes long, beginning at byte 1 of the record. Records are character formatted and are sorted in ascending order. The requirements for the INPFIL and OUTFIL control statements are determined by default.

Example 3:

```
// JOB SRTEXMP3,8ØØØ,9ØØØ                                        1
// DVC 5Ø                  )                                     2
// VOL DSP1ØØ              (  INPUT FILE DEVICE                  3
// LBL INPUT               <  ASSIGNMENT SET                     4
// LFD SORTIN1             )                                     5
// DVC 5Ø                  )                                     6
// VOL DSP1ØØ              (  OUTPUT FILE DEVICE                 7
// LBL OUTPUT              <  ASSIGNMENT SET                     8
// LFD SORTOUT,,INIT       )                                     9
// DVC 51                  )                                    10
// VOL DSP1Ø1              |                                    11
// LBL WORK                <  WORK FILE DEVICE                  12
// LFD DMØ1,,INIT          |  ASSIGNMENT SET                    13
// EXEC SORT               )                                    14
/$                                                              15
 SORT FIELDS=(1,4,CH),SORTP=(1,3)  )                            16
 END                               <  SORT PROGRAM              17
/*                                 |                            18
/&                                 )                            19
// FIN                                                          20
```

Line Number	Explanation
1	The JOB statement defines the job named SRTEXMP3 to the system and minimum and maximum hexadecimal main storage bytes required to run the job.
2—5	Assigns the input file. (See 3.1.1.)
6—9	Assigns the output file. (See 3.1.1.)
10—13	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.
14	Initiates the execution of the independent sort/merge.
15—18	The data set containing the independent sort/merge control statements.
16	The SORT statement specifies: ■ a 4-byte character key field beginning at byte 1 of the data records to be sorted; ■ partition 3 of the input file contains the records to be sorted; and ■ the sorted records are to be written to a single-partition output file (SORTOUT).

Line Number	Explanation
17—18	Marks the end of the sort control statements.
19	Marks the end of the job stream.
20	Marks the end of card reader operations.

Example 4 illustrates the job control stream required for performing an independent disk sort using multipartitioned disk input and output files. The job named SRTEXMP4 sorts the data records contained in partition 3 of the input file SORTIN1 and then writes the sorted records to partition 3 of the output file SORTOUT. Records to be sorted are character formatted and the sort sequence is ascending. The key field for the sort begins in byte 2 of each record and is four bytes long. Because the output file consists of four partitions (NPTN=4), the user program must define all four partitions so that the output file can be opened by independent sort/merge. To facilitate this requirement, the user's data set must include the BLKSIZE, the SIZE, the UOS, the TYPE, and the RCSZ keyword parameters to define each partition in the output file.

Example 4:

```
// JOB SRTEXMP4,,7ØØØ,9ØØØ                                                1
// DVC 5Ø               ⎫                                                 2
// VOL DSP1ØØ           ⎬ INPUT FILE DEVICE                               3
// LBL INPUT            ⎬ ASSIGNMENT SET                                  4
// LFD SORTIN1          ⎭                                                 5
// DVC 5Ø               ⎫                                                 6
// VOL DSP1ØØ           ⎬ OUTPUT FILE DEVICE                              7
// LBL OUTPUT           ⎬ ASSIGNMENT SET                                  8
// LFD SORTOUT,,INIT    ⎭                                                 9
// DVC 51               ⎫                                                10
// VOL DSP1Ø1           ⎬ WORK FILE DEVICE                               11
// LBL WORK             ⎬ ASSIGNMENT SET                                 12
// LFD DMØ1,,INIT       ⎭                                                13
// EXEC SORT                                                             14
/$                                                                       15
 SORT FIELDS=(2,4,CH),SORTP=(3,3)                                        16
 OUTFIL BLKSIZE=(4ØØ,25Ø,8ØØ,2ØØ),SIZE=(2Ø,1Ø,5Ø,2Ø),                 C 17
              UOS=(5Ø,Ø,25,Ø),TYPE=(F,V,F,F),RCSZ=(4Ø,5Ø,8Ø,4Ø),      C 18
              NPTN=4                                                     19
  END                                                                    20
/*                                                                       21
/&                                                                       22
// FIN                                                                   23
```

Line Number	Explanation
1	The JOB statement defines the job named SRTEXMP4 and minimum and maximum main storage bytes 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—13	Assigns the disk work file. (See 3.1.1.)
14	Initiates the execution of the independent sort/merge.
15—19	The data set containing the independent sort/merge control statements.
16	The SORT statement specifies: ■ a 4-byte character key field at byte 2 of the data records to be sorted; and ■ the input and output files are multipartitioned files, with the input record read from input file partition 3 and the sorted records written to output file partition 3.
17—19	The OUTFIL statement defines: ■ the output block sizes; ■ percentage of the output file reserved for each partition; ■ percentage of secondary storage allocation to be suballocated to each partition as needed; ■ output record types; and ■ output record sizes for each partition in the output file. Block size is 400 bytes in output file partition 1, 250 bytes in partition 2, 800 bytes in partition 3, and 200 bytes in partition 4. Twenty percent of the output file is assigned to partition 1, 10 percent to partition 2, 50 percent to partition 3, and 20 percent to partition 4. The percentage of secondary storage for each partition is 50, 0, 25, and 0 percent, respectively. Output record types for partitions 1 through 4 are fixed, variable, fixed, and fixed, respectively. Record sizes for partitions 1 through 4 are 40 bytes, 50 bytes, 80 bytes, and 40 bytes. There are four partitions in the output file.

Example 5 illustrates the job control stream required for performing an indpendent disk sort using multipartitioned input and output files and the COPY=ALL keyword parameter. The COPY=ALL keyword parameter allows all partitions of the input file not involved in the sort to be copied directly into the corresponding partitions of the output file. Records in the partitions being sorted are processed in the usual manner. In this example, the job named SRTEXMP5 sorts only the data records contained in partition 3 of the 4-partitioned input file SORTIN1. The sorted records are written to partition 3 of the output file SORTOUT. Data records in the remaining partitions of the input file are not sorted but are copied into the corresponding partitions in the output file as specified by the COPY=ALL keyword parameter. In this type of sort, only the size (SIZE) of the output file partitions needs to be defined. The expansion percentage (UOS) for each partition is assumed as 0.

Example 5:

```
// JOB SRTEXMP5,,7ØØØ,9ØØØ                                                1
// DVC 5Ø              )                                                  2
// VOL DSP1ØØ          |                                                  3
// LBL INPUT           > INPUT FILE DEVICE ASSIGNMENT SET                 4
// LFD SORTIN1         )                                                  5
// DVC 5Ø              )                                                  6
// VOL DSP1ØØ          |                                                  7
// LBL OUTPUT          > OUTPUT FILE DEVICE ASSIGNMENT SET                8
// LFD SORTOUT,,INIT   )                                                  9
// DVC 51              )                                                 10
// VOL DSP1Ø1          |                                                 11
// LBL WORK            > WORK FILE DEVICE ASSIGNMENT SET                 12
// LFD DMØ1,,INIT      )                                                 13
// EXEC SORT                                                             14
/$                                         )                             15
 SORT FIELDS=(2,4,CH),SORTP=(3,3),COPY=ALL > SORT PROGRAM                16
 OUTFIL SIZE=(2Ø,1Ø,5Ø,2Ø),NPTN=4          )                             17
 END                                                                     18
/*                                                                       19
/&                                                                       20
// FIN                                                                   21
```

Line Number	Explanation
1	The JOB statement defines the job named SRTEXMP5 to the system and the minimum and maximum main storage hexadecimal bytes 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—13	Assigns the disk work file. (See 3.1.1.)

Line Number Explanation

14 Initiates the execution of the independent sort/merge.

15—19 The data set containing the independent sort/merge control statements.

16 The SORT statement specifies:

- a 4-byte character key field at byte 2 of the data records to be sorted;
- only partition 3 of the input file is involved in the sort; and
- all other partitions of the input file are copied to their corresponding partitions in the output file.

17 The OUTFIL statement defines the percentage of the output file to be allocated for each partition in the output file. There are four partitions in the output file (NPTN=4).

18—19 Marks the end of sort control statements.

20 Marks the end of the job stream.

21 Marks the end of the card reader operations.

Example 6 illustrates the job control stream required for performing an independent disk sort involving multipartitioned disk files and demonstrating the use of the selective COPY feature. In this example, the job named SRTEXMP6 sorts the data records in partition 3 of the input file SORTIN1 and writes the sorted records to partition 1 in the output file SORTOUT. The selective COPY keyword parameter specifies that partitions 1 and 2 of the input file are to be copied directly into partitions 2 and 4, respectively, of the output file. The characteristics of partitions copied from the input file to the output file remain the same. It is best, to fully define the output file in all cases.

Example 6:

```
// JOB SRTEXMP6,,7ØØØ,9ØØØ                                                      1
// DVC 5Ø                    )                                                  2
// VOL DSP1ØØ                |                                                  3
// LBL INPUT                 > INPUT FILE DEVICE ASSIGNMENT SET                 4
// LFD SORTIN1               )                                                  5
// DVC 5Ø                    )                                                  6
// VOL DSP1ØØ                |                                                  7
// LBL OUTPUT                > OUTPUT FILE DEVICE ASSIGNMENT SET                8
// LFD SORTOUT,,INIT         )                                                  9
// DVC 51                    )                                                 10
// VOL DSP1Ø1                |                                                 11
// LBL WORK                  > WORK FILE DEVICE ASSIGNMENT SET                 12
// LFD DMØ1,,INIT            )                                                 13
// EXEC SORT                                                                   14
/$                                                                             15
 SORT FIELDS=(1,4),SORTP=(1,3),COPY=(1.2,2.4)  )                               16
 OUTFIL SIZE=(5Ø,2Ø,2Ø,1Ø),NPTN=4              > SORT PROGRAM                  17
 END                                           )                               18
/*                                                                             19
/&                                                                             20
// FIN                                                                         21
```

Line Number	Explanation
1	The JOB statement defines: ■ the job named SRTEXMP6 to the system; and ■ minimum and maximum main storage bytes 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—13	Assigns the disk work file. (See 3.1.1.)
14	Initiates the execution of the independent sort/merge.
15—19	The data set containing the independent sort/merge control statements.

Line Number Explanation

16 The SORT statement specifies:

- a 4-byte character key field at byte 1 of the data records to be sorted;
- data records in partition 3 of the input file are to be sorted and written into partition 1 of the output file; and
- partitions 1 and 2 of the input file are to be copied into partitions 2 and 4, respectively, of the output file.

17 The OUTFIL statement defines the percentage of the OUTPUT file to be allocated for each of the four partitions comprising the output file. There are four output file partitions (NPTN=4).

18—19 Marks the end of sort control statements.

20 Marks the end of the job stream.

21 Marks the end of card reader operations.

4.4. JOB CONTROL STREAM TO PERFORM INDEPENDENT TAPE SORTS

Both examples 7 and 8 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 7 illustrates a typical job control stream required to perform an independent sort/merge operation using tape for the input, output, and work files. The job named SRTEXMP7 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 phase 3 of the sort.

Example 7:

```
// JOB SRTEXMP7,,7ØØØ,9ØØØ,2                                          1
// DVC 9Ø              )                                               2
// VOL MASTER          > INPUT FILE DEVICE ASSIGNMENT SET              3
// LFD SORTIN1         )                                               4
// DVC 9Ø,IGNORE       )                                               5
// VOL TAPEØ1          ( REDEFINED INPUT DEVICE ASSIGNMENT             6
// LFD SMØ1            ) SET TO WORK FILE                              7
// DVC 91              )                                               8
// VOL TAPEØ2          > WORK FILE DEVICE ASSIGNMENT SET               9
// LFD SMØ2            )                                              10
// DVC 92              )                                              11
// VOL TAPEØ3          > WORK FILE DEVICE ASSIGNMENT SET              12
// LFD SMØ3            )                                              13
// DVC 92,IGNORE       )                                              14
// VOL MASTER          ( REDEFINED WORK FILE DEVICE                   15
// LFD SORTOUT         ) ASSIGNMENT SET TO OUTPUT FILE                16
// EXEC SORT                                                          17
/$                                                                    18
 SORT FIELDS=(8,1Ø,CH)                          )                     19
 RECORD LENGTH=(8Ø),TYPE=F                      )                     20
 INPFIL BLKSIZE=8ØØ,OPEN=RWD,CLOSE=RWI          >  SORT PROGRAM       21
 OUTFIL BLKSIZE=8ØØ,OPEN=RWD,CLOSE=RWI          (                     22
 OPTION SHARE=SMØ1,RESERV=SMØ3                  )                     23
 END                                            )                     24
/*                                                                    25
/&                                                                    26
// FIN                                                                27
```

Line Number	Explanation
1	The JOB statement defines the job named SRTEXMP7 to the system and minimum and maximum main storage bytes (in hexadecimal) 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-byte 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 and Figures 3—15 and 3—16.)
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 8 illustrates a typical job control stream required for restarting an interrupted tape sort performed by independent sort/merge. The sort itself is identical with that described in example 7. (See example 7 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 8 gives independent sort/merge the pass recovery point at which the sort is resumed.

Example 8:

```
// JOB SRTEXM8,,7ØØØ,9ØØØ,2
// OPR 'SORT EXAMPLE 8'
// OPR 'TAPE SORT WITH RESTART'
// DVC 9Ø                       )
// VOL MASTER                   > INPUT FILE DEVICE
// LFD SORTIN1                  ) ASSIGNMENT SET
// DVC 9Ø,IGNORE                )
// VOL TAPEØ1                   > REDEFINED INPUT DEVICE ASSIGNMENT
// LFD SMØ1                     ) SET TO WORK FILE
// DVC 91                       )
// VOL TAPEØ2                   > WORK FILE DEVICE ASSIGNMENT SET
// LFD SMØ2                     )
// DVC 92                       )
// VOL TAPEØ3                   > WORK FILE DEVICE ASSIGNMENT SET
// LFD SMØ3                     )
// DVC 92,IGNORE                )
// VOL MASTER                   > REDEFINED WORK FILE DEVICE ASSIGNMENT
// LFD SORTOUT                  ) SET TO OUTPUT FILE
// EXEC SORT
/$
 SORT FIELDS=(8,1Ø,CH)                                  )
 RECORD LENGTH=(8Ø),TYPE=F                              )
 INPFIL BLKSIZE=8ØØ,OPEN=RWD,CLOSE=RWI                  > SORT PROGRAM
 OUTFIL BLKSIZE=8ØØ,OPEN=RWD,CLOSE=RWI                  )
 OPTION SHARE=SMØ1,RESERV=SMØ3,RESTART,CSPRAM=YES       )
 END                                                    )
/*
// PARAM RESUME=(PASS,023)
/&
// FIN
```

4.5. JOB CONTROL STREAM TO PERFORM AN INDEPENDENT DEFAULT SORT

The default sort is so named because all information supplied to independent sort/merge is automatically defaulted in the absence of sort control statements. In the following example, notice 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, independent 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, independent 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 9:

```
// JOB SRTEXMP9,,7ØØØ,9ØØØ                                               1
// OPR 'SORT EXAMPLE 6'                                                  2
// OPR 'DEFAULT SORT'                                                    3
// DVC 5Ø                  )                                             4
// VOL DSP1ØØ              }  INPUT FILE DEVICE ASSIGNMENT SET           5
// LBL INPUT               |                                             6
// LFD SORTIN1             )                                             7
// DVC 5Ø                  )                                             8
// VOL DSP1ØØ              }  OUTPUT FILE DEVICE ASSIGNMENT SET          9
// LBL OUTPUT              |                                            10
// LFD SORTOUT,,INIT       )                                            11
// DVC 51                  )                                            12
// VOL DSP1Ø1              }  WORK FILE DEVICE ASSIGNMENT SET           13
// LBL WORK                |                                            14
// LFD DMØ1,,INIT          )                                            15
// EXEC SORT                                                            16
/&                                                                      17
// FIN                                                                  18
```

Line Number	Explanation
1	The JOB statement defines the job named SRTEXMP9 and minimum and maximum main storage bytes 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 independent sort/merge default sort.
17	Marks the end of control stream.
18	Marks the end of card reader operations.

PART 3. SUBROUTINE SORT/MERGE

5. Subroutine Sort/Merge Basic Concepts

5.1. GENERAL

For greater control over the sort/merge process than independent sort/merge provides, use the subroutine sort/merge. Naturally, the benefits you receive for this control cost something — an increase in the programming that you must do. You will have to program many of the activities that are done automatically by independent sort/merge.

Writing your own routines requires a good working knowledge of basic assembler language (BAL) or COBOL and data management macros. Section 6 discusses a disk subroutine sort/merge program example showing the use of BAL instructions and data management macros. Appendix C provides some subroutine sort/merge interface requirements for the COBOL programmer. Most users will use the independent sort/merge; however, the subroutine sort/merge is available if you have special sorting and data reduction problems.

The same disk sort problem used to illustrate independent sort/merge is used to illustrate subroutine sort/merge. This enables you to compare the programming needed for each method and help you decide which technique most adequately fills your needs.

5.2. SORT PROBLEM: A SOLUTION

To recapitulate, the disk sort problem general specifications are:

SYSTEM:	OS/3
PROGRAM:	Subroutine Sort/Merge Disk Sort
FUNCTION:	

1. This program sorts and merges an unordered file of employee records.

2. It is a disk sort.

3. It uses a sort key to sort and merge records.

4. The sort key is the employee number.

5. Employee number is located in the first eight byte positions of each record (0—7).

6. Records are sorted in ascending order.

INFORMATION:

1. This program needs a work file, $SCR1, to perform the sort/merge.

2. Work files are assigned to disk device 50.

INPUT AND OUTPUT:

1. Both input and ouput files are fixed-length, blocked records.

2. Each record contains 80 bytes.

3. Each block contains 5 records.

OUTPUT:

The program produces an output file of records sorted in ascending order.

Figure 2—1 summarizes these specifications.

Between your input stage and the output results, the program you write activates the subroutine sort/merge to perform the sort and return control to your program. The sort/merge modules reside in the system load library file ($Y$LOD) located on the SYSRES volume. When your program activates the subroutine sort/merge, it calls the sort/merge modules into main storage from $Y$LOD (Figure 5—1).

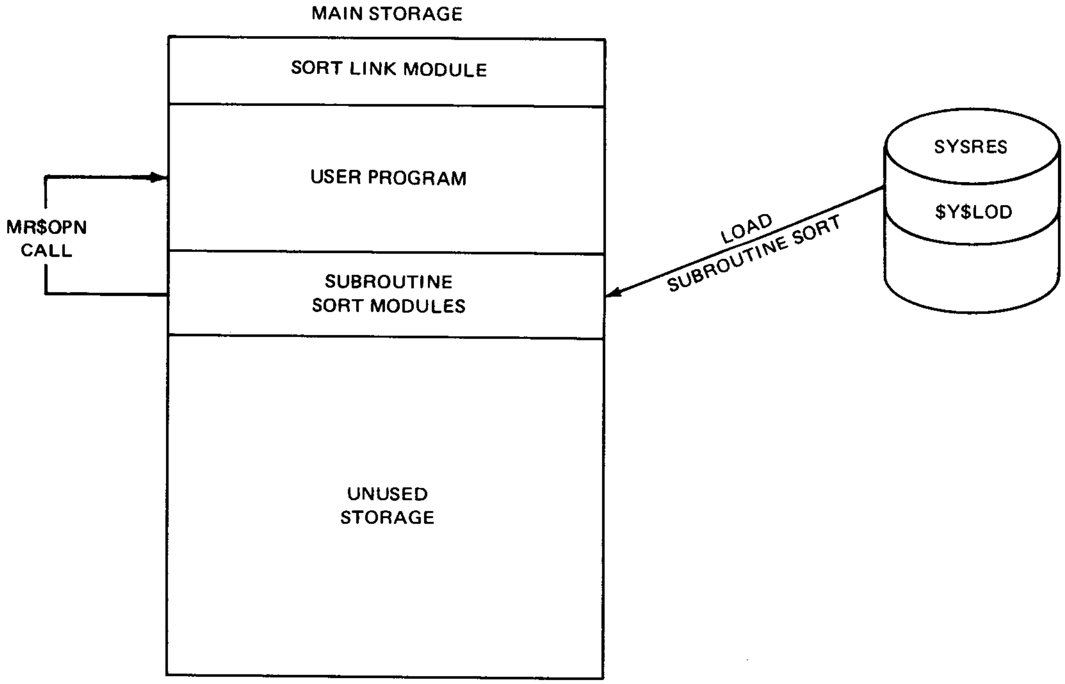


Figure 5—1. Calling in Sort/Merge Modules (Phase 0)

Before sorting can begin, your program must read the input file records from an input device. In this case, the input device is a disk named DSP028. Your program reads input records block by block from DSP028 into an I/O area in main storage called a *buffer area.* Buffer areas compensate for the differences in speed between low-speed I/O devices and high-speed main storage processing.

Using two buffer areas for record processing substantially increases sort speed. The disk sort program we are building illustrates this. This increase occurs because we can read records into one buffer while we empty the other buffer into a work area for further processing (Figure 5—2).

Records are passed one at a time to the subroutine sort where they are sorted into strings of sequenced data. The strings are stored on disk work files to be merged in phases 2 and 3 of the sort.

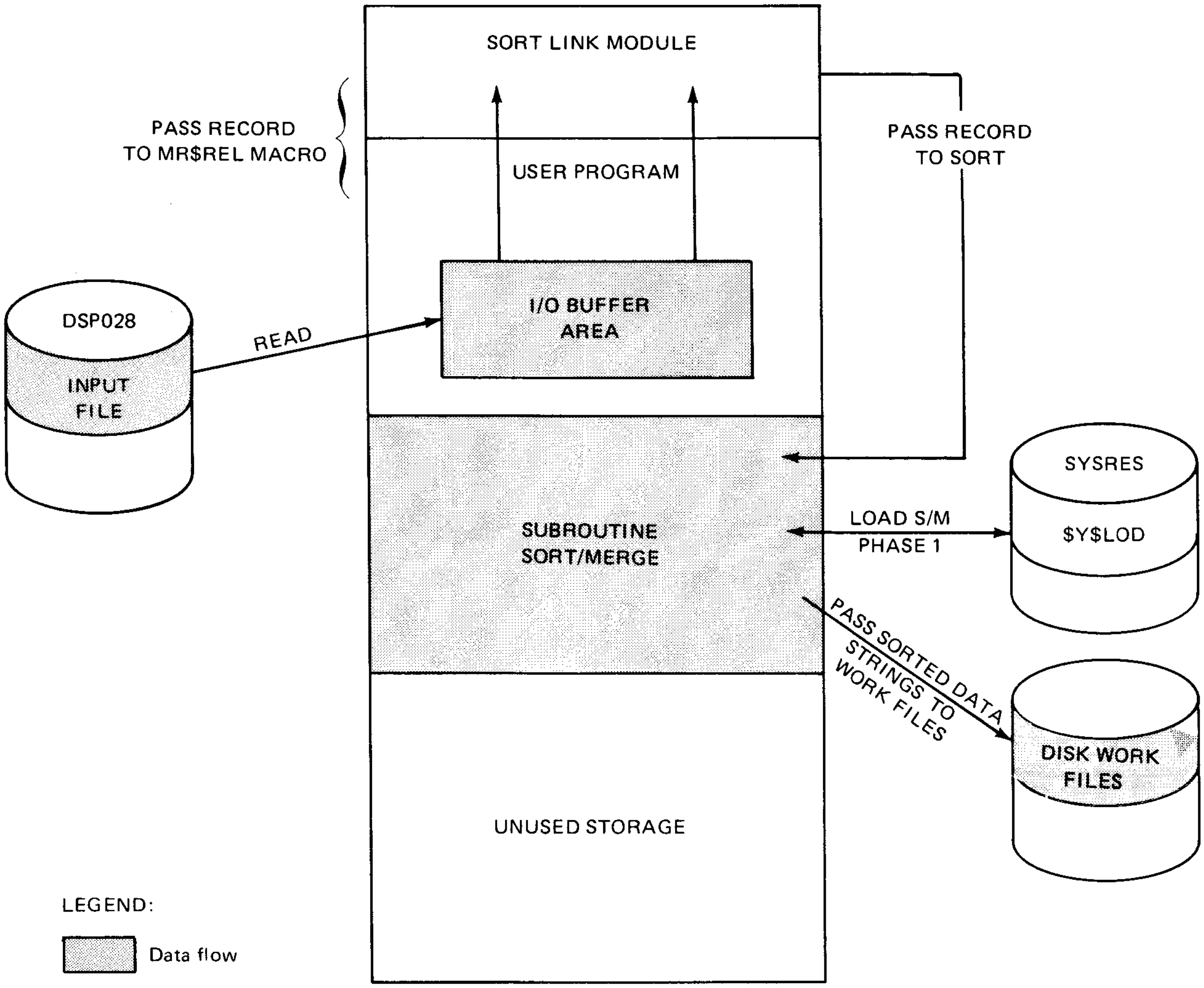


Figure 5—2. Reading Unsorted Input Records (Phase 1)

When all the input records have been read, passed to the subroutine sort/merge for the sorting, and ordered into sequenced strings of data on the work files, the strings are repeatedly merged to produce a single string of sorted data (Figure 5—3).

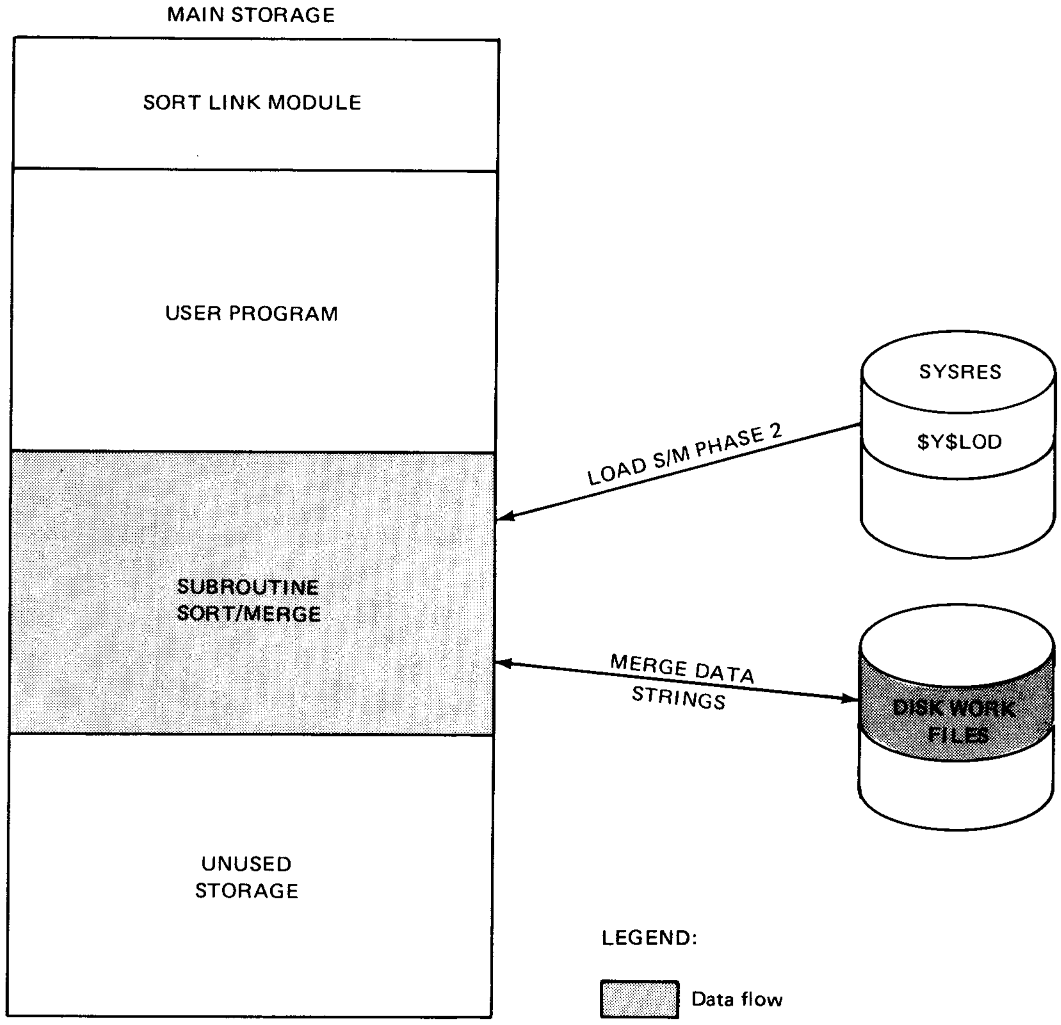


Figure 5—3. Sorting Input Records and Building Record Strings (Phase 2)

When a single merge pass produces one string of ordered records, the sort returns to your program, which may then request the return of records one at a time in the order desired. Your program can then put the records in the output buffer and write them to your output file (Figure 5—4).

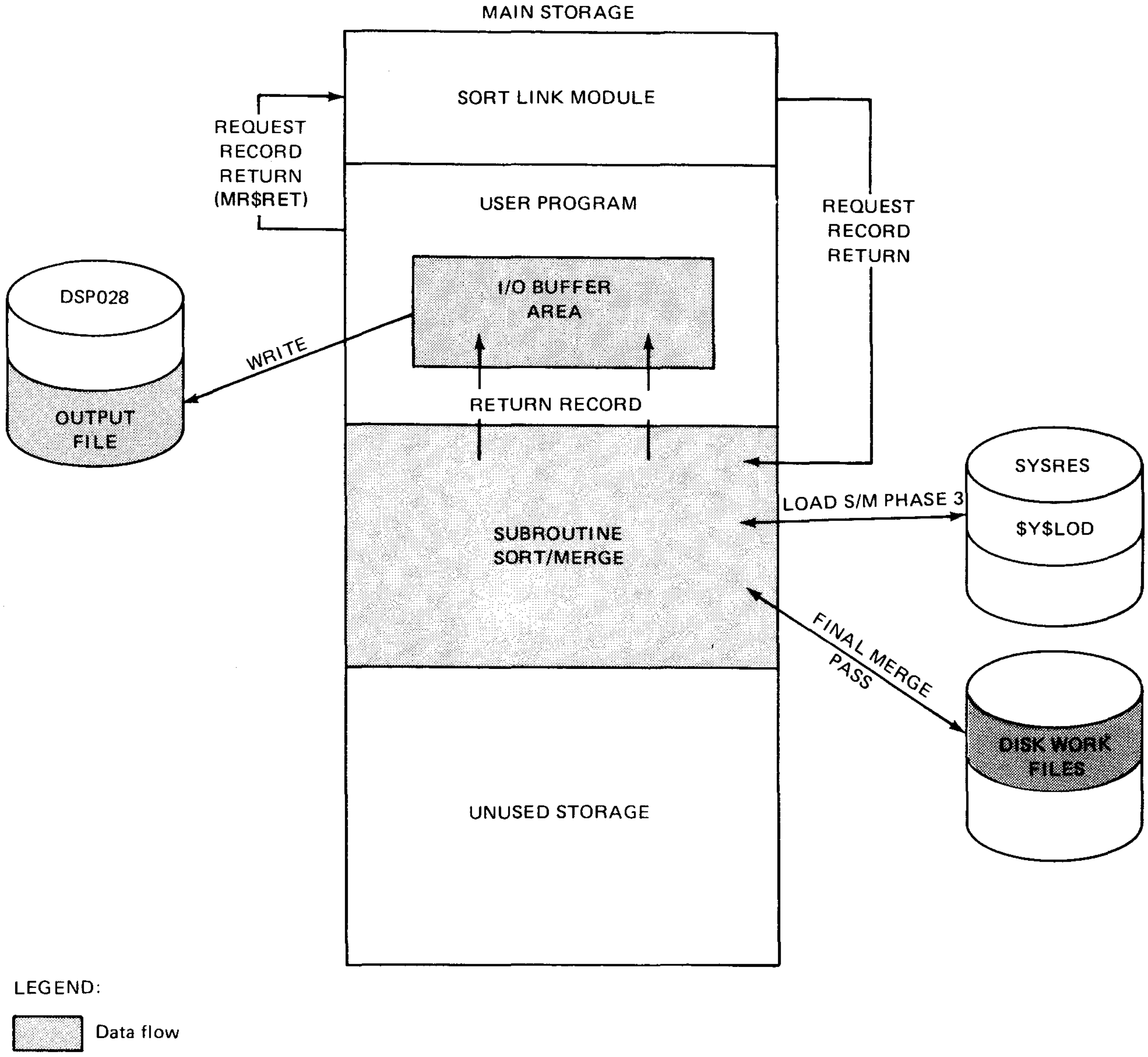


Figure 5—4. Writing Sorted Records to the Output File (Phase 3)

5.3. WHAT SORT/MERGE DOES FOR YOU

Subroutine sort/merge is a sort/merge utility that you can call from your program via macro instructions. The sort accepts unordered data from your program and returns it to your program in the order you specified.

5.3.1. Software Framework

The subroutine sort/merge operation consists of two to four phases. Each phase employs a specified sort/merge module to perform a distinct function. As each phase of the sort/merge is performed, the sort/merge modules needed during that phase are loaded into main storage and executed. Sort/merge modules are interrelated yet independent modules residing as *load modules* in $Y$LOD.

This brings us to the question of what happens during the subroutine sort/merge phases and how we access the sort/merge load modules. To answer the first question, let's take a closer look at the sort/merge phases. The answer to the second question should be clear when we examine what your program must supply to subroutine sort/merge.

5.3.2. Subroutine Sort/Merge Phases

Now that you know where the sort/merge modules reside and have a general idea of their functions, you realize that sort/merge must be called by your program in order to begin sorting records. To call the subroutine sort/merge, you must link the SG$ORT object module to your program. This module resides in $Y$OBJ and is automatically linked to your program when you specify the label MR$ORT as an EXTRN. SG$ORT initiates the subroutine sort/merge when the MR$OPN macro instruction is executed.

5.3.2.1. Phase 0: Sort Initialization and Assignment

Phase 0, the sort initialization and assignment phase, is the first phase executed. It collects and analyzes all information required by the following phases in determining the overall sort/merge requirements. It extracts this information from the data your program provides via parameter statements either in the MR$PRM sort macro or in the PARAM job control statement. When the assignment function is completed, phase 0 passes control to phase 1 or, in a merge-only procedure, to phase 3.

5.3.2.2. Phase 1: Initial Sort

In the beginning of phase 1, subroutine sort/merge accepts successive records from your program, compares sort keys, and initially sorts them according to your specification (e.g., ascending or descending sequence). During this phase, the records are accumulated in sequential lists called *record strings.* These sequenced record strings are then written out to tape or disk.

If you remember your first two blocks of data records with key fields for comparison, the input and initial sort process would look like Figure 5—5.

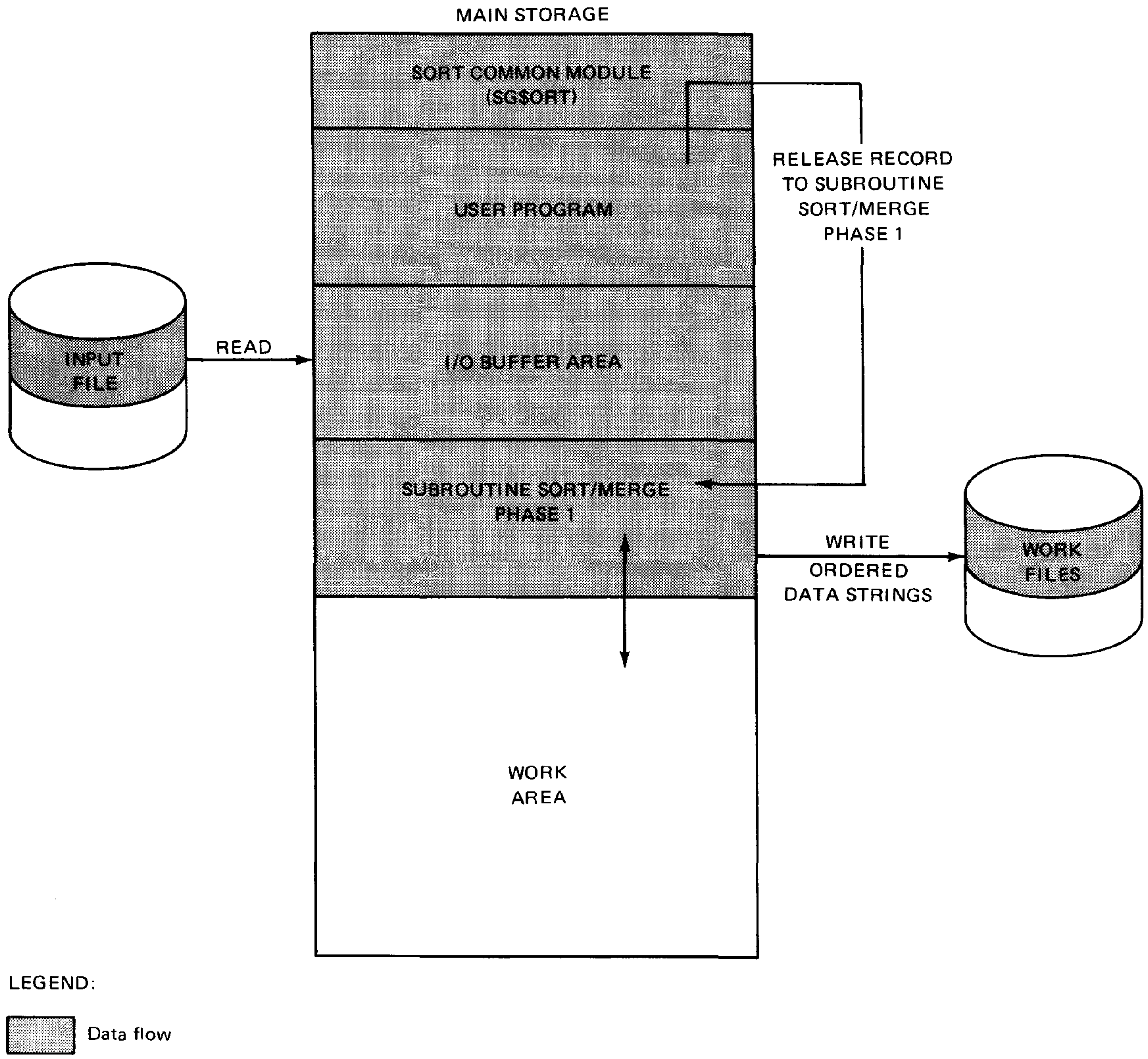


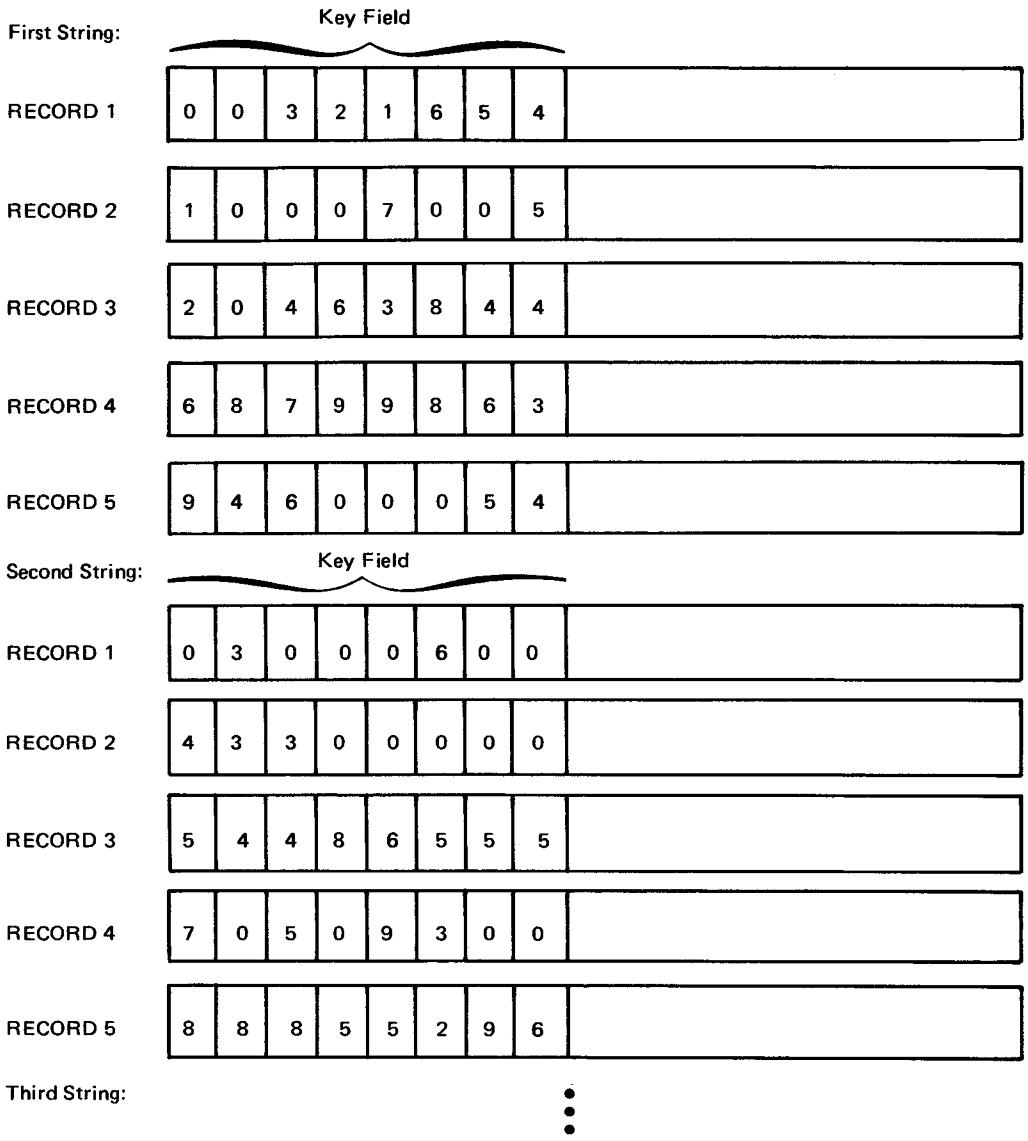
Figure 5—5. Data Route (Phase 1)

If you assign inadequate auxiliary storage to your program, the sort job step terminates. However, if your work files are on disk, you can prevent your program from aborting by including instructions that will repeatedly check the availability of work space. (See 6.6.)

5.3.2.3. Phase 2: Preliminary Merge

Phase 2 is initiated by the release of the last record to the subroutine sort/merge. Subroutine sort/merge repeatedly merges the record strings produced during phase 1 so that each successive pass produces fewer but longer sequential record strings. It continues this process until only one final merge is needed to produce a single string of sorted records. At this point, the subroutine sort/merge passes control to phase 3 for the final merge.

If the record strings produced during phase 1 can be sequenced in one merge pass, phase 2 is unnecessary and it is bypassed. This occurs when input to phase 1 is small or closely resembles the final sequence desired, or a large amount of main storage is available to the program. When a bypass occurs, phase 2 is skipped and control passes from phase 1 to phase 3 for the final merge. If we required phase 2, the record strings at the end of the phase would look like this:



First String: Key Field

RECORD 1 0 0 3 2 1 6 5 4

RECORD 2 1 0 0 0 7 0 0 5

RECORD 3 2 0 4 6 3 8 4 4

RECORD 4 6 8 7 9 9 8 6 3

RECORD 5 9 4 6 0 0 0 5 4

Second String: Key Field

RECORD 1 0 3 0 0 0 6 0 0

RECORD 2 4 3 3 0 0 0 0 0

RECORD 3 5 4 4 8 6 5 5 5

RECORD 4 7 0 5 0 9 3 0 0

RECORD 5 8 8 8 5 5 2 9 6

Third String:

5.3.2.4. Phase 3: Final Merge

Phase 3 performs the final merge of the sequenced record strings and produces a single string of sorted records that would look like this:

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	

⋮

After the final merge of the record strings is completed, phase 3 makes the records available to your program. At this point your program is responsible for requesting the return of the sorted records and for writing them into your output file. The subroutine sort/merge module operations and their interface with your main program are shown in Figure 5—6.

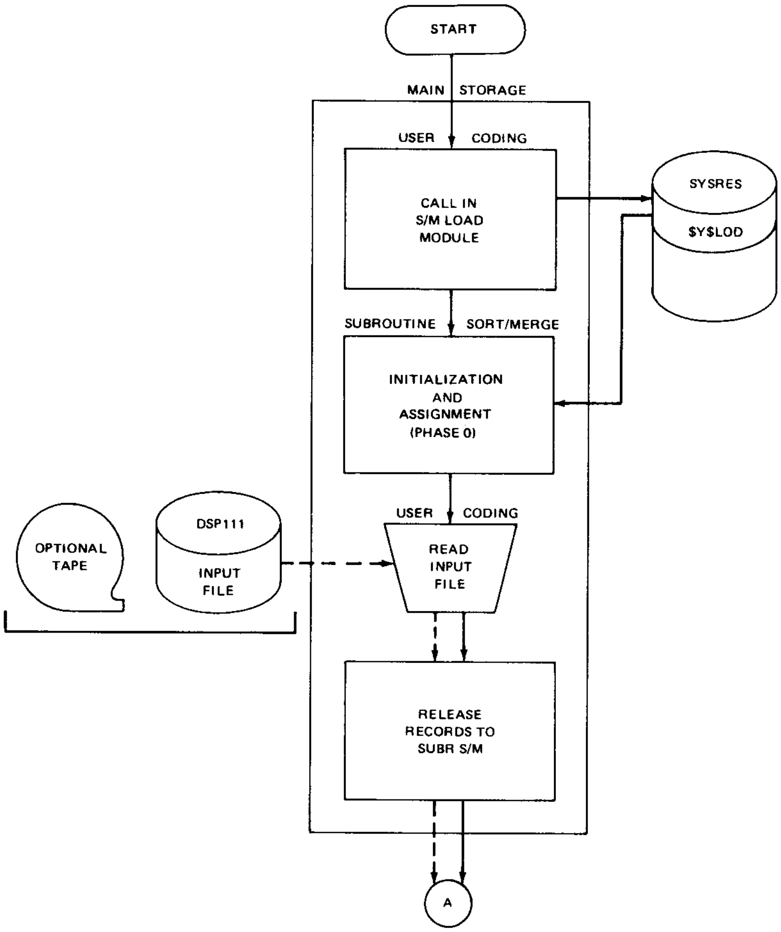


Figure 5—6. Subroutine Sort/Merge Operational Phases (Part 1 of 2)

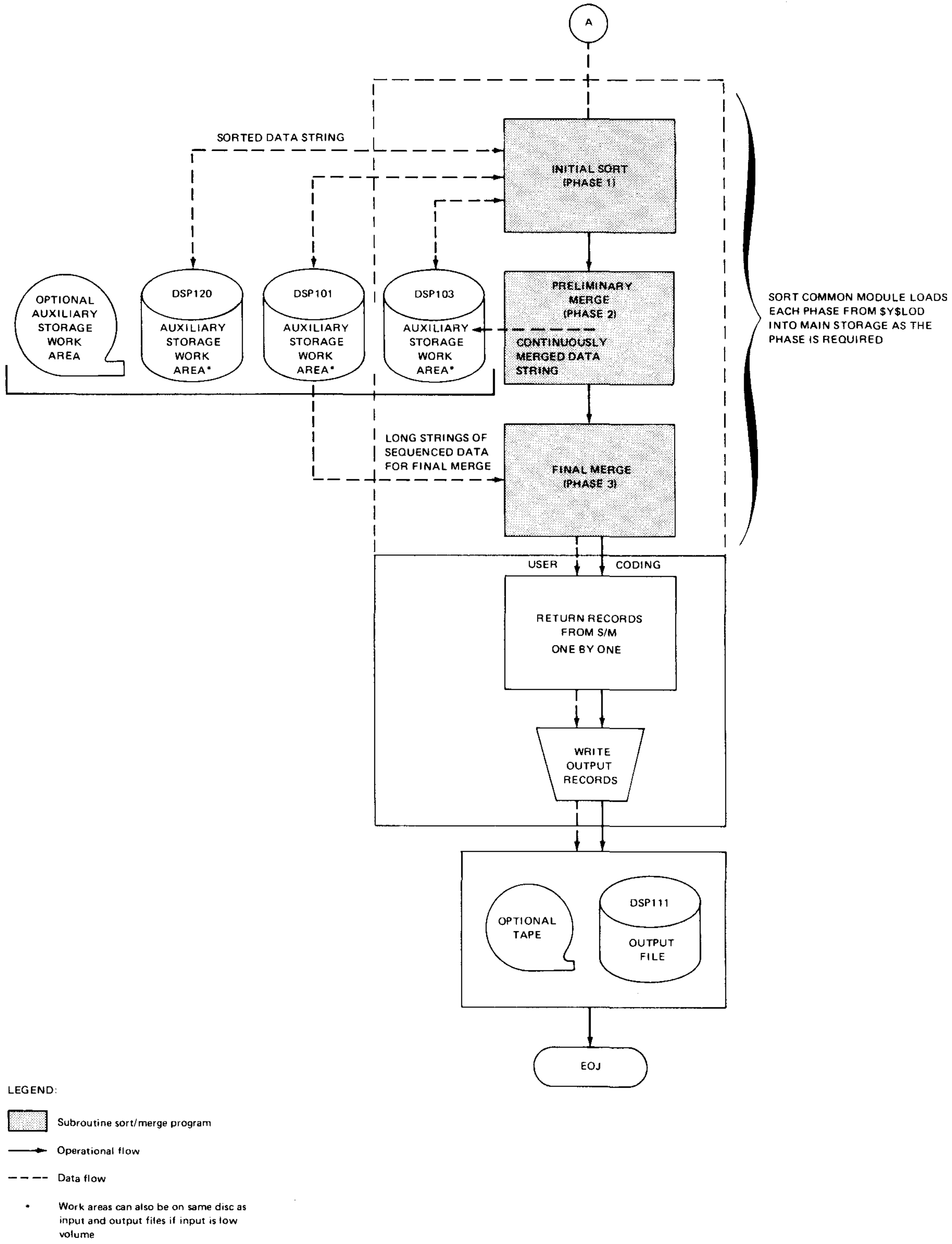

A
SORTED DATA STRING
INITIAL SORT (PHASE 1)
OPTIONAL AUXILIARY STORAGE WORK AREA
DSP120 AUXILIARY STORAGE WORK AREA*
DSP101 AUXILIARY STORAGE WORK AREA*
DSP103 AUXILIARY STORAGE WORK AREA*
PRELIMINARY MERGE (PHASE 2)
CONTINUOUSLY MERGED DATA STRING
SORT COMMON MODULE LOADS EACH PHASE FROM $Y$LOD INTO MAIN STORAGE AS THE PHASE IS REQUIRED
LONG STRINGS OF SEQUENCED DATA FOR FINAL MERGE
FINAL MERGE (PHASE 3)
USER
CODING
RETURN RECORDS FROM S/M ONE BY ONE
WRITE OUTPUT RECORDS
OPTIONAL TAPE
DSP111 OUTPUT FILE
EOJ
LEGEND:
Subroutine sort/merge program
Operational flow
Data flow
* Work areas can also be on same disc as input and output files if input is low volume

Figure 5—6. Subroutine Sort/Merge Operational Phases (Part 2 of 2)

Now that the importance of your main program cooperation with subroutine sort/merge becomes quite evident, let's examine how you handle the needed interfacing activities (Section 6).

6. Subroutine Sort/Merge Requirements You Supply

6.1. GENERAL

To use the subroutine sort/merge, your program must establish a communication link with it. Like any other communication problem, a language is needed as a medium to convey information. Basic assembler language (BAL) or COBOL is your medium for communicating with subroutine sort/merge. Consequently, a good working knowledge of BAL or COBOL is needed if you plan to use subroutine sort/merge. This section discusses a disk subroutine sort/merge program example showing the use of BAL instructions and data management macros. Appendix C provides some subroutine sort/merge interface requirements for the COBOL programmer.

BAL is not your only communicating tool. You use a set of *sort/merge macro instructions* to activate subroutine sort/merge. These macro instructions, which you code as part of your program, are expanded into a sequence of machine instructions that bring the subroutine sort/merge modules into your program as they are needed. A final, essential communicating tool is the job control stream consisting of control statements that name the devices used by your program and the subroutine sort/merge modules, describe label and space allocations, and call for the execution of assembly and linkage editor software routines as they are needed.

An easy way to remember the subroutine sort/merge requirements you supply is to think of the word "IDEAS". Each letter of this word represents something your program must do when you use subroutine sort/merge:

I Initiate the operation.

D Define files.

E Explain sort/merge run requirements.

A Activate subroutine sort/merge services.

S Stop or end the sort/merge process.

Before coding any program, a flowchart is helpful. The flowchart for the disk sort program might look like Figure 6—1.

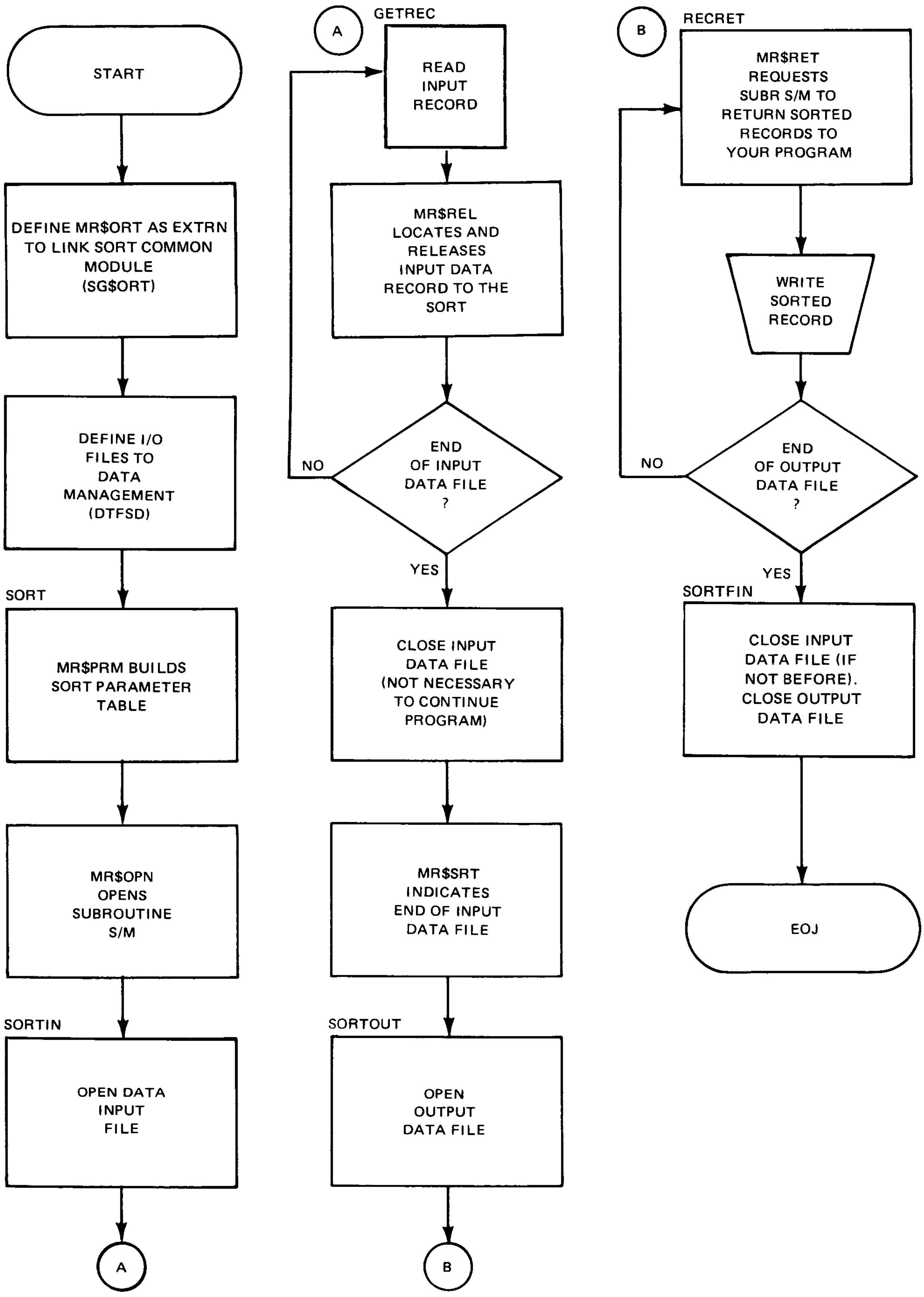

START
DEFINE MR$ORT AS EXTRN TO LINK SORT COMMON MODULE (SG$ORT)
DEFINE I/O FILES TO DATA MANAGEMENT (DTFSD)
SORT
MR$PRM BUILDS SORT PARAMETER TABLE
MR$OPN OPENS SUBROUTINE S/M
SORTIN
OPEN DATA INPUT FILE
A
A
GETREC
READ INPUT RECORD
MR$REL LOCATES AND RELEASES INPUT DATA RECORD TO THE SORT
NO
END OF INPUT DATA FILE ?
YES
CLOSE INPUT DATA FILE (NOT NECESSARY TO CONTINUE PROGRAM)
MR$SRT INDICATES END OF INPUT DATA FILE
SORTOUT
OPEN OUTPUT DATA FILE
B
B
RECRET
MR$RET REQUESTS SUBR S/M TO RETURN SORTED RECORDS TO YOUR PROGRAM
WRITE SORTED RECORD
NO
END OF OUTPUT DATA FILE ?
YES
SORTFIN
CLOSE INPUT DATA FILE (IF NOT BEFORE). CLOSE OUTPUT DATA FILE
EOJ

Figure 6—1. Disk Sort Program Flowchart

6.2. INITIATING THE OPERATION

The first thing you must do is name your program (we will use SRTEXMPL) and set the location counter to 0. The location counter always contains the address of the current instruction. To set the location counter to 0, use the START assembler directive.

```
LABEL     ΔOPERATIONΔ                    OPERAND
1         10         16
------------------------------------------------------------------
SRTEXMPL  START      Ø
```

Part of initiating subroutine sort/merge is to establish a communication interface between your program and the subroutine sort/merge program via a *sort common module*. The sort common module (SG$ORT) is a standard interface module that resides in the system object library file ($Y$OBJ). To establish the communication interface between your program and the subroutine sort/merge, you must link the sort common module to your program in the link edit run.

The linkage editor links the sort common module (SG$ORT) in $Y$OBJ to the user object module produced by the assembler.

To specify linkage, define the entry point for the common sort module in your program by naming MR$ORT as an external reference (EXTRN). This is done by coding line 2 as follows:

```
1.|SRTEXMPL START      Ø
2.|         EXTRN      MR$ORT
```

When the linkage editor processes your program, EXTRN tells it that MR$ORT is not defined in your program but refers to an object module which must be linked to it. The linkage editor makes the sort common module part of your program when it builds the load module for your program in the job run library file ($Y$RUN). This must be done before your program is loaded into main storage for execution. Once your program load module is loaded into main storage, the sort common module loads phase 0 into main storage, and the sort common module remains there for the duration of subroutine sort/merge processing and provides a link between your program and the subroutine sort. Phase 0 loads the other phases (Figure 5—6).

Until now, the interfacing procedures we've discussed would look like Figure 6—2.

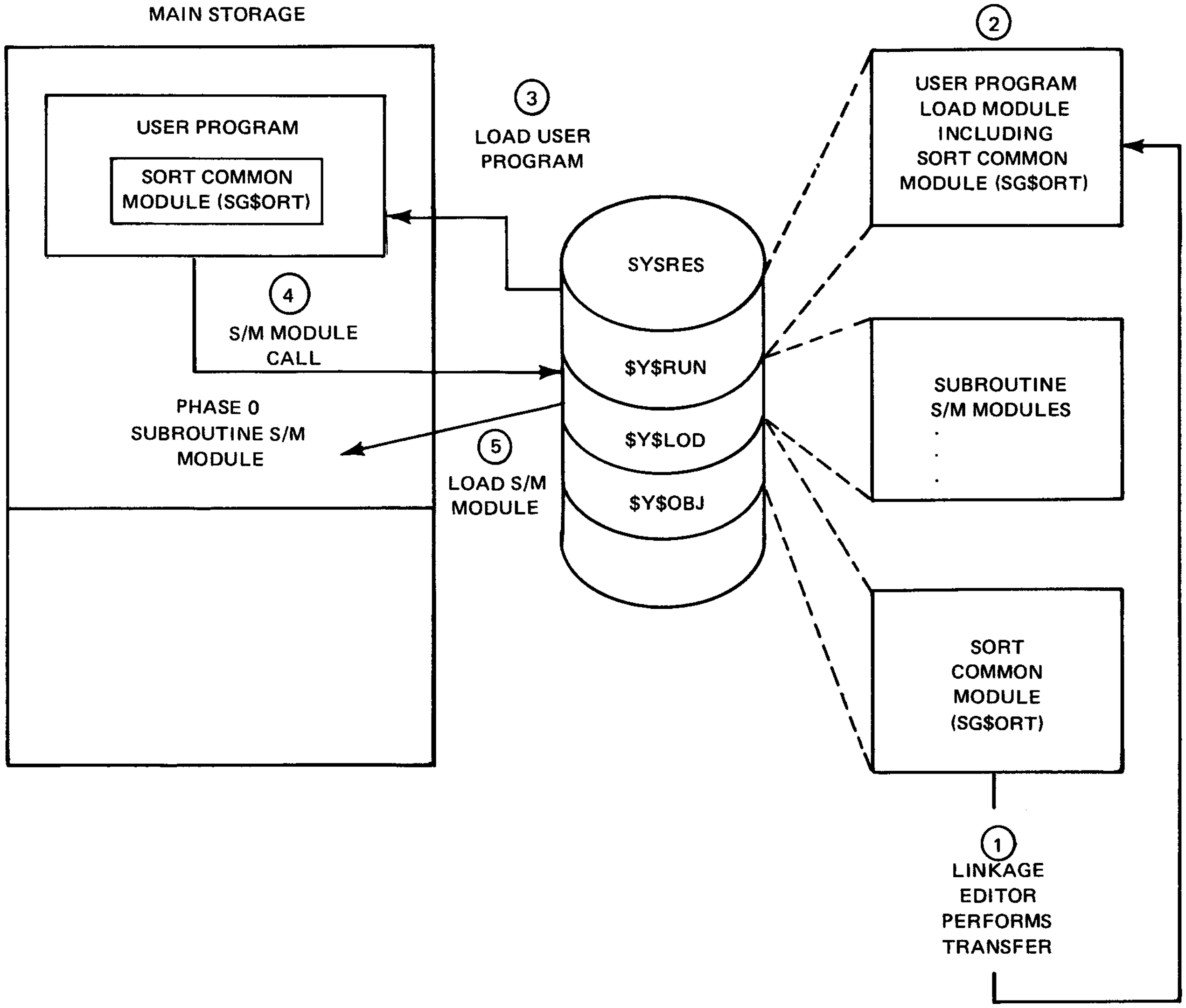

MAIN STORAGE
USER PROGRAM
SORT COMMON MODULE (SG$ORT)
4
S/M MODULE CALL
PHASE 0 SUBROUTINE S/M MODULE
3
LOAD USER PROGRAM
5
LOAD S/M MODULE
SYSRES
$Y$RUN
$Y$LOD
$Y$OBJ
2
USER PROGRAM LOAD MODULE INCLUDING SORT COMMON MODULE (SG$ORT)
SUBROUTINE S/M MODULES
SORT COMMON MODULE (SG$ORT)
1
LINKAGE EDITOR PERFORMS TRANSFER

Figure 6—2. Sort Common Module as Initial Interface

Naturally, you want to make your program relocatable. This can be done by using base register addressing; in our program, we will use base register 4. To do this, we code:

```
LABEL    ΔOPERATIONΔ                                OPERAND
1         10         16
-----------------------------------------------------------------------
          BALR       4,Ø
          USING      *,4
```

The branch and link assembler instruction loads the starting address of your program into register 4. When your program is loaded into main storage, its starting address is loaded into register 4, the base register. The 0 operand indicates that no branching is to occur.

The USING assembler directive assigns general register 4 to your program as the base register. The asterisk (*) operand indicates that the value assumed to be in register 4 when the program is assembled is the current value in the location counter.

Next we must branch to the beginning of our program. This is accomplished by coding:

```
LABEL   ΔOPERATIONΔ                     OPERAND
1       10       16
-----------------------------------------------------------------
        B        START
```

START is the label of the first instruction in our subroutine sort/merge program.

We have now completed the initialization of our program. To summarize, the coding for our disk sort program up to this point looks like this:

```
SRTEXMPL START   Ø
         EXTRN   MR$ORT                 DEFINES MR$ORT AS AN EXTRN
*                                       LINKS SORT COMMON MODULE TO
*                                       YOUR PROGRAM
         BALR    4,Ø
         USING   *,4
         B       START
```

6.3. DEFINING FILES

The software supplied by Sperry Univac includes another powerful component called *data management*, an elaborate group of routines that handle several types of processing, such as sequential, random, and indexed-sequential. When using subroutine sort/merge, you must provide your own I/O routines. Each record is read in order of its physical location on the tape or disk. (Subroutine tape sorts are shown in Section 9.) The several access methods, and therefore the sort/merge, can process only files defined by the DTF declarative macros peculiar to tape (DTFMT) or disk (DTFSD, DTFDA, DTFNI, or DTFMI) input files. When you want to sort records in a file, you must tell the data management software that the file you are processing is an input or an output file. The TYPEFLE keyword parameter serves this purpose. In order to operate properly, data management also needs specific information defining your program's data files.

NOTE:

In this discussion, all references to data management mean basic data management. Users of consolidated data management should refer to the consolidated data management concepts and facilities, UP-8825 (current version) for information on defining files.

In our discussion of your data files and the resultant files you wanted after subroutine sort/merge execution (5.2), we looked at some of the file definitions, such as record size, record format, and block size. Each of these specifications, in addition to other file information, must appear in your program in a section called file definitions. For example, you may define the files for sequential disk processing by using the DTFSD macro instruction. Data management uses your file definitions to supply file information to the system when your program requires it.

You can find the format of the DTFSD, DTFDA, DTFNI, DTFMI, or DTFMT macro instructions in the data management user guide, UP-8068 (current version). We will not repeat them here; if you need more information about these macro instructions, see the user guide. The coding for the input and output file definitions on our disk sort example program is illustrated in Figure 6—3.

```
LABEL    ΔOPERATIONΔ                      OPERAND
1        10       16                                                    72
-----------------------------------------------------------------------------
INPUT    DTFSD    BLKSIZE=4ØØ,RECSIZE=8Ø,IOAREA1=BUFF1,IOAREA2=BUFF2,    C
                  IOREG=(2),RECFORM=FIXBLK,ERROR=IOERROR,OPTION=YES,     C
                  EOFADDR=EOF,TYPEFLE=INPUT
OUTPUT   DTFSD    BLKSIZE=4ØØ,RECSIZE=8Ø,IOAREA1=BUFF1,IOAREA2=BUFF2,    C
                  IOREG=(2),RECFORM=FIXBLK,ERROR=IOERROR,OPTION=YES,     C
                  TYPEFLE=OUTPUT
```

Figure 6—3. Data Management Macro Specifications

In this disk sort problem, we're specifying:

- a maximum block size of 400 bytes;
- a record size of 80 bytes;
- an IOAREA1 called BUFF1 for the primary I/O buffer area;
- an additional IOAREA2 called BUFF2 to speed up I/O processing;
- a record format of fixed blocks; and
- an end-of-file address called EOF, which specifies the symbolic name of your program's routine to handle the end of input file processing.

You do not need an end-of-file routine for the output file because that is handled by the end of job processing.

In addition, we're specifying the optional parameters ERROR, OPTION, TYPEFLE, and IOREG. The ERROR=IOERROR parameter specifies that your program includes a routine labeled IOERROR to handle unrecoverable errors. OPTION=YES indicates that both input and output files are optional; i.e., you won't always use both of them. Although omitting the file type parameter still generates the TYPEFLE=INPUT parameter by default, we have input and output sort files to process so we use TYPEFLE=OUTPUT. This parameter indicates that the second DTFSD is for an output file. The TYPEFLE=INPUT parameter reads header/trailer labels for the input file, and the TYPEFLE=OUTPUT parameter writes header/trailer labels for the output file. IOREG specifies the register used for incrementing record addresses during the reading and writing of records. Here we are using general register 2 as the index register.

When you specify IOAREA1 and IOAREA2, you must also define how much main storage is required to handle the block size you indicated on the BLKSIZE parameter. Thus, somewhere in your program you write the *define storage* statements as illustrated in lines 2 and 3 of the following coding:

```
    LABEL    ΔOPERATIONΔ                     OPERAND
    1        10        16

1. |         DS        0H
2. |         DS        CL8
3. | BUFF1   DS        CL4ØØ
4. |         DS        CL8
5. | BUFF2   DS        CL4ØØ
6. | SAVEAREA DS       18F
```

Not only does data management associate IOAREA1 and IOAREA2 with their names, BUFF1 and BUFF2, but it also looks for the needed space, indicated by character length 400 (CL 400). This data management space accommodates alternate input and output block processing. If your input files are on 8416 or 8418 fixed-sector disks, or if you want to make your program device independent, you must allow a multiple of 256 bytes for each buffer area. Since your input and output block size is 400 bytes, you would need 512 bytes for each buffer area. Data management requires two other define storage statements, DS CL8 (lines 2 and 4 on preceding coding form). These statements designate eight additional bytes of storage immediately preceding the first and second I/O areas (BUFF1 and BUFF2 in this case), and a save area defined as 18 full words (72 bytes) of storage, SAVEAREA DS 18F (line 6 on preceding coding form). This is the area where your program saves the contents of any register used during execution of GET and PUT imperative macro instructions.

Data management requires eight bytes before each buffer on output but it does not require eight bytes before each buffer on input. I/O buffers must be half-word aligned (see line 1 on preceding coding form), and the save area must be full-word aligned. It should be noted at this point that there are two ways of providing the save area address to data management: loading the address into general register 13 before entering the data management imperative macro (see line 9, Figure 8—5) or specifying the label of the area via the SAVAREA keyword parameter in your DTF. For more details about the SAVAREA keyword parameter, refer to the data management user guide, UP-8068 (current version). Using the SAVAREA keyword frees register 13 for other use by your program. Up to this point, our coding looks like Figure 6—4.

```
      LABEL    ΔOPERATIONΔ                        OPERAND
      1        10        16                                                            72

 1.  | SRTEXMPL START    Ø .
 2.  |          EXTRN    MRSORT                   DEFINES MRSORT AS AN EXTRN
 3.  | *                                          LINKS SORT COMMON MODULE TO
 4.  | *                                          YOUR PROGRAM
 5.  |          BALR     4,Ø
 6.  |          USING    *,4
 7.  |          B        START
 8.  | INPUT    DTFSD    BLKSIZE=4ØØ,RECSIZE=8Ø,IOAREA1=BUFF1,IOAREA2=BUFF2,           C
 9.  |                   IOREG=(2),RECFORM=FIXBLK,ERROR=IOERROR,OPTION=YES,            C
10.  |                   EOFADDR=EOF,TYPEFLE=INPUT
11.  | OUTPUT   DTFSD    BLKSIZE=4ØØ,RECSIZE=8Ø,IOAREA1=BUFF1.IOAREA2=BUFF2,           C
12.  |                   IOREG=(2),RECFORM=FIXBLK,ERROR=IOERROR,OPTION=YES,            C
13.  |                   TYPEFLE=OUTPUT

20a. |          DS       0H
21.  |          DS       CL8                      STORAGE REQUIRED FOR DM
22.  | BUFF1    DS       CL4ØØ
22a. |          DS       CL8
23.  | BUFF2    DS       CL4ØØ
24.  | SAVEAREA DS       18F                      DATA MANAGEMENT SAVE AREA
```

Figure 6—4. Subroutine Sort/Merge Disk Sort Coding — Part 1

6.4. EXPLAINING RUN REQUIREMENTS TO SUBROUTINE SORT/MERGE

Your program must describe its sort/merge requirements to subroutine sort/merge. You use the MR$PRM macro instruction to do this. Subroutine sort/merge uses the information specified by MR$PRM to build a sort parameter table. Each keyword parameter you specify with MR$PRM becomes an entry in the sort parameter table. (See Appendix B.)

Since the MR$PRM macro instruction has many parameters, we are going to show its format in two parts. The first part illustrates only the required parameters. After discussing the use of these parameters, the second part of the format shows the optional parameters followed by a discussion of their use. To complete the explanation, 6.4.3 explains the MR$PRM parameters we've specified for the disk sort program. Finally, 6.10 provides the entire MR$PRM macro format and a table that summarizes the use of the parameters.

6.4.1. Required MR$PRM Parameters

The MR$PRM format for the required parameters is:

LABEL	ΔOPERATIONΔ	OPERAND

[symbol]	MR$PRM	{ FIELD=(strt-pos-1,lgth-1[,form-1] [,seq-1] [,order-1]
		[,...,strt-pos-n,lgth-n[,form-n] [,seq-n] [,order-n]]) } ,
		{ RSOC=symbol, }
		FIN=symbol,
		IN=symbol,
		OUT=symbol,
		RCSZ=max-bytes,
		STOR= { symbol
		(symbol, number-of-bytes) }

The first thing you must do is choose either the FIELD or RSOC parameter. One or the other of these parameters is required but not both. If you are sorting by key field comparison, you indicate the FIELD parameter. It has subparameters that define the sort key fields to subroutine sort/merge. The key field definition includes starting position, length, data format, sorting sequences, and order of significance. You must specify at least the starting position and the length of key field. Specifications for the other subparameters are generated by default.

By writing a decimal number for starting position, you indicate the starting point of a key field relative to the beginning of the record. For subroutine sort/merge, there are two numbering scales for bytes in records: the *byte number* and the *byte position number*. Both byte numbers and byte position numbers proceed from most significant to least significant (left to right); however, byte numbers begin at 1 and increase, while byte position numbers begin at 0 and increase. Remember to specify key field starting positions by *byte position* in the record, not by byte number.

Using your record layout for the disk sort as an example (Figure 1—2), notice that the first key field starts at byte 1 of each record. You would specify 0 for the *strt-pos-1* subparameter because byte 1 corresponds with byte position 0 of the record (Figure 6—5).

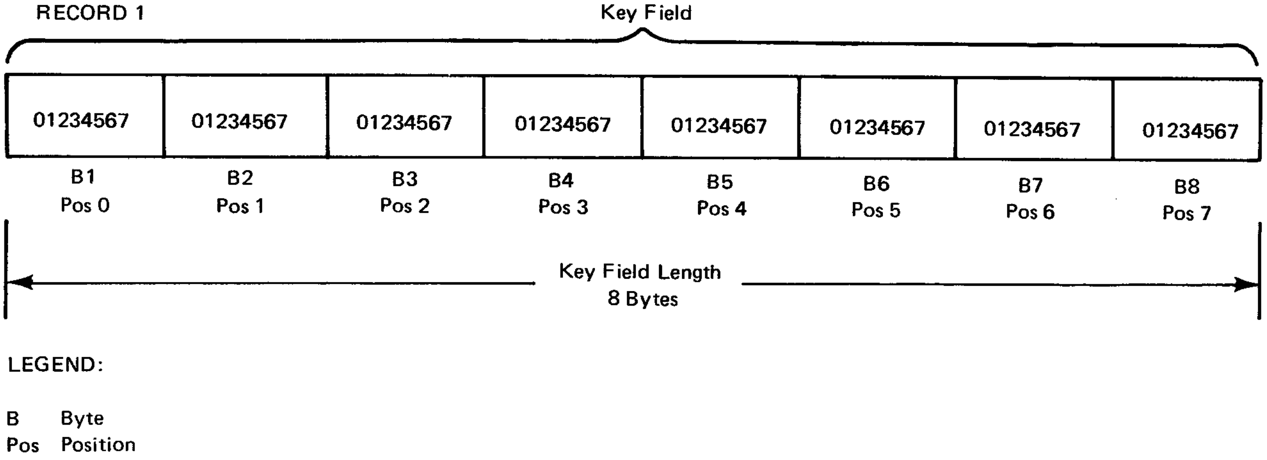


Figure 6—5. Key Field on Byte Boundary

All key fields, with the exception of binary key fields, start on a full-byte boundary so you can easily specify their starting points by using the byte position number in the record. When you want to specify a binary key field, the starting position is not limited to a byte boundary but can start at any bit position within a byte. Sometimes you might need to specify the binary key field starting position in a *byte-bit* format. Suppose that instead of starting in record byte 1 or byte position number 0, your 8-byte key field starts in bit position 2 of record byte 6. You would specify 5.2 for byte position number 5, bit 2 (Figure 6—6).

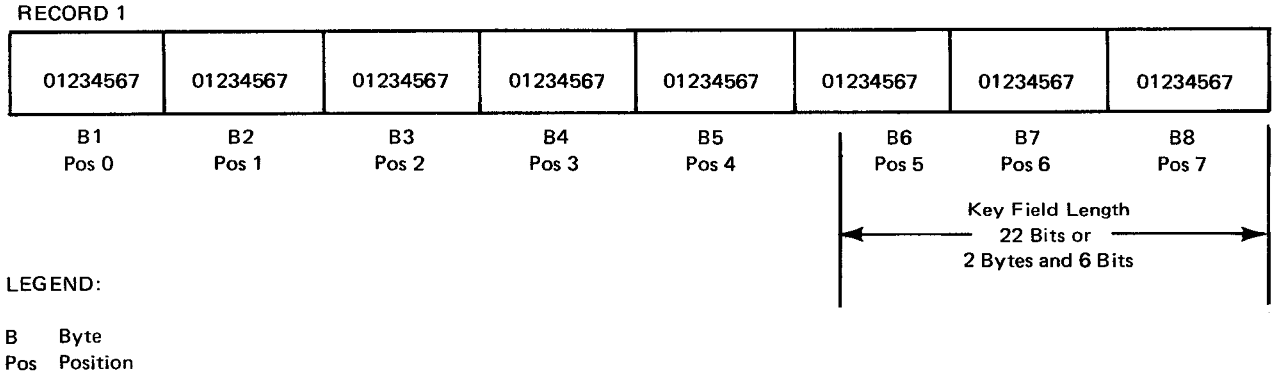


Figure 6—6. Binary Key Field with Bit-Byte References

The key field length subparameter (*lgth*) is also a mandatory specification. When you specify a key field in full bytes, *lgth* is a whole number indicating the total number of bytes the field occupies relative to the byte position number you specified in the *strt-pos* subparameter (Figure 6—5). Since your record key fields from the disk sort example are each eight bytes, you would write an 8 for the *lgth* subparameter as follows:

```
LABEL    ΔOPERATIONΔ                    OPERAND
1         10         16
-------------------------------------------------------------
          MR$PRM     FIELD=(Ø,8)
```

A binary key field's length is based upon the number of full bytes plus the number of bits the field occupies. Using Figure 6—6, you would specify 2.6 for the *lgth* subparameter, indicating a total of 22 bits or 2 bytes and 6 bits.

The *form* subparameter is not mandatory. It is a 2- or 3-character code that specifies the key field's data format.

If you did not specify one of the format codes in Table 6—1, the default would be CH for character code (*form*).

Table 6—1. Data Format Codes (Part 1 of 2)

Format Code	Description	Maximum Allowable Field Length (Bytes)
AC	Character (EBCDIC in ASCII collation sequence)	1—256
ASL	ASCII leading sign numeric	2—256
AST	ASCII trailing sign numeric	2—256
BI	Unsigned binary	1 bit—256
[redacted]	Character (EBCDIC or ASCII)	1—256
CLO	Overpunched leading sign numeric	1—256
CSL	Leading sign numeric	2—256
CST	Trailing sign numeric	2—256
CTO	Overpunched trailing sign numeric	1—256
FI	Fixed-point integer	1—256

Table 6—1. Data Format Codes (Part 2 of 2)

Format Code	Description	Maximum Allowable Field Length (Bytes)
FL	Floating point	1—256
MC	Multiple character, user-specified collating sequence	1—256
PD	Packed decimal	1—32
USQ	Character, user-specified collation sequence	1—256
ZD	Zoned decimal	1—32

Seq, the sorting sequence subparameter, could be A for ascending or D for descending. By not writing a specification, you accept ascending sequence, the default condition.

As many as 255 different key fields may be specified. The *order* subparameter designates the significance of multiple key fields from major to minor. The major key field is always numbered 1; the next most significant key field is 2; and so on up to the maximum specification of 255 key fields. If you omit the *order* subparameter, subroutine sort/merge assumes the order in which you define the key fields to be the order of significance. If you use *order* for one field, you must use it for all fields.

In the following coding example, line 1 describes a single key field. The key field *strt-pos-1* begins in byte position 0 and extends for seven bytes (*lgth-1*). The key field's data format (*form-1*) is EBCDIC in ASCII collation sequence (AC). The D indicates a descending sort sequence (*seq-1*). Line 2 describes three keys. Each key has its own parameter specifications. The first key has a starting position of byte position 5 extending through byte position 12 (eight bytes). The format is assumed character (EBCDIC or ASCII), and sort sequence is assumed ascending by default. The first key field is the second most significant key field (*order-1*). The second key field starts in byte position number 16 and extends through byte position number 18 (three bytes). Character format and ascending sort sequence are assumed by default, and the second key field is the major key since *order-2* indicates 1. Finally, the third key field starts in byte position number 58 and extends through byte position number 67 (10 bytes). Again, by default, the format is assumed to be character and sequence, ascending. Key field 3 is the third in order of major to minor key fields.

```
     LABEL    ΔOPERATIONΔ                        OPERAND
     1        10        16

1.            MR$PRM      FIELD=(Ø,7,AC,D)
2.            MR$PRM      FIELD=(5,8,,,2,16,3,,,1,58,1Ø,,,3)
```

Instead of specifying the FIELD parameter, you could choose the record sequencing own-code parameter (RSOC). If you decided to write your own routine for record sequencing, you would code RSOC and the symbolic name of your own-code routine:

```
LABEL    ΔOPERATIONΔ                            OPERAND
1        10       16

         MR$PRM        RSOC=MYROUT
```

This parameter overrides the FIELD parameter if you specify both FIELD and RSOC. RSOC is discussed in 8.2.

The sort/merge macro that initializes subroutine sort/merge is discussed in 6.5. Once initialization is complete, subroutine sort/merge looks for the entry address of your program. You define this entry location by specifying a symbolic name via the IN parameter of the MR$PRM sort macro:

```
         MR$PRM        IN=MYOPN
```

After sort/merge is complete and subroutine sort/merge is ready to return records to your program, it looks for the location within your program where it can return control. The OUT parameter symbolic name specifies this return location:

```
         MR$PRM        OUT=MYCLSE
```

As soon as the last sorted record is returned to your program, you've reached the output end-of-data and you must tell subroutine sort/merge where to pass control. The FIN parameter indicates your symbolic name for the output end-of-data routine:

```
         MR$PRM        FIN=MYEND
```

In addition to the areas you've set aside for the program itself and for input/output buffers, you need space in main storage for the subroutine sort/merge modules and for sort/merge operations.

Using the STOR parameter, you can indicate either:

- the symbolic name of the first main storage location available for subroutine sort/merge; or

- the symbolic name and maximum number of bytes (decimal) available in main storage, starting at that name.

If you do not give a maximum number of bytes, subroutine sort/merge uses main storage locations starting at the address you specify (e.g., WORK) to the upper limit of main storage allocated to your job region (Figure 6—7).

```
LABEL   ΔOPERATIONΔ                     OPERAND
1         10       16
________________________________________________________
          MR$PRM      STOR=WORK
```

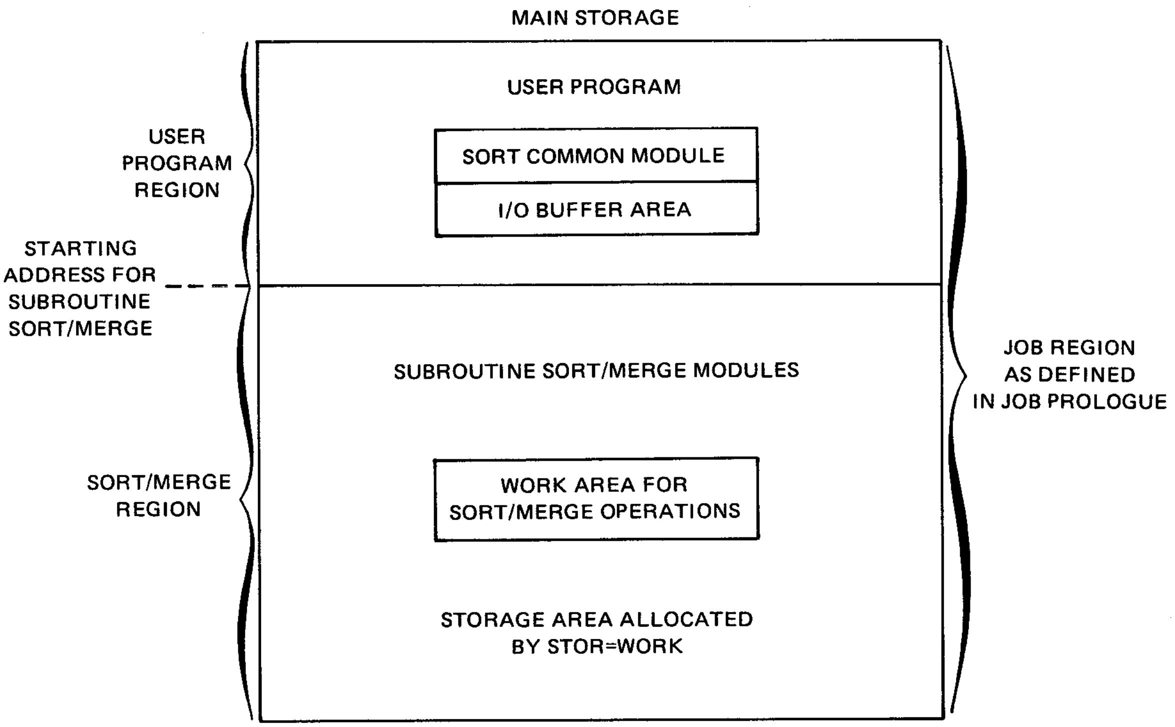

MAIN STORAGE
USER PROGRAM
SORT COMMON MODULE
I/O BUFFER AREA
USER PROGRAM REGION
STARTING ADDRESS FOR SUBROUTINE SORT/MERGE
SUBROUTINE SORT/MERGE MODULES
WORK AREA FOR SORT/MERGE OPERATIONS
STORAGE AREA ALLOCATED BY STOR=WORK
SORT/MERGE REGION
JOB REGION AS DEFINED IN JOB PROLOGUE

Figure 6—7. Main Storage Area Allotted by STOR without Number of Bytes Specified

If, for example, you specify a maximum number of main storage bytes by writing STOR=(WORK,15000), the main storage area allocated for subroutine sort/merge would extend 15,000 bytes from your starting address of WORK. Main storage space allocation would look like Figure 6—8.

```
          MR$PRM      STOR=(WORK,15ØØØ)
```

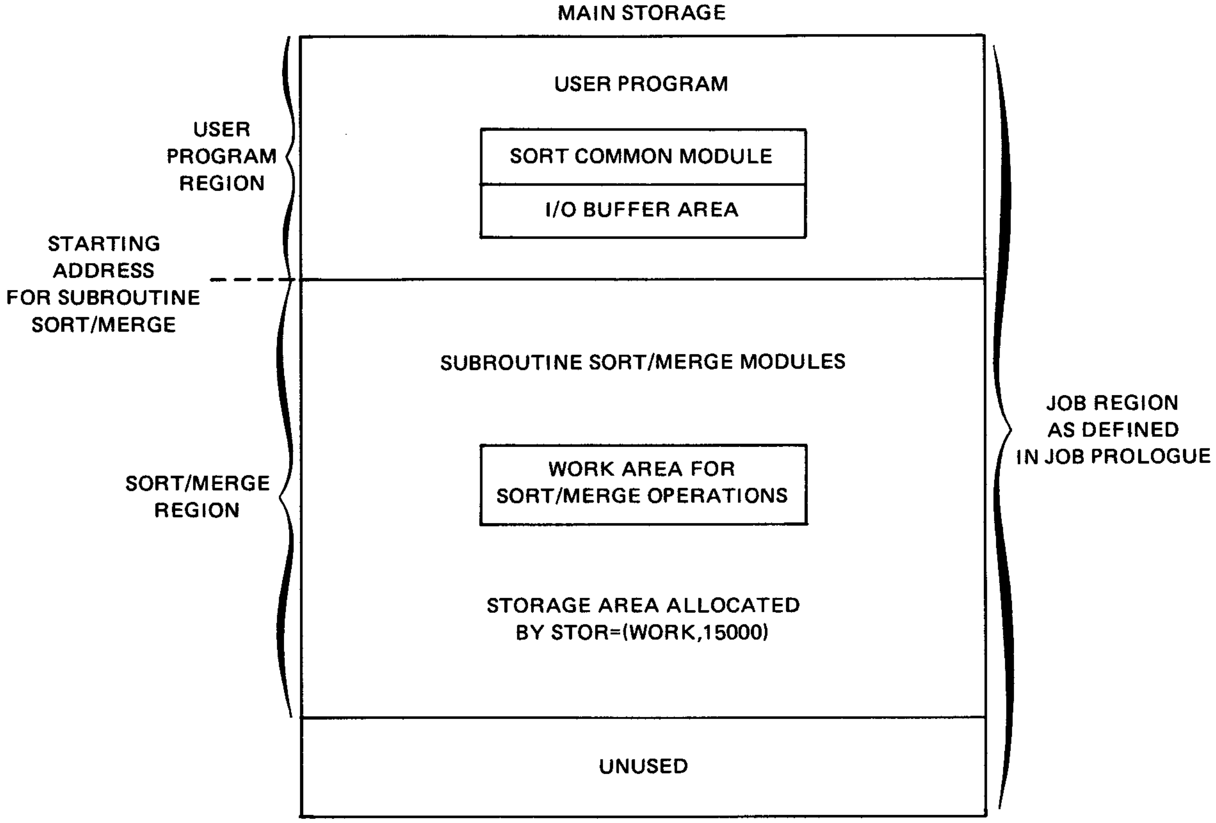

MAIN STORAGE
USER PROGRAM
SORT COMMON MODULE
I/O BUFFER AREA
USER PROGRAM REGION
STARTING ADDRESS FOR SUBROUTINE SORT/MERGE
SUBROUTINE SORT/MERGE MODULES
WORK AREA FOR SORT/MERGE OPERATIONS
SORT/MERGE REGION
STORAGE AREA ALLOCATED BY STOR=(WORK,15000)
JOB REGION AS DEFINED IN JOB PROLOGUE
UNUSED

Figure 6—8. Main Storage Area Allotted by STOR Specifying Maximum Number of Bytes

If you use the STOR parameter to specify the amount of main storage available to subroutine sort/merge, be sure to allocate a sufficient amount. See 1.6.1 for minimum main storage requirements.

The last required MR$PRM parameter is RCSZ. You must specify the size of fixed-length data records or the maximum size of variable-length data records to be sorted. Indicate a decimal number of bytes after the equal sign, e.g., RCSZ=80.

```
LABEL     ΔOPERATIONΔ                          OPERAND
1          10          16
________________________________________________________________
           MR$PRM      RCSZ=8Ø
```

Size specified for variable-length records must include the 4-byte record length field that precedes each record. If a tag sort has been indicated (ADDROUT keyword parameter specified), the record size must equal the combined length of all key fields specified plus the 10-byte record access address field. Maximum allowable record size depends somewhat upon the system hardware configuration.

This completes our discussion of the required MR$PRM parameters.

6.4.2. Optional MR$PRM Parameters

In addition to required parameters, MR$PRM has many optional parameters. Some are more frequently used than others. The following format shows all the MR$PRM optional parameters:

LABEL	△OPERATION△	OPERAND
[symbol]	MR$PRM	$\left[\text{,ADDROUT=}\begin{Bmatrix}\text{A}\\\text{D}\end{Bmatrix}\right]$
		[,ADTABL=symbol]
		$\left[\text{,BIN=}\begin{Bmatrix}\text{bytes}\\\text{(min-bytes,size-1,freq-1 [,...,size-n,freq-n])}\end{Bmatrix}\right]$
		$\left[\text{,CALC = }\begin{Bmatrix}\text{NO}\\\text{YES}\end{Bmatrix}\right]$
		$\left[\text{,CSPRAM=}\begin{Bmatrix}\text{NO}\\\text{OPTION}\\\text{YES}\end{Bmatrix}\right]$
		$\left[\text{,}\begin{Bmatrix}\text{DISC=}\begin{Bmatrix}\text{(address,max-disk-file-number)}\\\text{max-disk-file-number}\end{Bmatrix}\\\text{TAPE=}\begin{Bmatrix}\text{label-type}\\\text{(label-type,max-file-number)}\end{Bmatrix}\end{Bmatrix}\right]$
		$\left[\text{,DROC=}\begin{Bmatrix}\text{DELETE}\\\text{symbol}\end{Bmatrix}\right]$
		$\left[\text{,MERGE=}\begin{Bmatrix}\text{NO}\\\text{YES}\end{Bmatrix}\right]$
		$\left[\text{,NOCKSM=}\begin{Bmatrix}\text{D}\\\text{T}\end{Bmatrix}\right]$
		[,PAD=bytes]
		$\left[\text{,PRINT=}\begin{Bmatrix}\text{ALL}\\\text{CRITICAL}\\\text{NONE}\end{Bmatrix}\right]$
		[,RESERV =sort-filename]
		[,RESUME=(PASS,recovery-number)]
		[,SHARE=sort-filename]
		[,SIZE=number]
		[,USEQ=(to-address,from-address)]

In order to help you relate these optional parameters with their functions, we will discuss them under these categories:

- Device assignment parameters
- Record definition parameters
- Restart parameter
- Miscellaneous parameters

Before each categorical explanation we will list the parameters to be discussed.

6.4.2.1. Device Assignment Parameters

Parameters used to define devices include:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	MR$PRM	$\left[, \begin{Bmatrix} \text{DISC=}\begin{Bmatrix} \text{(address,max-disk-file-number)} \\ \text{max-disk-file-number} \end{Bmatrix} \\ \text{TAPE=}\begin{Bmatrix} \text{label-type} \\ \text{(label-type,max-file-number)} \end{Bmatrix} \end{Bmatrix} \right]$ [,RESERV =sort-filename] [,SHARE=sort-filename]

The DISC and TAPE parameters identify the storage medium assigned to your work files. You must first decide whether to use tape or disk. Suppose you choose disk. You would decide whether to specify:

- address and maximum disk file number; or
- maximum disk file number.

The *address* subparameter specifies the symbolic name of a list of your own user-supplied disk file names. The *max-disk-file-number* subparameter specifies the maximum number of files available to sort/merge. This number must not exceed eight. Line 1 in the following coding shows an example of the *address* and *max-disk-file-number* specification. On the other hand, you can specify only the *max-disk-file-number*. This indicates the maximum number of standard disk file names (not to exceed eight) assigned to subroutine sort/merge. Line 2 of the following coding shows a maximum of seven disks to be used for work files.

```
   LABEL    ΔOPERATIONΔ                      OPERAND
   1         10        16

1.          MR$PRM     DISC=(MYLABEL,7)
2.          MR$PRM     DISC=7
```

By using the TAPE parameter, you can identify the tape labels you want for all work (scratch) tape files in your program and specify the maximum number of sort files that may be assigned for subroutine sort/merge use. If you chose tape as your storage medium, you have to decide whether to specify:

- label type (NO or STD); or
- label type and maximum file number.

Tapes are either unlabeled or labeled standard. In the following example, the first specification indicates that you are assigning unlabeled tapes as scratch tape files; the second specification assigns standard label tapes as scratch tape files.

```
   LABEL   ΔOPERATIONΔ                       OPERAND
   1        10       16
1.|         MR$PRM    TAPE=NO
2.|         MR$PRM    TAPE=STD
```

If you specify both *label type* and *max-file-number*, you write the label type and a decimal number to indicate a maximum number of tape files you want assigned as working storage. The minimum is three; the maximum is six. For example, if you want to use standard labels and a maximum of four auxiliary working storage tapes, you code:

```
            MR$PRM    TAPE=(STD,4)
```

This TAPE parameter specifies only the assignment of standard labels to four tape work files. It does not assign standard sort tape file names. The LFD job control statement does that.

If you omit both the DISC and TAPE parameters, subroutine sort/merge will determine the type and number of work files from your LFD statements in the job control stream. For tape files, standard labels are assumed.

The RESERV parameter reserves a tape unit for use by a sort work file and by an output file. Sort/merge uses the tape unit as a work file during phases 0, 1, and 2. At the beginning of phase 3 when sort/merge transfers control to your program at the address you specify on the OUT parameter, the work file is closed and rewound to the unload point. After you dismount it and mount your data output file, the reserved tape unit accepts your output file on the same device (Figure 6—9). You might specify a standard tape sort file name (SM01,...,SM06) as follows:

```
            MR$PRM    RESERV=SMØ4
```

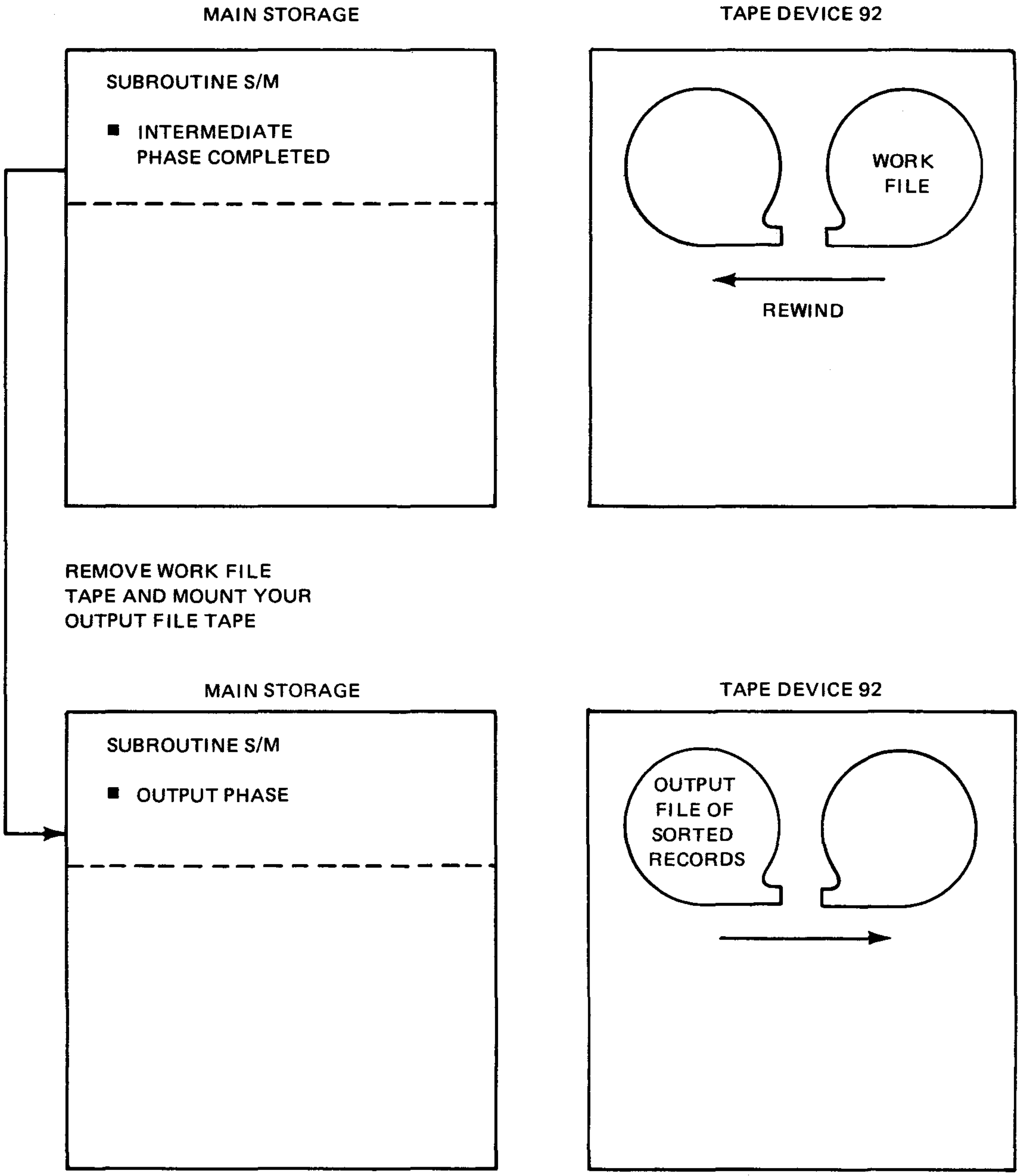

MAIN STORAGE
TAPE DEVICE 92
SUBROUTINE S/M
■ INTERMEDIATE PHASE COMPLETED
WORK FILE
REWIND
REMOVE WORK FILE TAPE AND MOUNT YOUR OUTPUT FILE TAPE
MAIN STORAGE
TAPE DEVICE 92
SUBROUTINE S/M
■ OUTPUT PHASE
OUTPUT FILE OF SORTED RECORDS

Figure 6—9. Same Work File Device Reserved for Output File Processing

The SHARE parameter allows a tape unit assigned to subroutine sort/merge to be used (shared) as a device for an input file during the initial sort phase of subroutine sort/merge and as a work file during the remaining phases of sort/merge operation. You designate a standard sort tape name (SM01,...,SM06) for the SHARE parameter:

```
LABEL    ΔOPERATIONΔ                    OPERAND
1         10         16
--------------------------------------------------------------------------
          MR$PRM     SHARE=SMØ1
```

The action taken might look like Figure 6—10.

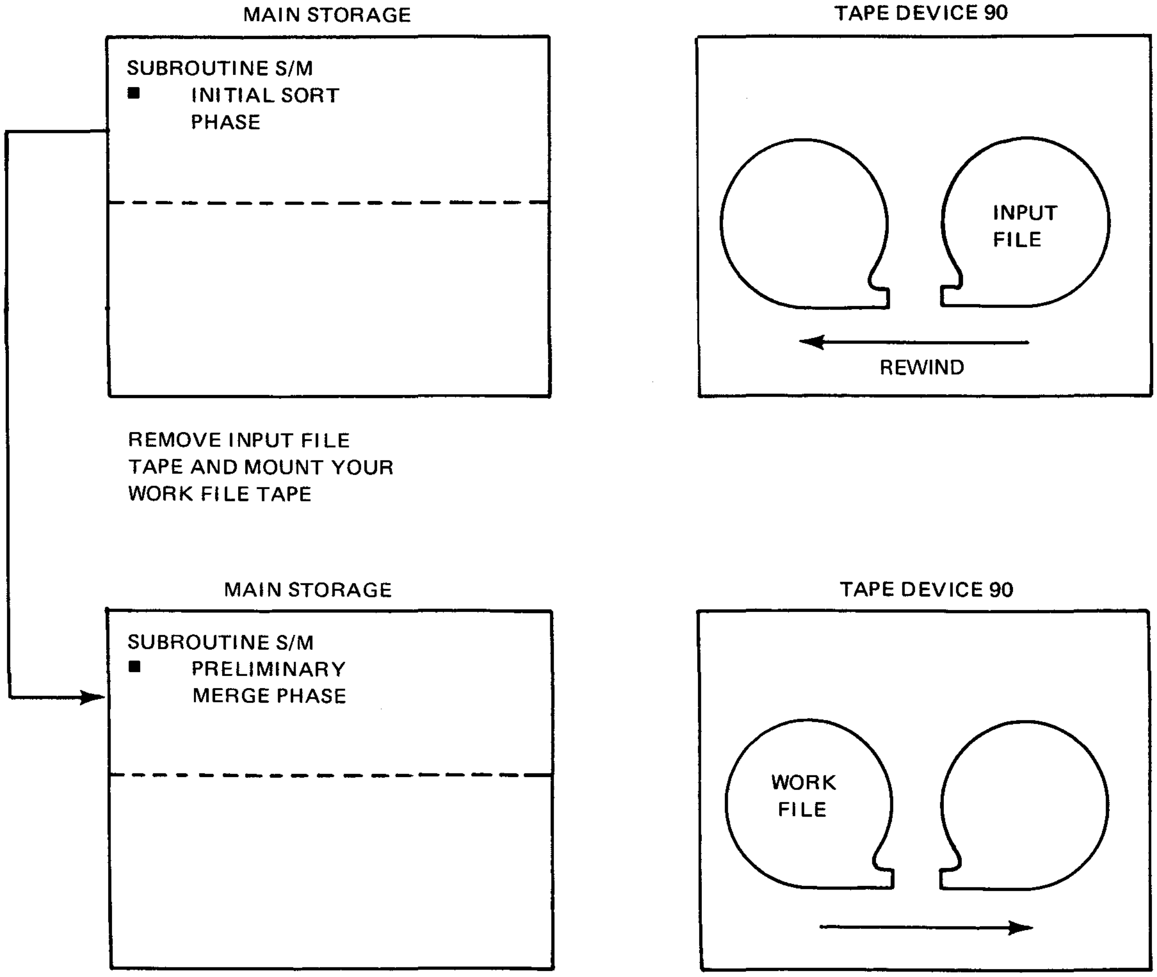


Figure 6—10. Same Input Device Shared between Input File and Sort Work File during Subroutine Sort/Merge Phases

Remember, a shared tape cannot be reserved and a reserved device cannot be shared. Associate the SHARE parameter with a dual-purpose input device and the RESERV parameter with a dual-purpose output device.

6.4.2.2. Record Definition Parameters

The following parameters define records:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	**MR$PRM**	**$\left[\text{,ADDROUT=}\begin{Bmatrix}\text{A}\\\text{D}\end{Bmatrix}\right]$** **$\left[\text{,BIN=}\begin{Bmatrix}\text{bytes}\\\text{(min-bytes,size-1,freq-1 [,...,size-n,freq-n])}\end{Bmatrix}\right]$** **[,USEQ=(to-address,from-address)]**

The ADDROUT parameter is related to a special sort application called a tag sort. The tag sort performed by subroutine sort/merge is a method of sorting in which the output file contains only the direct access addresses, or the addresses and key fields, of the records in the original file. The first 10 bytes of each reconstructed record contain the direct access address field. The total length of all key fields per tag sort record cannot exceed 256 bytes. A tag sort can be performed only when input is from a nonindexed or IRAM disk file. Multiple input files cannot be tag sorted. When input is from an IRAM file, the output of a tag sort is an IRAM file without an index.

When you want to perform a tag sort, you tell sort/merge via the ADDROUT parameter:

```
LABEL    ∆OPERATION∆                        OPERAND
1          10        16
--------------------------------------------------------------------
           MR$PRM      ADDROUT=A
           MR$PRM      ADDROUT=D
```

The ADDROUT parameter has two options. The D specifies that both the address field and the record key fields are returned to your own program in the sorted record. The A specifies that the sorted records returning to your program include only the address field. If you want to construct a separate file containing the sorted key fields you need and you also want to save the original addresses of the whole record that you tag sorted, specify D. You use A if you don't need to know the key field contents of the sorted records but want only their addresses for retrieving the entire original record at a later time. Figures 6—11 and 6—12 show unsorted key fields from four records and the resulting records returned to your output file after a tag sort. It is not the intent to show actual record formats in Figures 6—11 and 6—12 but only to illustrate the concept of record sorting by key fields and the outputs produced by a tag sort operation. To retrieve the disk address of the record for a tag sort, you must define the input file as a DTFNI file and use the imperative macro, NOTE, to obtain the record address. The NOTE macro must never be used with a DTFSD file. It is issued only to a DFFNI file.

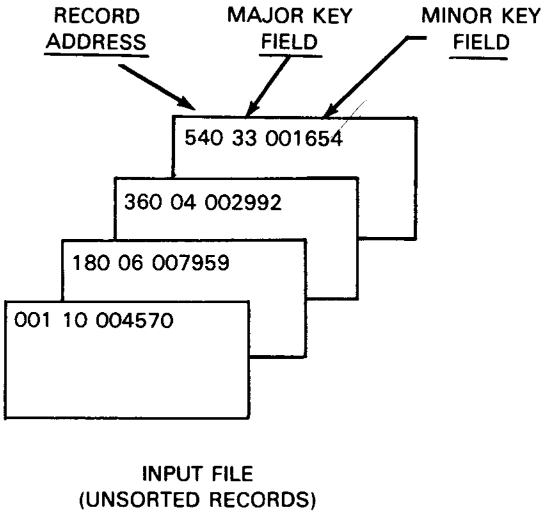


Figure 6—11. Input File, Unsorted Records (Additional Data Fields Not Shown)

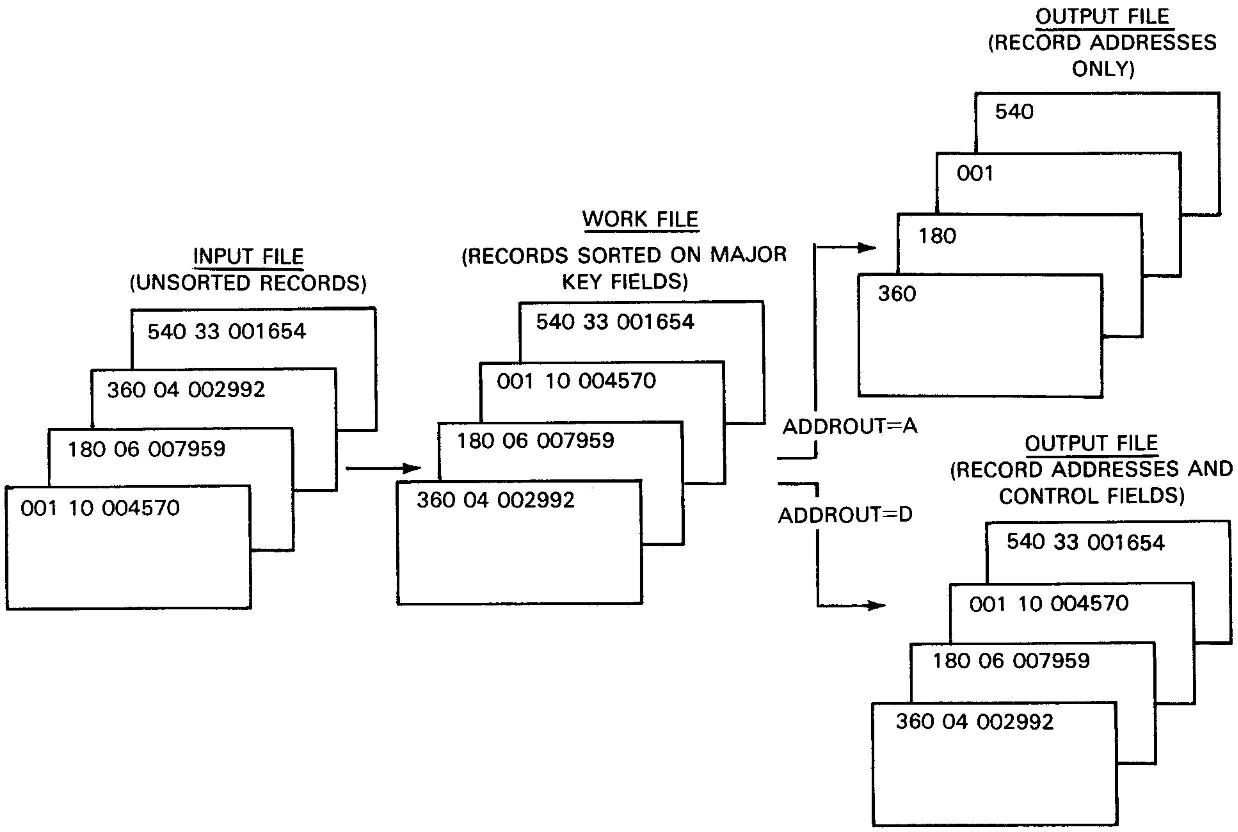
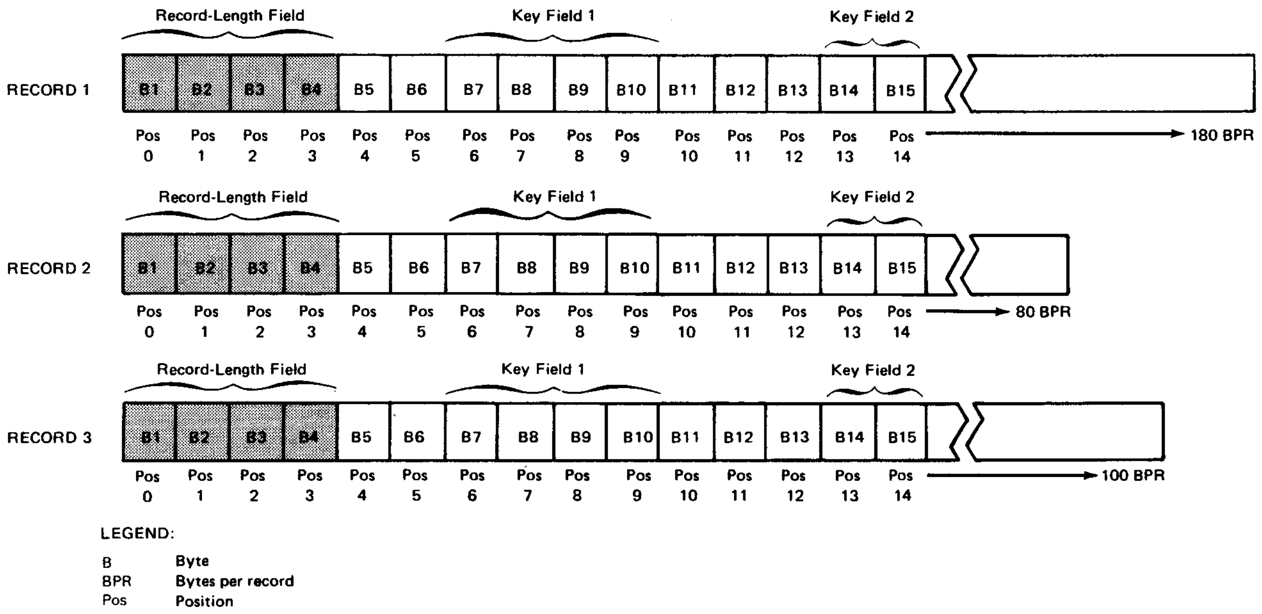


Figure 6—12. Tag-Sorted Output Files

Although the BIN parameter is shown as optional, it is required if your records are variable length. To conserve main storage space and provide optimum processing speed, subroutine sort/merge divides variable-length records into fixed-length subrecords called *bins*. Remembering that a 4-byte *record-length* field is considered a part of variable-length records, several of them with key fields might look like Figure 6—13.


Record-Length Field
Key Field 1
Key Field 2
RECORD 1
B1 B2 B3 B4 B5 B6 B7 B8 B9 B10 B11 B12 B13 B14 B15
Pos 0 Pos 1 Pos 2 Pos 3 Pos 4 Pos 5 Pos 6 Pos 7 Pos 8 Pos 9 Pos 10 Pos 11 Pos 12 Pos 13 Pos 14
180 BPR
RECORD 2
80 BPR
RECORD 3
100 BPR
LEGEND:
B Byte
BPR Bytes per record
Pos Position

·ords and BIN Size

There are two formats for the BIN parameter. The first format allows you to define the size of these subrecords (bins), and the second format allows you to supply information that subroutine sort/merge uses to calculate the bin size for you. If you specify the bin size yourself, remember that the size must be large enough so that the first bin may contain all the sort key fields within a record as well as the 4-byte record-length field. Examining Figure 6—13 to determine the number of bytes for the format 1 BIN parameter, notice that each record contains two key fields which extend 15 bytes into the record. Therefore, the minimum number you can specify is 15. However, since you have record lengths of 180, 80, and 100 bytes, all divisible by 20, a more efficient bin size to specify might be 20.

```
LABEL   ΔOPERATIONΔ                     OPERAND
1        10       16
-------------------------------------------------------------
         MR$PRM    BIN=2Ø
```

Suppose you have the same record information from Figure 6—13 but you decide to let subroutine sort/merge calculate the bin size. To calculate this number, subroutine sort/merge needs:

- the minimum number of bytes which can accommodate all sort key fields for each variable-length record plus the 4-byte record length field (*min-bytes*);
- the record length (*size-1*) appearing most frequently in the input file; and
- the number or percentage of *size-1* records in the input file (*freq-1*). If the number specified is less than 100, subroutine sort/merge assumes it to be a percentage. If 100 or greater, it is assumed to be an estimate of the number of records in the file.

The same information can optionally be specified for additional record sizes appearing in the input file. The following coding specifies that 15 bytes are needed to accommodate all key fields, that 50 percent of your input file contains 180-byte records, and that there are approximately two hundred 80-byte records and three hundred 100-byte records in the file. You need not specify every record size appearing in your input file.

```
         MR$PRM    BIN=(15,18Ø,5Ø,8Ø,2ØØ,1ØØ,3ØØ)
```

In our discussion of the FIELD parameter, we learned that there are many format codes used to perform collation sequences (Table 6—1). If you have a collation sequence for 8-bit character data differing from EBCDIC or ASCII representation, you may specify USQ on the *form-1* subparameter of the FIELD keyword parameter. In addition to the FIELD parameter, you specify the USEQ parameter of the MR$PRM macro.

```
LABEL    ΔOPERATIONΔ                         OPERAND
1         10        16
----------------------------------------------------------------------
          MR$PRM     FIELD=(Ø,8,USQ)
                           .
                           .
                           .
                           .
                           .
                     USEQ=(MYCODE,CODTRAN)
```

The *to-address* subparameter on the USEQ parameter specifies the address of a 256-byte table that translates the record fields into your own collation sequence. The *from-address* subparameter is the address of a 256-byte table that translates the fields back to the original data format code for output.

Usually one table is sufficient to perform the necessary translations and since both positional subparameters must be specified, you code the same address on both subparameters. Thus, you would probably write the following coding if one table is sufficient for the translations:

```
          MR$PRM     FIELD=(Ø,8,USQ),
                           .
                           .
                           .
                           .
                           .
                     USEQ=(MYCODE,MYCODE)
```

6.4.2.3. Restart Parameter

Suppose that somewhere in the middle of merging records into your desired sequence, the sort/merge program was interrupted. The number of collation passes previously made is shown on the system console. To restart your tape sort, you code the most recent collation pass number on the RESUME parameter.

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	MR$PRM	[,RESUME=(PASS,recovery-number)]

```
LABEL    ΔOPERATIONΔ                         OPERAND
1         10        16
_______________________________________________________________________
          MR$PRM        RESUME=(PASS,Ø53)
```

Instead of coding RESUME on the MR$PRM macro instruction and having to reassemble your program, you can enter it from the job control stream by submitting a PARAM job control statement (6.12), as in the following example:

```
// PARAM RESUME        =(PASS,Ø53)
```

In order to enter RESUME on a PARAM statement, you must have coded CSPRAM=YES on your MR$PRM macro (6.4.2.4).

Only tape sorts can be restarted. The disk cannot be repositioned as a tape is repositioned for a restart.

6.4.2.4. Miscellaneous Parameters

The remaining optional parameters are:

```
  LABEL   |  ΔOPERATIONΔ  |                OPERAND
==========|===============|==========================================
[symbol]  |  MR$PRM       | [ ,ADTABL=symbol]
          |               |
          |               | ⎡         ⎧ NO  ⎫ ⎤
          |               | ⎣ ,CALC = ⎩ YES ⎭ ⎦
          |               |
          |               | ⎡         ⎧ NO     ⎫ ⎤
          |               | ⎢ ,CSPRAM=⎨ OPTION ⎬ ⎥
          |               | ⎣         ⎩ YES    ⎭ ⎦
          |               |
          |               | ⎡       ⎧ DELETE ⎫ ⎤
          |               | ⎣ ,DROC=⎩ symbol ⎭ ⎦
          |               |
          |               | ⎡        ⎧ NO  ⎫ ⎤
          |               | ⎣ ,MERGE=⎩ YES ⎭ ⎦
          |               |
          |               | ⎡         ⎧ D ⎫ ⎤
          |               | ⎣ ,NOCKSM=⎩ T ⎭ ⎦
          |               |
          |               | [ ,PAD=bytes]
          |               |
          |               | ⎡        ⎧ ALL      ⎫ ⎤
          |               | ⎢ ,PRINT=⎨ CRITICAL ⎬ ⎥
          |               | ⎣        ⎩ NONE     ⎭ ⎦
          |               |
          |               | [ ,SIZE=number]
```

Just as MR$PRM builds the sort parameter table, the ADTABL parameter allows you to generate additional parameter tables and link them to the existing sort parameter table. It is important to code ADTABL as the last parameter of the MR$PRM it is used in, because subroutine sort/merge ignores all parameter entries following the ADTABL parameter (Figure 6—14). This symbolic label may be the beginning of an additional parameter table or any number of parameter tables. In addition to coding the ADTABL parameter last on your MR$PRM macro, you must create another sort parameter table in the current program or reference a sort parameter table from another program. To link tables within the same program later in the program, you indicate the symbolic label specified on the ADTABL and write a MR$PRM there as follows (line 13):

```
      LABEL    ΔOPERATIONΔ           OPERAND
      1         10        16                                            72
     ------------------------------------------------------------------------
 1.  | SORT     MR$PRM       FIELD=(Ø,8),                                 C
 2.  |                      IN=SORTIN,                                    C
 3.  |                      OUT=SORTOUT,                                  C
 4.  |                      FIN=SORTFIN,                                  C
 5.  |                      RCSZ=8Ø                                       C
 6.  |                      SOTR=WORK                                     C
 7.  |                      PAD=12,                                       C
 8.  |                      ADTABL=MYTABL
 9.  |                      .
10.  |                      .
11.  |                      .
12.  |                      .
13.  | MYTABL   MR$PRM       DISC=4,                                      C
14.  |                      ADDROUT=D
```

Figure 6—14. ADTABL Parameter Adding Table Entries within the Same Program

To reference sort parameter tables from other programs, you must indicate your symbolic name from the ADTABL parameter as an external reference in your program and as an entry point in the program being referenced (Figure 6—15). If duplicate fields exist in the two parameter tables, the first occurrence is used.

```
       LABEL    ΔOPERATIONΔ          OPERAND                      COMMENT
       1        10          16                                               72
      ______________________________________________________________________
      |SORT     MR$PRM       FIELD=(Ø,8),  \                                 C
      |                     IN=SORTIN,     |                                 C
      |                     OUT=SORTOUT,   |                                 C
      |                     FIN=SORTFIN,   ( FIRST PROGRAM SORT              C
      |                     RCSZ=8Ø,       > PARAMETER TABLE                 C
      |                     STOR=WORK,     |                                 C
      |                     PAD=16,        |                                 C
      |                     ADTABL=MYTABL  |
  ┌──>|           EXTRN     MYTABL         /
  |   |
ADDED |
AFTER |
FIRST |SRTAB    MR$PRM       FIELD=(12,4),\                                  C
TABLE |                     IN=SORTIN,    |                                  C
  |   |                     OUT=SORTOUT,  |                                  C
  |   |                     FIN=SORTFIN,  \ SECOND PROGRAM SORT              C
  └─{ |MYTABL               RCSZ=12Ø,     ( PARAMETER TABLE                  C
      |                     DISC=4,       |                                  C
      |                     ADDROUT=D     |
      |         ENTRY       MYTABL        /
```

Figure 6—15. ADTABL Parameter Referencing Table in Previous Program

Another very useful optional parameter is the CALC parameter. This parameter may be specified only for disk sorts. If you want the subroutine sort/merge to calculate optimum working storage, display information produced during sort initialization, and then terminate the job step, you must indicate CALC=NO (line 1).

```
1.          MR$PRM      CALC=NO
2.          MR$PRM      CALC=YES
```

The YES specification (line 2) causes the subroutine sort/merge to calculate optimum working storage, display sort information, and proceed with the sort as defined by the current sort parameter table. In either case, you must have specified the SIZE parameter in MR$PRM, as well as all required record description keyword parameters. The information displayed specifies the estimated sort time in minutes and the number of cylinders required for work space.

In the SIZE parameter, you indicate the approximate number of records to be sorted. This permits subroutine sort/merge to optimize its procedures. If you omit the SIZE parameter, subroutine sort/merge assumes a file of 25,000 records and the sort may not be optimized.

```
LABEL    ΔOPERATIONΔ           OPERAND
1        10        16                                                      72
-------------------------------------------------------------------------//----
         MR$PRM    SIZE=2ØØØ,                                               C
                   CALC=YES
```

Subroutine sort/merge also accepts sort/merge parameters from the job control stream by means of the PARAM job control statement. See 6.12 for parameters you can submit to subroutine sort/merge via the job control stream at run time. If you use this convenient method of entering sort/merge parameters in the sort parameter table, you specify your intention via the CSPRAM keyword parameter by coding YES (line 2). If you omit the CSPRAM parameter or code CSPRAM=NO (line 1), subroutine sort/merge will not look for PARAM statements in the job control stream.

```
1.|      MR$PRM    CSPRAM=NO
2.|      MR$PRM    CSPRAM=YES
```

It is advisable to specify CSPRAM=YES. Then, if you decide to add other parameters to your sort parameter table, you may do so; or, if you don't, the execution of your program is not affected. If you choose the default condition of CSPRAM=NO, you have to recode the MR$PRM macro and recompile your program to add parameters. Only BIN, DISC, NOCKSM, RESERV, RESUME, SHARE, and TAPE may be entered into the parameter table via the PARAM job control statement.

Suppose you know that your data files contain a large quantity of records with equal key fields. To avoid unnecessary key field comparison and redundancy in your output file, there is a convenient method of eliminating or combining these records with equal key fields. It's called data reduction own-code routine (DROC). This parameter allows you to specify automatic data reduction to be performed by subroutine sort/merge or by your own-code routine. Remember that record fields are duplicated in your files and that these whole records may be either eliminated or combined. Therefore, all records in your data files for data reduction must be fixed-length records. Never specify the DROC parameter for variable-length records. If you specify DELETE (otherwise known as auto delete), subroutine sort/merge performs data reduction automatically (line 1). Subroutine sort/merge uses registers to handle the saving and deleting of records with duplicate keys. For a more detailed description of how it performs deletion, read 7.3.

```
1.|      MR$PRM    DROC=DELETE
2.|      MR$PRM    DROC=MYWORK
```

Otherwise, the *symbol* you indicate on the DROC parameter specifies the symbolic label of your own-code data reduction routine entry address (line 2). Own-code routines can delete records with equal keys, summarize duplicate keys creating new records, or use a combination of keeping, deleting, or summarizing records. Thus, if you are interested in combining keys (summarizing) or a combination of deleting and combining, you must write your own routine and specify its name on the DROC parameter of the MR$PRM macro.

Subroutine sort/merge is capable of performing a merge-only application. You tell subroutine sort/merge to perform merge-only via the MERGE parameter:

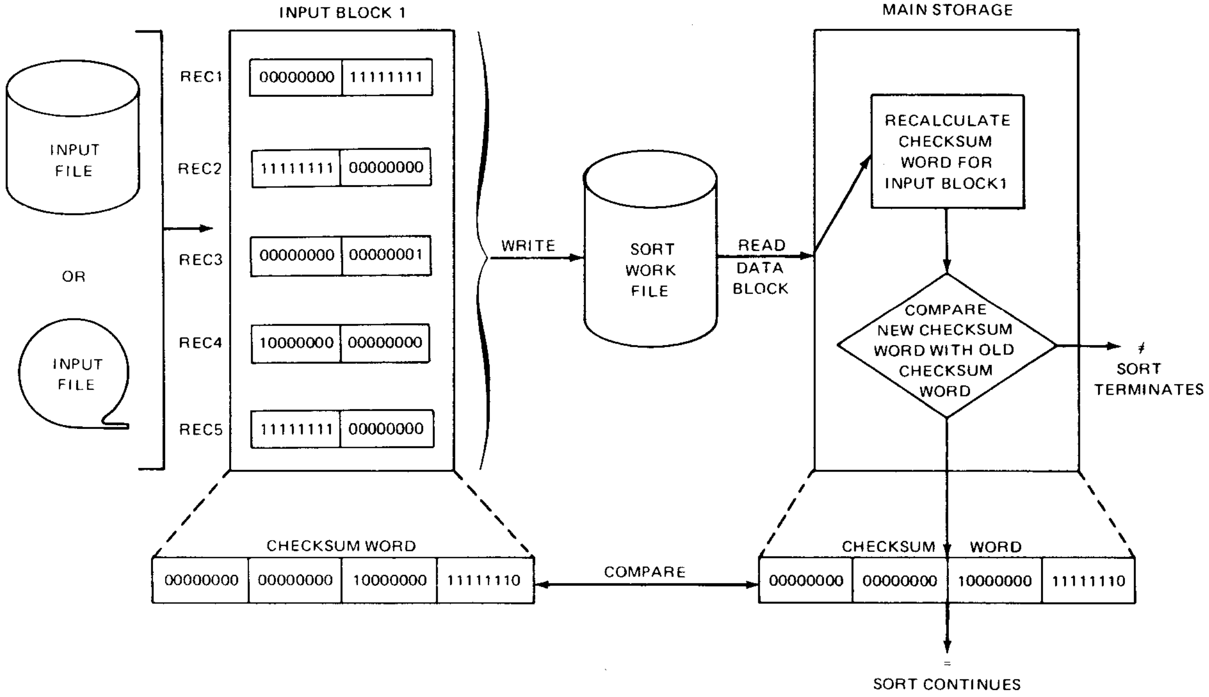
```
      LABEL    ΔOPERATIONΔ          OPERAND
      1         10        16
   ┌──────────────────────────────────────────────────────
1. │            MR$PRM      MERGE=YES
2. │            MR$P$M      MERGE=NO
```

If you omit this parameter, subroutine sort/merge assumes NO by default, a merge-only operation is not performed. You can also specify that this is *not* a merge-only operation by coding NO (line 2).

Normally, the subroutine sort/merge generates a *checksum word* for each output data block written to the tape or disk working storage areas. The checksum word provides a check of data integrity during read and write transfer operations (I/O processing) between the sort/merge operation and the sort work files.

The checksum word is calculated by logically summing, into a 1-word field, the records in the data block before they are written out to the sort work file. This checksum word is placed in the data block that is written to the sort work file.

Later, after the data blocks are read back into main storage from the sort work file, a checksum word is recalculated. Data integrity is then verified by comparing the new checksum word with the old checksum word. If the new word equals the old, the sort continues. If the comparison is unequal, the sort terminates. The checksum operation works as follows:



You can suppress this calculation of the checksum word by specifying the NOCKSM keyword parameter for the device type to which output data blocks are written.

```
        LABEL    ΔOPERATIONΔ          OPERAND
        1        10        16
      ------------------------------------------------------------
  1. |           MR$PRM       NOCKSM=D
  2. |           MR$PRM       NOCKSM=T
```

The D indicates no checksum word calculation for blocks written to disk (line 1). Specify T for no checksum word calculations on blocks written to tape (line 2). Since checksum word calculations are time consuming, it is wise to specify this parameter.

Another special optional parameter, PAD, allows you to augment the sort parameter table beyond its generated length. This enables you to enter additional parameters into the table from your own program at run time. The decimal number you enter specifies the number of additional bytes to be added to the sort parameter table. Remember that these bytes must be expressed in multiples of four.

The PAD parameter is used with the ADTABL parameter. The ADTABL specifies the name of the additional sort parameter table or the table being referenced in another program, and PAD specifies the number of extra bytes required for the additional parameter table entries.

The following coding illustrates two PAD parameters:

```
  1. |          MR$PRM       PAD=12
  2. |          MR$PRM       PAD=8
```

Also, see Figure 6—14, line 7 and Figure 6—15, line 7.

Subroutine sort/merge generates messages that are displayed on the system console or written in the job control spool log. The parameter PRINT allows you to specify that you want all messages (ALL), only critical messages (CRITICAL), or no messages (NONE) written into the system message log. If you omit PRINT, all messages are written to the system message log.

6.4.3. MR$PRM for the Disk Sort Program

Let's consider the specifications you might make in the parameter table for the disk sort program. Specifying the required parameters and other parameters pertinent to your disk sort (Figure 6—16), you might write:

```
LABEL   ΔOPERATIONΔ           OPERAND
1       10        16                                                      72
SORT    MR$PRM     FIELD=(Ø,8,CH),                                         C
                  IN=SORTIN,                                               C
                  OUT=SORTOUT,                                             C
                  FIN=SORTFIN,                                             C
                  RCSZ=8Ø,                                                 C
                  STOR=WORK,                                               C
                  DISC=4
```

Figure 6—16. Subroutine Sort/Merge Disk Sort Coding —Part 2

The FIELD parameter indicates that each of your record key fields start in byte position number 0, are eight bytes long, and are in the EBCDIC or ASCII character format (CH). Since your sorting sequence is ascending, you don't need to code an A for the *seq-1* subparameter because A is the normal default. In this case, you are not sorting on more than one key field so there is no need to specify the *order-1* subparameter for major or minor sort key fields. You assign the name SORTIN to the entry location of your program by using the IN parameter. You also assign the name SORTOUT to the location in your program where subroutine sort/merge can return control after it has sorted the records and it is ready to return them to your program.

When subroutine sort/merge returns the last sorted record to your program, it looks for the name of your output end-of-date routine. In your FIN parameter, you specified the name SORTFIN. Your records for the disk sort are 80-byte, fixed-length data records, so you specify a record size of 80 bytes.

On the STOR parameter, you indicate that the name of the first main storage location available for working storage is WORK and that this area extends to the upper limit of main storage allocated to your job region (Figure 6—7).

Since your sort/merge is being performed on disk, you specify in the DISC parameter that you want to use disk space for additional working storage on the sort. For this program, you choose to indicate only the maximum number of standard disk file names assigned to subroutine sort/merge. The 4 specifies that four file names are assigned to subroutine sort/merge.

6.5. ACTIVATING THE SUBROUTINE SORT/MERGE (MR$OPN)

Once you define the input and output files (DTFSD), establish the communications interface with subroutine sort/merge modules (MR$ORT), define the sort requirements (MR$PRM), and reserve input and output buffer areas, you need some way to activate phase 0 of subroutine sort/merge. The MR$OPN imperative macro instruction generates linkage to call the sort/merge initialization module into main storage. This module performs the initialization procedure of phase 0 before actual sort/merge execution. You may choose to open the input data files before or after you open the subroutine sort/merge; however, you must be sure to open both subroutine sort/merge and your input data files before releasing records to the sort.

A label on the MR$OPN macro is optional, but for the operand you must indicate either the symbolic label (address) of the sort parameter table or the number 1 indicating register 1 where you have previously loaded the address of your sort parameter table. A blank operand field will also indicate that register 1 was loaded with the parameter table address. In our disk program, we indicate the symbolic label of the sort parameter table on the MR$OPN macro instruction. Continuing the disk sort program coding from the last coding examples of Figure 6—4 and Figure 6—16, you would write:

```
        LABEL   ΔOPERATIONΔ          OPERAND
        1       10        16
       -------------------------------------------------------------------------
   25. | START   EQU       *
   26. |         MR$OPN     SORT                  OPEN THE SUBROUTINE SORT/MERGE
   27. | SORTIN  LA         13,SAVEAREA           POINT TO SAVE AREA
   28. |         OPEN       INPUT                 OPEN THE INPUT FILE
   29. | GETREC  GET        INPUT                 GET RECORD FROM INPUT FILE
   30. |         LR         1,2                   LOAD R1 WITH RECORD ADDRESS
   31. | *                                        ASSUMING R2 IS SPECIFIED AD
   32. | *                                        THE I/O REGISTER IN THE INPUT DTF
```

When the MR$OPN has opened the subroutine sort/merge, it passes control to your program at the address you specified in the IN keyword parameter of the MR$PRM macro instruction. According to your specification on the IN parameter for the disk sort program, your program receives control at the address of symbolic label SORTIN.

At this point, you begin your own program input routine. This routine loads the data management save area address into register 13, opens your input data file (if you haven't already opened it), reads each record, and sets the address of the record in register 1, preparing it for release to the sort. You label your first input routine instruction SORTIN because you want your program to receive control from subroutine sort/merge at that point. To set the address of the data management save area, you load it into register 13 via the *load-address* (LA) instruction (line 27).

Next, by issuing the OPEN imperative macro instruction (line 28), you open the input data file you named INPUT in your DTFSD. With the file open, you can read the input file by designating the GET imperative macro instruction (line 29). Since you plan to read many records and you'll need to repeat this instruction, you label it GETREC, giving yourself a place to return for reading subsequent records. Data management automatically loads the first data record address into register 2 when you specify IOREG=(2) on the DTFSD macro. Because the sort/merge expects the address of the record being released to it to be in register 1, you must load register 1 with the record address. In this example, a *load register* (LR) instruction is used.

6.6. GETTING DATA INTO THE SORT PROCESS

You've read the record and now you must pass it to subroutine sort/merge before returning to read subsequent records.

The MR$REL macro generates code to release unsorted records one at a time to the subroutine sort/merge for processing.

```
      LABEL   ΔOPERATIONΔ          OPERAND
      1       10      16
    ┌────────────────────────────────────────────────────────────────────
33. │         MR$REL                       RELEASE RECORD TO THE SORT
34. │         B           GETREC           GET NEXT RECORD
```

After the transfer occurs, subroutine sort/merge returns control to your program at the instruction immediately following the MR$REL macro. Now you want to read the next record, so you branch back to your GET instruction labeled GETREC. Reading records, setting the address in register 1, and releasing records to the sort procedure are repeated until the end of input file is reached. When data management reaches the end-of-input file, it looks for the symbolic address of your end-of-file routine. Your EOFADDR specification on the input DTFSD data management macro indicates the name EOF. This is your means of exiting the read record loop.

If you are using disk work files and are not certain whether you have assigned enough auxiliary storage, you can include a routine that will check on the availability of work area before each record is passed to the subroutine sort/merge. When control is returned to your program immediately following the MR$REL macro instruction, register 1 will be set to a positive value if more records can be accepted or to a negative value if work space may be insufficient to complete the sort. Use a *load and test register* (LTR) instruction to set register 1, followed by a *branch minus* (BM) instruction, which will cut short the read record loop. You have several alternatives at this point. You can complete the sort with only the records read thus far by branching to EOF; branch to your error processing routine, IOERROR (6.9, line 53); or write a special routine to handle this condition in some other way. In the example, we have labeled this routine NOROOM.

```
33a. │        LTR         1,1              LOAD R1 AND CHECK FOR NEGATIVE VALUE
33b. │        BM          NOROOM           GO TO 'NOROOM' ROUTINE
```

6.7. PASSING CONTROL TO OUTPUT PROCESS

After you read the last data record of the input file and reach the end of file, you designate the end-of-file routine and issue a CLOSE imperative macro to close the input file (although this is not required for continuing your program).

This is followed by the MR$SRT sort macro, which tells the subroutine sort/merge that you have reached the end-of-input data and that it may now complete the process of sorting and merging to produce the final results.

```
      LABEL   ΔOPERATIONΔ         OPERAND
      1        10        16
35. | EOF      EQU        *                    THIS LOCATION IS SPECIFIED
36. | *                                        AS THE END OF FILE ADDRESS
37. | *                                        IN THE INPUT DTF
38. |          CLOSE      INPUT                CLOSE THE INPUT FILE
39. |          MR$SRT                          TELLS THE SORT THAT THE
40. | *                                        END OF FILE HAS BEEN REACHED.
```

After the sort receives all input data records, it completes a preliminary merge of record strings. Subroutine sort/merge may skip this phase if your input file is small. The final merge (phase 3) always occurs, and subroutine sort/merge looks in your sort parameter table for the symbolic label you indicated on the OUT parameter of the MR$PRM. It passes control to this label address when it is ready to return the records to your program. Since you designated SORTOUT as the symbolic label for the disk sort program, the subroutine sort/merge returns control to that label address which is the beginning of your output routine.

6.8. DRAWING DATA FROM THE SORT PROCESS

Your output routine coding might continue as follows:

```
41. | SORTOUT  EQU        *                    OUT ADDRESS
42. |          OPEN       OUTPUT               OPEN THE OUTPUT FILE
43. | RECRET   MR$RET                          REQUEST A RECORD RETURNED.
44. |          MVC        Ø(8Ø,2),Ø(1)         MOVE THE SORTED RECORD TO
45. | *                                        THE OUTPUT BUFFER AREA.R2 IS
46. | *                                        ASSUMED SPECIFIED AS THE I/O
47. | *                                        REGISTER IN THE OUTPUT DTF.
48. |          PUT        OUTPUT               OUTPUT THE RECORD RETURNED
49. |          B          RECRET
```

To begin your output routine, you open the output file (line 42) and request subroutine sort/merge to return sorted records to your program via the MR$RET sort macro (line 43). Records are released to your program one at a time. Consequently, the MR$RET macro must execute for each returning sorted record. Because record writing is a repetitive process, and the MR$RET must execute for each record, you assign a symbolic label to the MR$RET macro instruction to develop your output record processing loop (line 43). MR$RET returns the address of sorted records one at a time to register 1 and returns control to your program at the line of coding immediately following the MR$RET macro instruction. When you open the output file and specify IOREG=(2) in the DTFSD data management macro, register 2 is set to the location in the buffer where the first record is to be moved. When control returns to your program (following the MR$RET), the record to which register 1 points must be moved to the location where register 2 points (line 44). When you issue a PUT macro, data management updates the address in register 2 to the location for the next record, tests to see if the buffer is full, and, if it is full, writes the block to the output file.

Next, you write the sorted record to disk via the PUT imperative macro (line 48) and issue an unconditional branch (line 49) to the MR$RET macro, which you labeled RECRET (line 43). This loop repeats until it reaches the end of data indicating that all sorted records have been returned to your program.

When subroutine sort/merge has returned all sorted records, it looks for a point in your program to pass control and exit the return records loop. This point is specified as a symbolic label address in the FIN parameter of your sort parameter table. You indicated the label address FIN=SORTFIN in your disk sort parameter table, so subroutine sort/merge returns control to your program at SORTFIN (the beginning of your close output file routine). In this way, you exit the return record loop.

6.9. ENDING THE SORT RUN

Programming the ending of a sort run follows the same basic procedure as ending any other program. If there are no other calculations or data manipulations to be performed on the sorted data, you issue the CLOSE imperative macro to close the output data file and an EOJ supervisor macro to notify the supervisor that the job step is completed.

```
       LABEL    ΔOPERATIONΔ          OPERAND                      COMMENTS
       1        10          16
    -------------------------------------------------------------------------------
50. | SORTFIN  EQU         *                        FIN ADDRESS
51. |          CLOSE       OUTPUT                   CLOSE THE OUTPUT FILE.
52. |          EOJ                                  END OF JOB STEP.
```

Another good addition to any program is an error routine to tell the system what procedures it should take when an error occurs. If you include an error routine, you must indicate the name of your error routine on the ERROR parameter of the DTFSD statements for both input and output files. (See Figure 6—3, ERROR=IOERROR.) Suppose you name your error processing routine IOERROR. You might use the following approach to handle an error condition by using the CANCEL supervisor macro (line 54) to halt the current job run.

```
53. | IOERROR  EQU         *
54. |          CANCEL                               CANCEL THE JOB.
55. |          LTORG                                DEFINE ALL LITERALS HERE.
56. | WORK     EQU         *                        START OF SORT WORK AREA.
57. | *                                             THIS SET UP ALLOWS THE SORT
58. | *                                             TO USE ALL MEMORY FROM
59. | *                                             THIS LOCATION TO THE END OF
60. | *                                             THE JOB REGION.
61. |          END         SRTEXMPL
```

Finally, to lead into the necessary job control statements for your disk sort program, you would write the LTORG assembler directive (line 55), which generates into your source module all previously defined literals.

In the STOR parameter of the MR$PRM macro, you specified the symbolic label WORK. That name indicated the starting address of the main storage area available to subroutine sort/merge. Here, at the end of your program, you place your designation of that area (line 61). Your equate (EQU*) statement with the current location counter symbol (*) tells sort/merge that it should use the area starting with the address in the current location counter (now showing the address of the end of your program) to the end of the job region as the space for its main storage work area. Figure 6—17 shows the coding of your disk sort program to this point and the diagram following it, Figure 6—18, illustrates your program's interface with the subroutine sort/merge.

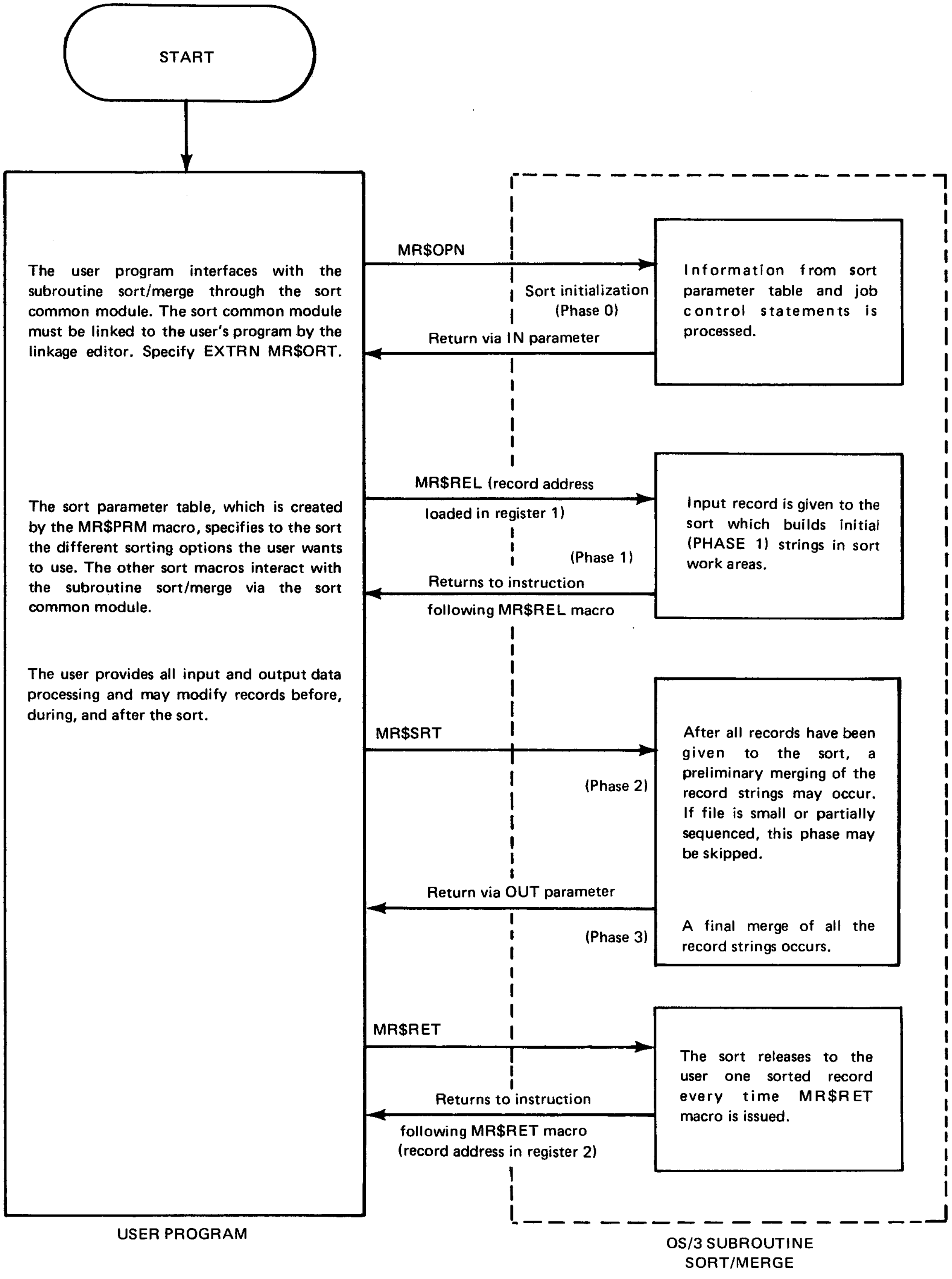
Notice the use of equate statements in this program coding. In all cases except the last, these statements are located at the beginning of input, sort, output, end, or error routines, as indicated by their labels. Use of the equate statement is a valuable programming technique that allows you to change or insert instructions at these points at a later time.

```
SRTEXMPL START Ø                                SETS LOCATION COUNTER TO ZERO.
*        EXTRN MR$ORT                           MR$ORT DEFINES AN EXTRN.
*                                               LINKS COMMON SORT MODULE
*                                               TO YOUR PROGRAM.
         BALR  4,Ø
         USING *,4
         B     START
INPUT    DTFSD BLKSIZE=4ØØ,RECSIZE=8Ø,IOAREA1=BUFF1,IOAREA2=BUFF2,      C
               IOREG=(2),RECFORM=FIXBLK,ERROR=IOERROR,OPTION=YES,       C
               EOFADDR=EOF,TYPEFLE=INPUT
OUTPUT   DTFSD BLKSIZE=4ØØ,RECSIZE=8Ø,IOAREA1=BUFF1,IOAREA2=BUFF2,      C
               IOREG=(2),RECFORM=FIXBLK,ERROR=IOERROR,OPTION=YES,       C
               TYPEFLE=OUTPUT
*
*
SORT     MR$PRM FIELD=(Ø,7,CH),                                         C
               IN=SORTIN,                                               C
               OUT=SORTOUT,                                             C
               FIN=SORTFIN,                                             C
               RCSZ=8Ø,                                                 C
               STOR=WORK,                                               C
               DISC=4                                                   C
*
*              DATA MANAGEMENT WORK AREA
         DS    OH
*        DS    CL8
BUFF1    DS    CL4ØØ
         DS    CL8
BUFF2    DS    CL4ØØ
SAVEAREA DS    18F                        DATA MANAGEMENT SAVE AREA
*
*
```

Figure 6—17. Disk Sort Program Coding (Part 1 of 2)

```
START    EQU    *                                                   25
         MR$OPN SORT             OPEN THE SORT/MERGE SUBROUTINE     26
SORTIN   LA     13,SAVEAREA      POINT TO SAVE AREA                 27
         OPEN   INPUT            OPEN THE INPUT FILE                28
GETREC   GET    INPUT            GET RECORD FROM INPUT FILE         29
*
         LR     1,2              LOAD R1 WITH RECORD ADDRESS        30
*                                ASSUMING R2 IS SPECIFIED AS        31
*                                THE I/O REGISTER IN THE INPUT      32
*                                DTF.
         MR$REL                  RELEASE RECORD TO THE SORT         33
         B      GETREC           GET NEXT RECORD                    34
*
EOF      EQU    *                THIS LOCATION IS SPECIFIED         35
*                                AS THE END OF FILE ADDRESS         36
*                                IN THE INPUT DTF.                  37
         CLOSE  INPUT            CLOSE THE INPUT FILE.              38
*
         MR$SRT                  TELLS THE SORT THAT THE END-       39
*                                OF-DATA HAS BEEN REACHED.          40
SORTOUT  EQU    *                OUT ADDRESS                        41
         OPEN   OUTPUT           OPEN THE OUTPUT FILE.              42
RECRET   MR$RET                  REQUEST A RECORD RETURNED.         43
         MVC    Ø(8Ø,2),Ø(1)     MOVE THE SORTED RECORD TO          44
*                                THE OUTPUT BUFFER AREA.            45
*                                R2 IS ASSUMED SPECIFIED AS         46
*                                THE I/O REGISTER IN THE            47
*                                OUTPUT DTF.
         PUT    OUTPUT           OUTPUT THE RECORD RETURNED.        48
         B      RECRET           REQUEST NEXT RECORD.               49
*
SORTFIN  EQU    *                FIN ADDRESS                        50
         CLOSE  OUTPUT           CLOSE THE OUTPUT FILE.             51
         EOJ                     END OF JOB STEP.                   52
*
*               ERROR ADDRESS FOR DATA MANAGEMENT
*
IOERROR  EQU    *                                                   53
         CANCEL                  CANCEL THE JOB                     54
         LTORG                   DEFINE ALL LITERALS HERE           55
WORK     EQU    *                START OF SORT WORK AREA.           56
*                                THIS SETUP ALLOWS THE SORT         57
*                                TO USE ALL MEMORY FROM             58
*                                THIS LOCATION TO THE END OF        59
*                                THE JOB REGION.                    60
         END SRTEXMPL                                               61
```

Figure 6—17. Disk Sort Program Coding (Part 2 of 2)



The user program interfaces with the subroutine sort/merge through the sort common module. The sort common module must be linked to the user's program by the linkage editor. Specify EXTRN MR$ORT.

The sort parameter table, which is created by the MR$PRM macro, specifies to the sort the different sorting options the user wants to use. The other sort macros interact with the subroutine sort/merge via the sort common module.

The user provides all input and output data processing and may modify records before, during, and after the sort.

Information from sort parameter table and job control statements is processed.

Input record is given to the sort which builds initial (PHASE 1) strings in sort work areas.

After all records have been given to the sort, a preliminary merging of the record strings may occur. If file is small or partially sequenced, this phase may be skipped.

A final merge of all the record strings occurs.

The sort releases to the user one sorted record every time MR$RET macro is issued.

Figure 6—18. User Program Interface with Subroutine Sort/Merge

6.10. SUBROUTINE SORT/MERGE MACRO PARAMETER USAGE

We've examined required and optional MR$PRM macro parameters and a typical disk sort program MR$PRM specification. Figure 6—19 illustrates the entire MR$PRM macro format.

```
LABEL      | ΔOPERATIONΔ |                          OPERAND
-----------+-------------+------------------------------------------------------------------------
[symbol]   | MR$PRM      | ⎧ FIELD=(strt-pos-1,lgth-1[,form-1] [,seq-1] [,order-1]                    ⎫
           |             | ⎨          [,...,strt-pos-n,lgth-n[,form-n] [,seq-n] [,order-n]])         ⎬ ,
           |             | ⎩ RSOC=symbol,                                                             ⎭
           |             |
           |             | FIN=symbol,
           |             |
           |             | IN=symbol,
           |             |
           |             | OUT=symbol,
           |             |
           |             | RCSZ=max-bytes,
           |             |
           |             | STOR= { symbol                        }
           |             |       { (symbol, number-of-bytes)     }
           |             |
           |             | [ ,ADDROUT= { A } ]
           |             | [           { D } ]
           |             |
           |             | [ ,ADTABL=symbol]
           |             |
           |             | [ ,BIN={ bytes                                            } ]
           |             | [      { (min-bytes,size-1,freq-1[,...,size-n,freq-n])   } ]
           |             |
           |             | [ ,CALC = { NO  } ]
           |             | [         { YES } ]
           |             |
           |             | [ ,CSPRAM= { NO  } ]
           |             | [          { YES } ]
           |             |
           |             | [   ⎧ DISC= { (address,max-disk-file-number) }    ⎫ ]
           |             | [   ⎪       { max-disk-file-number           }    ⎪ ]
           |             | [ , ⎨                                               ⎬ ]
           |             | [   ⎪ TAPE= { label-type                     }    ⎪ ]
           |             | [   ⎩       { (label-type,max-file-number)   }    ⎭ ]
           |             |
           |             | [ ,DROC= { DELETE } ]
           |             | [        { symbol } ]
           |             |
           |             | [ ,MERGE= { NO  } ]
           |             | [         { YES } ]
           |             |
           |             | [ ,NOCKSM= { D } ]
           |             | [          { T } ]
           |             |
           |             | [, PAD=bytes]
           |             |
           |             | [          { ALL      } ]
           |             | [ ,PRINT=  { CRITICAL } ]
           |             | [          { NONE     } ]
```

Figure 6—19. Complete MR$PRM Macro Format (Part 1 of 2)

LABEL	ΔOPERATIONΔ	OPERAND
	MR$PRM	**[,RESERV =sort-filename]**
	(cont)	**[,RESUME=(PASS,recovery-number)]**
		[,SHARE=sort-filename]
		[,SIZE=number]
		[,USEQ=(to-address,from-address)]

Figure 6—19. Complete MR$PRM Macro Format (Part 2 of 2)

Table 6—2 summarizes sort/merge macro instructions required for subroutine sort/merge execution in single-cycle sort/merge, merge-only, or internal (main storage) sort/merge operations including MR$PRM subparameter use.

6.11. ASSEMBLING, LINKING, AND EXECUTING YOUR PROGRAM

Up to this point, you have written your program. Now you must assemble, link, and execute it. This is done by embedding your program in a job control stream. The job control stream consists of job control statements that name devices used by your program and the subroutine sort/merge; describe labels and space allocations; and assemble, link, and execute your program.

6.11.1. Assembling the Program

When you submit your program (including the job control statements before and after the source coding) to the assembler, it prepares a machine language program from your program's source code. This machine language is called object code; the assembler's translation of your source code to object code is an object module that the assembler places in the temporary job run library file ($Y$RUN). The whole process is called the *assembly run.*

On the assembly run, no data is manipulated. The assembler simply analyzes each statement and converts it into a form acceptable to the machine. Instructions called assembler control directives direct the operation of the assembler. In your disk sort program, the START assembler directive sets the initial location counter value. The END directive indicates the end of your source program and the location where control is transferred after your program is loaded into main storage.

You specify the EXTRN assembler directive and the assembler includes it in your program object module as an unresolved external reference. The EXTRN directive tells the assembler that you want the linkage editor to call in the sort common module in object code form from the object library file ($Y$OBJ).

Table 6—2. Summary of Sort/Merge Parameter Usage

Macros and Parameters	Single-Cycle Sort/Merge				Merge-Only		Internal Sort/Merge	
	Disk		Tape					
	Required	Optional	Required	Optional	Required	Optional	Required	Optional
MR$PRM	X		X		X		X	
MR$OPN	X		X		X		X	
MR$REL	X		X				X	
MR$SRT	X		X				X	
MR$RET	X		X				X	
MG$REL					X			
MG$RET					X			
Normal Linkage Sort Parameter Table Entries								
DROC		X		X		X		X
FIN	X		X		X		X	
IN	X		X		X		X	
OUT	X		X				X	
RSOC		X		X		X		X
Device Assignment Sort Parameter Table Entries								
DISC		X						
RESERV				X				
SHARE				X				
STOR	X		X		X		X	
TAPE				X				
Record Definition Sort Parameter Table Entries								
ADDROUT		X		X				X
BIN		X		X				X
FIELD	X		X		X		X	
RCSZ	X		X		X		X	
USEQ		X		X		X		X
Restart Sort Parameter Table Entries								
RESUME				X				
Miscellaneous Sort Parameter Table Entries								
ADTABL		X		X		X		X
CALC		X						
CSPRAM		X		X		X		X
MERGE					X			
SIZE		X		X				X
NOCKSM		X		X				
PAD		X		X		X		X
PRINT		X		X		X		X

Another assembler directive, LTORG, tells the assembler to generate all literals that were not previously defined in your source program. In other words, the assembler builds a *literal table*, a collection of constant values assigned to symbolic names.

6.11.2. Link Editing the Program

You now have an object module representing your source program in $Y$RUN. The linkage editor begins its activities by taking the object module as its input. If you elect to write a control stream for the link edit job step, linkage editor scans its control stream data set for linkage editor control statements and finds the LOADM and INCLUDE statements which tell it to name the load module it is creating SRTEXM and to include the object module named SRTEXMPL. (See Figure 6—24, lines 18 through 22.) Otherwise, if you use the short way of linking the object module named SRTEXMPL, you use the LINK job control procedure instead of the linkage editor statements. (See Figure 6—21, line 8.) The linkage editor also scans your program object module for external references and finds MR$ORT. It looks for MR$ORT in $Y$OBJ, finds it is an entry point to the object module SG$ORT, and includes SG$ORT in the load module SRTEXM. Normally, linkage editor places the load modules it produces in the temporary job run library file ($Y$RUN) unless you specify that the load modules be placed in your user load library (a file separate from the system resident library files).

6.11.3. Executing the Program

Now you have a load module that is acceptable to the system for the execution run. At this point, you need the sort data files and device assignment set information. You supplied the device assignment data after the linkage editor jproc call. (See Figure 6—21, lines 9 through 21.) At the end of your job control stream, the EXEC statement tells the supervisor to execute your load module named SRTEXM. Your program load module normally comes from $Y$RUN and the execution begins. In the execution run, the load modules for the subroutine sort/merge are called from $Y$LOD into main storage, as needed by your program. When the sort/merge phases are completed, your sorted records are written to the output files on the volumes and devices you specified in the job control stream. (See Figure 6—21, lines 13 through 16.)

Figure 6—20 illustrates the assembly, linkage edit, and execution runs for a subroutine sort/merge disk sort program.

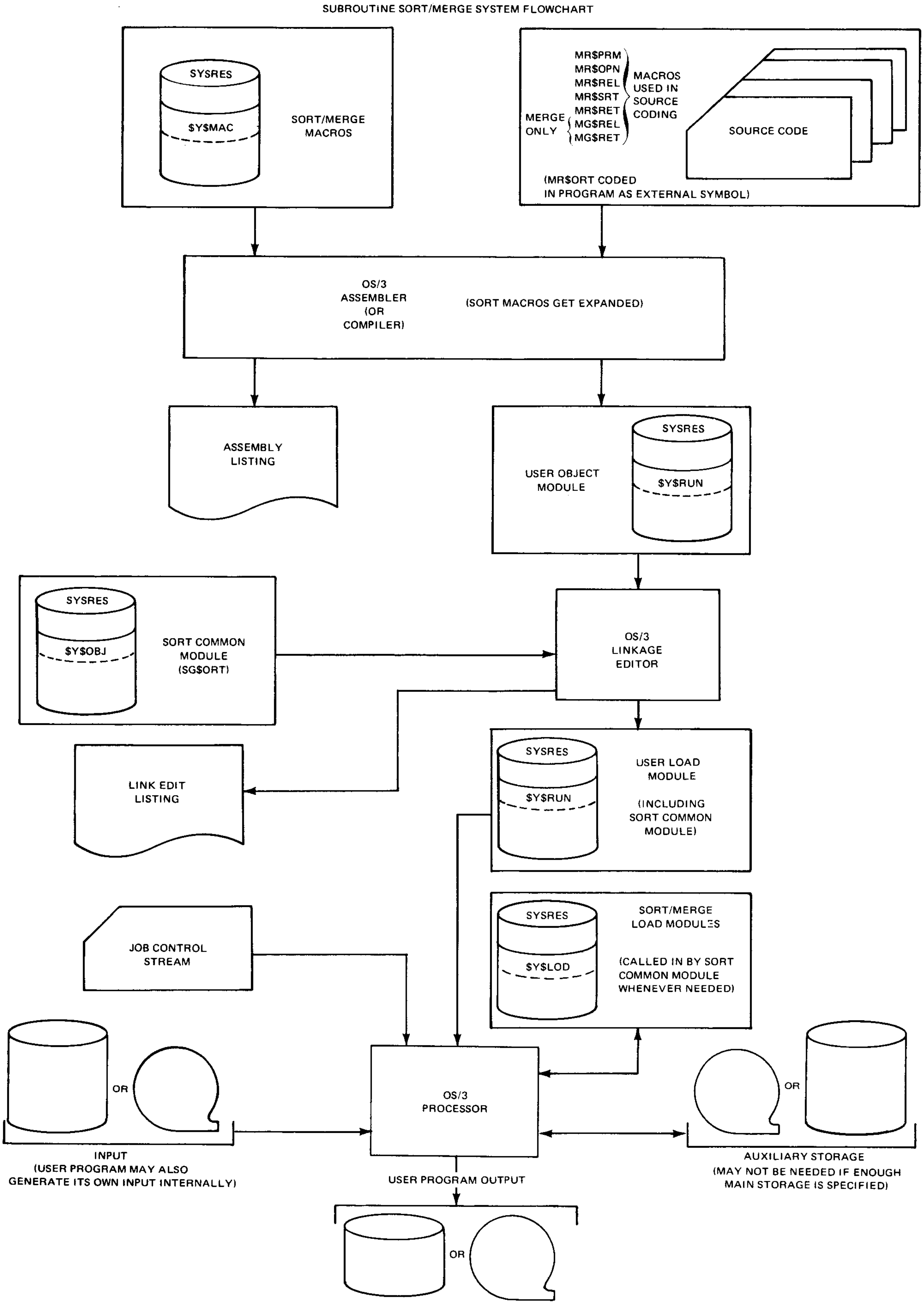

SUBROUTINE SORT/MERGE SYSTEM FLOWCHART
SYSRES
$Y$MAC
SORT/MERGE MACROS
MR$PRM
MR$OPN
MR$REL
MR$SRT
MR$RET
MG$REL
MG$RET
MERGE ONLY
MACROS USED IN SOURCE CODING
SOURCE CODE
(MR$ORT CODED IN PROGRAM AS EXTERNAL SYMBOL)
OS/3 ASSEMBLER (OR COMPILER)
(SORT MACROS GET EXPANDED)
ASSEMBLY LISTING
USER OBJECT MODULE
SYSRES
$Y$RUN
SYSRES
$Y$OBJ
SORT COMMON MODULE (SG$ORT)
OS/3 LINKAGE EDITOR
LINK EDIT LISTING
SYSRES
$Y$RUN
USER LOAD MODULE
(INCLUDING SORT COMMON MODULE)
SYSRES
$Y$LOD
SORT/MERGE LOAD MODULES
(CALLED IN BY SORT COMMON MODULE WHENEVER NEEDED)
JOB CONTROL STREAM
OR
INPUT
(USER PROGRAM MAY ALSO GENERATE ITS OWN INPUT INTERNALLY)
OS/3 PROCESSOR
OR
AUXILIARY STORAGE
(MAY NOT BE NEEDED IF ENOUGH MAIN STORAGE IS SPECIFIED)
USER PROGRAM OUTPUT
OR

Figure 6—20. Assembly, Linkage Edit, and Execution Run System Flowchart

6.11.4. Typical Subroutine Disk Sort Job Control Stream

In order to schedule your program and allocate 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 6—21 shows the entire job control stream required for our disk sort program, the input data before the sort, and the output data after the sort.

It is important to note that this example employs very low volume input files. Under normal disk sort conditions, input files are much larger, and the same disk used as it is in this example for input, output, and the work file will result in the least efficient sort. The most efficient disk sort is achieved when you use one work file per disk and a separate disk for the input and output files.

```
1        10        20
// JOB SRTEXMPL ,,7ØØØ,9ØØØ,2
// DVC 20 // LFD  PRNTR
// WORK1
// WORK2
// EXEC ASM
/$
    .  )
    .  }             YOUR       PROGRAM   CODING
    .  )
/*
//SRTEXM LINK SRTEXMPL
// DVC 5Ø
// VOL DSPØ28
// LBL MYFILE1
// LFD INPUT
// DVC 5Ø
// VOL DSPØ28
// LBL MYFILE2
// LFD OUTPUT,,INIT
// DVC 5Ø
// VOL DSPØ28
// EXT ST,C,,CYL,5
// LBL $SCR1
// LFD DMØ1
```

Figure 6—21. Disk Sort Program Job Control Stream (Part 1 of 5)

```
     1        10        20
22. // EXEC SRTEXM,$Y$RUN
23. /&
24. // FIN
```

DATA FILE BEFORE SORT

```
BLK# REC# BKSZ RCSZ  ........10 ........20 ........30 ........40 ........50 ........60 ........70 ........80

   1    1   80   80  00321654
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     0032165400 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   2    1   80   80  10007005
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     1000700500 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   3    1   80   80  68799863
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     6879986300 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   4    1   80   80  94600054
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     9460005400 00000000CC 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   5    1   80   80  20463844
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     2046384400 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   6    1   30   80  54486555
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     5448655500 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   7    1   80   80  03000600
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     0300060000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   8    1   80   80  88855296
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     8885529600 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   9    1   80   80  43300000
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     4330000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  10    1   80   80  70509300
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     7050930000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  11    1   80   80  DAYTONOH
                     CCEEDDDC44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     4183656800 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  12    1   80   80  STLOUISM
                     EEDDECED44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     2336492400 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  13    1   80   80  YORKPEN
                     EDDDDCD444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     8692755000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  14    1   80   80  NEWARKNJ
                     DCECDDDD44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
```

Figure 6—21. Disk Sort Program Job Control Stream (Part 2 of 5)

DATA FILE BEFORE SORT (cont)

```
BLK# REC# BKSZ RCSZ  ........10 ........20 ........30 ........40 ........50 ........60 ........70 ........80

                     5561925100 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  15    1   80   80  MIAMIFLA
                     DCCDCCDC44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     4914963100 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  16    1   80   80  OH00001
                     DCFFFFF444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     6800001000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  17    1   80   80  MO000004
                     DDFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     4600000400 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  18    1   80   80  PA000007
                     DCFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     7100000700 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  19    1   80   80  NJ000012
                     DDFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     5100001200 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  20    1   80   80  FL000006
                     CDFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     6300000600 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  21    1   80   80  33655307
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     3365530700 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  22    1   80   80  10985469
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     1098546900 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  23    1   80   80  98654777
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     9865477700 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  24    1   80   80  68548833
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     6854883300 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  25    1   80   80  40675987
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     4067598700 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  26    1   80   80  77330659
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     7733065900 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  27    1   80   80  90675004
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     9067500400 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  28    1   80   80  09436750
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     0943675000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  29    1   80   80  11766325
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     1176632500 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

  30    1   80   80  50964097
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     5096409700 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000
```

Figure 6—21. Disk Sort Program Job Control Stream (Part 3 of 5)

DATA FILE AFTER SORT

```
BLK# REC# BKSZ RCSZ  ........10 ........20 ........30 ........40 ........50 ........60 ........70 ........80

   1    1  400   80  DAYTONOH
                     CCEEDDDC44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     4183656800 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   1    2  400   80  FL000006
                     CDFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     6300000600 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   1    3  400   80  MIAMIFLA
                     DCCDCCDC44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     4914963100 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   1    4  400   80  M0000004
                     DDFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     4600000400 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   1    5  400   80  NEWARKNJ
                     DCECDDDD44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     5561925100 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   2    1  400   80  NJ000012
                     DDFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     5100001200 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   2    2  400   80  OH00001
                     DCFFFFF444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     6800001000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   2    3  400   80  PA000007
                     DCFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     7100000700 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   2    4  400   80  STLCUISM
                     EEDDECED44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     2336492400 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   2    5  400   80  YORKPEN
                     EDDDDCD444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     8692755000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   3    1  400   80  00321654
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     0032165400 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   3    2  400   80  03000600
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     0300060000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   3    3  400   80  09436750
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     0943675000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   3    4  400   80  10007005
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     1000700500 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   3    5  400   80  10985469
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     1098546900 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   4    1  400   80  11766325
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     1176632500 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   4    2  400   80  20463844
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     2046384400 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   4    3  400   80  33655307
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     3365530700 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   4    4  400   80  40675987
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     4067598700 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   4    5  400   80  43300000
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     4330000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   5    1  400   80  50964097
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     5096409700 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   5    2  400   80  54486555
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     5448655500 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000
```

Figure 6—21. Disk Sort Program Job Control Stream (Part 4 of 5)

DATA FILE AFTER SORT (cont)

```
BLK# REC# BKSZ RCSZ  ........10 ........20 ........30 ........40 ........50 ........60 ........70 ........80

   5    3  400   80  68548833
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     6854883300 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   5    4  400   80  68799863
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     6879986300 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   5    5  400   80  70509300
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     7050930000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   6    1  400   80  77330659
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     7733065900 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   6    2  400   80  88855296
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     8885529600 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   6    3  400   80  90675004
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     9067500400 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   6    4  400   80  94600054
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     9460005400 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000

   6    5  400   80  98654777
                     FFFFFFFF44 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444 4444444444
                     9865477700 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000
```

Figure 6—21. Disk Sort Program Job Control Stream (Part 5 of 5)

SRTEXMPL is the 8-character alphanumeric name of your job (Figure 6—21, line 1). 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. The number 2 indicates that no more than two tasks can be active at the same time in any job step. A *task* is a unit of work that the supervisor schedules.

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.

// DVC 20 assigns device number 20 to the printer device designated by the system filename, PRNTR (Figure 6—21, line 2). The two following job control statements, // WORK1 and // WORK2 (Figure 6—21, lines 3 and 4), are job control procedure (jproc) calls that allot temporary files for the assembly job step by automatically supplying the DVC, VOL, EXT, LBL, and LFD parameter information you would otherwise have to specify for assembler use. Two of these temporary files are needed by the assembler so that it can assemble an object module from the source code you supply immediately after the start-of-data (/$) control statement (Figure 6—21, line 6).

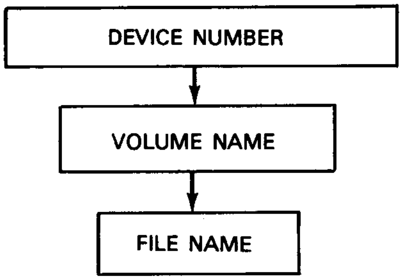
Finally, the // EXEC ASM statement (Figure 6—21, line 5) tells the system to load and execute the assembler. The /$ indicates the start-of-data to the assembler. This data is your program.

At the end of your source coding, you code a /* delimiter statement (Figure 6—21, line 7) to indicate the end of data (your program) to the assembler.

So far, we have generated an object module called SRTEXMPL (label of the START assembler directive) and it is in $Y$RUN. Now we must use the linkage editor to prepare a load module. The simplest way to do this is to use the LINK job control procedure call (Figure 6—21, line 8). The LINK jproc generates a load module called SRTEXM from the object module (called SRTEXMPL). Load module SRTEXM is then automatically placed in $Y$RUN, unless you specify an alternate library via the OUT keyword parameter. For more information about job control procedures, refer to the job control user guide, UP-8065 (current version).

When you execute the load module SRTEXM (Figure 6—21, line 22), you tell the supervisor to retrieve it from $Y$RUN. Otherwise, the supervisor searches for the SRTEXMPL load module in the $Y$LOD first before going to $Y$RUN. Thus, by specifying $Y$RUN, you save processing time.

Your next series of job control statements (Figure 6—21, lines 9 through 21) 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.



Each device assignment set begins with a DVC statement that assigns a device number (Figure 6—21, lines 9, 13, and 17). For specific I/O device numbers, check the list of device types and features in the job control user guide, UP-8065 (current version).

Your first DVC statement assigns device number 50 to your input file named MYFILE1 (Figure 6—21, lines 9 and 11). The second DVC statement assigns the same device to your output file named MYFILE2 (lines 13 and 15). Looking at the next DVC statement (line 17), notice that device number 50 is assigned for the sort work file $SCR1. 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 10, 14, and 18). The name you assign to your input and output file volume is the alphanumeric name DSP028 (lines 10 and 14). For the sort work file volume name you specify the same volume, DSP028 (line 18).

To provide disk space for the sort work file 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 set for the sort work file. The EXT statement applies to the first volume specified on the immediately preceding VOL statement (line 19). Notice that there is no EXT statement for either input or output files because these files already exist. 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 cylinders, and the 5 indicates the number of cylinders allocated for the work file.

Data management needs to know the file names you designate for your program. The LBL job control statement supplies this information by specifying label information for tape or disk volumes. Only one LBL statement is allowed per device assignment set. You specify the disk sort program's input file identifier as MYFILE1 (line 11), the output file identifier as MYFILE2 (line 15), and the sort work file identifier as $SCR1 (line 20).

To link the file information in the job control stream with the data management file definition, you specify the DTF file label on the LFD job control statement of the device assignment set for each file (lines 12 and 16). Thus your first two LFD statements in the job control stream would specify the names INPUT and OUTPUT. Although job control allows 8-character names, data management requires that logical file names not exceed seven characters, the first of which must be alphabetic. Because the logical file names on the LFD statements (lines 12 and 16) come from the file label on the data management DTF macros, lines 12 and 16 must be the same as the file names in the labels of corresponding DTF declarative macros. They also must not exceed seven characters. The INIT parameter on the LFD statement for the output file (line 16) indicates that you want to start writing at the beginning of the file, overlaying its previous contents.

When specifying the LFD statement for your sort work file, you must specify the link file name DM01 or $SCR1, because only these standard names are recognized internally by data management for the sort work file area. Thus, the third LFD statement specifies the name DM01 (line 21). An easier way to allocate work areas on disk is with the WORK jproc call. A WORK jproc automatically generates a device assignment set allocating system scratch space as a work area. The format for a jproc call that would take the place of lines 17 through 21 is //DM01 WORK1 or just // WORK1. The WORK jproc, used without parameters, allocates 4000 blocks of 256-bytes each (equivalent to one cylinder) of scratch space on your system resident device (SYSRES) or the disk containing your system run library ($Y$RUN). You can increase the amount of work space and specify the use of other disk devices through optional parameters. For more information about the WORK jproc, see the job control user guide, UP-8065 (current version).

After you execute your program load module (line 22), the /& delimiter card must indicate the end of your job control stream and the FIN job control statement, the end of the card reader operation. Figure 6—22 shows the job control stream required to assemble, link, and execute your disk sort program using subroutine sort/merge. Figure 6—23 illustrates the results of your device assignment set specifications.

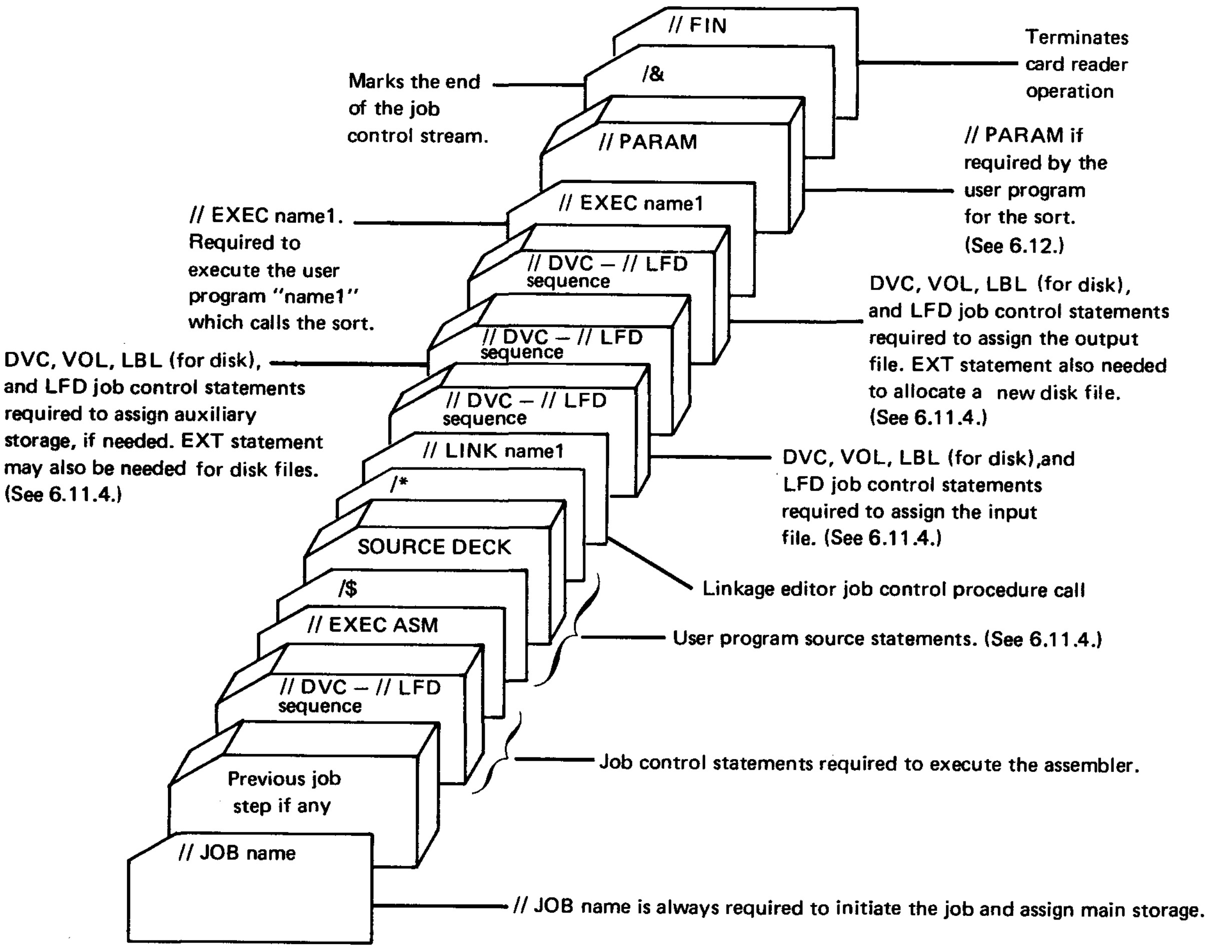

// FIN
/&
// PARAM
// EXEC name1
// DVC – // LFD sequence
// DVC – // LFD sequence
// DVC – // LFD sequence
// LINK name1
/*
SOURCE DECK
/$
// EXEC ASM
// DVC – // LFD sequence
Previous job step if any
// JOB name
Terminates card reader operation
Marks the end of the job control stream.
// PARAM if required by the user program for the sort. (See 6.12.)
// EXEC name1. Required to execute the user program "name1" which calls the sort.
DVC, VOL, LBL (for disk), and LFD job control statements required to assign the output file. EXT statement also needed to allocate a new disk file. (See 6.11.4.)
DVC, VOL, 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 6.11.4.)
DVC, VOL, LBL (for disk),and LFD job control statements required to assign the input file. (See 6.11.4.)
Linkage editor job control procedure call
User program source statements. (See 6.11.4.)
Job control statements required to execute the assembler.
// JOB name is always required to initiate the job and assign main storage.

Figure 6—22. Typical Job Control Stream for a Subroutine Sort/Merge Application

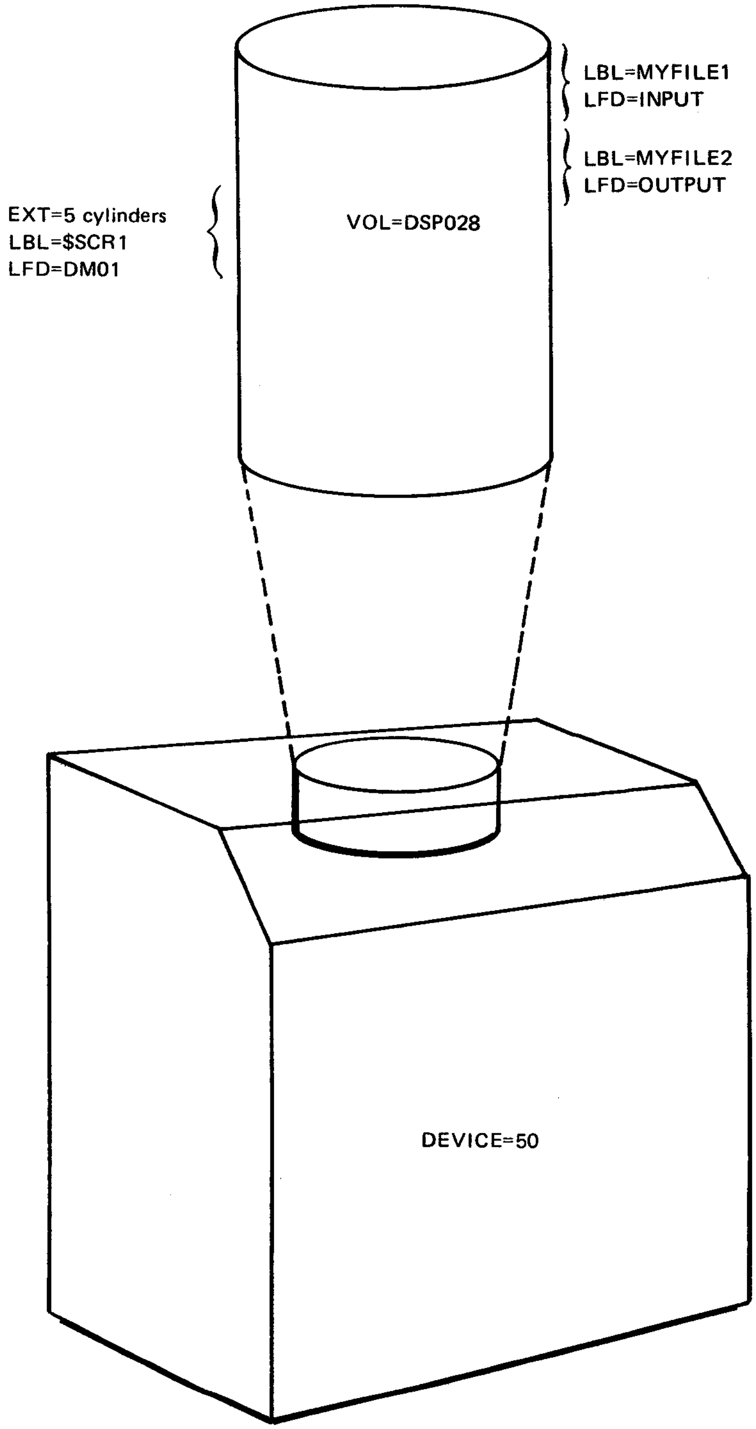


Figure 6—23. Device Assignment Results

6.11.4.1. Alternate Job Control Stream

The job control stream shown in Figure 6—21 illustrates shortcuts in assigning work files to the assembler and the linkage editor. If you choose to use the standard job control and link editor statements equivalent to the WORK and LINK job control procedures, you can do so (Figure 6—24); however, it makes lengthier coding and does not increase efficiency. To set up the assembler and linkage editor work files, write DVC, VOL, EXT, LBL, and LFD job control statements; to write a data stream for the linkage editor, use the LOADM and INCLUDE linkage editor control statements, as shown in Figure 6—24.

```
1        10        20
// JOB SRTEXMPL,,7ØØØ,9ØØØ,2
// DVC 2Ø // LFD PRNTR
// DVC RES
// EST ST,,1,BLK,(256,4ØØØ)
// LBL $SCR1
// LFD $SCR1                                  = // WORK1
// DVC RUN                                      // WORK2
// EXT ST,,1,BLK,(256,4ØØØ)
// LBL $SCR2
// LFD $SCR2
// EXEC ASM
/$
    .
    .         YOUR     PROGRAM    CODING
    .
/*
// WORK1
// EXEC LNKEDT
/$                                            = // LINK SRTEXMPL
 LOADM   SRTEXM
 INCLUDE SRTEXMPL
/*
// DVC 5Ø
// VOL DSP111                                   DEVICE ASSIGNMENT SET 1
// LBL MYFILE1
// LFD INPUT
// DVC 5Ø
// VOL DSP111                                   DEVICE ASSIGNMENT SET 2
// LBL MYFILE2
// LFD OUTPUT,,INIT
// DVC 51
// VOL DSP12Ø
// EXT ST,CYL,,C,2Ø                             DEVICE ASSIGNMENT SET 3
// LBL SRTWK1
// LFD DMØ1
// EXEC SRTEXM
/&
// FIN
```

Figure 6—24. Disk Sort Program Alternate Job Control Stream

Notice that your load module name in the EXEC statement (line 36) must specify the name from the LOADM control statement (line 20).

Using the WORK jproc statement (line 17) without any of its optional parameters generates:

- the device and volume numbers of your SYSRES volume;
- an extent of 4000 256-byte blocks;
- the label name $SCR1; and
- the LFD name $SCR1.

6.11.5. Job Control Stream for Tape Work File Assignment

If you want to use tape work files, your program requires the following series of job control statements in your job control stream:

```
      1         10        20
1.  // DVC 9Ø
2.  // VOL TAP 15Ø
3.  // LBL SRTWK1
4.  // LFD SMØ3
```

Line 1 specifies the logical device unit number. Lines 2 and 3 specify the volume serial number and file label. The LBL statement is optional for tape files. Line 4 gives the standard sort tape file name, SM03. Three to six tape work files are required when you are using tape auxiliary storage. You must assign the LFD names SM01, SM02, and SM03 if you are using three tape files, SM04 for one additional file, and so on.

6.12. SUBMITTING SORT PARAMETER TABLE ENTRIES VIA THE JOB CONTROL STREAM

You can change, add, delete, or override existing parameters in the sort parameter table by coding PARAM job control statements in your control stream. Only the following keyword parameters can be accepted from the control stream at program execution time:

BIN	Bin size
DISC	Disk work file allocation
NOCKSM	Checksum suppression
RESERV	Tape work file device reserved for output file

RESUME	Resumption of interrupted tape sort
SHARE	Tape unit shared by input file and work file
TAPE	Tape work file allocation

To code parameters you want to include in the control stream, use the same keyword format as described for the MR$PRM macro instruction (Figure 6—19) and begin writing your PARAM statement keyword parameters in column 10. Separate each parameter by a comma and, if necessary, continue through column 71. If more parameters must be included on that PARAM statement, follow the last keyword parameter by a comma and code a nonblank character in column 72 to indicate more parameters to come. You may also submit multiple PARAM statements as in the following example.

```
1       10      20
_____________________________________________
// PARAM TAPE=(STD,6),SHARE=SMØ1
// PARAM NOCKSM=T
```

PARAM control statements should appear in your job control stream immediately following the EXEC statement that initiates execution of your program. The following example illustrates the proper placement of the PARAM job control statement to add keyword parameters to the sort parameter table in the disk sort program.

```
1. | // EXEC SRTEXM
2. | // PARAM DISC=7,NOCKSM=D
3. | /&
4. | // FIN
```

Line 1 specifies the sort program to be executed. On line 2, the first keyword parameter would change the disk sort program's original specification of four disks to seven disks for sort work file use. The NOCKSM keyword parameter specifies no checksum calculations on disk. Both parameters are being added to the sort parameter table from the job control stream.

In addition to coding PARAM job control statements, you must include the CSPRAM=YES keyword parameter in your sort parameter table via the MR$PRM macro. Otherwise, if you do not specify the CSPRAM or specify CSPRAM=NO, control stream processing is bypassed. To avoid recompiling your program, it is wise to specify CSPRAM=YES on your original program. If you do not add keyword parameters from the job control stream, the CSPRAM=YES specification won't affect your program's execution. For an example of a subroutine tape sort with a restart capability using a PARAM statement, see 9.3, line 30 and line 112.

7. Subroutine Sort/Merge User Own-Code Routines

7.1. DEFINITION

Subroutine sort/merge handles two types of user own-code routines during sort processing:

- Record sequence own-code (RSOC)
- Data reduction own-code (DROC)

Whenever you use own-code routines, you must indicate that you are using them and what you are naming them. By writing the RSOC or DROC keyword parameters in the MR$PRM sort macro, you can fulfill both of these requirements. The result is that your keyword specifications appear in the sort parameter table.

Both RSOC and DROC routines require registers 11, 12, 14, and 15 for communication with the subroutine sort/merge. All other registers are available for use by the own-code routine. Information contained in the registers and the action to be performed depend on the specific own-code routine executed.

7.2. RECORD SEQUENCE OWN-CODE ROUTINE (RSOC)

Using the RSOC routine provides a powerful method of handling sort sequences that involve more than a comparison for ascending or descending sequences. It enables you to write your own routine for record comparisons that might include a variety of record key field tests. RSOC allows you to compare the key fields of two records and to set the condition code to indicate the order you want. If you specify RSOC on the MR$PRM macro, do not specify the FIELD parameter. Nevertheless, the RSOC parameter overrides the FIELD parameter if you should forget and specify both.

When two records are ready to be compared to determine which should precede the other, subroutine sort/merge transfers control to your own-code routine at the address (symbolic label name) you specified on your RSOC keyword parameter. Subroutine sort/merge places the RSOC address in register 15 and stores the subroutine sort/merge return address in register 14.

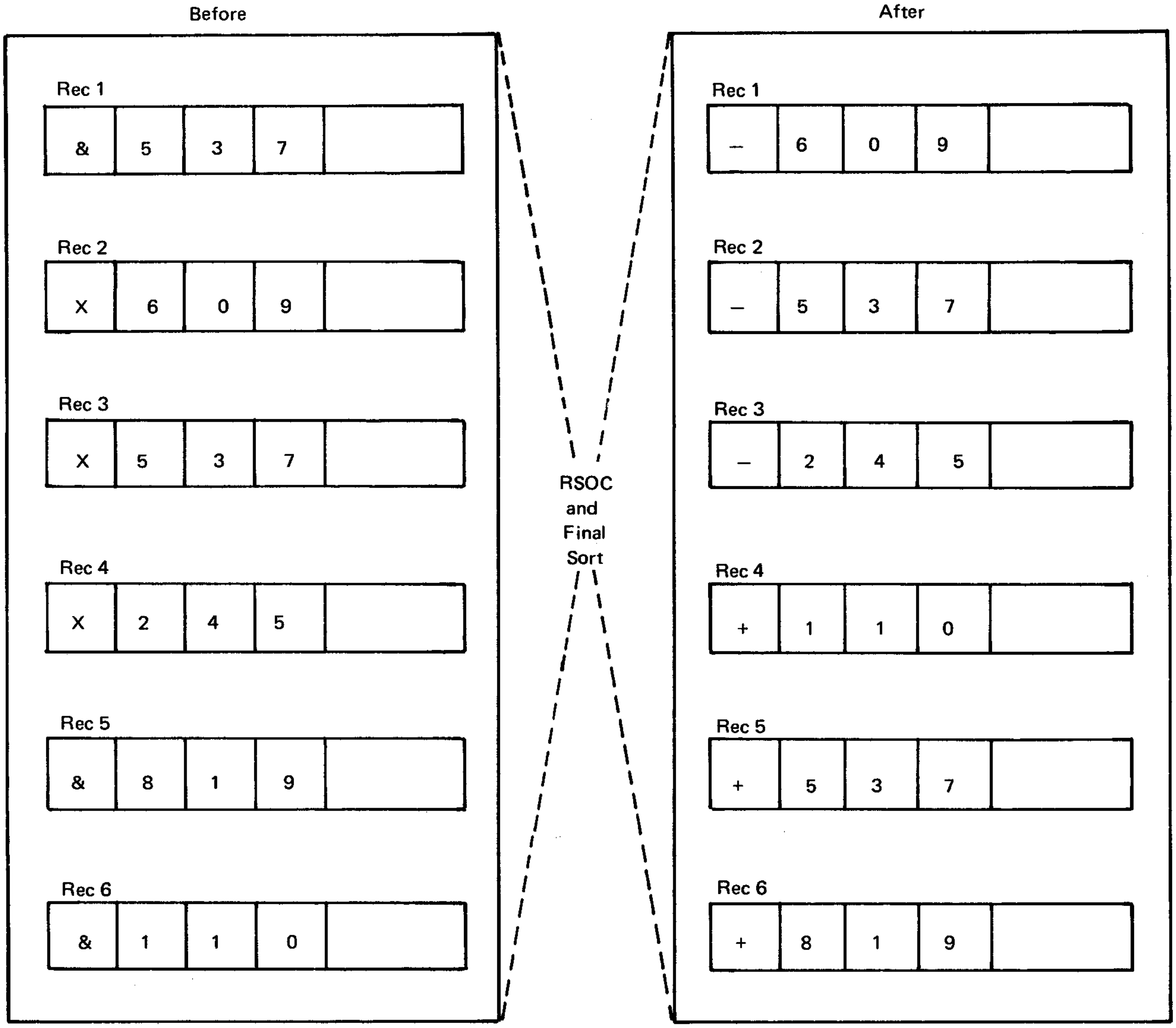
The first instruction in your own-code routine must be the USING assembler directive. It must assign register 15 for use as the base register of your RSOC routine. Your RSOC routine automatically receives the addresses of the two records to be compared in registers 11 and 12. For variable-length records,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

After you set the condition codes resulting from the comparison, you may optionally write a DROP assembler directive to disengage the use of base register 15 before you return control to the subroutine sort/merge via a branch to register 14. A sketch of the key instructions needed for using RSOC follows:

```
LABEL    ΔOPERATIONΔ        OPERAND                         ΔCOMMENTS
1         10         16

          MR$PRM     RSOC=MYROUT
  .)
  .}                                 YOUR PROGRAM
  .)

MYROUT    USING      *,15            ASSIGN BASE R15 TO OWN-CODE ROUTINE

  .)
  .}                                 YOUR OWN-CODE ROUTINE
  .)
          DROP       15              DISENGAGE BASE R15
          BR         14
  .}                                 RETURNS TO THE SUBROUTINE SORT/MERGE
  .)
```

For a complete program example illustrating a user own-code routine for a subroutine sort, see 9.4.

Use of the RSOC routine is not frequent but its availability can prove very valuable. For example, your company might use a nonstandard arithmetic sign with data. In this case, an RSOC routine can provide the necessary transsystem sign interpretation. The following diagram illustrates file contents before and after the execution of a RSOC routine to arrange the file in ascending sequence:



	Before	After
Rec 1	& 5 3 7	– 6 0 9
Rec 2	X 6 0 9	– 5 3 7
Rec 3	X 5 3 7	– 2 4 5
Rec 4	X 2 4 5	+ 1 1 0
Rec 5	& 8 1 9	+ 5 3 7
Rec 6	& 1 1 0	+ 8 1 9

LEGEND:

X = –
& = +

7.3. DATA REDUCTION OWN-CODE ROUTINE (DROC)

DROC routines concern final disposition of fixed-length records with equal key field values. DROC routines should not be used with variable-length records. When you specify the DROC keyword parameter on the MR$PRM sort macro, you can specify:

- automatic data reduction by deletion of duplicate records (DELETE), also called auto delete; or

- the name of your own-code routine, which can handle the data reduction in three ways:

  1. by deleting one of the records containing equal keys;

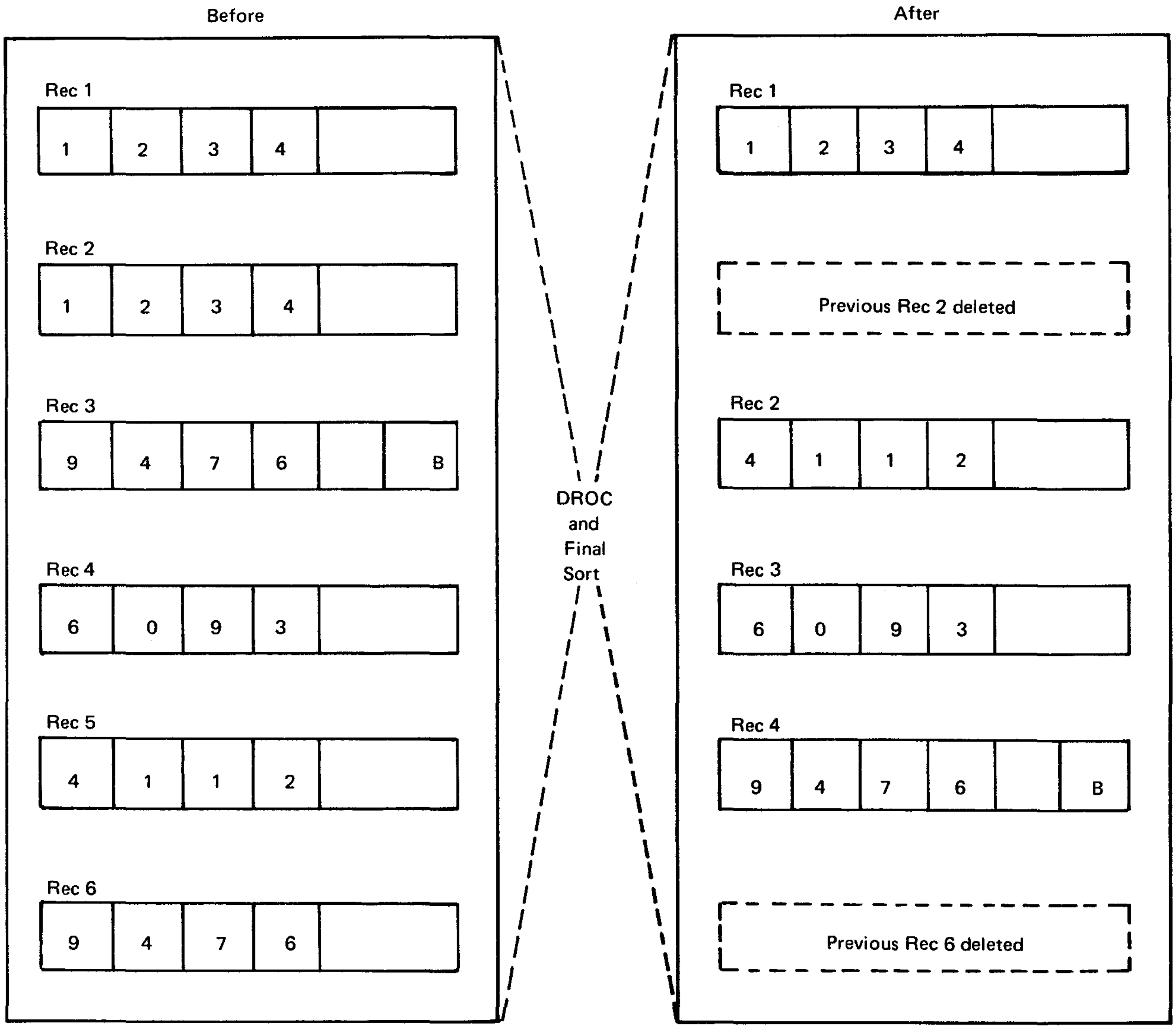
  2. by combining data contained in the two records to create a new record; and

  3. by using a combination of keeping, deleting, and combining records with duplicate keys.

Like the RSOC routine, DROC uses register 15 as a base register to contain its address. Thus, the first instruction in your DROC routine should be the USING assembler directive specifying register 15 as a base register. Registers 11 and 12 contain the addresses of the two records with equal keys. If you wish to retain only one record, the retained record address is in register 11 and the deleted record address is in register 12 unless, in your own-code routine, you overlay the address in register 11. Such an overlay forces the deletion of the address in register 11 and uses the address in register 12 as your saved record address. Normally, register 11 addresses the saved record and control returns to subroutine sort/merge four bytes beyond the subroutine sort/merge return address specified in register 14. If you want to retain both records, control must return to subroutine sort/merge at the address specified in register 14. Because register 14 contains the subroutine sort/merge return address, take great care not to change its contents.

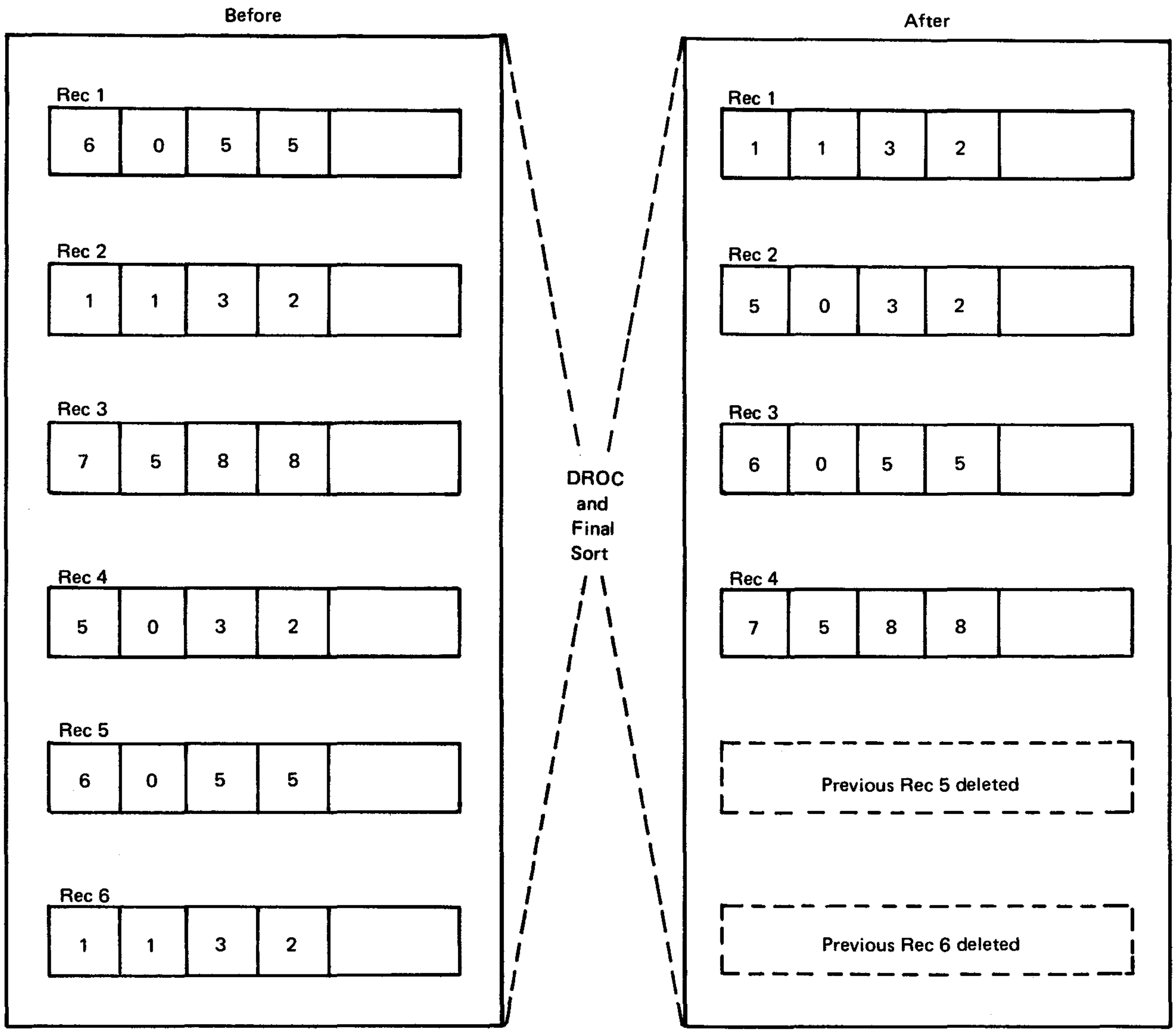
To end your DROC routine, you return control to independent sort/merge at the address specified in register 14. You may optionally include the DROP assembler directive to disengage register 15 from use as your routine's base register. The following shows the coding required to specify your own DROC routine. Notice in the diagram the file contents before and after the execution of a DROC routine which specifies your own-code routine symbolic label name, MYWORK.

```
LABEL   ΔOPERATIONΔ        OPERAND                  ΔCOMMENTS
1        10         16
--------------------------------------------------------------------------
         MR$PRM     DROC=MYWORK
  . \
  .  }                                YOUR PROGRAM
  . /
MYWORK   USING       *,15             ASSIGN BASE R15 TO OWN-CODE ROUTINE
  . \
  .  }      .                         YOUR OWN-CODE ROUTINE
  . /
         DROP        15               DISENGAGE BASE R15
         BR          14
  . \
  . /                                 RETURNS TO THE SUBROUTINE SORT/MERGE
```



You also have the alternative specification of DELETE on the DROC parameter. Before using this, you should be very sure that the records are exactly duplicated or that the key fields you need are exactly duplicated, because the subroutine sort/merge performs automatic data reduction by arbitrarily deleting one of the records with equal keys. Your program receives no control in this instance. The illustration which follows shows the coding required and a file before and after the execution of a DROC=DELETE specification in the MR$PRM sort macro.

```
LABEL    ΔOPERATIONΔ        OPERAND                         COMMENTS
1          10         16
---------------------------------------------------------------------------
           MR$PRM       DROC=DELETE
```


Before
Rec 1
6 0 5 5
Rec 2
1 1 3 2
Rec 3
7 5 8 8
Rec 4
5 0 3 2
Rec 5
6 0 5 5
Rec 6
1 1 3 2
DROC
and
Final
Sort
After
Rec 1
1 1 3 2
Rec 2
5 0 3 2
Rec 3
6 0 5 5
Rec 4
7 5 8 8
Previous Rec 5 deleted
Previous Rec 6 deleted

8. Special Subroutine Sort/Merge Applications

8.1. TAG SORT

A tag sort produces a sorted output file that contains only the direct access address or the address and key fields of records. The main purpose of a tag sort is to reduce the amount of storage required for your data files when you want the same files sorted in several different ways. A tag file allows you to access your original file in the sequence you desire without having to duplicate its entire contents. A tag sort can be performed only if you have nonindexed or IRAM disk files.

By specifying the ADDROUT parameter on the MR$PRM macro instruction, you indicate that you want to perform a tag sort. If you specify ADDROUT=A, only the 10-byte record address is returned to your program. If you specify ADDROUT=D, sort/merge returns both the address and the record key fields to your program. The length of a tag sort record cannot exceed 256 bytes, including the 10-byte address field. Sometimes you may be interested in creating a new file of key fields, as well as saving the addresses of the records they came from for later reference. If this is your need, specify ADDROUT=D. Otherwise, specify ADDROUT=A to indicate that you want only the addresses of the tag sort records returned to your program. In this case, you would only be interested in sorting the tag sort records and saving their addresses but not contents. The addresses would still enable you to retrieve their record contents at a later time. (See 6.4.2.2.)

Tag sort records are not available to your own-code routines (RSOC and DROC). Because the records are reconstructed during a tag sort, you may not know the exact location of key fields in the tag sort record. It is up to you to obtain the disk address of that input record being reconstructed and place it into the 10-byte address field of the new tag sort record. To do this, you first define the file using the DTFNI data management macro instruction and in your program's record reading routine immediately after reading each record (GET macro), issue the NOTE imperative macro to place the address of the record into a program-addressable field of the DTFNI file table, designated *filenameB*. You may address this area by concatenating the letter B to your 7-character logical file name. Refer to data management user guide, UP-8068 (current version) for the uses of the NOTE macro and the *filenameB* field. The following coding example illustrates the key instructions needed for a tag sort that returns only the address field to your program. Remember, the NOTE imperative macro must be used with the DTFNI, not the DTFSD, declarative macro.

```
LABEL    ΔOPERATIONΔ       OPERAND                    ΔCOMMENTS
1        10       16
----------------------------------------------------------------------------
         MR$PRM       ADDROUT=A
  .  )
  .  }                                          YOUR PROGRAM OPENING INSTRUCTIONS
  .  )
INPUT    DTFNI        RECSIZE=1ØØ,IOREG=(2),
  .  )
  .  }                                          OTHER DATA MANAGEMENT MACROS
  .  )
GETREC   GET    INPUT                           READS RECORD
         NOTE   INPUTB                          PLACES INPUT REC ADDR IN DATA
*                                               MANAGEMENT DEFINED AREA,INPUTB
         MVC          TAGREC+4(6),INPUTB        PLACES INPUT REC ADDR IN TAG
*                                               SORT REC ADDR AREA
         MVC          TAGREC+1Ø(8Ø),Ø(2)        PLACES KEY FIELDS IN TAG SORT
*                                               RECORD
         MR$REL                                 RELEASE RECORD TO THE SORT
         B            GETREC                    GET NEXT RECORD
TAGREC   DC           XL1Ø'ØØ'
```

8.2. RESTART FACILITIES

If your program is interrupted in the middle of a tape sort/merge, there is a way to restart it from the point of interruption. By coding the RESUME parameter on your MR$PRM macro instruction, or on a PARAM job control statement, you can indicate that you want to recover your tape sort. You must specify the most recent collation pass number displayed on the system console. (See 6.4.2.3). For additional program examples, see 9.3.

8.3. MERGE-ONLY FUNCTION

The merge-only function combines two or more similarly ordered (presorted) input files into one output file arranged in the same order as the input files. The merge-only function can combine 2 to 16 previously sequenced files into one final output file.

In a situation that requires merge-only, you start with a number of files presorted in some sequence. You are interested in expanding the size of your data files while reducing the number of files you have to work with. At the same time, you don't want to resort any files. As long as the files you are combining have been presorted in the same sequence (i.e., ascending or descending), your application is definitely a merge-only operation. Because the merge-only function is a part of the subroutine sort/merge, you must indicate to subroutine sort/merge that you want merge-only processing by writing the MERGE=YES parameter on your MR$PRM macro instruction. This places the merge-only indication in your sort parameter table.

8.3.1. What Merge-Only Does for You

The subroutine merge-only operation is activated in basically the same way as the sort/merge, with two exceptions: the sort macro, MR$SRT, is not needed, and the release and return macros, MR$REL and MR$RET, are replaced by MG$REL and MG$RET. These two macros are unique to merge-only processing.

Their formats are:

LABEL	△OPERATION△	OPERAND
[symbol]	MG$REL	

LABEL	△OPERATION△	OPERAND
[symbol]	MG$RET	

When you initiate the merge-only operation, phase 3 is performed. Multiple input files of the same sequence must be combined so that the one final output file, though expanded, has the same overall sequence. To determine the proper sequence, subroutine sort/merge performs a tournament sort to find the record that meets the output file sequence that you specified in your program. Initially, your program releases the first record of each input file to subroutine sort/merge for comparison by pairs. Subroutine sort/merge continues until a final comparison results in a single *winner* record. A tournament sort is similar to the elimination process used in a tennis match or tournament playoff. Figures 8—1 and 8—2 depict how this process occurs.

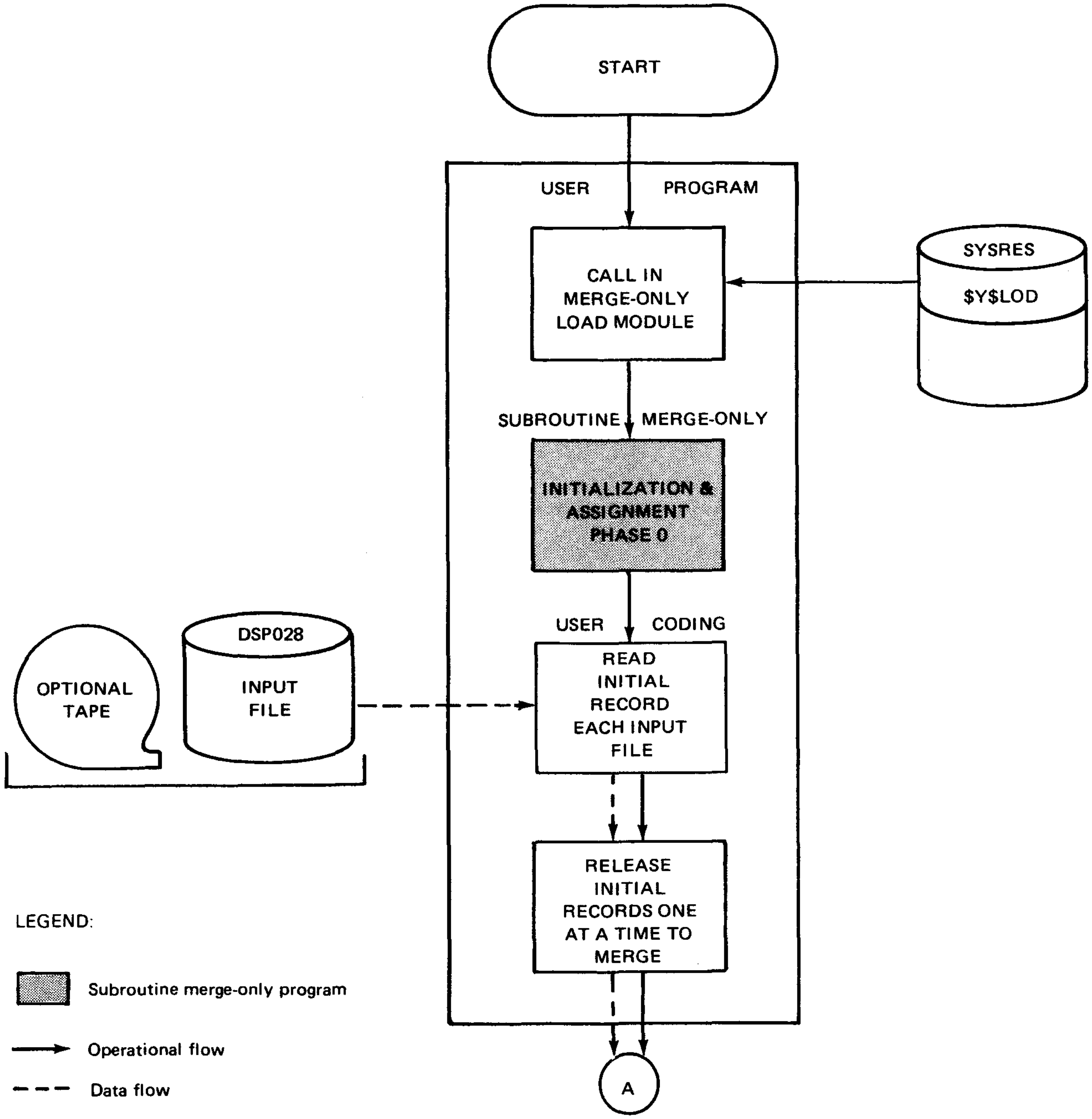

START
USER PROGRAM
CALL IN MERGE-ONLY LOAD MODULE
SYSRES
YLOD
SUBROUTINE MERGE-ONLY
INITIALIZATION & ASSIGNMENT PHASE 0
USER CODING
OPTIONAL TAPE
DSP028
INPUT FILE
READ INITIAL RECORD EACH INPUT FILE
RELEASE INITIAL RECORDS ONE AT A TIME TO MERGE
A
LEGEND:
Subroutine merge-only program
Operational flow
Data flow

Figure 8—1. Subroutine Merge-Only Operational Phases (Part 1 of 2)

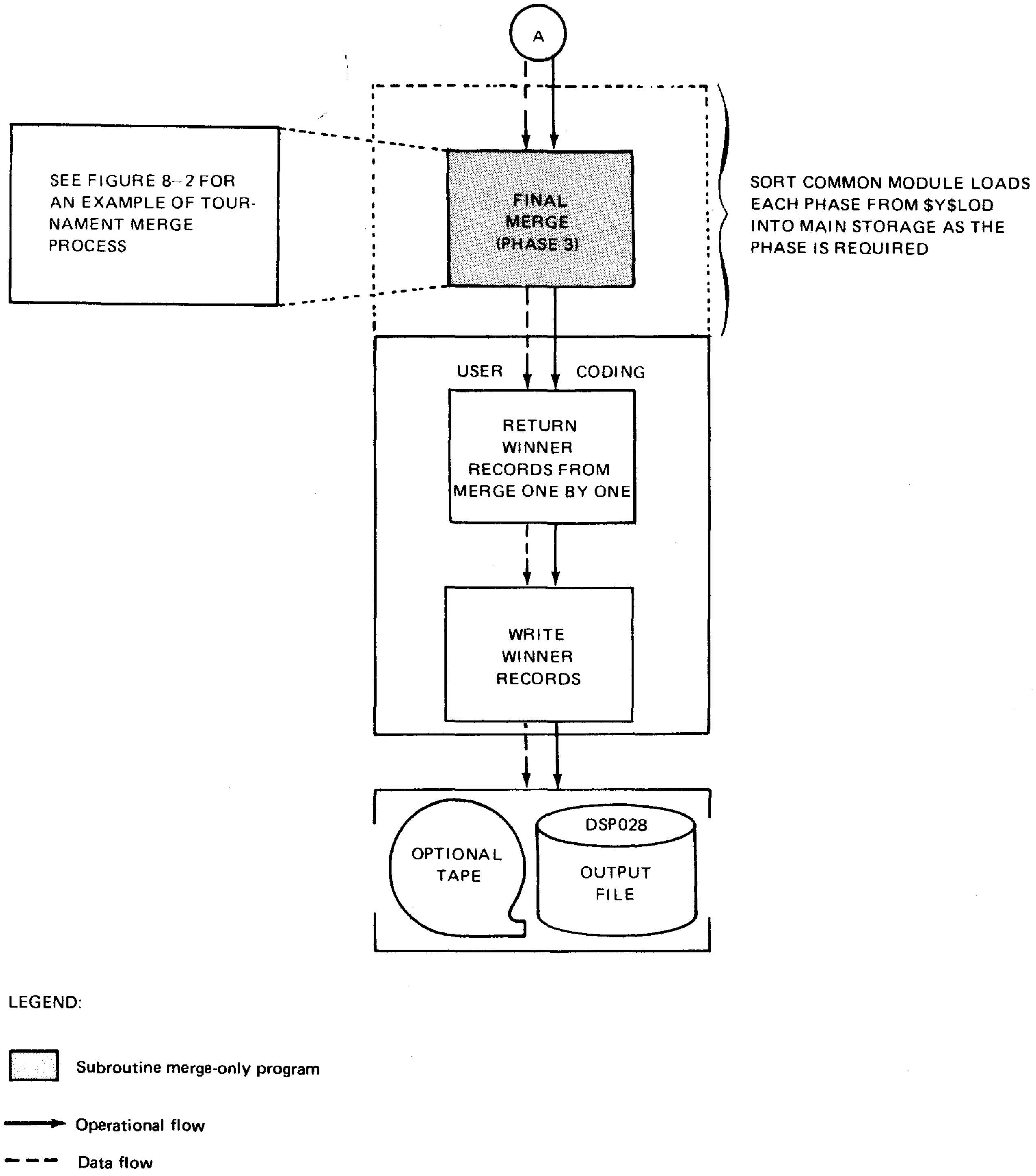

A
SEE FIGURE 8-2 FOR AN EXAMPLE OF TOURNAMENT MERGE PROCESS
FINAL MERGE (PHASE 3)
SORT COMMON MODULE LOADS EACH PHASE FROM $Y$LOD INTO MAIN STORAGE AS THE PHASE IS REQUIRED
USER
CODING
RETURN WINNER RECORDS FROM MERGE ONE BY ONE
WRITE WINNER RECORDS
OPTIONAL TAPE
DSP028
OUTPUT FILE
LEGEND:
Subroutine merge-only program
Operational flow
Data flow

Figure 8—1. Subroutine Merge-Only Operational Phases (Part 2 of 2)

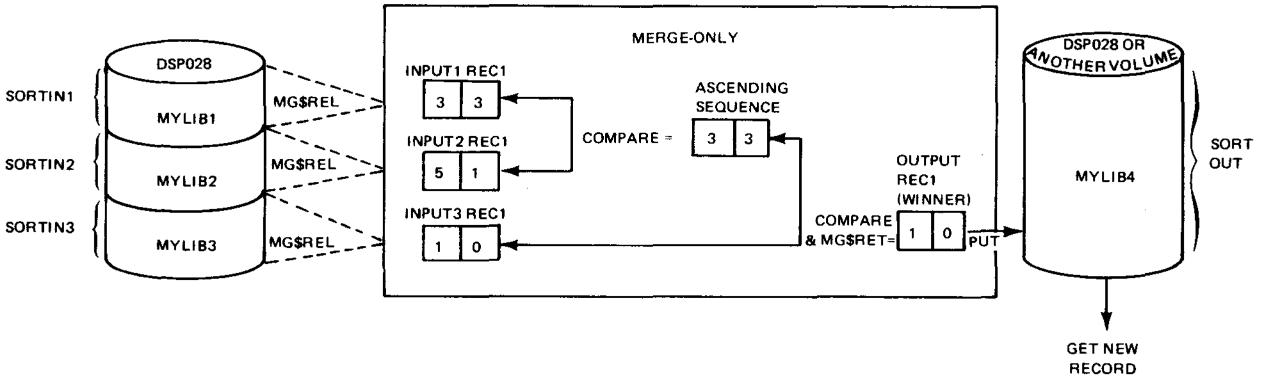


Figure 8—2. Initial Comparison for Winner Record

The record selected as the winner is returned to your program and the file identifier points your program to the next record to be released. After the first record is released, each new record released to the merge is always obtained from the input file associated with the returned winner record. The other records involved in the merge do not return to your program but remain in the merge for the next comparison. This and all succeeding comparisons are initiated as soon as your program replaces the returned winner record with the new record to be included in the merge via the MG$RET macro. This new record is always the next record of the winner record's input file. The merge process repeats until subroutine sort/merge processes all records from each input file and returns them to your program. Figure 8—3 illustrates this.

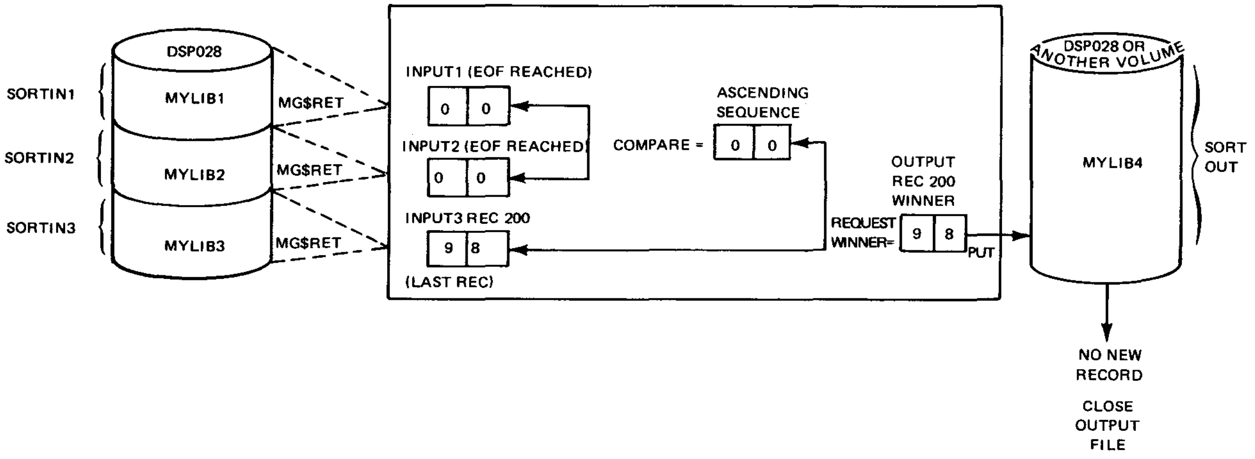


Figure 8—3. End of File Merge-Only Processing

8.3.2. Merge-Only Requirements You Supply (MG$REL and MG$RET)

Before we start to explain a sample merge-only program, let's look at a flowchart of that program (Figure 8—4) and the job description that follows.

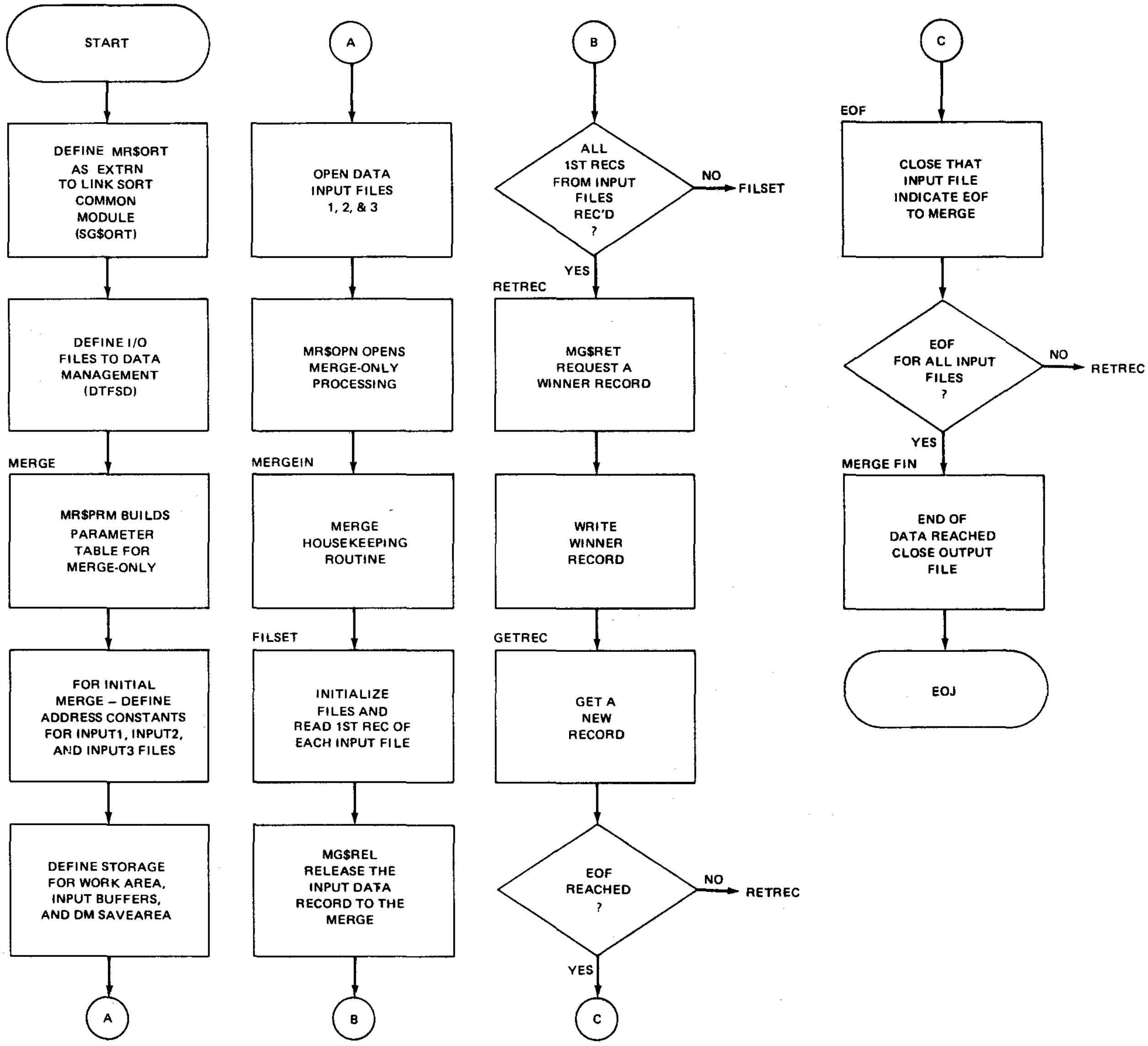

START
DEFINE MR$ORT AS EXTRN TO LINK SORT COMMON MODULE (SG$ORT)
DEFINE I/O FILES TO DATA MANAGEMENT (DTFSD)
MERGE
MR$PRM BUILDS PARAMETER TABLE FOR MERGE-ONLY
FOR INITIAL MERGE – DEFINE ADDRESS CONSTANTS FOR INPUT1, INPUT2, AND INPUT3 FILES
DEFINE STORAGE FOR WORK AREA, INPUT BUFFERS, AND DM SAVEAREA
A
A
OPEN DATA INPUT FILES 1, 2, & 3
MR$OPN OPENS MERGE-ONLY PROCESSING
MERGEIN
MERGE HOUSEKEEPING ROUTINE
FILSET
INITIALIZE FILES AND READ 1ST REC OF EACH INPUT FILE
MG$REL RELEASE THE INPUT DATA RECORD TO THE MERGE
B
B
ALL 1ST RECS FROM INPUT FILES REC'D ?
NO
FILSET
YES
RETREC
MG$RET REQUEST A WINNER RECORD
WRITE WINNER RECORD
GETREC
GET A NEW RECORD
EOF REACHED ?
NO
RETREC
YES
C
C
EOF
CLOSE THAT INPUT FILE INDICATE EOF TO MERGE
EOF FOR ALL INPUT FILES ?
NO
RETREC
YES
MERGE FIN
END OF DATA REACHED CLOSE OUTPUT FILE
EOJ

Figure 8—4. Subroutine Merge-Only Program Flowchart

MERGE-ONLY PROBLEM: A SOLUTION

SYSTEM: OS/3

PROGRAM: Subroutine Merge-Only Program

FUNCTION:

1. This program merges records of three previously sequenced files to produce a single output file.

2. It is a disk merge.

3. Previously sequenced files are in ascending sequence.

INFORMATION:

1. This program needs a table of file addresses to help locate input files for initiation of the first record merge from each file.

2. Buffer and output processing areas must be reserved in main storage for input and output file processing.

3. All three input files are assigned to disk device 51.

INPUT & OUTPUT:

1. Both input and output files use fixed-length, blocked records.

2. Each record contains 80 bytes.

3. The first input file contains ten records per block, the second input file, five records, and the third input file, twenty records.

OUTPUT:

The program produces a single output file of records merged in ascending order from three input files.

After coding your initial job control statements and assigning a base register to your program to make it relocatable (lines 1 through 11), you issue the EXTRN assembler directive, which links the sort common module from $Y$OBJ to your program (line 12), and you define your input and output files to data management (lines 15 through 26).

```
       1       10        20                                                     72
 1.  | // JOB MRGEXMPL,,7ØØØ,9ØØØ,2
 2.  | // DVC 2Ø  // LFD PRNTR
 3.  | // WORK1
 4.  | // WORK2
 5.  | // EXEC ASM
 6.  | /$
 7.  | MRGEXMPL START Ø
 8.  |          BALR  4,Ø
 9.  |          USING *,4
10.  |          LA    13,SAVEAREA
11.  |          B     START
12.  |          EXTRN MR$ORT              DEFINES THE SORT COMMON MODULE
13.  | *                                  TO BE INCLUDED BY THE LINKAGE
14.  | *                                  EDITOR
15.  | INPUT1   DTFSD BLKSIZE=8ØØ,RECSIZE=8Ø,IOAREA1=BUFF1,                     C
16.  |                IOREG=(2),RECFORM=FIXBLK,ERROR=IOERROR,OPTION=YES,         C
17.  |                EOFADDR=EOF,TYPEFLE=INPUT
18.  | INPUT2   DTFSD BLKSIZE=8ØØ,RECSIZE=8Ø,IOAREA1=BUFF2,                     C
19.  |                IOREG=(2),RECFORM=FIXBLK,ERROR=IOERROR,OPTION=YES,         C
20.  |                EOFADDR=EOF,TYPEFLE=INPUT
21.  | INPUT3   DTFSD BLKSIZE=16ØØ,RECSIZE=8Ø,IOAREA1=BUFF3,                    C
22.  |                IOREG=(2),RECFORM=FIXBLK,ERROR=IOERROR,OPTION=YES,         C
23.  |                EOFADDR=EOF,TYPEFLE=INPUT
24.  | OUTPUT   DTFSD BLKSIZE=4ØØ,RECSIZE=8Ø,IOAREA1=OUTØ1,IOAREA2=OUTØ2,       C
25.  |                WORKA=YES,RECFORM=FIXBLK,ERROR=IOERROR,OPTION=YES,         C
26.  |                TYPEFLE=OUTPUT
```

Your MR$PRM macro instruction, which creates the parameter table for your program, supplies all the information needed by the subroutine sort/merge to perform the merge. All the following parameters coded for the merge-only program example are required (lines 27 through 32). For other optional merge-only parameters that may be included in the sort parameter table generated by MR$PRM, see 6.10.

```
        LABEL   ΔOPERATIONΔ       OPERAND                          ΔCOMMENTS
        1         10       16                                                       72
27.   | MERGE     MR$PRM    IN=MERGEIN,                                             C
28.   |                    FIN=MERGEFIN,                                            C
29.   |                    STOR=WORK,                                               C
30.   |                    RCSZ=8Ø,                                                 C
31.   |                    FIELD=(Ø,8,CH),                                          C
32.   |                    MERGE=YES
```

Initially, the subroutine sort/merge needs a way to locate the first record of each input file. Lines 33—36 show the way to provide that information to your program when it begins the initial merge comparison. In addition, you must define an 18-full-word (72-byte) save area which is full-word area aligned and a save area of 18 full words for data management use (line 33).

```
33. | SAVEAREA DS      18F
34. | FILTABL  DC      A(INPUT1)         ADDRESS OF IN1 DTF
35. |          DC      A(INPUT2)         ADDRESS OF IN2 DTF
36. |          DC      A(INPUT3)         ADDRESS OF IN3 DTF
```

To begin your program, you open all the input files and the output file (lines 38 through 41). By issuing the MR$OPN (line 42) and referencing the table from your MR$PRM sort parameter table specifications (line 27), you open the subroutine merge (lines 37 through 44).

```
37. | START    EQU     *
38. |          OPEN    INPUT1
39. |          OPEN    INPUT2            OPEN INPUT FILES
40. |          OPEN    INPUT3
41. |          OPEN    OUTPUT            OPEN OUTPUT FILE
42. |          MR$OPN   MERGE            OPEN SORT/MERGE SUBROUTINE
43. | *                                  REFERENCING MR$PRM MACRO
44. | *                                  GENERATED AT MERGE
```

The next routines consist of handling the registers that receive initial file addresses and index later file address references. You must read the initial record of each input file before you release it to the merge via the MG$REL macro (line 54). This means that you must increment the full length of your input file to get to the second file (line 55). This is the value of setting up your file table of address constants earlier in lines 34 through 36.

```
     LABEL    ΔOPERATIONΔ      OPERAND                    ΔCOMMENTS
     1        10      16
45.  MERGEIN  EQU     *                           IN ADDRESS
46.           LA      5,3(Ø,Ø)                    LOAD R5 WITH THE NUMBER OF
47.  *                                            INPUT FILES
48.           LA      6,FILTABL                   GET FILE TABLE ADDRESS
49.  FILSET   L       1Ø,Ø(6)                     LOAD DTF ADDR IN R10 AND
50.  *                                            USE AS AN INDEX TO IDENTIFY
51.  *                                            INPUT FILE TO MERGE AND TO
52.  *                                            YOUR PROGRAM.
53.           BAL     7,GETREC                    GET FIRST RECORD FOR EACH FILE.
54.           MG$REL                              RELEASE RECORD TO MERGE.
55.           LA      6,4(Ø,6)                    INCREMENT TO NEXT DTF ADDR.
56.           BCT     5,FILSET                    TEST FOR LAST INPUT FILE.
57.  *                                            IF YES CONTINUE. IF NO GET FIRST
58.  *                                            RECORD OF NEXT FILE.
```

Before continuing, let's examine the function of the MG$REL macro instruction. The MG$REL is used to release only the initial record of each previously sequenced data file to the subroutine merge. After the initial record of each input file has been released and the merge begins, do not use MG$REL macro for releasing any subsequent records to the merge-only. Issue the MG$REL macro only after your program has:

- defined input and output files and assigned devices on which they are located;
- created the interface between subroutine sort/merge and your program (EXTRN MR$ORT);
- defined merge-only processing (MR$PRM);
- opened input and output files; and
- initiated subroutine sort/merge for merge-only processing (MR$OPN).

Two registers, R1 and R10, play important roles in receiving and storing addresses used by the subroutine sort/merge. Before releasing the initial record of an input file to subroutine sort/merge, you must identify both the record to be released and the file it belongs to. You identify the records and files by loading the address of the record's first byte into register 1 (line 67, GETREC routine) and the address or identifier of the file into register 10 (line 49, FILSET routine).

```
59.  RETREC   EQU     *
60.           MG$RET                              REQUEST A WINNER RECORD.
61.           BAL     7,PUTREC                    WRITE RECORD TO OUTPUT FILE.
62.           BAL     7,GETREC                    GET NEW RECORD FROM INPUT FILE.
63.           B       RETREC                      GET A NEW WINNER RECORD.
64.  GETREC   EQU     *                           GET A RECORD ROUTINE.
65.           LR      1,1Ø                        POINT TO INPUT FILE DTF.
66.           GET     (1)                         GET RECORD.
67.           LR      1,2                         POINT TO NEW RECORD.
68.           BR      7                           RETURN
```

Your record and file identification coding must precede the MG$REL macro (line 53 branches out to the GETREC routine, which points to the next record in line 67 before branching back to the MG$REL macro). After you release the initial record of the first input file, the subroutine sort/merge returns control to your program at the instruction immediately after the MG$REL macro (line 55). Your program must then point to (identify) the next input file, where you must retrieve the first record for release to the merge (line 55). You create a processing loop from the initial record accessing to its release. When the initial records from each input file have been released, your program can request the subroutine sort/merge to compare them. Select one winner record that fulfills the output sequence requirements you specified, and return it to your program. To return the winner record, you issue the MG$RET macro (line 60). Once you issue MG$RET and it executes, the MG$REL macro is no longer required to release subsequent records to the merge. The succeeding MG$RET executions automatically release the next winner record. In addition, MG$RET initiates each succeeding merge process just by requesting the return of a record.

Because of the double function of the MG$RET macro after the initial input file records are merged, you must be cautious to avoid overlaying a previous winner record with the next new record for merge, when submitting subsequent records for merge-only processing. If you do not write your winner record to the output file before the next MG$RET execution, the next record is called in, destroying your previous winner record. This can easily occur because subroutine sort/merge does not move records accessed by your program during the merge-only processing. Subroutine sort/merge, however, does make the winner record available to your program by placing the address of its first byte into register 1 and by returning control to the instruction immediately following the MG$RET macro (lines 67, 68, 63, and 61) in your program.

At this point, you must make certain that your program does not lose the winner record by having it returned to your program and consequently overlayed by the next record. This can occur because register 1 is the same register in which your program identifies the address of the next record to be released to the merge. To avoid this error, place your winner record into the output or work area (lines 69 through 72) before placing the address of the next record to be released into register 1 (line 67). The following coding shows how to avoid overlaying the winner record in our subroutine merge-only program:

```
       LABEL   ΔOPERATIONΔ      OPERAND                   ΔCOMMENTS
       1       10        16
59.  RETREC    EQU       *
60.            MG$RET                             REQUEST A WINNER RECORD.
61.            BAL       7,PUTREC                 WRITE RECORD TO OUTPUT FILE.
62.            BAL       7,GETREC                 GET NEW RECORD FROM INPUT FILE.
63.            B         RETREC                   GET A NEW WINNER RECORD.
64.  GETREC    EQU       *                        GET A RECORD ROUTINE.
65.            LR        1,1Ø                     POINT TO INPUT FILE DTF.
66.            GET       (1)                      GET RECORD.
67.            LR        1,2                      POINT TO NEW RECORD
68.            BR        7                        RETURN.
69.  PUTREC    EQU       *                        PUT RECORD ROUTINE.
70.            MVC       WORKAREA,Ø(1)            MOVE WINNER REC TO WORKAREA.
71.            PUT       OUTPUT,WORKAREA          PUT OUT THE RECORD.
72.            BR        7                        RETURN.
```

After you write the winner record to your output file, your program must always replace that record in the merge with the next record from the winner record input file (lines 62 and 64 through 68). Subroutine sort/merge enforces this requirement by placing the identifier of the winner record input file in register 10 at the same time it returns the winner record address to your program. You use this file identifier (address) from register 10 as a pointer to locate the next record you want released to the merge (lines 60 through 66). Thus, it is very important that you be careful not to alter the contents of register 10; otherwise, the merge will be in error.

After obtaining the next record from the selected file, your program must load this record address into register 1 (line 67). Execution of the MG$RET macro instruction then releases the new record to the merge for processing (PUTREC and GETREC routines).

The entire cycle repeats until your program encounters an end-of-file condition for one of the input files (identified by the file address in register 10). Your program must close this depleted file and indicate an end-of-file condition to the subroutine sort/merge before releasing additional records to the merge (EOF routine, lines 73 through 77).

```
        LABEL    ΔOPERATIONΔ        OPERAND                  ΔCOMMENTS
        1        10        16
      ------------------------------------------------------------------------------
73.   EOF        EQU       *                     END-OF-FILE ADDRESS.
74.              LR        1,1Ø                  LOAD R1 WITH DTF ADDRESS.
75.              CLOSE     (1)                   CLOSE THAT INPUT FILE.
76.              XR        1,1                   INDICATE EOF CONDITION TO MERGE.
77.              B         RETREC                REQUEST ANOTHER WINNER.
```

By loading binary 0's into register 1 and executing the MG$RET macro instruction, your program can indicate end-of-file status to the subroutine sort/merge (lines 59, 60, 76, and 77).

The merge is complete when all input files have been closed and the last winner record has been returned to your program. Subroutine sort/merge looks for the symbolic label specified on the FIN parameter of your sort parameter table (lines 28 and 73).

```
78.   MERGEFIN   EQU       *                     FIN ADDRESS.
79.              CLOSE     OUTPUT                CLOSE OUTPUT FILE.
80.              EOJ
81.   IOERROR    EQU       *
82.              CANCEL                          CANCEL THE JOB.
83.              LTORG                           DEFINE LITERALS HERE.
```

In this routine, you close the output file and indicate an end-of-job condition to job control (lines 79 and 80). Finally, you add the error routine named IOERROR as specified in your DTF data management macros (lines 16, 19, 22, 25, 81, and 82). The LTORG assembler directive in line 83 defines all literals from your program and line. Line 84 defines the work area specified in your output DTF (line 25). Data management also requires an additional eight bytes of storage before each output buffer, and all I/O buffers must be half-word aligned (lines 85 and 87). Lines 86 and 88 indicate 400-byte buffer areas for each output buffer, and lines 89—91 define the three 400-byte input buffers.

```
     LABEL   ΔOPERATIONΔ       OPERAND                          ΔCOMMENTS
     1          10       16
84.  WORKAREA DS         CL8Ø
85.           DS         4H                    HALFWORD ALIGN
86.  OUTØ1    DS         CL4ØØ                 OUTPUT AREA1
87.           DS         4H
88.  OUTØ2    DS         CL4ØØ                 OUTPUT AREA2
89.  BUFF1    DS         CL8ØØ                 INPUT AREA1
90.  BUFF2    DS         CL4ØØ                 INPUT AREA2
91.  BUFF3    DS         CL16ØØ                INPUT AREA3
92.  WORK     EQU        *
93.           END        MRGEXMPL
```

Line 92 equates the value of the location counter at this point in your program to the beginning of the work area you specified in your MR$PRM macro (line 29). The END assembler control directive indicates the end of your source program. Figure 8—5 illustrates a printout of the entire source program.

```
//ΔJOBΔMRGEXMPL,,7ØØØ,9ØØØ,2                                                  1
//ΔDVCΔ2Ø     // LFD PRNTR                                                     2
//ΔWORK1                            )                                          3
//ΔWORK2                            }= // ASM                                  4
//ΔEXECΔASM                         )                                          5
/$                                                                             6
MRGEXMPL  START Ø                                                              7
          BALR  4,Ø                                                            8
          USING *,4                                                            9
          LA    13,SAVEAREA                                                   10
          B     START                                                         11
          EXTRN MR$ORT                                                        12
*                                                                             13
*                                                                             14
INPUT1    DTFSD BLKSIZE=8ØØ,RECSIZE=8Ø,IOAREA1=BUFF1,                 C       15
                IOREG=(2),RECFORM=FIXBLK,ERROR=IOERROR,OPTION=YES,    C       16
                EOFADDR=EOF,TYPEFLE=INPUT                                     17
INPUT2    DTFSD BLKSIZE=4ØØ,RECSIZE=8Ø,IOAREA1=BUFF2,                 C       18
                IOREG=(2),RECFORM=FIXBLK,ERROR=IOERROR,OPTION=YES,    C       19
                EOFADDR=EOF,TYPEFLE=INPUT                                     20
INPUT3    DTFSD BLKSIZE=16ØØ,RECSIZE=8Ø,IOAREA1=BUFF3,                C       21
                IOREG=(2),RECFORM=FIXBLK,ERROR=IOERROR,OPTION=YES,    C       22
                EOFADDR=EOF,TYPEFLE=INPUT                                     23
OUTPUT    DTFSD BLKSIZE=4ØØ,RECSIZE=8Ø,IOAREA1=OUTØ1,IOAREA2=OUTØ2,   C       24
                WORKA=YES,RECFORM=FIXBLK,ERROR=IOERROR,OPTION=YES,    C       25
                TYPEFLE=OUTPUT                                                26
*
*               THE FOLLOWING MACRO GENERATES THE PARAMETER TABLE
*
MERGE     MR$PRM IN=MERGEIN,                                          C       27
                FIN=MERGEFIN,                                         C       28
                STOR=WORK,                                            C       29
                RCSZ=8Ø,                                              C       30
                FIELD=(Ø,8,CST),                                      C       31
                MERGE=YES                                                     32
*
```

Figure 8—5. Subroutine Merge-Only Program Coding (Part 1 of 3)

```
SAVEAREA  DS      18F              DATA MANAGEMENT SAVE AREA
FILTABL   DC      A(INPUT1)        ADDRESS OF IN1 DTF
          DC      A(INPUT2)        ADDRESS OF IN2 DTF
          DC      A(INPUT3)        ADDRESS OF IN3 DTF
START     EQU     *
          OPEN    INPUT1           OPEN
          OPEN    INPUT2               INPUT
          OPEN    INPUT3                   FILES
          OPEN    OUTPUT           OPEN OUTPUT FILE
          MR$OPN MERGE             OPEN S/M REFERENCING MR$PRM MACRO
*                                  IN ADDRESS

MERGEIN   EQU     *
          LA      5,3(Ø,Ø)         LOAD R5 WITH THE NUMBER OF INPUT
*                                  FILES.
          LA      6,FILTABL        GET FILE TABLE ADDRESS
FILSET    L       1Ø,Ø(6)          LOAD DTF ADDR IN R10, USE AS INDEX
*                                  TO IDENTIFY INPUT FILE TO MERGE
*                                  AND TO YOUR PROGRAM.

          BAL     7,GETREC         GET FIRST REC FOR EACH FILE.
          MG$REL                   RELEASE REC TO MERGE.
          LA      6,4(Ø,6)         INCREMENT TO NEXT DTF ADDR.
          BCT     5,FILSET         TEST FOR LAST INPUT FILE: IF YES,
*                                  CONTINUE. IF NO, GET FIRST REC OF
*                                  NEXT FILE.
RETREC    EQU     *
          MG$RET                   REQUEST WINNER REC
          BAL     7,PUTREC         WRITE REC TO OUTPUT FILE
          BAL     7,GETREC         GET NEW REC FROM INPUT FILE
          B       RETREC           GET NEW WINNER REC.
GETREC    EQU     *                GET a REC ROUTINE
          LR      1,1Ø             POINT TO INPUT FILE DTF.
          GET     (1)              GET RECORD.
          LR      1,2              POINT TO NEW RECORD
          BR      7                RETURN
PUTREC    EQU     *                PUT REC ROUTINE
          MVC     WORKAREA, Ø(1)   MOVE WINNER RECORD TO WORKAREA
          PUT     OUTPUT,WORKAREA  PUT OUT THE REC
          BR      7                RETURN
EOF       EQU     *                END-OF-FILE ADDR
          LR      1,1Ø             LOAD R1 WITH DTF ADDR
          CLOSE (1)                CLOSE THAT INPUT FILE
          XR      1,1              INDICATE EOF CONDITION TO MERGE
          B       RETREC           REQUEST ANOTHER WINNER
MERGEFIN   EQU    *                FIN ADDRESS
          CLOSE OUTPUT             CLOSE OUTPUT FILE
          EOJ
*
*         ERROR ADDRESS FOR DATA MANAGEMENT
*
IOERROR   EQU    *
          CANCEL                   CANCEL JOB
          LTORG                    DEFINE LITERALS HERE
*
*
```

Figure 8—5. Subroutine Merge-Only Program Coding (Part 2 of 3)

```
WORKAREA  DS    CL8Ø                                              84
          DS    4H                                                85
OUTØ1     DS    CL4ØØ                OUTPUT AREA1                 86
          DS    4H                                                87
OUTØ2     DS    CL4ØØ                OUTPUT AREA2                 88
BUFF1     DS    CL8ØØ                INPUT AREA1                  89
BUFF2     DS    CL4ØØ                INPUT AREA2                  90
BUFF3     DS    CL16ØØ               INPUT AREA3                  91
WORK      EQU   *                                                 92
          END   MRGEXMPL                                          93
/*                                 )                              94
//ΔWORK1                           )                              95
// EXEC LNKEDT                     )                              96
/$                                 ) = //MERØ1 LINK MRGEXMPL      97
 LOADM MERØ1                       )                              98
 INCLUDE MRGEXMPL                  )                              99
/*                                 )                             100
// DVC 51                                                        101
// VOL DSPØ28                                                    102
// LBL MYLIB1                                                    103
// LFD INPUT1                                                    104
// DVC 51                                                        105
// VOL DSPØ28                                                    106
// LBL MYLIB2                                                    107
// LFD INPUT2                                                    108
// DVC 51                                                        109
// VOL DSPØ28                                                    110
// LBL MYLIB3                                                    112
// LFD INPUT3                                                    113
// DVC 51                                                        114
// VOL DSPØ28                                                    115
// LBL MYLIB4                                                    116
// LFD OUTPUT,,INIT                                              117
// EXEC MERØ1,$Y$RUN                                             118
/&                                                               119
// FIN
```

Figure 8—5. Subroutine Merge-Only Program Coding (Part 3 of 3)

To eliminate extra coding, lines 3, 4, and 5 can be replaced by the ASM jproc call, which automatically supplies two work areas for the assembler. Also, lines 95 through 100 may be replaced by the single jproc call //MER01 LINK MRGEXMPL.

8.3.3. Assembling, Link Editing, and Executing Subroutine Merge-Only Program

The process of assembling, link editing, and executing the subroutine merge-only program is basically the same as our subroutine sort/merge disk sort program (6.11). Job control statements precede and follow the subroutine merge-only program. Some execute the assembler which produces an object module. The linkage editor uses this object module as input to create a load module. Further job control following the source program specifies device assignment sets and end statements (line 101 through 119). They tell us that the three input files named MYLIB1, 2, and 3 are contained on the same volume, DSP028, on the same input device 51 and that after merge processing, the records will be written to one output file, MYLIB4 on that same volume DSP028 on device 51. The load module to be executed can be found in the $Y$RUN library. Refer to the system flowchart (Figure 6—20) which depicts assembly, linkage edit, and execution runs. Figure 8—6 illustrates your program's interface with subroutine merge-only.

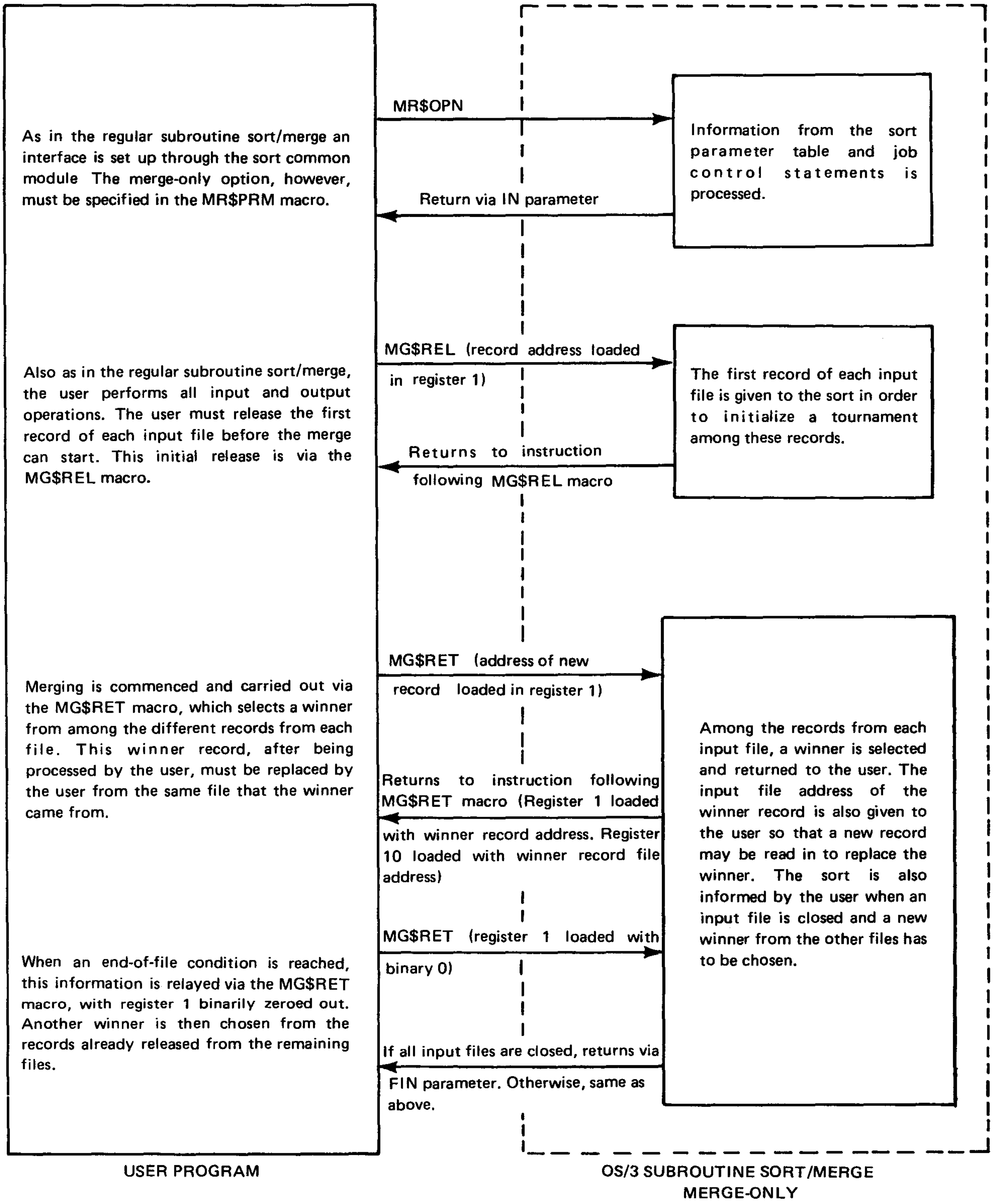

As in the regular subroutine sort/merge an interface is set up through the sort common module The merge-only option, however, must be specified in the MR$PRM macro.
MR$OPN
Information from the sort parameter table and job control statements is processed.
Return via IN parameter
Also as in the regular subroutine sort/merge, the user performs all input and output operations. The user must release the first record of each input file before the merge can start. This initial release is via the MG$REL macro.
MG$REL (record address loaded in register 1)
The first record of each input file is given to the sort in order to initialize a tournament among these records.
Returns to instruction following MG$REL macro
Merging is commenced and carried out via the MG$RET macro, which selects a winner from among the different records from each file. This winner record, after being processed by the user, must be replaced by the user from the same file that the winner came from.
MG$RET (address of new record loaded in register 1)
Among the records from each input file, a winner is selected and returned to the user. The input file address of the winner record is also given to the user so that a new record may be read in to replace the winner. The sort is also informed by the user when an input file is closed and a new winner from the other files has to be chosen.
Returns to instruction following MG$RET macro (Register 1 loaded with winner record address. Register 10 loaded with winner record file address)
When an end-of-file condition is reached, this information is relayed via the MG$RET macro, with register 1 binarily zeroed out. Another winner is then chosen from the records already released from the remaining files.
MG$RET (register 1 loaded with binary 0)
If all input files are closed, returns via FIN parameter. Otherwise, same as above.
USER PROGRAM
OS/3 SUBROUTINE SORT/MERGE MERGE-ONLY

Figure 8—6. User Program Interface with Subroutine Merge-Only

9. Subroutine Sort/Merge Program Examples

9.1. GENERAL

This section contains complete program coding examples and an explanation of:

- A subroutine tape sort
- A tape sort using the PARAM statement to add parameters to the sort parameter table
- A tape sort using a record sequence own-code routine (RSOC)
- An internal (main storage) sort
- A disk sort using consolidated data management

Each example illustrates the job control stream requirements needed to assemble, link, and execute the program. Following each example is a line-by-line description of what each instruction or group of instructions does.

9.2. SUBROUTINE TAPE SORT

The following example illustrates the general requirements for performing a typical subroutine sort operation using tape work files and disk input and output files.

```
// JOB SRTEXMP2,,7ØØØ,9ØØØ,2                                                    001
// DVC 2Ø  // LFD PRNTR                                                          002
// WORK1                                                                         003
// WORK2                                                                         004
// EXEC ASM                                                                      005
/$                                                                               006
SRTEXMPL START Ø                                                                 007
         BALR  4,Ø                                                               008
         USING *,4                                                               009
         B     START                                                             010
         EXTRN MR$ORT                  THIS DEFINE THE COMMON SORT               011
*                                      MODULE FOR INCLUSION BY THE               012
*                                      LINKAGE EDITOR.                           013
*                                                                                014
INPUT    DTFSD BLKSIZE=4Ø96,RECSIZE=256,IOAREA1=BUFF1,                  C        015
               IOREG=(2),RECFORM=FIXBLK,ERROR=IOERROR,OPTION=YES,       C        016
               EOFADDR=EOF,TYPEFLE=INPUT                                         017
```

(continued)

```
OUTPUT   DTFSD BLKSIZE=4096,RECSIZE=256,IOAREA1=BUFF1,                C   018
               IOREG=(2),RECFORM=FIXBLK,ERROR=IOERROR,OPTION=YES,     C   019
               TYPEFLE=OUTPUT                                             020
*                                                                         021
*                                                                         022
SORT     MR$PRM IN=SORTIN,                                            C   023
               OUT=SORTOUT,                                           C   024
               FIN=SORTFIN,                                           C   025
               STOR=WORK.                                             C   026
               TAPE=(NO,3),                                           C   027
               RCSZ=256,                                              C   028
               FIELD=(0,8,CH)                                             029
*                                                                         030
*        DATA MANAGEMENT WORK AREAS                                       031
         DS    0H                                                         032
         DS    CL8                                                        033
BUFF1    DS    CL4096                  IOAREA                             034
SAVEAREA DS    18F                     DATA MANAGEMENT SAVE AREA.         035
*                                                                         036
*                                                                         037


START    EQU   *                                                          038
         MR$OPN SORT                 OPEN SORT/MERGE SUBROUTINE.          039
SORTIN   LA    13,SAVEAREA           LOAD R13 WITH ADDR OF DM             040
*                                    SAVE AREA.                           041
         OPEN  INPUT                 OPEN THE INPUT FILE                  042
*                                                                         043
*              INPUT AND RECORD RELEASE ROUTINE.                          044
*                                                                         045
GETREC   EQU   *                                                          046
         GET   INPUT                 GET RECORD FROM INPUT FILE.          047
         LR    1,2                   LOAD R1 WITH ADDR OF RECORD          048
*                                    TO BE RELEASED.                      049
         MR$REL                      RELEASE RECORD TO THE SORT.          050
         B     GETREC                GET THE NEXT RECORD.                 051
*                                                                         052
*              EOF ROUTINE                                                053
*                                                                         054
EOF      EQU   *                                                          055
         CLOSE INPUT                 CLOSE THE INPUT FILE                 056
         MR$SRT                      INFORM THE SORT OF THE               057
*                                    EOF CONDITION.                       058
*                                                                         059
*              OUTPUT AND RECORD RETURN ROUTINE                           060
*                                                                         061
SORTOUT  EQU   *                                                          062
         OPEN  OUTPUT                 OPEN THE OUTPUT FILE.               063
*                                                                         064
REQREC   MR$RET                       REQUEST THE RETURN OF A             065
*                                     RECORD                              066
         MVC   0(256,2),0(1)          MOVE THE SORTED RECORD TO OUTPUT    067
*                                     BUFFER AREA.                        068
*                                                                         069
         PUT   OUTPUT                 OUT PUT THE RECORD                  070
         B     REQREC                 REQUEST NEXT RECORD                 071
*                                                                         072
*              END OF SORT ROUTINE                                        073
*                                                                         074
```

```
SORTFIN  EQU   *                                                          075
         CLOSE OUTPUT                     CLOSE THE OUTPUT FILE.          076
         EOJ                                                              077
*                                                                         078
*              ERROR ADDRESS FOR DATA MANAGEMENT                          079
*                                                                         080
IOERROR  EQU   *                                                          081
         CANCEL                           CANCEL THE JOB                  082
         LTORG                            DEFINE ALL LITERALS HERE        083
WORK     EQU   *                          SORT WORK AREA.                 084
         END   SRTEXMPL                                                   085
/*                                                                        086
// WORK1                                                                  087
// EXEC  LNKEDT                                                           088
/$                                                                        089
 LOADM   SORTØ2                                                           090
 INCLUDE SRTEXMPL                                                         091
/*                                                                        092
// DVC 5Ø                                                                 093
// VOL DSPØØ1                                                             094
// LBL SORTIN                                                             095
// LFD INPUT                                                              096
// DVC 5Ø                                                                 097
// VOL DSPØØ1                                                             098
// LBL SORTOUT                                                            099
// LFD OUTPUT                                                             100
// DVC 9Ø                                                                 101
// VOL SCRCH1                                                             102
// LFD SMØ1                                                               103
// DVC 91                                                                 104
// VOL SCRCH2                                                             105
// LFD SMØ2                                                               106
// DVC 92                                                                 107
// VOL SCRCH3                                                             108
// LFD SMØ3                                                               109
// EXEC SORTØ2,$Y$RUN                                                     110
/&                                                                        111
// FIN                                                                    112
```

Line Number   Explanation

1   The name of the job is SRTEXMP2; it requires 28,672 decimal (7000_{16}) bytes of main storage as a minimum and requests 32,768 decimal (9000_{16}) bytes maximum. A maximum of two tasks can be active simultaneously in any job step.

2   These job control statements assign the printer to the subroutine sort/merge for displaying messages during program execution.

3—4   The WORK1 and WORK2 statements set up temporary files for the assembler job step.

5   This statement initiates the execution of the assembler.

6   /$ job control delimiter statement indicates the start-of-data to the assembler.

Line Number — Explanation

7—11 This group of assembler directives and instructions initializes your location counter to zero, assigns register 4 as a base register, and defines the sort common module.

15—20 These two DTFSD statements describe input and output files to data management.

23—29 This MR$PRM macro sets up the sort parameter table and is referenced later by the MR$OPN macro (line 39). For information about these parameters, see 6.4.

32 This DS statement half-word aligns the I/O buffer.

33 This DS statement provides the 8-byte area required by data management before BUFFER 1.

34—35 These define storage statements set up 4096 bytes for input/output buffer 1 and 18 full words (72 bytes) of storage for the data management save area, which must be full-word aligned.

38—39 The MR$OPN macro opens the subroutine sort/merge.

40—42 The sort input routine opens the input data file.

46—51 The input and record release routine reads records and releases them to the sort.

55—58 The end-of-file routine closes the input data file and tells the sort that it has reached the end-of-file (MR$SRT).

62—63 The sort output routine opens the output data file.

65—71 This routine returns records from the sort/merge, writes the record, and requests the next record.

75—77 The end of sort routine closes the output file and notifies job control that the end-of-job condition was reached.

81—82 The IOERROR routine is the error handling routine for data management.

83 LTORG assembler control directive defines all literals at this point in your program.

84—86 The EQU statement indicates that the address of the current location counter be used as the beginning of the main storage work area you designated in the STOR parameter (line 26). The END assembler directive concludes your source program and the /* is a job control delimiter statement indicating the end-of-data (your source program) to the assembler.

Line Number	Explanation
87	Sets up a temporary work file for the link edit step.
88	This statement executes the linkage editor.
89—92	These statements indicate the data set to control the building of the load module named SORT02.
93—96	Assigns the input file named SORTIN to volume DSP001, on device 50.
97—100	Assigns the output file named SORTOUT to volume DSP001 on device 50.
101—109	Assigns the tape sort work files with LFD names SM01, SM02, and SM03 to volumes SCRCH1, SCRCH2, and SCRCH3 on devices 90, 91, and 92, respectively.
110	Executes your program named SORT02, which is found in $Y$RUN library.
111	Marks the end of the job stream.
112	Marks the end of reader operations.

9.3. SUBROUTINE TAPE SORT WITH RESTART USING PARAM STATEMENT

The following example illustrates requirements to perform a restart after a subroutine sort/merge program is interrupted. This example includes the RESUME parameter via the job control PARAM statement and the CSPRAM parameter indication.

```
// JOB SRTEXM13,,7ØØØ,9ØØØ                                                                1
// DVC 2Ø                                                                                 2
// LFD PRNTR                                                                              3
// WORK1                                                                                  4
// WORK2                                                                                  5
// EXEC ASM                                                                               6
/$                                                                                        7
SRTEXMPL START  Ø                                                                         8
         BALR   4,Ø                                                                       9
         USING  *,4                                                                      10
         B      START                                                                    11
         EXTRN  MR$ORT                   THIS DEFINES THE COMMON SORT                    12
*                                        MODULE FOR INCLUSION BY THE                     13
*                                        LINKAGE EDITOR.                                 14
*                                                                                        15
INPUT    DTFSD  BLKSIZE=4Ø96,RECSIZE=256,IOAREA1=BUFF1,                         C        16
                IOREG=(2),RECFORM=FIXBLK,ERROR=IOERROR,OPTION=YES,              C        17
                EOFADDR=EOF,TYPEFLE=INPUT                                                18
OUTPUT   DTFSD  BLKSIZE=4Ø96,RECSIZE=256,IOAREA1=BUFF1,                         C        19
                IOREG=(2),RECFORM=FIXBLK,ERROR=IOERROR,OPTION=YES,              C        20
                TYPEFLE=OUTPUT                                                           21
*                                                                                        22
*                                                                                        23
```

```
SORT     MR$PRM IN=SORTIN,                                             C
               OUT=SORTOUT,                                            C
               FIN=SORTFIN,                                            C
               STOR=WORK,                                              C
               TAPES=(NO,3),                                           C
               RCSZ=256,                                               C
               CSPRAM=YES,                                             C
               FIELD=(Ø,8,CH)
*
*              DATA MANAGEMENT WORK AREAS
         DS    OH
         DS    CL8
BUFF1    DS    CL4Ø96                   IOAREA
SAVEAREA DS    18F                      DATA MANAGEMENT SAVE AREA
*
*
START    EQU   *
         MR$OPN SORT                    OPEN SORT/MERGE SUBROUTINE
SORTIN   LA    13,SAVEAREA              LOAD REG 13 WITH ADDR OF DM
*                                       SAVE AREA.
         OPEN  INPUT                    OPEN THE INPUT FILE.
*
*              INPUT AND RECORD RELEASE ROUTINE.
*
GETREC   EQU   *
         GET   INPUT                    GET RECORD FROM INPUT FILE.
         LR    1,2                      LOAD REG 1 WITH ADDR OF
*                                       RECORD TO BE RELEASED.
         MR$REL                         RELEASE RECORD TO THE SORT.
         B     GETREC                   GET THE NEXT RECORD.
*
*              EOF ROUTINE
*
EOF      EQU   *
         CLOSE INPUT                    CLOSE THE INPUT FILE AND
         MR$SRT                         INFORM THE SORT OF THE END-
                                        OF-DATA CONDITION.
*
*              OUT      PUT AND RECORD RETURN ROUTINE.
*
SORTOUT  EQU   *
         OPEN  OUTPUT                   OPEN THE OUTPUT FILE.
*
REQREC   MR$RET                         REQUEST THE RETURN OF A
*                                       RECORD.
         MVC   Ø(256,2),Ø(1)            MOVE THE SORTED RECORD TO THE
*                                       OUTPUT BUFFER AREA
         PUT   OUTPUT                   OUTPUT THE RECORD.
         B     REQREC                   REQUEST THE NEXT RECORD.
*
*              END OF SORT ROUTINE.
*
SORTFIN  EQU   *
         CLOSE OUTPUT                   CLOSE THE OUTPUT FILE.
         EOJ
*
*              ERROR ADDRESS FOR DATA MANAGEMENT.
*
IOERROR  EQU   *
         CANCEL                         CANCEL THE JOB.
         LTORG                          DEFINE ALL LITERALS HERE.
```

```
WORK     EQU    *                              SORT WORK AREA.                 85
         END    SRTEXMPL                                                       86
/*                                                                             87
// WORK1                                                                       88
// EXEC LNKEDT                                                                 89
/$                                                                             90
 LOADM    SORTØ3                                                               91
 INCLUDE  SRTEXMPL                                                             92
/*                                                                             93
// DVC 5Ø                                                                      94
// VOL DSPØØ1                                                                  95
// LBL SORTIN                                                                  96
// LFD INPUT                                                                   97
// DVC 5Ø                                                                      98
// VOL DSPØØ1                                                                  99
// LBL SORTOUT                                                                100
// LFD OUTPUT                                                                 101
// DVC 9Ø                                                                     102
// VOL SCRCH1                                                                 103
// LFD SMØ1                                                                   104
// DVC 91                                                                     105
// VOL SCRCH2                                                                 106
// LFD SMØ2                                                                   107
// DVC 92                                                                     108
// VOL SCRCH3                                                                 109
// LFD SMØ3                                                                   110
// EXEC SORTØ3,$Y$RUN                                                         111
// PARAM RESUME=(PASS,233)                                                    112
/&                                                                            113
// FIN                                                                        114
```

Line Number	Explanation
1	The JOB statement names the program SRTEXM13 and specifies approximately 28,000 decimal (7000_{16}) bytes minimum main storage and 32,000 decimal (9000_{16}) bytes maximum main storage.
2—5	Assigns the printer to the job and sets up two temporary work files.
6	Executes the assembler.
7	/$ indicates the start-of-data (your source program) to the assembler.
8—14	These instructions set the location counter to zero, register 4 as base register to the program, and define the sort common module.
16—21	DTFSD statement describes input and output files to data management.
24—31	SORT is the label of the MR$PRM macro that specifies sort parameter table entries. (See 6.4 for details.) Line 30 must be specified if you intend to enter the RESUME parameter via a PARAM statement in line 112.
34	This DS statement half-word aligns the I/O buffer.
35	This DS statement provides the 8-byte area required by data management before BUFFER 1.

Line Number	Explanation
36—37	These statements define storage areas for the I/O buffer and data management save area. The 72-byte data management save area must be full-word aligned.
40—41	MR$OPN opens the subroutine sort/merge by specifying the name of the sort parameter table (line 24).
42—44	This sort input routine opens the input file.
48—53	This sort input routine reads records and releases them to the sort.
57—60	This end-of-file routine closes the input file and indicates the end-of-data (MR$SRT).
64—72	This output sort routine opens the output file, requests the return of sorted records, and writes sorted records.
76—78	The end of sort routine, SORTFIN, closes the output file and tells job control that the end-of-job condition was reached.
82—83	IOERROR is the error routine for data management.
84	LTORG defines all literals.
85—87	The EQU, END, and /* statements specify the beginning of the sort work area, the end of the source program, and the end-of-data to the assembler.
88—89	WORK1 provides a temporary work file to the linkage editor and EXEC executes the linkage editor.
90—93	This is the data set to the linkage editor (the load module SORT03).
94—101	Disk input and output data files named SORTIN and SORTOUT are assigned to volume DSP001, on device 50.
102—110	Tape work files with LFD names SM01, SM02, and SM03 are assigned to volumes SCRCH1, 2, and 3 on devices 90, 91, and 92, respectively.
111	Your program named SORT03 is executed from the $Y$RUN library.
112—113	The PARAM statement includes the RESUME parameter to provide the restart capability. (See 6.4.2.3 for more details.) The /& delimiter statement indicates the end-of-job to job control.
114	Marks the end of reader operations.

9.4. SUBROUTINE TAPE SORT USING OWN-CODE ROUTINE

The following example shows the use of a record sequence own-code routine (RSOC).

```
// JOB SRTEXM15,,7ØØØ,9ØØØ
// DVC 2Ø
// LFD PRNTR
// WORK1
// WORK2
// EXEC ASM
/$
SRTEXMPL  START  Ø
          BALR   4,Ø
          USING  *,4
          B      START
          EXTRN  MR$ORT                 THIS DEFINES THE COMMON SORT
*                                       MODULE FOR INCLUSION BY THE
*                                       LINKAGE EDITOR.
*
INPUT     DTFSD  BLKSIZE=4ØØ,RECSIZE=8Ø,IOAREA1=BUFF1,                        C
                 IOREG=(2),RECFORM=FIXBLK,ERROR=IOERROR,OPTION=YES,           C
                 EOFADDR=EOF,TYPEFLE=INPUT
OUTPUT    DTFSD  BLKSIZE=4ØØ,RECSIZE=8Ø,IOAREA1=BUFF1,                        C
                 IOREG=(2),RECFORM=FIXBLK,ERROR=IOERROR,OPTION=YES,           C
                 TYPE=OUTPUT
*
*
SORT      MR$PRM IN=SORTIN,                                                   C
                 OUT=SORTOUT,                                                 C
                 FIN=SORTFIN,                                                 C
                 STOR=WORK,                                                   C
                 TAPES=(NO,3),                                                C
                 RCSZ=8Ø,                                                     C
                 RSOC=RECCMPR
*
*
*                DATA MANAGEMENT WORK AREAS
           DS    OH
           DS    CL8
BUFF1      DS    CL4ØØ                    IOAREA
SAVEAREA   DS    18F                      DATA MANAGEMENT SAVE AREA
*
*
START      EQU   *
           MR$OPN SORT                    OPEN SORT/MERGE SUBROUTINE
SORTIN     LA    13,SAVEAREA              LOAD REG 13 WITH ADDR OF DM
*                                         SAVE AREA.
           OPEN  INPUT                    OPEN THE INPUT FILE.
*
*                INPUT AND RECORD RELEASE ROUTINE.
*
GETREC     EQU   *
           GET   INPUT                    GET RECORD FROM INPUT FILE.
           LR    1,2                      LOAD REG 1 WITH ADDR OF
*                                         RECORD TO BE RELEASED.
           MR$REL                         RELEASE RECORD TO THE SORT.
           B     GETREC                   GET THE NEXT RECORD.
*
*                EOF ROUTINE
*
```

```
EOF      EQU    *
         CLOSE  INPUT               CLOSE THE INPUT FILE AND
         MR$SRT                     INFORM THE SORT OF THE END-
*                                   OF-DATA CONDITION.
*
*            RECORD RETURN AND OUTPUT ROUTINE.
*
SORTOUT  EQU    *
         OPEN   OUTPUT              OPEN THE OUTPUT FILE.
*
REQREC   MR$RET                     REQUEST THE RETURN OF A
*                                   RECORD.
         MVC    Ø(8Ø,2),Ø(1)        MOVE SORTED REC TO OUTPUT
*                                   BUFFER AREA
         PUT    OUTPUT              OUTPUT THE RECORD.
         B      REQREC              REQUEST THE NEXT RECORD.
*
*            END OF SORT ROUTINE.
*
SORTFIN  EQU    *
         CLOSE  OUTPUT              CLOSE THE OUTPUT FILE.
         EOJ
*
*            RSOC ROUTINE
*
RECCMPR  EQU    *
         USING *,15
*            IN THIS LOCATION A ROUTINE IS TO BE INSERTED TO PERFORM
*            KEY COMPARISONS. REGISTERS 11 AND 12 CONTAIN THE ADDRESS
*            OF THE RECORDS TO BE COMPARED. IF THE RECORD POINTED TO
*            BY REGISTER 11 IS THE WINNER, THE CONDITION CODE IS TO BE
*            SET TO LOW (CC=1). IF THE RECORD FOR THE ADDRESS IN
*            REGISTER 12 IS THE WINNER, THE CONDITION CODE IS TO BE
*            SET TO HIGH (CC=2). IF THE TWO RECORDS ARE EQUAL. THE
*            CONDITION CODE IS TO BE SET TO EQUAL (CC=Ø). THE RSOC
*            ROUTINE RETURNS TO THE SORT VIA REGISTER 14.
*
*
         CLC    Ø(8,11),Ø(12)        COMPARE FOR ASCENDING SEQUENCE.
*                                    IF THE SEQUENCE WERE DESCENDING
*                                    REGISTER 11 AND REGISTER 12
*                                    WOULD BE SWITCHED SO THAT THE
*                                    INSTRUCTION WOULD READ:
*                                    CLC   4(1Ø,12),4(11)
         DROP   15                   DISENGAGE USE OF R15 AS RSOC BASE REG
         BR     14                   RETURN TO THE SORT WITH THE
*                                    CONDITION CODE SET BY THE
*                                    COMPARE INSTRUCTION.
*
*
*            ERROR ADDRESS FOR DATA MANAGEMENT
*
IOERROR  EQU    *
*        CANCEL                      CANCEL THE JOB.
*
*
*
         LTORG                       DEFINE ALL LITERALS HERE TO
*                                    FREE THE WORK AREA.
```

```
WORK      EQU   *                     SORT WORK AREA                              115
          END   SRTEXMPL                                                          116
/*                                                                                117
// WORK1                                                                          118
// EXEC LNKEDT                                                                    119
/$                                                                                120
 LOADM    SORTØ3                                                                  121
 INCLUDE  SRTEXMPL                                                                122
/*                                                                                123
// DVC 5Ø                                                                         124
// VOL DSPØØ1                                                                     125
// LBL SORTIN                                                                     126
// LFD INPUT                                                                      127
// DVC 5Ø                                                                         128
// VOL DSPØØ1                                                                     129
// LBL SORTOUT                                                                    130
// LFD OUTPUT                                                                     131
// DVC 9Ø                                                                         132
// VOL SCRCH1                                                                     133
// LFD SMØ1                                                                       134
// DVC 91                                                                         135
// VOL SCRCH2                                                                     136
// LFD SMØ2                                                                       137
// DVC 92                                                                         138
// VOL SCRCH3                                                                     139
// LFD SMØ3                                                                       140
// EXEC SORTØ3,$Y$RUN                                                             141
/&                                                                                142
// FIN                                                                            143
```

Line Number	Explanation
1—6	The program named SRTEXM15 uses approximately 28,000 decimal (7000_{16}) bytes minimum main storage space and approximately 32,000 decimal (9000_{16}) bytes maximum main storage. Two temporary work files and a printer (if needed) are made available to the assembler and it is executed.
7	This is the start-of-data to the assembler.
8—14	These instructions set the location number counter to zero, designate register 4 as the base register, and define the sort common module (MR$ORT).
16—21	DTFSD macros define the input and output files to data management.
24—30	MR$PRM defines the sort parameter table. Notice the name of the record sequence own-code routine is RECCMPR (line 30).
24—30	For more details, see 6.4.1.
34	This DS statement half-word aligns the I/O buffer.
35	This DS statement provides the 8-byte area required by data management before BUFFER 1.

Line Number	Explanation
36—37	These instructions define storage for the half-word aligned I/O buffer area and the data management 72-byte save area, which must be full-word aligned.
40—41	The MR$OPN macro opens the subroutine sort/merge.
42—44	The sort input routine opens the input file.
48—53	This input routine reads input records and releases them to the sort.
57—59	The end-of-file routine closes the output file and informs the sort of the end-of-data condition.
64—66	The sort output routine opens the output file.
67—72	This routine requests the return of records from the sort and writes the record.
76—78	The end of sort routine closes the output file and informs job control that end-of-job was reached.
82—105	RECCMPR is the name of the user's own-code routine for record sequencing.
108—109	IOERROR is the data management error handling routine.
113	LTORG assembler directive defines all literals.
115—117	This EQU statement points to the beginning of the work area. The END assembler directive names the source module that is ending and /* indicates to job control that end-of-data was reached.
118—123	Execute the linkage editor by using one temporary work file. /$ and /* mark the beginning and end of the data set used by the linkage editor.
124—131	Both disk input file SORTIN and disk output file SORTOUT on volume DSP001 use the same device 50.
132—140	Tape work files named SM01, 02, and 03, on volumes SCRCH1, 2, and 3 reside on devices 90, 91, and 92, respectively.
141—142	Execute the program SORT02 from $Y$RUN library and indicate end-of-job to job control (/&).
143	Marks end of reader operation.

9.5. SUBROUTINE INTERNAL SORT

The distinguishing characteristic of an internal sort/merge is that the entire sort process is accomplished in main storage without the use of tape or disk work files. The general program coding for an internal-only sort is identical to that for a subroutine disk or tape sort (Figure 6—17 and 9.2) except for the following modifications:

- The DISC and TAPE keyword parameters specified in the MR$PRM macro instruction for the tape and disk sorts are omitted for the internal sort.

- Disk and tape work files are not assigned for internal sorts. (The assignment of work files in the examples for tape and disk sorts appears in the job control stream.)

An internal sort/merge is feasible only if the input file is relatively small, because all of the data must be in main storage at the same time. If you do not assign adequate main storage, the sort will terminate. See 1.7.1 for minimum main storage requirements.

9.6. SUBROUTINE DISK SORT USING CONSOLIDATED DATA MANAGEMENT

The following example shows a subroutine disk sort operation using consolidated data management to define input and output files.

```
 // JOB SRTEXMPL,,7ØØØ,9ØØØ,2
 // DVC 2Ø // LFD PRNTR
 // WORK1
 // WORK2
 // EXEC ASM
 /$
 SRTEXMPL START Ø                        SETS LOCATION COUNTER TO ZERO.
          EXTRN MR$ORT                   MR$ORT DEFINES AN EXTRN.
 *                                       LINKS COMMON SORT MODULE
 *                                       TO YOUR PROGRAM.
         BALR  4,Ø
         USING *,4
         B     START
SORTRIB  RIB   BFSZ=512,RCSZ=8Ø,IOA1=BUFF1,IOA2=BUFF2,WORK=YES,            C
               RCFM=FIX,OPTN=YES,MODE=SEQ
INPUT    CDIB
OUTPUT   CDIB
USING    CD$CDIB,5
VTOC     CDIB=YES
 *
 *
SORT     MR$PRM FIELD=(Ø,7,CH),                                            C
               IN=SORTIN,                                                  C
               OUT=SORTOUT,                                                C
               FIN=SORTFIN,                                                C
               RCSZ=8Ø,                                                    C
               STOR=WORK,                                                  C
               DISC=4
```

(continued)

```
*                DATA MANAGEMENT WORK AREA
         DS      OH
BUFF1    DS      CL512
BUFF2    DS      CL512
INOUTBUF DS      CL8Ø
*
*
START    EQU     *
         MR$OPN SORT                      OPEN THE SORT/MERGE SUBROUTINE
SORTIN   LA      5,INPUT
         OPEN    INPUT,(SORTRIB)          OPEN THE INPUT FILE
         TM      CD$ISUCC,L'CD$ISUCC      SUCCESSFUL OPERATION?
         BZ      IOERROR                  IF NOT, BRANCH TO IOERROR.
GETREC   EQU     *
         DMINP   INPUT,INOUTBUF           GET RECORD FROM INPUT FILE
         TM      CD$IEOF,L'CD$IEOF        INPUT FILE EMPTY?
         BO      EOF                      IF EMPTY, BRANCH TO EOF.
         TM      CD$ISUCC,L'CD$ISUCC
         BZ      IOERROR
         LA      1,INOUTBUF               LOAD R1 WITH RECORD ADDRESS.
         MR$REL                           RELEASE RECORD TO THE SORT.
         B       GETREC                   GET NEXT RECORD.
*
EOF:     EQU     *                        THIS LOCATION IS SPECIFIED
*                                         AS THE END OF FILE ADDRESS.
         CLOSE   INPUT                    CLOSE THE INPUT FILE.
         TM      CD$ISUCC,L'CD$ISUCC
         BZ      IOERROR
*
         MR$SRT                           TELLS THE SORT THAT THE END
*                                         OF FILE HAS BEEN REACHED.
SORTOUT  EQU     *                        OUT ADDRESS.
         LA      5,OUTPUT
         OPEN    OUTPUT,(SORTRIB)         OPEN THE OUTPUT FILE.
         TM      CD$ISUCC,L'CD$ISUCC
         BZ      IOERROR
RECRET   MR$RET                           REQUEST A RECORD RETURNED.
         LA      2,INOUTBUF               LOAD R2 WITH BUFFER ADDRESS
         MVC     Ø(8Ø,2),Ø(1)             MOVE THE SORTED RECORD TO
*                                         THE OUTPUT BUFFER AREA.
         DMOUT   OUTPUT,INOUTBUF          OUTPUT THE RECORD RETURNED.
         TM      CD$ISUCC,L'CD$ISUCC
         BZ      IOERROR
         B       RECRET
*
SORTFIN  EQU     *                        FIN ADDRESS
         CLOSE   OUTPUT                   CLOSE THE OUTPUT FILE.
         TM      CD$ISUCC,L'CD$ISUCC
         BZ      IOERROR
         EOJ                              END OF JOB STEP.
*
```

(continued)

```
*            ERROR ADDRESS FOR DATA MANAGEMENT                            80
*                                                                         81
IOERROR  EQU   *                                                          82
         CANCEL                          CANCEL THE JOB.                  83
         LTORG                           DEFINE ALL LITERALS HERE.        84
WORK     EQU   *                         START OF SORT WORK AREA.         85
*                                        THIS SETUP ALLOWS THE SORT       86
*                                        TO USE ALL MEMORY FROM           87
*                                        THIS LOCATION TO THE END OF      88
*                                        THE JOB REGION.                  89
         END   SRTEXMPL                                                   90
/*                                                                        91
//SRTEXM LINK SRTEXMPL                                                    92
// DVC 5Ø                                                                 93
// VOL DSPØ28                                                             94
// LBL MYFILE1                                                            95
// LFD INPUT                                                              96
// DVC 5Ø                                                                 97
// VOL DSPØ28                                                             98
// LBL MYFILE2                                                            99
// LFD OUTPUT,,INIT                                                      100
// DVC 5Ø                                                                101
// VOL DSPØ28                                                            102
// EXT ST,C,,CYL,5                                                       103
// LBL $SCR1                                                             104
// LFD DMØ1                                                              105
// EXEC SRTEXM,$Y$RUN                                                    106
/&                                                                       107
// FIN                                                                   108
```

Line Number	Explanation
1—5	The program named SRTEXMPL uses approximately 28,000 decimal (7000_{16}) bytes of minimum main storage space and approximately 32,000 decimal (9000_{16}) bytes of maximum main storage. Two temporary work files and a printer (if needed) are made available to the assembler, and the assembler is executed.
6	This is the start-of-data to the assembler.
7—13	These instructions set the location counter to zero, set register 4 as base register to the program, and define the sort common module.
14—17	These instructions specify two files named INPUT and OUTPUT, each with a CDIB macroinstruction. In addition, a RIB labeled SORTRIB is specified as having a buffer size of 512 bytes, a record size of 80 bytes, an IOA1 called BUFF1 for the primary I/O buffer area, an additional IOA2 called BUFF2 to speed up I/O processing, and sequential access. The WORK parameter indicates that all input and output operations take place using data contained in a work area. OPTN=YES indicates that all files associated with SORTRIB are optional; i.e., they won't always be used.

Line Number — Explanation

Only one RIB macro is needed because it defines the characteristics of both files INPUT and OUTPUT.

18—19 The VTOC macro generates a DSECT, which in effect is a map of the CDIB. The USING directive associates register 5 with the address of the first CDIB byte, CD$CDIB. Statements generated by VTOC then fix the indicators CD$ISUCC (lines 40, 46, 55, 63, 70, and 76) and CD$IEOF (line 44) as offsets from register 5. All you do, as a result, is load register 5 with the address of any actual CDIB, and you can test the bit indicators in the CDIB as symbols rather than having to know where exactly in the CDIB they lie. The use of register 5 with VTOC does not affect register 4 because the remainder of the program continues to use register 4 as its base register.

22—28 This MR$PRM macro sets up the sort parameter table and is referenced later by the MR$OPN macro (line 37). For information about these parameters, see 6.4.

30 This DS statement half-word aligns the I/O buffer.

31—32 These define storage statements set up 512 bytes for input/output buffers 1 and 2, and 80 bytes of storage for the data management save area, which must be full-word aligned.

36—37 The MR$OPN macro opens the subroutine sort/merge.

38 Establishes the base register as described in line 18.

39 The sort input routine opens the input data file.

40—41 Before doing any I/O operation, you want to link bit indicators CD$ISUCC (line 40) and CD$IEOF (line 44) to the file whose condition they are to test. You do this by loading register 5 with the address of the input file CDIB, an operation that takes place at location SORTIN. As long as register 5 remains unchanged, CD$ISUCC and CD$IEOF will reflect the condition of file INPUT.

When you open your input file (line 39), you associate it with the RIB named SORTRIB. Lines 40 and 41 contain a pair of instructions that recur throughout the program. The *test under mask* (TM) instruction at line 40 tests the successful-operation indicator CD$ISUCC that is set during the preceding OPEN operation. The *branch on zero* (BZ) instruction at line 41 causes a branch to routine IOERROR only if the CD$ISUCC indicator has been set off (the I/O operation has failed for some reason); otherwise, the operation has been successful and control passes to the next sequential instruction. You code these TM and BZ instructions after each data management macro in your program.

Line Number   Explanation

42—48 With the file open, you can read the input file by designating the DMINP imperative macro (line 43). Because you plan to read many records and you will need to repeat this instruction, you label it GETREC, giving yourself a place to return for reading subsequent records. Data management automatically loads the first data record address into register 2 when you specify IORG=(2) on the RIB macro. Because sort/merge expects the address of the record being released to it to be in register 1, you must load register 1 with the record address (in this case, the work area INOUTBUF). In this example, a *load address* (LA) instruction is used (line 48).

49—50 These instructions release the record to the sort and get the next record, until the end of the input file is reached.

52—59 When the last record is read, the CDIB end-of-file indicator CD$IEOF is set on. The TM and BO instructions at lines 44—45, which before have passed control to the next instruction, now cause a branch to the routine beginning at EOF. The end-of-file routine closes the input data file and tells the sort that it has reached end-of-file (MR$SRT).

60—62 This is the sort output routine. You load register 5 with the address of the OUTPUT file CDIB (line 61). This causes the bit indicator CD$ISUCC to reflect the condition of the file OUTPUT. Line 62 opens the output file.

65—72 This routine returns each record from sort/merge, writes the record, and requests the next record.

74—78 The end-of-sort routine closes the output file and notifies job control that the end-of-job condition was reached.

82—83 The IOERROR routine is the error handling routine for data management.

84 The LTORG assembler control directive defines all literals at this point in the program.

85—91 The EQU statement indicates that the address of the current location counter is to be used as the beginning of the main storage work area you designated in the STOR parameter (line 27). The END assembler directive concludes your source program and the /* is the job control delimiter statement indicating the end-of-data (your source program) to the assembler.

92 This is the LINK job control procedure call, which generates a load module called SRTEXM from the object module (called SRTEXMPL).

93—96 Assigns the input file named INPUT to volume DSP028 on device 50. The INIT parameter on line 100 indicates that you want to start writing at the beginning of the file, overlaying its previous contents.

Line Number Explanation

101—105 Assigns the disk sort work file with LFD name DM01 to volume DSP028 on device 50. The EXT statement specifies that your work file is accessed via the system address technique (ST), allocates contiguous space for the extent (C), specifies that space must be allocated in cylinders (CYL), and allocates 5 cylinders for the work file.

106 Executes your program named SRTEXM, which is found in the $Y$RUN library.

107 Marks the end of the job stream.

108 Marks the end of reader operations.

PART 4. SYSTEM/3, 32, and 34 COMPATIBLE SORT

10. System/3, 32, and 34 Compatible Sort Basic Concepts

10.1. GENERAL

The SORT3 program, as introduced in Part 1, assists you in sorting and merging your data files with only a minimum amount of user program intervention. Simplicity is the word which best describes the use of this program because, basically, it does all the work for you. It reads your data files, selectively sorts and merges the data according to your specifications, and then writes the data to your output file. You do not have to concern yourself with opening and closing files, supplying read and write routines, passing data to the sort program or retrieving sorted data for the output. Your responsibility is to provide the data files to be sorted, and to prepare the control stream necessary to define the sort and execute the program. In relation to the entire job, your program responsibility, as shown in Figure 10—1, is involved only with the input step of running SORT3. Details for preparing both control statements and sort specifications are covered in Section 11.

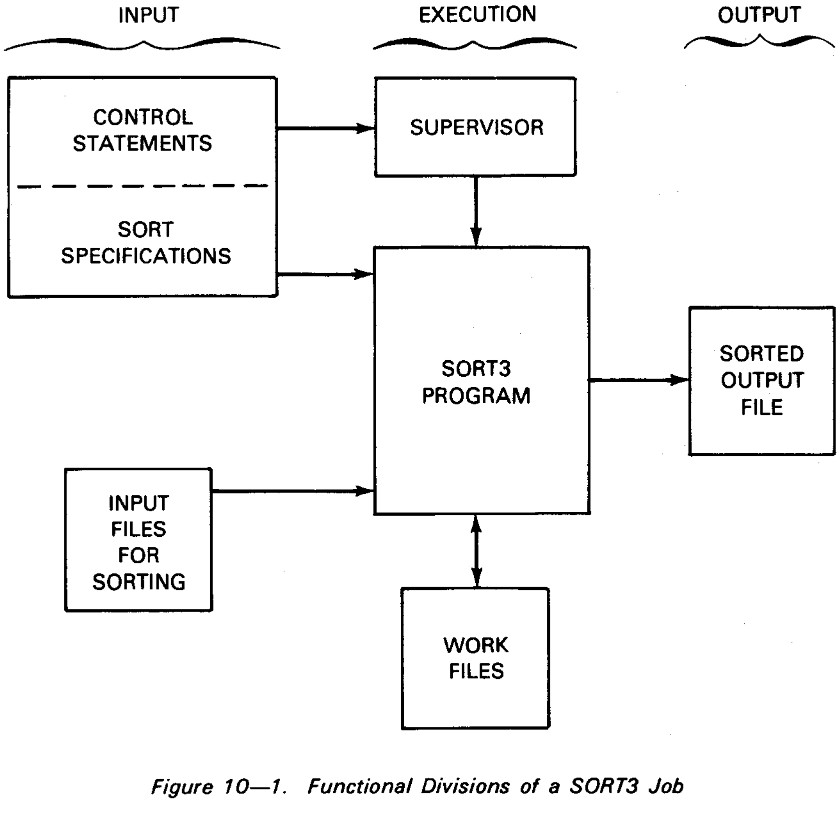


Figure 10—1. Functional Divisions of a SORT3 Job

10.2. EXECUTION OF THE SORT3 PROGRAM

Execution of the SORT3 program takes place after the system input device has read your control stream. In the discussion of program execution, take note of the interplay of activities between your control stream input, the system, and the SORT3 program. The entire sort/merge operation centers around the elements supplied by both you and SORT3.

Program execution begins when the EXEC statement (JCL) or RUN statement (OCL) is read from the control stream you submitted to the system for running your job. This statement signals the system to load the system driver program (Figure 10—2) for SORT3 into main storage. The system driver program provides the interface between the system and the remaining SORT3 program modules. The first action taken by the driver program is to call into main storage the sort modules needed to initialize the sort process. The loading of the modules signals the sort program to accept your sort specifications and execute the first phase of the program: initialization and assignment. As explained in Section 1, the sort program is modular and the various modules used in the sort process reside in the system load library file ($Y$LOD) on the SYSRES volume. When one phase is completed, it signals the driver program to load the next group of modules into main storage and execute the next phase in the sort process.

10.3. SOFTWARE FRAMEWORK OF SORT3 PROGRAM

SORT3 consists of four operational phases that are normally executed in the following sequence:

- Phase 0 — Sort initialization and assignment
- Phase 1 — Data input read and internal sort
- Phase 2 — Preliminary merge
- Phase 3 — Final merge and output

In cases where the input file is partially sequenced or is small enough so one final merge produces the required output sequence, SORT3 bypasses phase 2 and proceeds to phase 3, where the records are read into main storage, merged, and written to the output file. Figure 10—3 shows the operational phases of the SORT3 program.

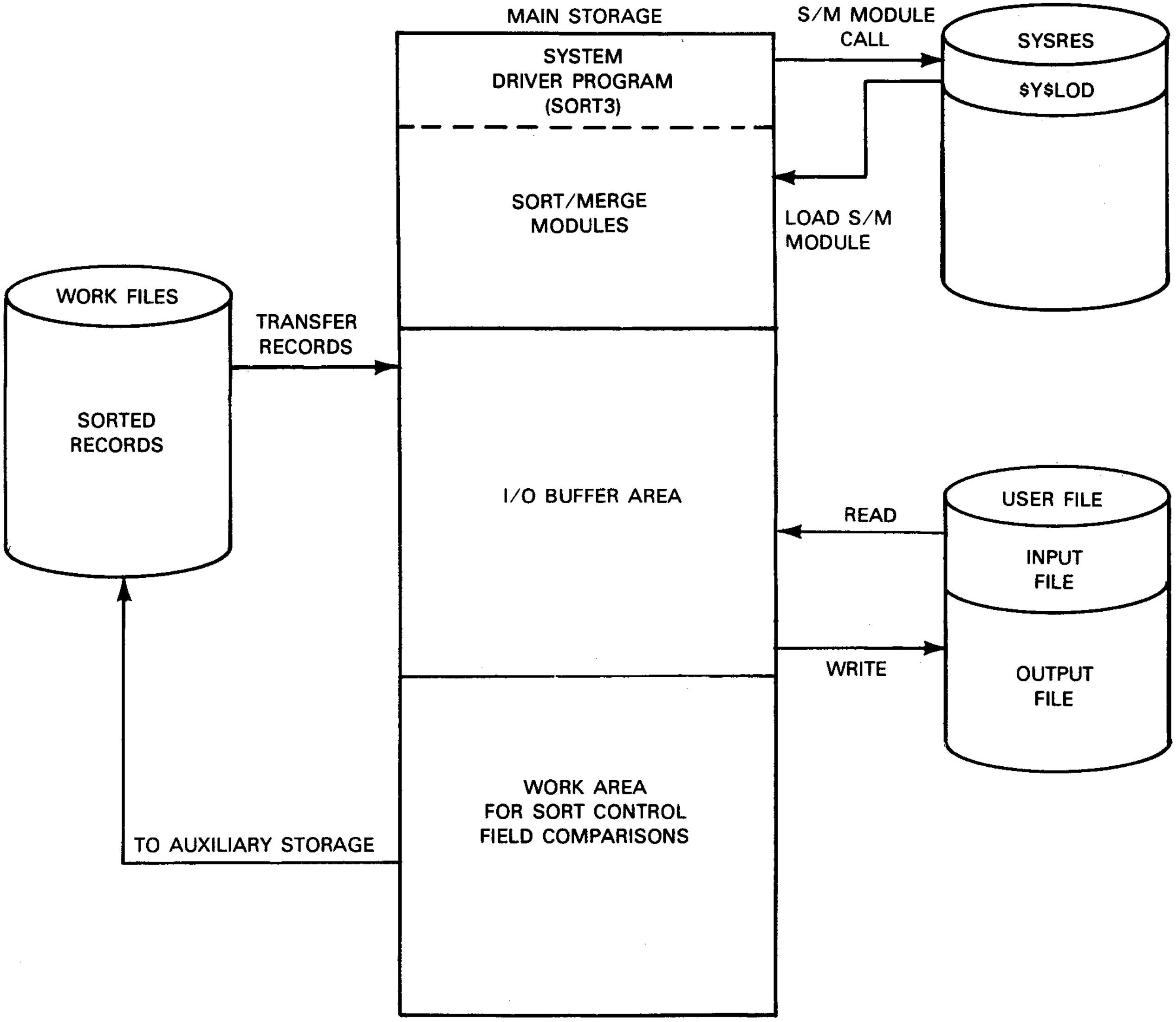


Figure 10—2. Execution of SORT3 Program

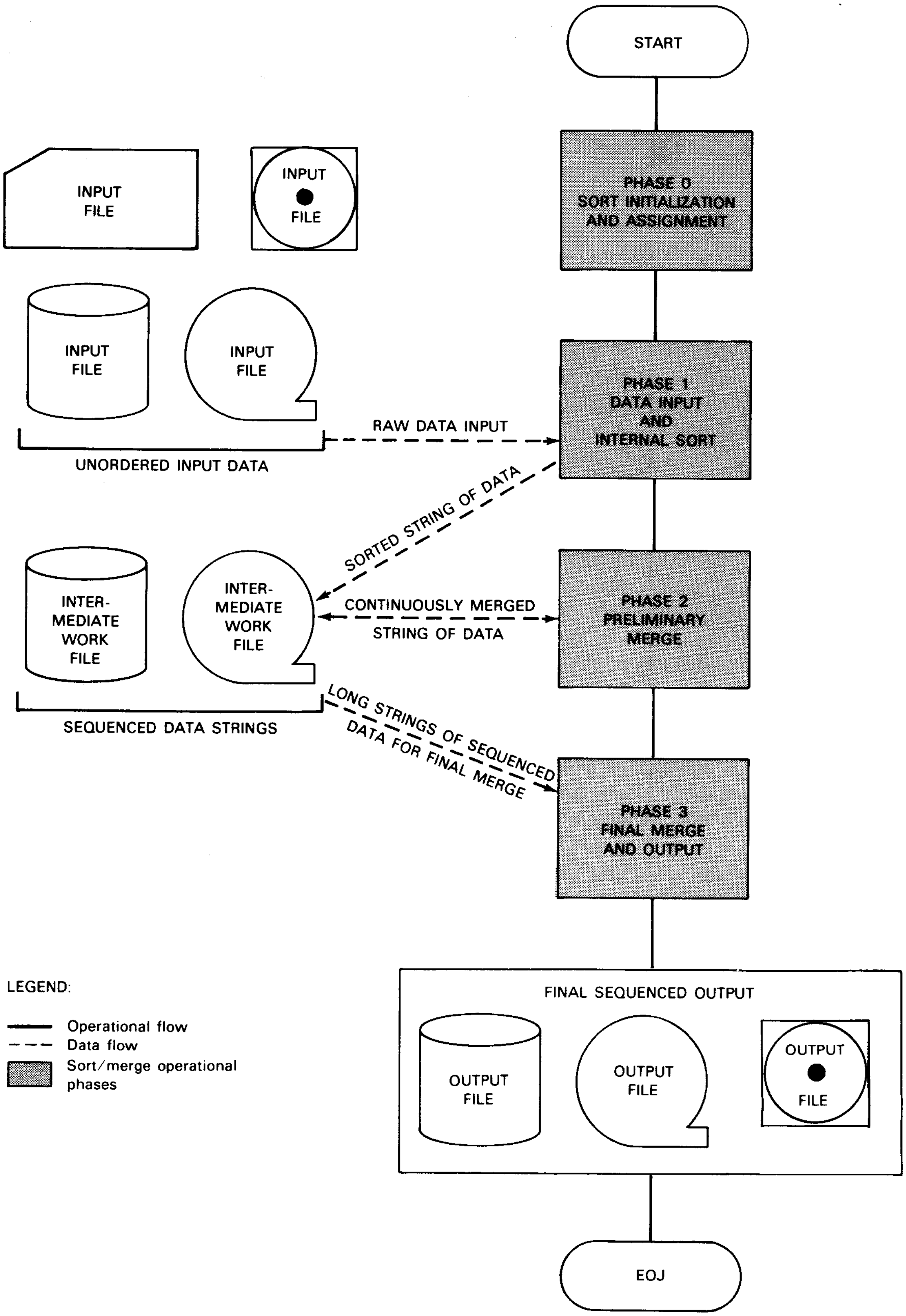


Figure 10—3. SORT3 Operational Phases

10.3.1. Phase 0: Sort Initialization and Assignment

Phase 0 (Figure 10—4) initializes the sort process by reading sort control statements from the job control stream. It validates both the content and syntax of these statements and then passes control to an assignment segment of the phase. By examining your parameters, the assignment segment determines the type of sort function to be performed. In addition, it builds a parameter table, sets up compare routines, and structures the SORT3 processor to perform only the sort functions you have specified. When the assignment segment completes its task, phase 0 passes control to phase 1.

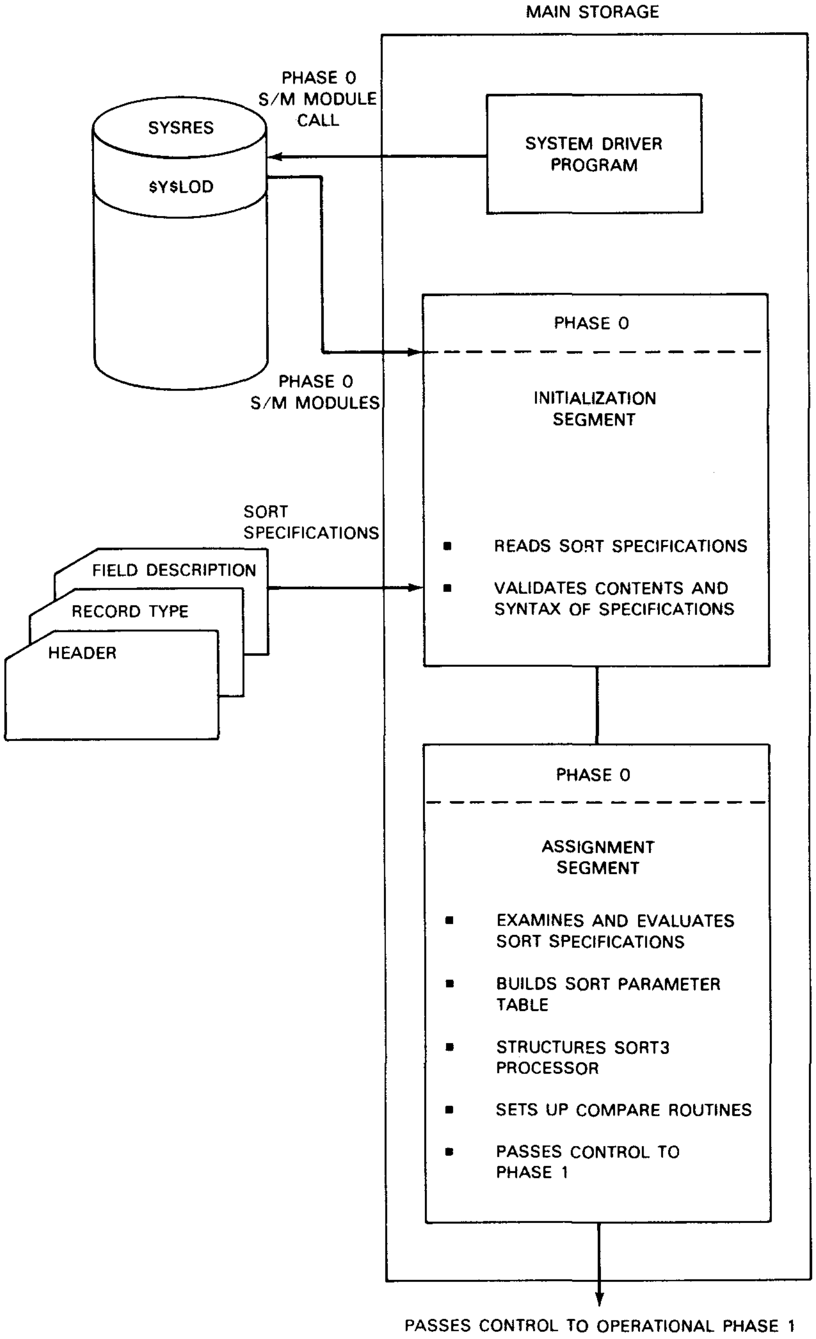

MAIN STORAGE
PHASE 0
S/M MODULE
CALL
SYSRES
YLOD
SYSTEM DRIVER
PROGRAM
PHASE 0
PHASE 0
S/M MODULES
INITIALIZATION
SEGMENT
SORT
SPECIFICATIONS
FIELD DESCRIPTION
RECORD TYPE
HEADER
■ READS SORT SPECIFICATIONS
■ VALIDATES CONTENTS AND SYNTAX OF SPECIFICATIONS
PHASE 0
ASSIGNMENT
SEGMENT
■ EXAMINES AND EVALUATES SORT SPECIFICATIONS
■ BUILDS SORT PARAMETER TABLE
■ STRUCTURES SORT3 PROCESSOR
■ SETS UP COMPARE ROUTINES
■ PASSES CONTROL TO PHASE 1
PASSES CONTROL TO OPERATIONAL PHASE 1

Figure 10—4. Operational Phase 0

10.3.2. Phase 1: Data Input and Internal Sort

When phase 1 receives control, it initiates an input routine that opens your input files, validates file labels, and reads the data records one at a time. (The location of your data files is determined by the device assignment sets in your control stream.) Before a record is passed to the internal sort routine of this phase for initial sorting, it is checked against the criteria of your sort specifications to determine whether it is to be included in the sort. If the record is to be included, phase 1 reformats the record into a sort record (according to your specifications) and passes the sort record to the internal sort routine. (The details of sort record handling are described in 10.4.)

Internal sorting is performed in main storage and produces strings of sequenced data that are written as intermediate files to auxiliary storage devices (tape or disk). If the number of data strings produced during the internal sort are few enough to be merged in one final merge, phase 2, the preliminary merge is bypassed and control passes to phase 3 for final merging and output to the output file. Otherwise, strings of sequenced data must be continuously merged into larger and larger data strings until only one final merge operation is required to produce an output file sequenced in the order you specified. Figure 10—5 illustrates data flow from the input file through internal sort processing.

When the internal sort is completed, control passes to either phase 2 or phase 3.

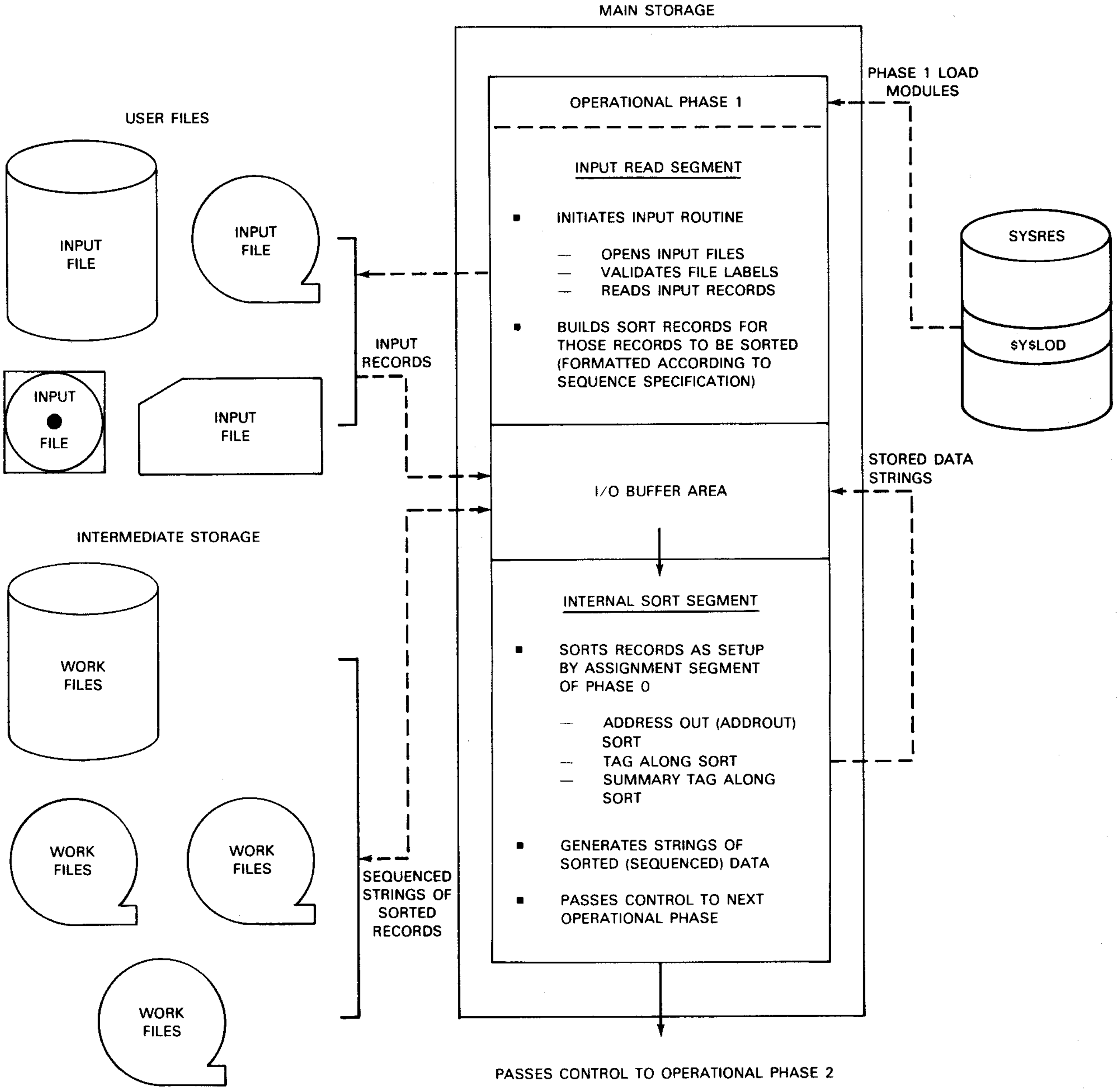

MAIN STORAGE
PHASE 1 LOAD MODULES
OPERATIONAL PHASE 1
USER FILES
INPUT READ SEGMENT
• INITIATES INPUT ROUTINE
— OPENS INPUT FILES
— VALIDATES FILE LABELS
— READS INPUT RECORDS
• BUILDS SORT RECORDS FOR THOSE RECORDS TO BE SORTED (FORMATTED ACCORDING TO SEQUENCE SPECIFICATION)
INPUT FILE
INPUT FILE
INPUT FILE
INPUT FILE
INPUT RECORDS
SYSRES
$Y$LOD
STORED DATA STRINGS
I/O BUFFER AREA
INTERMEDIATE STORAGE
INTERNAL SORT SEGMENT
• SORTS RECORDS AS SETUP BY ASSIGNMENT SEGMENT OF PHASE 0
— ADDRESS OUT (ADDROUT) SORT
— TAG ALONG SORT
— SUMMARY TAG ALONG SORT
• GENERATES STRINGS OF SORTED (SEQUENCED) DATA
• PASSES CONTROL TO NEXT OPERATIONAL PHASE
WORK FILES
WORK FILES
WORK FILES
WORK FILES
SEQUENCED STRINGS OF SORTED RECORDS
PASSES CONTROL TO OPERATIONAL PHASE 2

Figure 10—5. Operational Phase 1

10.3.3. Phase 2: Preliminary Merge

When phase 2 receives control, the module executed for it continuously merges data strings produced in phase 1. These merge passes occur between auxiliary storage devices, each successive merge producing longer and longer sequenced data strings. When only one final merge pass is needed to create a single sequenced string (final output string), phase 2 passes control to phase 3. Figure 10—6 shows long sequenced data strings ready to be given to the final merge phase as a result of phase 2 operations.

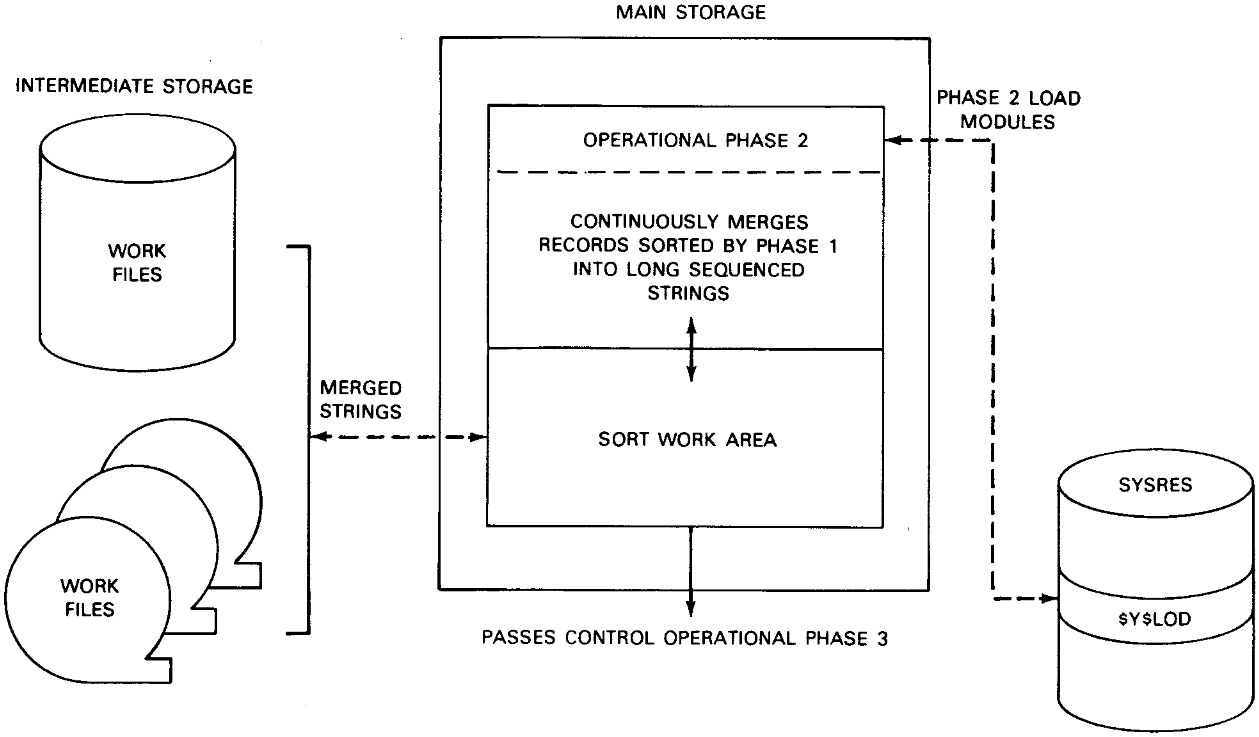


Figure 10—6. Operational Phase 2

10.3.4. Phase 3: Final Merge and Output

The final merge phase merges all data strings written to the work files into one sequenced string and passes it to the sort output routine. The output routine opens your output file, writes the output data, closes the output file, terminates the sort, and returns control to the system. Figure 10—7 shows the data flow during phase 3 execution.

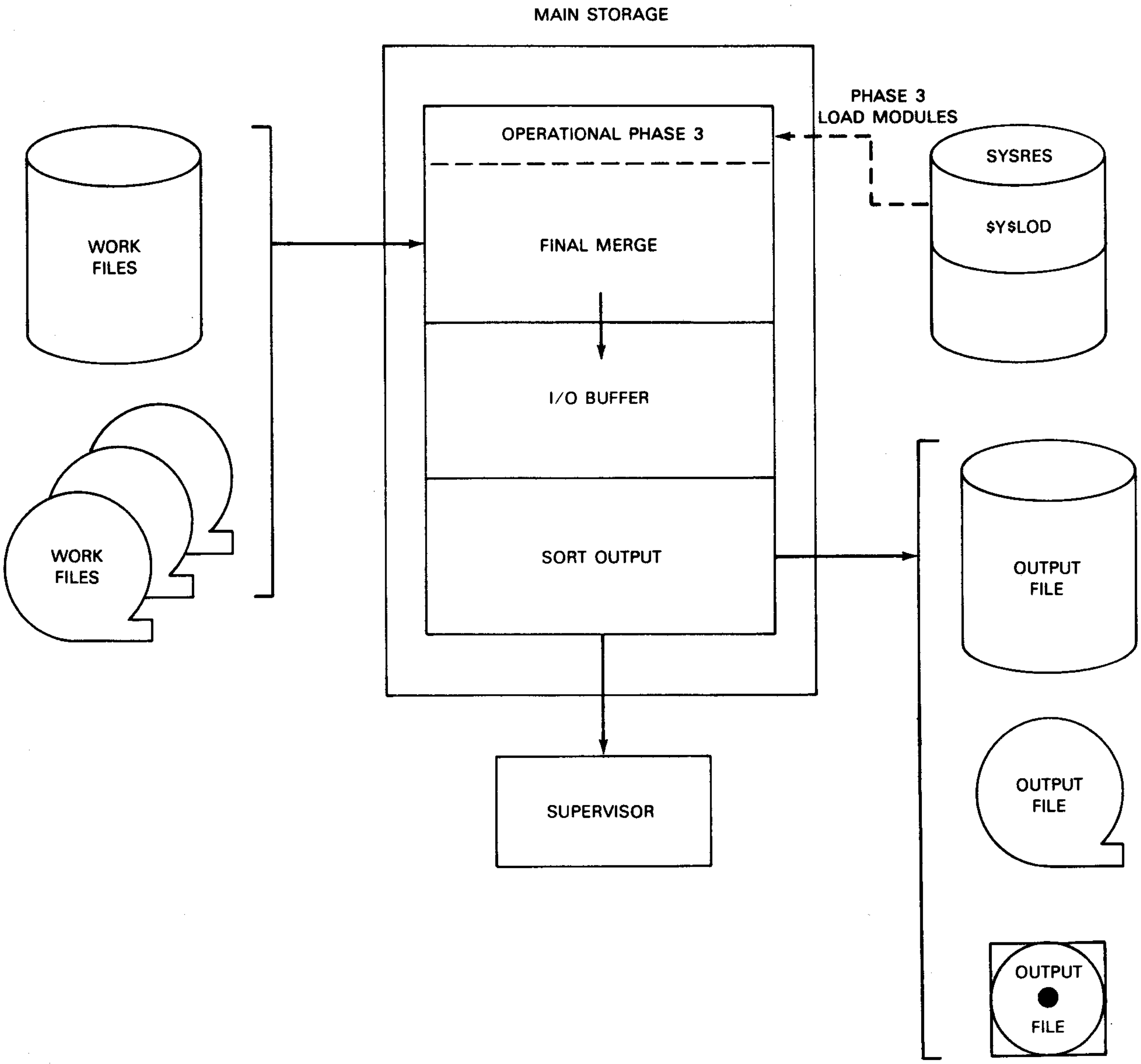
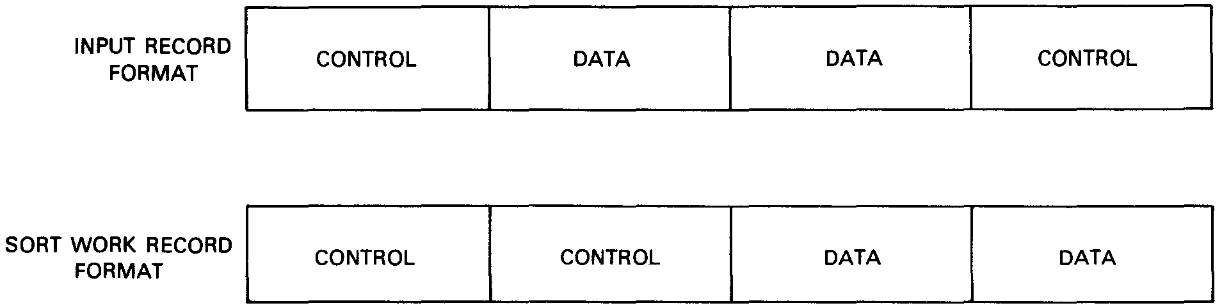


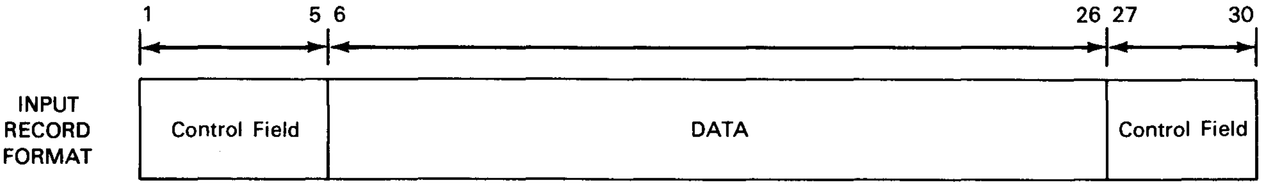
Figure 10—7. Operational Phase 3

10.4. RECORD HANDLING DURING SORT

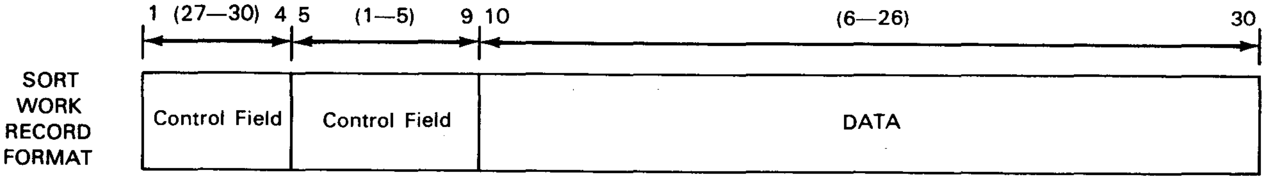
When your input records are read during phase 1 execution of the sort, SORT3 checks each record to see if it is to be accepted or rejected on the basis of your sort specifications. SORT3 builds a sort work record for each input record accepted into the sort. The sort work record is reformatted to increase the efficiency of the sort process. (SORT3 does not change the format of the actual input record.) In the reconstructed format, control or key fields are placed ahead of data fields unless, of course, the control fields are to be dropped during the sort.



The placement of specific control fields and data fields within the sort work record is defined by the parameters of your record type and field description specifications. For example, assume you have identified positions 27 through 30 of your input records as a primary control field, and positions 1 through 5 as a secondary control field. Positions 6 through 26 contain data. Your input record would appear as:



When SORT3 accepts the input record for sorting, it repositions the record fields according to the sequence you have specified. The primary control field appears first, the secondary field next, and so on until all the control fields are properly positioned and followed by the data fields. The sort work record would appear as:



The sort work record is sent for initial (internal) sorting after it is constructed. SORT3 uses the control fields to sequence the records in either ascending or descending order according to your sort specifications. When all the records are properly sequenced, SORT3 writes the records to your output file. The output record format is the same as the sort work record format unless SORT3 is instructed to drop control fields during the sort.

10.5. CHARACTERISTICS OF SORTS PERFORMED BY SORT3 PROGRAM

There are three types of sort jobs performed by SORT3: addrout (address out), tag-alone (data fields can *tag along* with control fields in the sorted records), and summary tag-along (data is summarized in the sorted records).

The output from an addrout sort job (Figure 10—8) consists of 10-byte binary relative record numbers of the records in the input file.

NOTE:

An addrout sort can be used to process disk files only.

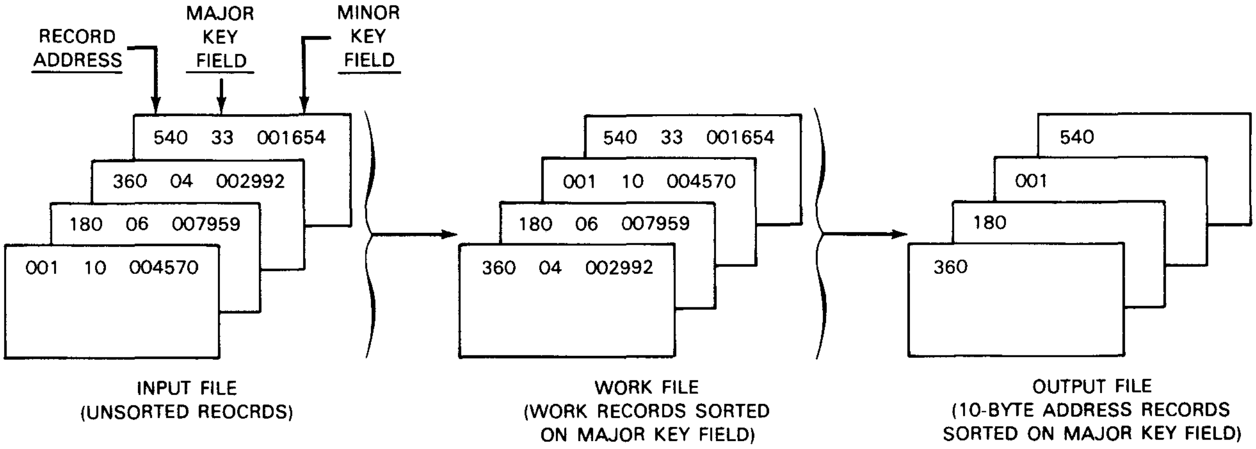


Figure 10—8. Example of Address Out (ADDROUT) Sort

The output for a tag-along sort (Figure 10—9) is a file of sorted records containing the following:

1. Control fields and data

2. Control fields only

3. Data only

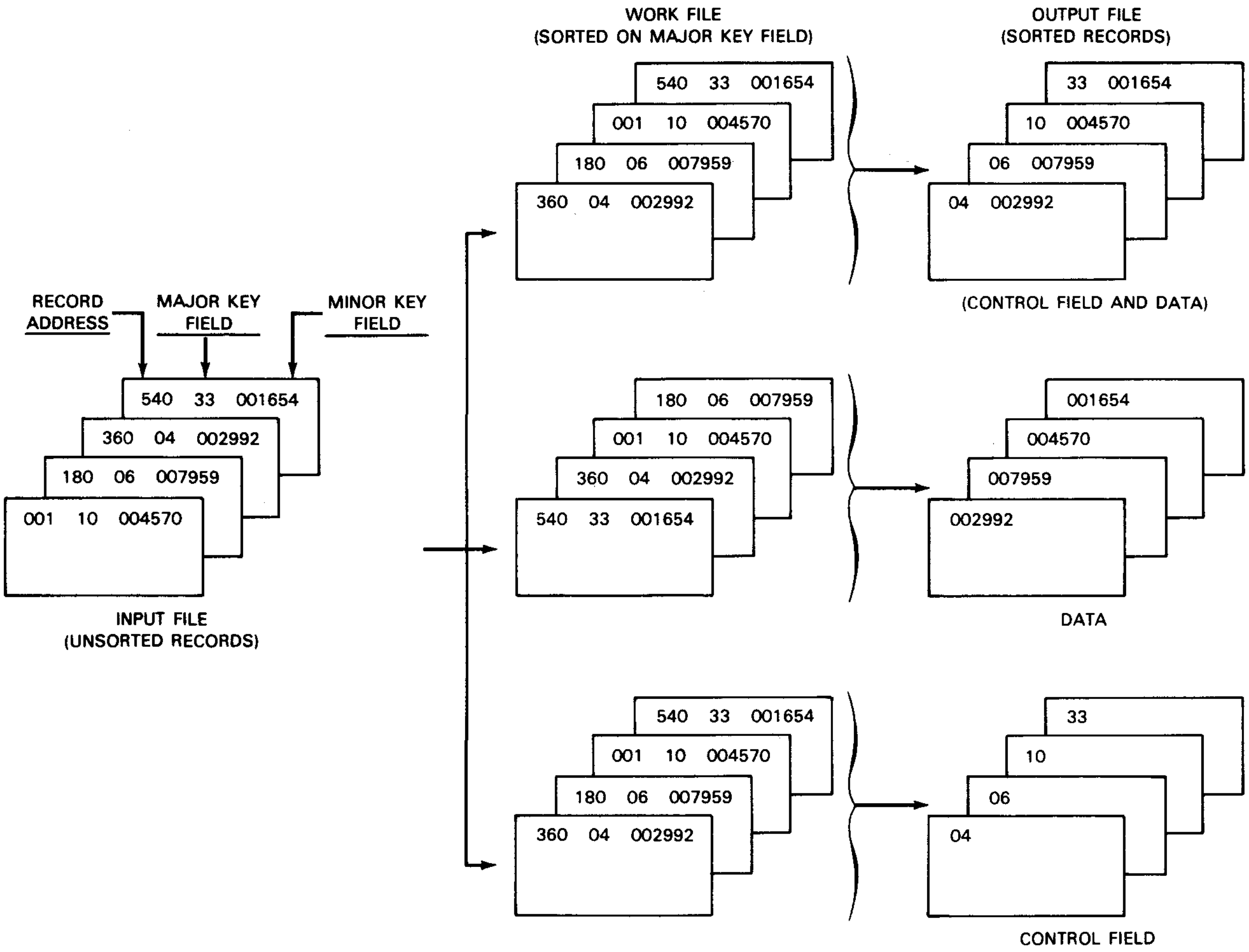


Figure 10—9. Example of a Tag-Along Sort

The output for a summary tag-along is a file of sorted records containing the following:

1. Control fields, data fields, and summary data
2. Control fields only
3. Data fields and summary data
4. Data fields only
5. Summary data fields only
6. Control fields and summary data fields

10.6. RUNNING SORT3 FROM A WORKSTATION

OS/3 provides you with the capability of running sort jobs that use the SORT3 program interactively from a workstation. The procedures used to build and prefile the control stream for a SORT3 job closely parallel those applicable to independent sort/merge. (See 3.5.)

If a job has been initiated from a workstation, all messages will be displayed on the workstation rather than the system console. This includes those messages that you have specified to be printed on the system printer. (See 11.3.3.1.)

11. System/3, 32, and 34 Compatible Sort Requirements You Supply

11.1. GENERAL

To run a SORT3 program, you are responsible for:

- Identifying your job to the system
- Assigning the devices needed for the sort
- Initiating execution of the sort program
- Defining the criteria for the sort

The first three items in our list of responsibilities are achieved through the use of control statements in the job stream for SORT3. The last item is accomplished by a set of sort specifications also included in the job stream. The detail involved in the preparation of the control statements and sort specifications depends upon the complexity of the sort, the system configuration in which you run the job and the size and format of your input files, to name a few.

Preparation of the sort specifications will probably be the largest task in setting up your job. As you know, these specifications define the criterion governing performance of the sort. However, SORT3 simplifies this requirement by accepting the same sort specifications that you used in the System/3, 32, or 34 environment, namely, the header, record type, and field description sequence specifications. The format for these specifications remains the same whether you use OS/3 job control or System/3 OCL to run SORT3. Your responsibility is to prepare the sort specifications (as described in 11.3) and include them as part of your control stream input.

The same does not hold true for the control statements needed to run your job. If you use OS/3 job control, the control statements in your control stream must conform to the OS/3 JCL conventions as defined in the job control user guide, UP-8065 (current version). If you use the OCL processor to run your job, the control statements appearing in your control stream must conform to the System/3 OCL conventions defined in the System/3 to OCL transition user guide, UP-8379 (current version). The advantage of having both methods is that you can use your existing control streams. For example, the user who has a control stream for running his sort job in a System/3, 32, or 34 environment can run that same control stream in an OS/3 environment by use of the OCL processor. For those using OS/3 job control, prepare your control statements and submit your job as you would for any other OS/3 job.

Typical job control streams for executing SORT3 under OS/3 JCL and the OCL processor are shown in Figures 11—1 and 11—2, respectively.

These job control streams can also be created and executed from a workstation. (See 3.5.)

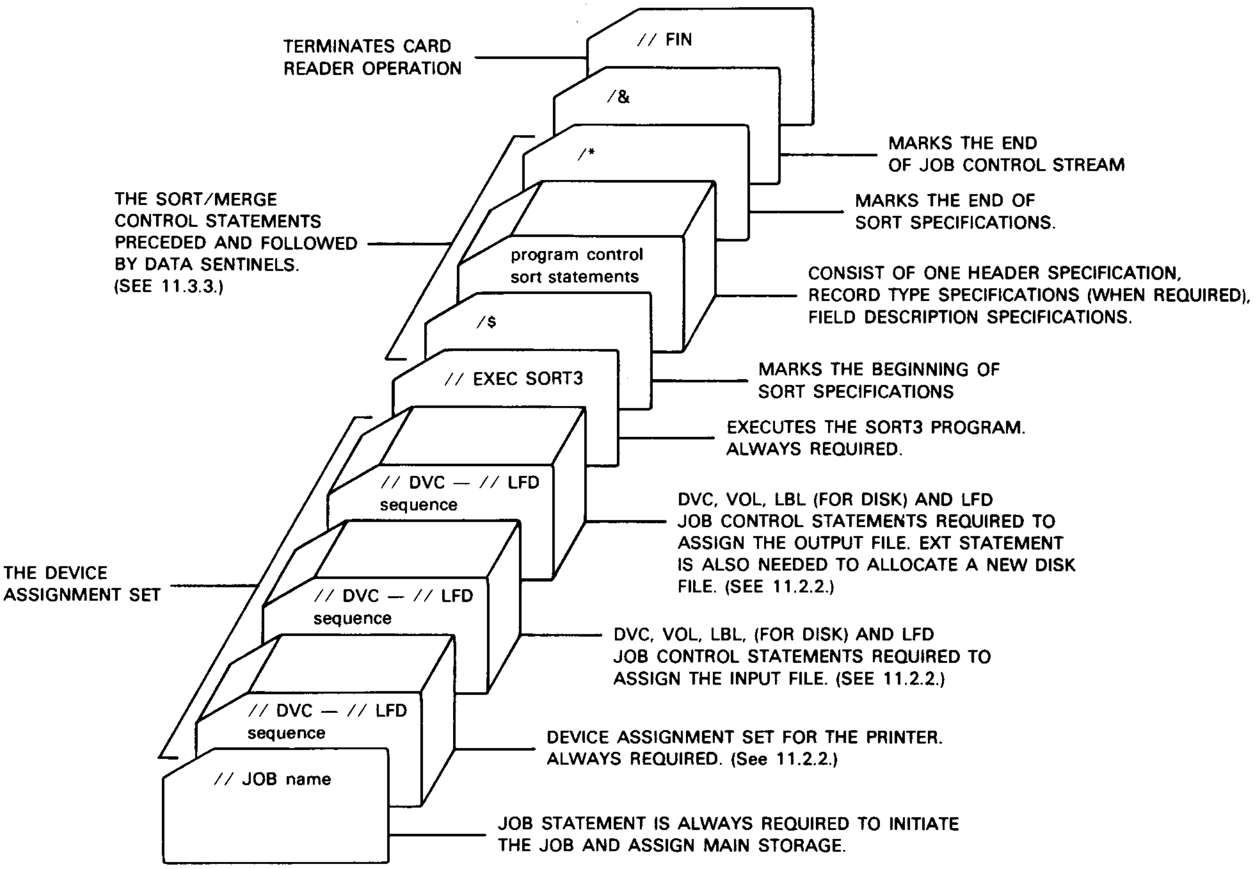

// FIN
TERMINATES CARD READER OPERATION
/&
MARKS THE END OF JOB CONTROL STREAM
/*
MARKS THE END OF SORT SPECIFICATIONS.
THE SORT/MERGE CONTROL STATEMENTS PRECEDED AND FOLLOWED BY DATA SENTINELS. (SEE 11.3.3.)
program control sort statements
CONSIST OF ONE HEADER SPECIFICATION, RECORD TYPE SPECIFICATIONS (WHEN REQUIRED), FIELD DESCRIPTION SPECIFICATIONS.
/$
MARKS THE BEGINNING OF SORT SPECIFICATIONS
// EXEC SORT3
EXECUTES THE SORT3 PROGRAM. ALWAYS REQUIRED.
// 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 11.2.2.)
THE DEVICE ASSIGNMENT SET
// DVC — // LFD sequence
DVC, VOL, LBL, (FOR DISK) AND LFD JOB CONTROL STATEMENTS REQUIRED TO ASSIGN THE INPUT FILE. (SEE 11.2.2.)
// DVC — // LFD sequence
DEVICE ASSIGNMENT SET FOR THE PRINTER. ALWAYS REQUIRED. (See 11.2.2.)
// JOB name
JOB STATEMENT IS ALWAYS REQUIRED TO INITIATE THE JOB AND ASSIGN MAIN STORAGE.

Figure 11—1. Typial Job Control Stream for Executing SORT3 under OS/3 Job Control

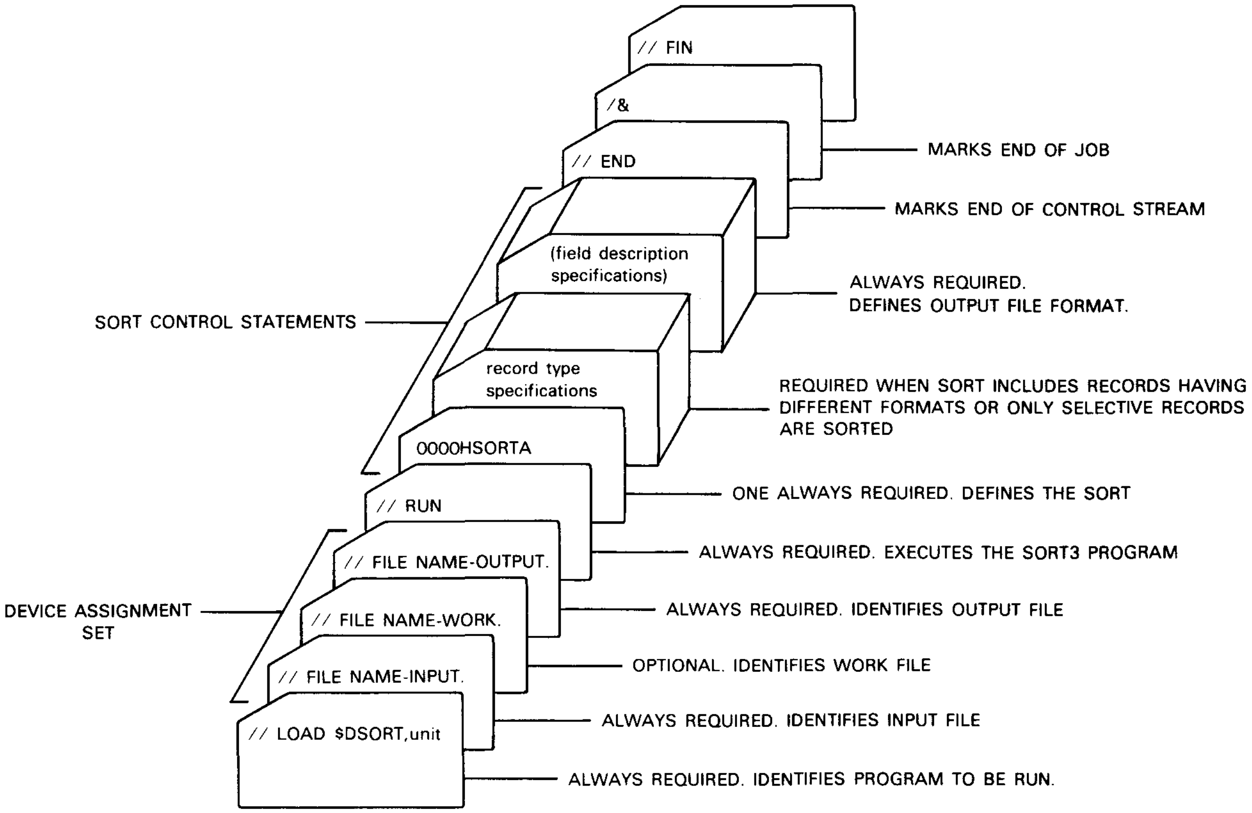

// FIN
/&
// END
(field description specifications)
record type specifications
OOOOHSORTA
// RUN
// FILE NAME-OUTPUT.
// FILE NAME-WORK.
// FILE NAME-INPUT.
// LOAD $DSORT,unit
SORT CONTROL STATEMENTS
DEVICE ASSIGNMENT SET
MARKS END OF JOB
MARKS END OF CONTROL STREAM
ALWAYS REQUIRED. DEFINES OUTPUT FILE FORMAT.
REQUIRED WHEN SORT INCLUDES RECORDS HAVING DIFFERENT FORMATS OR ONLY SELECTIVE RECORDS ARE SORTED
ONE ALWAYS REQUIRED. DEFINES THE SORT
ALWAYS REQUIRED. EXECUTES THE SORT3 PROGRAM
ALWAYS REQUIRED. IDENTIFIES OUTPUT FILE
OPTIONAL. IDENTIFIES WORK FILE
ALWAYS REQUIRED. IDENTIFIES INPUT FILE
ALWAYS REQUIRED. IDENTIFIES PROGRAM TO BE RUN.

Figure 11—2. Typical Job Control Stream for Executing SORT3 under OS/3 Operational Control Language (OCL) Processor

11.2. PREPARING JOB CONTROL STATEMENTS FOR YOUR SORT

The job control statements described in this section are used to direct the system in handling your SORT3 job in an OS/3 environment. They are responsible for:

- Identifying and scheduling your job
- Assigning system resources for your job
- Defining your input, work, and output files
- Initiating the SORT3 program
- Ending the job after the sort is completed

If you are running your sort under OCL processor, refer to the System/3 to OS/3 transition user guide, UP-8379 (current version), for information concerning the format and use of the OCL control statements.

11.2.1. Identifying and Scheduling Your Job

The first statement in the job stream is the JOB statement. It assigns a unique name to your job so that the system can distinguish it from other jobs being processed concurrently with your job. The JOB statement also specifies the minimum and maximum main storage requirements (in bytes) for the job and the priority of the job. The system will not schedule your job or allocate the required system resources to it if the JOB statement is omitted. The coding of a typical JOB statement may appear as:

```
// JOB PYRLSORT,,7000,9000
```

In the coding example of the JOB statement, PYRLSORT is the 8-character alphanumeric name assigned to your job. The double comma indicates that you have elected not to assign a priority level to the job. The system in this case assumes a normal priority. The hexadecimal values 7000 and 9000 represent the minimum number of main storage bytes needed to execute the largest job step of your job, and the maximum number of main storage bytes requested (not required) to execute the largest job step of this job.

11.2.2. Assigning Devices to Your Job

The next series of job control statements that appear in the job stream are the device assignment sets. Each device assignment set consists of as few as two job control statements (DVC and LFD), or as many as five job control statements (DVC, VOL, EXT, LBL, and LFD). The device assignment sets are used for the allocation of peripheral devices needed for printing messages, inputting data, handling data during processing, and collecting output data. They also identify the device type used, disk or tape volume mounted, and the files to be processed. Each device assignment set begins with a DVC statement that specifies the logical unit number for the device type upon which a particular file is mounted and ends with an LFD statement that associates a logical file name with that device. Detailed information about the device assignment statements and a list of specific I/O device numbers are provided in the job control user guide, UP-8065 (current version).

The first device that must be assigned for the sort job is the printer. SORT3 requires this device to print messages for operator action or information. The coding used to assign the printer may appear as:

```
// DVC 20 // LFD PRNTR
```

In this example, the printer to be assigned to your job is logical device 20. It must be assigned the system standard name PRNTR in the LFD statement.

Following the printer assignment set are the assignment sets for the input, work, and output files. The pattern of each set is similar. That is, the specifications for each file identify a device, a file on a volume, and a logical file name.

For example:

```
// DVC 65 // VOL SYS200 // LBL PAYROLL // LFD INPUT
// DVC 66 // VOL SCR200 // LBL $SCR1 // LFD DM01
// DVC 65 // VOL SYS200 // EXT SQ,C,CYL,5 // LBL EXEMPT // LFD OUTPUT
```

In this example, the first DVC statement assigns device number 65 to your input file named PAYROLL. The second DVC statement assigns device 66 to a temporary work or sort scratch file named $SCR1. The third DVC statement also assigns device number 65 to your output file EXEMPT. Unless your input files are very low volume, it is advisable to assign one device for each sort work file and another device for your input and output files. The sort operates more efficiently when one work file is assigned per device.

The VOL statements uniquely identify the volumes mounted on the devices you have assigned. The input and output files mounted on device 65 are on volume SYS200 and the work file is on volume SCR200.

By specifying the EXT job control statement in a device assignment set, you can provide disk space for sort work files, designate information needed to create new files, or extend existing disk files. Each EXT statement applies to the volume specified on the immediately preceding VOL statement. In the example, the EXT statement is specified for the output file to be created.

The LBL statements provide data management with the file identifier used to locate your file on the specified volume. Only one LBL statement is allowed per device assignment set. In the coding example, PAYROLL is the file identifier for the input file, $SCR1 is the identifier for the work file, and the EXEMPT is the identifier for the output file.

To associate the file information in your job control stream with the data management file definition, you must assign a logical or internal file name to each file. Logical file names are assigned via the LFD statement. For the SORT3 program, you must use the system standard names INPUT or INPUT1 through INPUT8 for the input file and OUTPUT for the output file. (You may assign a maximum of eight input files to your job providing they all contain the same size records.) The LFD statements for sort work files must specify the system standard names DM01 through DM03 or $SCR1 through $SCR3, in consecutive order starting with DM01 or $SCR1. Therefore, the LFD statement for the work file in the coding example is DM01.

Although the example shown uses disk devices exclusively for input, work, and output files, you are not limited to disk for these files. In addition to disk, input files may reside on card, magnetic tape, and diskette; work files can be on magnetic tape; and output files can be written to magnetic tape and diskette. (SORT3 supports the following types of input files: punch card, magnetic tape, diskettes, nonindexed and sequential nonindexed disk, and IRAM. Output files supported by SORT3 are magnetic tape, diskette, nonindexed and sequential nonindexed disk, and IRAM. SORT3 does not support ISAM files.) If the input file resides on a device other than disk and the output file is written to a disk file, then the output file is an IRAM file. If both the input file and the output file are on disk, the output file is the same type as the input file.

11.2.3. Initiating the Execution of the SORT3 Program

The // EXEC job control statement in your job control stream initiates the execution of the SORT3 program. When processed, the EXEC statement causes the root phase of the SORT3 program to be loaded from the system load library (SYSLOD) into the main storage. The // EXEC statement immediately follows the device assignment sets in the job control stream. The format of the // EXEC statement for SORT3 is:

```
// EXEC SORT3
```

11.2.4. Marking the End of Your Job

So far we have provided the system with all the control information and control data needed to execute your job. Now you must mark the end of your job so that job control does not confuse it with other jobs in the control stream. (This could occur when the system finishes executing your job and queries job control for more input.) To mark the end of job, place /$ job control statement at the end of the sort specifications in your job control stream.

If no other jobs follow your job in the control stream, you'll want to terminate the card reader operation. This is accomplished by including the // FIN control statement as the last statement in your job control stream.

11.3. SORT CONTROL SPECIFICATIONS FOR YOUR JOB

To determine which modules to include in the sort, SORT3 must be instructed how to conduct the sort. Directing sort execution is accomplished through the use of sort specifications. The specifications convey:

- what type of sort to perform;
- which record types to select for sorting;
- how to format the sorted records;
- how to format the output file; and
- what information (if any) is to be printed for user/operator use. The responsibility for supplying the specifications to SORT3 falls upon you, the user.

The media used to present the specifications to the SORT3 program is punched cards; however, job streams filed (via the general editor or job control language) in $Y$JCS can also be used. The SORT3 program accepts the specifications as control data and uses the information presented in their parameter fields as the criteria from which it structures the execution of its modules to sort the records of your file. The SORT3 specifications are:

- Header
- Record type
- Field description

The SORT3 specifications always follow the // EXEC statement in your control stream. A start-of-data sentinel (/$) marks the beginning of the specifications, and an end-of-data sentinel (/*) marks the end of the specifications in the control stream.

The three SORT3 specifications used to describe the sort and the information they convey to the SORT3 program are summarized in Table 11—1.

Table 11—1. SORT3 Specifications, Type and Function

Specification Type	Purpose
Header	■ Defines the type of sort conducted ■ Defines format of the sorted file ■ Defines the system information printed
Record type	■ Defines the record types to be included or omitted from the sort
Field description	■ Defines format of the output records

11.3.1. Determining the Sort Specifications Needed

The number of sort specifications that must appear in your job control stream is based upon the answer to two questions concerning your job.

1. Are all the records contained in your input file to be sorted?

2. Do all the records to be sorted have the same format?

If the answer to both questions is yes, you can bypass the normal specifiation requirement of header, record type, and field description, and provide only the header and field description specifications. The reason that SORT3 does not have to be selective in record processing is because all the records are included in the sort and they are all of the same type. (SORT3 considers the job to be an implied, include-all record type sort.) On the other hand, SORT3 must be selective in its record processing whenever the answer to either or both questions is no. Under these circumstances, you must identify the specific record types you want included in or omitted from the sort. Therefore, a record type specification must be included for each record type involved in the sort. The general rules for determining how to include sort specifications in your job control stream are:

1. One header specification is required for every sort job and it is always the first specification in the sequence.

2. A record type specification is required whenever the sort is not to include every record in your file or the records selected for the sort have different formats. Under these circumstances, a record type specification is required for each type of record.

3. Record type specifications are paired with field description specifications. A field description specification must be provided for each record type specification appearing in the job control stream.

4. Each field description specification immediately follows its associated record type specification.

5. The paired record type and field description specifications follow the header specification in the control stream.

The requirements for providing sort specifications and the sequence in which they must appear in your job control stream are summarized in Table 11—2.

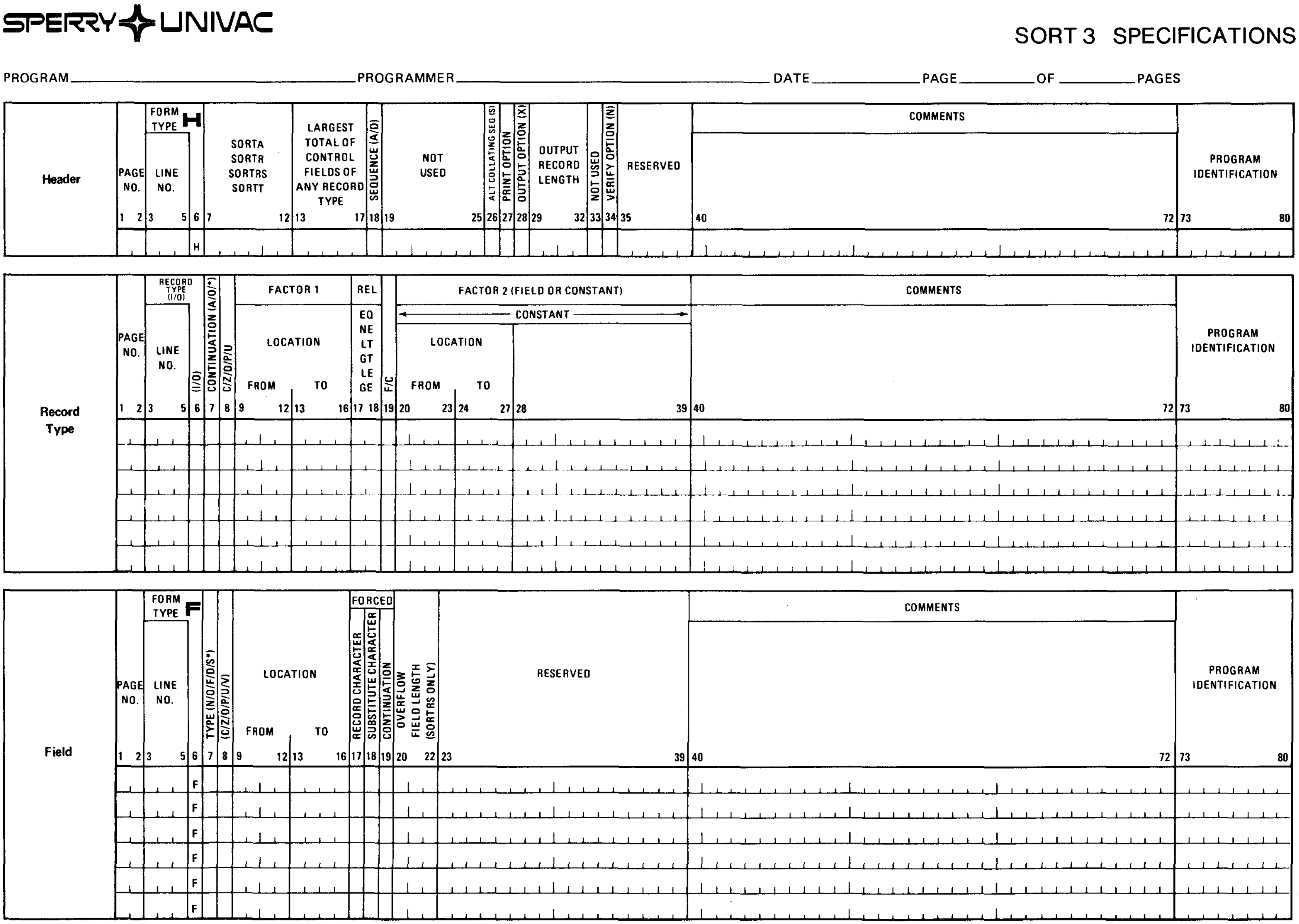
Table 11—2. Conditions Governing SORT3 Specification Requirements

Record Format	Number of Records to Be Sorted	Sequence Specifications Required (Arrange in Order Listed)
Same for all records to be sorted	All	1. Header 2. Field description
	Not all	1. Header 2. Record type 3. Field description
Several different formats for the records to be sorted	All or specified number from the file	First Record Type Format: 1. Header 2. Record type 3. Field description Second and Each Subsequent Type Format: 1. Record type 2. Field description

11.3.2. Numbering Your Sort Specifications

It is not required that you number the sort specifications appearing in your job control stream. However, numbering does avoid the possibility of getting the specifications out of sequence if you should accidentally drop or mix up the card deck containing the specifications. When used, every sort specification in your job control stream must be numbered so the SORT3 program can determine if a specification is out of order or if the entire specification sequence is arranged in a descending order. Because SORT3 is designed to process sort specifications numbered in ascending sequence, either of the other two conditions mentioned causes the program to issue a warning message to the operator and to halt the sort. The sort remains halted until the operator instructs SORT3 to continue processing or to terminate the job. To avoid this problem, make ceratin that each sort specification is properly numbered and that the entire sequence of specifications is arranged in ascending order by those numbers.

What constitutes a properly numbered sort specification? To help understand sort specification numbering, refer to the SORT3 Specifications form shown in Figure 11—3. The SORT3 Specification form is designed so each page of the form contains the facilities for specifying one header specification, one record type specification, and one field description specification. As you can see, the field columns of the specifications correspond to the columns of the punched cards used in your control stream. The purpose of the form is to provide you with an easy method of organizing and defining the sort specifications applicable to your sort. When you are satisified that the specifications are properly defined and are arranged in the order you want them processed, transferring them to the punched cards becomes a simple matter.



SPERRY UNIVAC

SORT 3 SPECIFICATIONS

PROGRAM ________ PROGRAMMER ________ DATE ________ PAGE ______ OF ______ PAGES

		FORM TYPE	H													
Header	PAGE NO.	LINE NO.		SORTA SORTR SORTRS SORTT	LARGEST TOTAL OF CONTROL FIELDS OF ANY RECORD TYPE	SEQUENCE (A/D)	NOT USED	ALT COLLATING SEQ (S)	PRINT OPTION	OUTPUT OPTION (X)	OUTPUT RECORD LENGTH	NOT USED	VERIFY OPTION (N)	RESERVED	COMMENTS	PROGRAM IDENTIFICATION
	1 2	3 5	6	7 12	13 17	18	19 25	26	27	28	29 32	33	34	35	40 72	73 80
			H													

		RECORD TYPE (I/O)				FACTOR 1		REL		FACTOR 2 (FIELD OR CONSTANT)			COMMENTS	PROGRAM IDENTIFICATION
										← CONSTANT →				
						LOCATION				LOCATION				
Record Type	PAGE NO.	LINE NO.	(I/O)	CONTINUATION (A/O/*)	C/Z/D/P/U	FROM	TO	EQ NE LT GT LE GE	F/C	FROM	TO			
	1 2	3 5	6	7	8	9 12	13 16	17 18	19	20 23	24 27	28 39	40 72	73 80

		FORM TYPE	F					FORCED					COMMENTS	
						LOCATION								
Field	PAGE NO.	LINE NO.		TYPE (N/O/F/D/S*)	(C/Z/D/P/U/V)	FROM	TO	RECORD CHARACTER	SUBSTITUTE CHARACTER	CONTINUATION	OVERFLOW FIELD LENGTH (SORTRS ONLY)	RESERVED		PROGRAM IDENTIFICATION
	1 2	3 5	6	7	8	9 12	13 16	17	18	19	20 22	23 39	40 72	73 80
			F											
			F											
			F											
			F											
			F											
			F											

Figure 11—3. SORT3 Specifications Form

Usually one page is sufficient to define all of the specifications needed to describe the sort to the SORT3 program. If necessary, specifications can be continued on subsequent pages. The important thing is to keep the pages arranged in ascending sequence. Therefore, every page containing specifications for your job must be assigned a 2-digit page number. Because the page number is applicable to every specification line appearing on the page, the number is also entered in the Page No. field (columns 1—2) of each specification line. For example, the first page is number 01; all the specification lines on that page are preceded with an 01 in columns 1 and 2. Subsequent pages would be numbered 02, 03, and so on.

The specification lines on each page are also numbered in ascending sequence. A 3-digit number specified in columns 3—5 (Line No. field) is used to identify each line. Make certain you define the specifiations in the order in which they are to be procesed by SORT3. It is suggested that you place a zero in column 5 of the Line No. field or leave this column of the field blank to allow you the capability of inserting additional or out-of-sequence specifications. This method of numbering eliminates renumbering existing specification lines whenever an insertion is required. To illustrate this, suppose you have used three lines to define a record type and have numbered the lines 010, 011, and 012. At this point you realize you omitted a line of the record type specification that should have been defined second in the sequence. Your numbering scheme leaves no room for insertion; therefore, you are forced to renumber all of the existing specification lines. Had your specification lines been numbered 010, 020, and 030, inserting the out-of-sequence specification line could have been accomplished simply by assigning it a number greater in value than 010 and less than 020. As you can see, any value from 1 to 9 assigned to column 5 will properly sequence the inserted line. Figure 11—4 illustrated sort specification page and line numbering. Specification line and page numbering is important because SORT3 compares the 5-digit number formed by the entries in the Page No. and Line No. fields as the specifications are read. Improperly sequenced sort specifications will terminate the job.

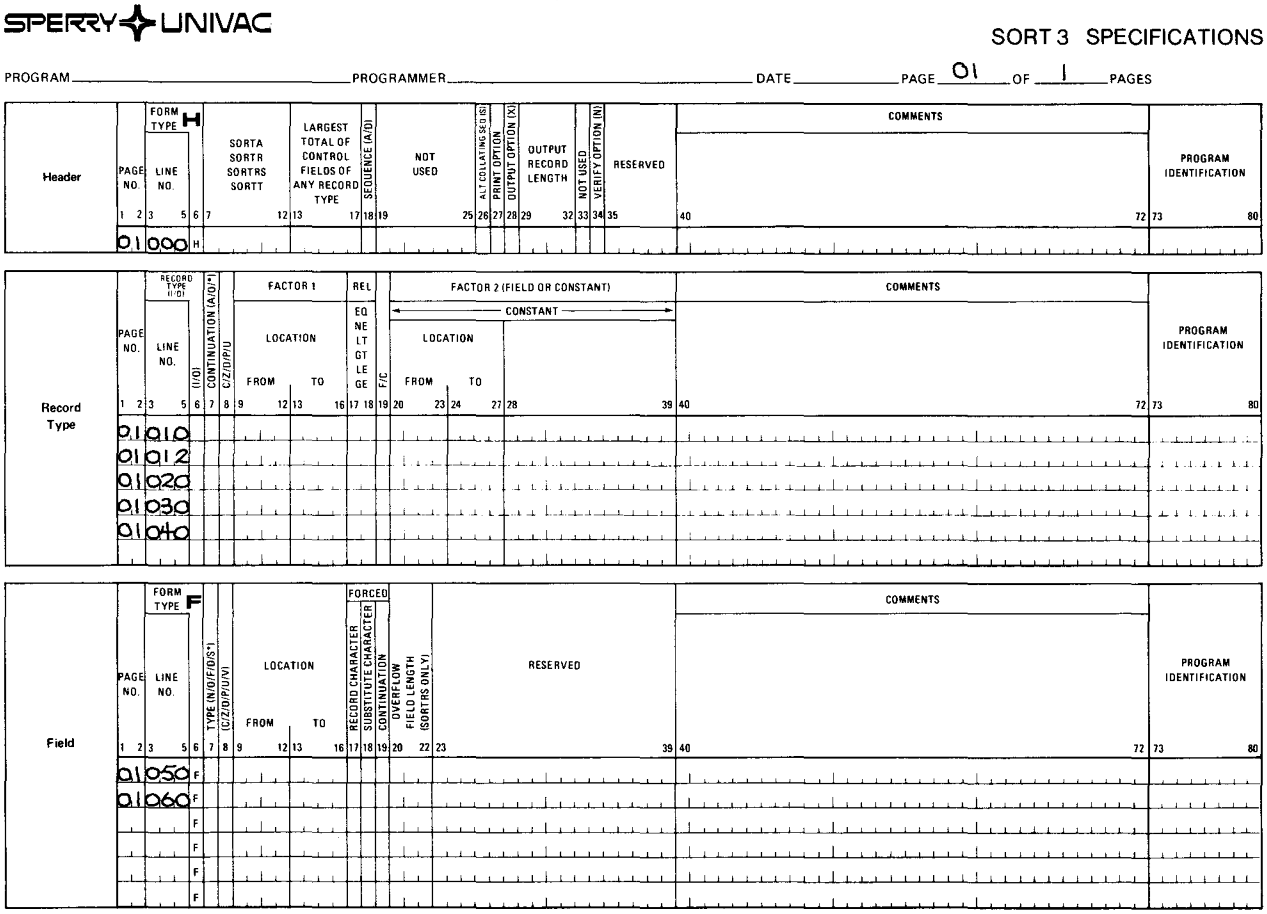
One other thing you must be concerned with when numbering sort specifications is the page and line numbers assigned to the header specification. The header specification must always be the first sort specification processed in the sequence. Therefore, it is always defined on page 01 and is given the line number 000.

11.3.3. Preparing the Sort Specifications

Now that you have determined which sort specifications are required for your job, it is time to define the specifications in a form recognizable to the SORT3 program: 80-column card format.

11.3.3.1. Header Specification

The first sort specification that you must prepare is the header specification. This specification allows you to identify the type of sort you want performed and to identify the criteria for formatting the output (sorted) file. Figure 11—5 shows the format of the header specification for each type of sort performed by the SORT3 program. The shaded areas identify the fields that you must consider when preparing the specifications.



SPERRY UNIVAC

SORT 3 SPECIFICATIONS

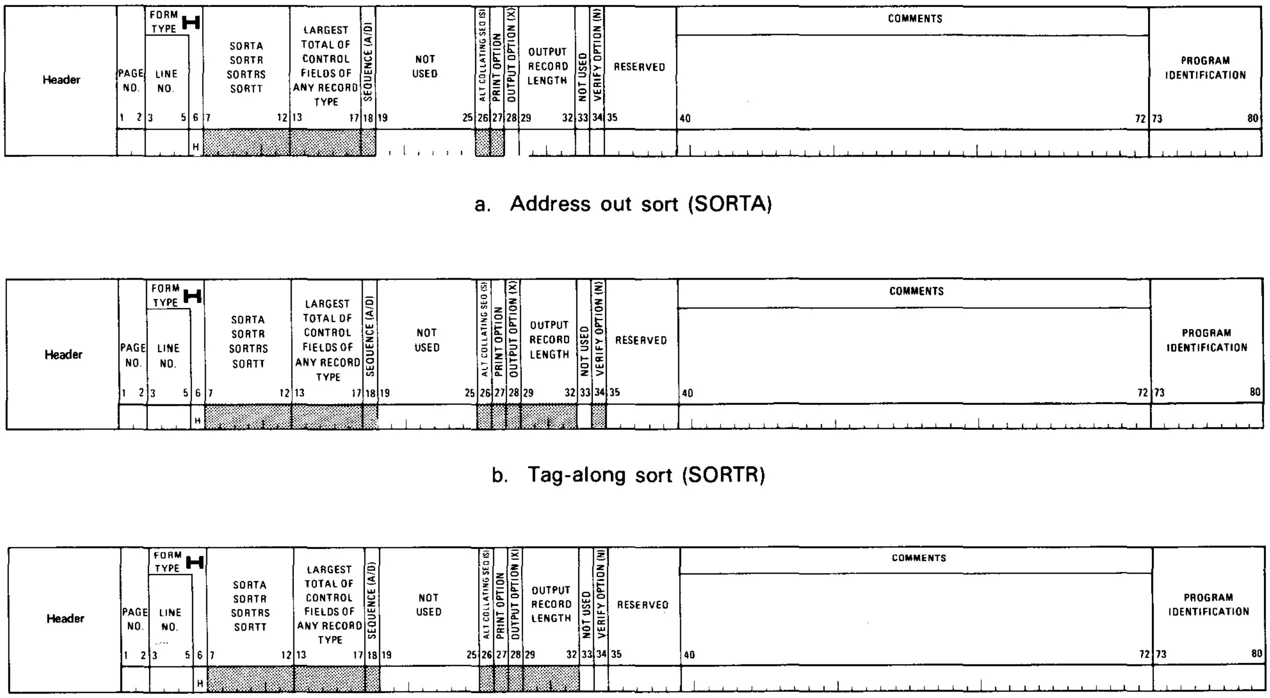
PROGRAM ______ PROGRAMMER ______ DATE ______ PAGE 01 OF 1 PAGES

	PAGE NO.	LINE NO.	FORM TYPE H	SORTA SORTR SORTRS SORTT	LARGEST TOTAL OF CONTROL FIELDS OF ANY RECORD TYPE	SEQUENCE (A/D)	NOT USED	ALT COLLATING SEQ (S)	PRINT OPTION	OUTPUT OPTION (X)	OUTPUT RECORD LENGTH	NOT USED	VERIFY OPTION (N)	RESERVED	COMMENTS	PROGRAM IDENTIFICATION
Header	1 2	3 5	6	7 12	13 17	18	19 25	26	27	28	29 32	33	34	35	40 72	73 80
	01	000	H													

	PAGE NO.	LINE NO.	RECORD TYPE (I/O)	CONTINUATION (A/O/*)	C/Z/D/P/U	FACTOR 1 LOCATION FROM	FACTOR 1 LOCATION TO	REL EQ NE LT GT LE GE	F/C	FACTOR 2 (FIELD OR CONSTANT) LOCATION FROM	FACTOR 2 LOCATION TO	CONSTANT	COMMENTS	PROGRAM IDENTIFICATION
Record Type	1 2	3 5	6	7	8	9 12	13 16	17 18	19	20 23	24 27	28 39	40 72	73 80
	01	010												
	01	012												
	01	020												
	01	030												
	01	040												

	PAGE NO.	LINE NO.	FORM TYPE F	TYPE (N/O/F/D/S*)	C/Z/D/P/U/V	LOCATION FROM	LOCATION TO	FORCED RECORD CHARACTER	FORCED SUBSTITUTE CHARACTER	CONTINUATION	OVERFLOW FIELD LENGTH (SORTRS ONLY)	RESERVED	COMMENTS	PROGRAM IDENTIFICATION
Field	1 2	3 5	6	7	8	9 12	13 16	17	18	19	20 22	23 39	40 72	73 80
	01	050	F											
	01	060	F											
			F											
			F											
			F											
			F											

Figure 11—4. Numbering Sort Specifications



	PAGE NO.	LINE NO.	FORM TYPE H	SORTA SORTR SORTRS SORTT	LARGEST TOTAL OF CONTROL FIELDS OF ANY RECORD TYPE	SEQUENCE (A/D)	NOT USED	ALT COLLATING SEQ (S)	PRINT OPTION	OUTPUT OPTION (X)	OUTPUT RECORD LENGTH	NOT USED	VERIFY OPTION (N)	RESERVED	COMMENTS	PROGRAM IDENTIFICATION
Header	1 2	3 5	6	7 12	13 17	18	19 25	26	27	28	29 32	33	34	35	40 72	73 80
			H	▨	▨	▨		▨	▨							

a. Address out sort (SORTA)

	PAGE NO.	LINE NO.	FORM TYPE H	SORTA SORTR SORTRS SORTT	LARGEST TOTAL OF CONTROL FIELDS OF ANY RECORD TYPE	SEQUENCE (A/D)	NOT USED	ALT COLLATING SEQ (S)	PRINT OPTION	OUTPUT OPTION (X)	OUTPUT RECORD LENGTH	NOT USED	VERIFY OPTION (N)	RESERVED	COMMENTS	PROGRAM IDENTIFICATION
Header	1 2	3 5	6	7 12	13 17	18	19 25	26	27	28	29 32	33	34	35	40 72	73 80
			H	▨	▨	▨		▨	▨	▨	▨		▨			

b. Tag-along sort (SORTR)

	PAGE NO.	LINE NO.	FORM TYPE H	SORTA SORTR SORTRS SORTT	LARGEST TOTAL OF CONTROL FIELDS OF ANY RECORD TYPE	SEQUENCE (A/D)	NOT USED	ALT COLLATING SEQ (S)	PRINT OPTION	OUTPUT OPTION (X)	OUTPUT RECORD LENGTH	NOT USED	VERIFY OPTION (N)	RESERVED	COMMENTS	PROGRAM IDENTIFICATION
Header	1 2	3 5	6	7 12	13 17	18	19 25	26	27	28	29 32	33	34	35	40 72	73 80
			H	▨	▨	▨		▨	▨	▨	▨					

c. Summary sort (SORTRS)

Figure 11—5. Header Specification Formats

Selecting the type of sort is accomplished by entering one of four sort names into the field formed by columns 7 through 12 of the specification. Although four sort names are provided for your selection, only three type sorts are possible: address out (SORTA), tag-along (SORTR), and tag-along summary (SORTRS). The SORTT entry is provided strictly for System/3 compatibility and, when specified, produces a tag-along sort the same as the SORTR entry. SORT3 will not execute if you omit this field from the specification. The sort type entry must be left-justified in the field.

In addition to telling SORT3 which sort to perform, you must provide control field and record sequencing information to the program. The control field information (specified in columns 13—17) tells SORT3 how large a buffer it must provide to accommodate the control fields used in sorting your input records. You can specify anything up to 256 bytes. But rather than arbitrarily assigning a value to this field, you can compute the number of bytes needed by totaling the lengths of the control fields for each record type involved in the sort. (Control field types are discussed under column 7 of the field description specification (11.3.3.3). Normal (N) control fields, opposite (O) control fields, and forced (F) control fields are included in this calculation.) The largest total is the value entered in the field. For example, three record types are described for your sort. The total length of the control fields for the first record type is 10 bytes; the total length for the second reocrd type is 12, and the total for the third record type is 8. A buffer size of 12 bytes can accommodate all three, so this is the value you would specify in columns 13—17. The entry must be right-justified in the field.

Record sequencing refers to how the SORT3 program sequences the records in the sorted output file. You can specify either ascending or descending by entering an A or D, respectively, in column 18.

With the exception of the Output Record Length field (columns 29—32), the remaining fields of the header specification need only be specified if you do not want the program's standard default options. For example, SORT3 uses a standard collating sequence for the sort. If you want to specify an alternate sequence, you must specify the character S in column 26. Of course, you must define characteristics of the alternate collation. This requires you to prepare ALTSEQ statements and place them immediately after the header specification in your job control stream. (ALTSEQ specifications are described in 11.3.4.) When you specify an alternate collating sequence, make certain you do not use a packed (P) or unpacked (U) Factor 1 in the record type specifiation. Otherwise, the proper records may not be included in or omitted from your output file. The reason is that the ALTSEQ specifications used to define your alternate collating sequence change the Factor 1 fields. This change may affect the unit position and sign of an unpacked decimal number or any one position of a packed decimal number. If it does, the basis of selecting records for the sort is unpredictable.

The Print Option field (column 27) is also optional. Normally, SORT3 prints (on the system printer) and displays (on the system console) sequence specifications, diagnostic messages, program status messages, action messages, and other system messages. This is the default case. You can limit or inhibit this service by specifying a 1, 2, or 3 in column 27. A 1 causes only program status messages, action messages, and other system messages to be printed and displayed. A 2 causes only action messages and other system messages to be printed and displayed. A 3 causes only other system messages to be printed and displayed. Note that if a job is initiated from a workstation, messages will be displayed on the workstation rather than the system console.

If you requested one of the tag-along sorts for your job (SORTR, SORTRS, or SORTT), you can have SORT3 drop control fields from the sorted output records by entering an X in column 28. (Control fields are normally dropped when opposite fields or an alternating collating sequence is specified. In both cases, SORT3 changes the control information during the sort so it is meaningless as data. If, under these circumstances, you want to retain the control information in the output record and keep it in a meaningful form, you must define the control field twice: once as control fields and once as data fields.

Previously, it was stated that the Output Record Length field (columns 29—32) was a required field. This is true whenever one of the tag-along sorts is to be performed. The entry in this field depends on whether control fields are dropped from the sorted output records (X in column 28). If the control fields are dropped, the length specified includes only data fields. The calculation is similar to that for columns 13—17; total the length of the data fields in each record type in the sort. Enter the largest value right-justified in the field. If control fields are retained in the output records, total the length of the data fields for each record type in the sort and add the largest value to the value entered in columns 13—17. Enter the sum right-justified into columns 29—32. Under both conditions, the value entered must not exceed 4096 bytes.

The VERIFY option (column 34) can be used to improve program performance (throughput) by requesting SORT3 not to verify the data written on the work files during the sort. If this field is blank, data verification is performed automatically. To inhibit this feature, enter an N in the field.

Columns 40 through 80 have no effect on program function. They are provided for your comments and program identification. They can be printed out whenever the Print Option field (column 27) is specified as 0 or left blank.

A summary of the field entries for the header specification is provided in Table 11—3.

Table 11—3. Column Summary for Header Specification (Part 1 of 2)

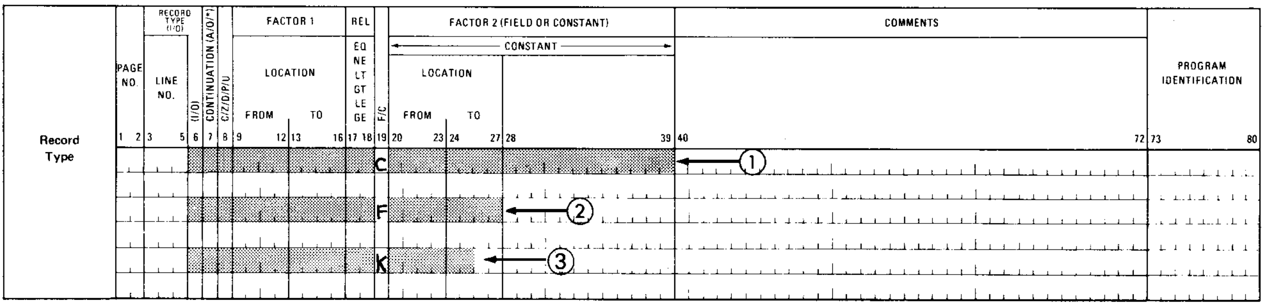
Columns	Entries	Explanation
1—2	00	Page number
3—5	000	Header number
6	H	Header identification
7—12	SORTA	Addrout sort job (disk files only)
	SORTR	Tag-along sort job
	SORTRS	Summary tag-along sort job
	SORTT	Tape sort (for System/3 compatibility only)
13—17	1—256	Longest control field used in sorting the reocrds (right-justified)
18	A	Records in sorted file to be in ascending order by control fields
	D	Records in sorted file to be in descending order by control fields
19—25	Blank	Not used

Table 11—3. Column Summary for Header Specification (Part 2 of 2)

Columns	Entries	Explanation
26	Blank	Use standard OS/3 collating sequence in compare operations.
	S	Use an alternate collating sequence in compare operations. ALTSEQ statements will define the collating sequence to be used.
27	0 or Blank	Print and display: Sequence specifications Diagnostic messages Program-status messages Action messages Other system messages
	1	Print and display: Program-status messages Action messages Other system messages
	2	Print and display action messages and other system messages.
	3	Print and display other system messages only.
28	Blank	Keep control fields in output records in tag-along sort job.
	X	Drop control fields from output records in tag-along sort job.
29-32	1-4096	Length of output records in tag-along sort job (right-justified)
33	Blank	Not used
34	N	Data written on the work file will not be verified.
35-39	Blank	Reserved
40-80	Blank or any OS/3 characters	Not used by SORT3. May be used for comments or program identification

11.3.3.2. Record Type Specification

The next sort specifications you must prepare for your job are the record type specifications. Record type specifications are used for defining the types of input records SORT3 is to include in or omit from the sort. Of course, there is no need to prepare record type specifications if SORT3 does not have to be selective in sorting the records of your input file. For example, if every record in your input files is to be sorted and the format of each record is the same, SORT3 does not have to decide which records to include or omit. You can, therefore, omit the record type specification from your job control stream. To SORT3, the omitted specifications imply an include-all record condition for your sort. On the other hand, a sort that includes or excludes specific record types requires you to identify these record types to the SORT3 program. In this case, you must include a record type specification for each record type to be sorted. Figure 11-6 shows the format of the record type specification and the field entries you must consider for each type of sort performed by SORT3.



LEGEND:

① Format when comparison involves input record field to a constant.

② Format when comparison involves two input record fields.

③ Format when comparison involves input record field to a keyword.

Figure 11—6. Record Type Specification Format

Before getting into the specifics of preparing the specification, let's briefly discuss how SORT3 identifies records. Records are selected or omitted on the basis of a test or comparison. That is, SORT3 looks at a particlar key field or fields (control and/or data) in each record of your input file and compares the data in that field to a constant, keyword, or the data in another field of the same record. (The data you are comparing is the Factor 1 field, and the data you are comparing it against is the Factor 2 field.) The results of the comparison determine whether the record is selected or omitted from the sort.

What role do you play in this procedure for sorting records? You establish the criteria upon which SORT3 makes its decisions. For example, the information coded in your record type specifications:

- defines the length and location of the Factor 1 and Factor 2 fields used in the comparison;
- provides the constant if it is used;
- provides the keyword;
- defines how the data contained in the factor fields is to be interpreted during the comparison;
- defines what the results of the comparison must be; and
- decides whether the record type based on the results of this test is to be included in or omitted from the sort.

The decision to include or omit the record type defined in the specification is based on your entry in column 6 (Record Type field). An I in this field tells SORT3 to include in the sort only those records that meet the comparison requirements set forth in the specification. An O in column 6 instructs SORT3 to omit those input records that meet the comparison requirements defined in the specification.

Why have an include an omit capability? Consider a file that contains many different types of records and you want to include only a few types in the sort. You automatically exclude all records not wanted by defining only those few types you do want sorted. Now, consider a time when you want to include all but a few record types in the sort. Rather than providing a description for each record included in the sort, you can simply describe the few records you want omitted. The remaining record types are automatically included in the sort. From this example, you can see the advantages of having both options available.

In normal practice, an omit record description is always followed by a special version on the include record description referred to as an include all record description. The include-all description is defined by entering the character I in column 6 and leaving blank the fields (columns 7—39) of the specification related to describing the record type. This tells SORT3 to include all record types in the sort not previously defined to be omitted or included. If you use the include-all version, only one can be specified per job and it must be the last record type defined for that job.

Because a record type description can extend beyond one line of code on the form, you must define the relationship of one line to another. Column 7 is used to define this relationship. A blank field tells SORT3 that this is the first line of code for the record description. An A entry states that this line of code is a continuation of the preceding line. (The A represents an AND function.) An O entry defines that the line of code applies to a different record type than the one described on the preceding line, but the field descriptions are common to both record types. (The O represents an OR function.) Comment lines are defined by an asterisk (*) in column 7. Comment lines have no effect on sort other than being printed if you have specified the Print option on the header specification.

How should SORT3 interpret the data in the Factor 1 and Factor 2 fields during compare operations? When these fields contain alphanumeric data, an entry of C, Z, or D must be entered in column 8 of the specification. The specific entry made depends upon what portion of the data you want included in the comparison. That is, alphanumeric data comprises two portions: a zone portion and a digit portion. The Z entry instructs SORT3 to use only the zone portion of the data. The D entry specifies only the digit portion and the C entry instructs SORT3 to use both portions. If the Factor 1 and Factor 2 fields contain numeric data, a P or U character must be specified in column 8. The P entry indicates the data is packed and the U entry indicates the data is unpacked. Packed numeric data always contains a sign (positive or negative) and only the digit portion of the data. Unpacked numeric data also contains a sign but includes both the zone and the digit portions of the data. As you can see, the data type, and the method of comparison used, have some influence on the length of the Factor 1 and Factor 2 fields. Table 11—4 lists the available entries for column 8 and the restrictions each places on the length of the Factor 1 and Factor 2 fields.

Table 11—4. Column 8 Entries and Their Effect on Factor 1 and 2 Field Lengths

Column Entry	Compare Operation Method	Maximum Allowable Length for Factor 1 and 2 Fields
C	Use both zone and digit portions of the bytes	256 bytes
Z	Use only the zone portion of the byte	1 byte
D	Use only the digit portion of the bytes	16 bytes
P	Packed numeric data	8 bytes or 15 digits and a sign
U	Unpacked numeric data	16 digits

The remaining fields of the specifications pertain to defining the Factor 1 and Factor 2 fields for the sort, defining the conditions for field comparisons, etc.

First, let's discuss setting up the test conditions for the comparison. You have six test conditions available to use in the comparison between the Factor 1 and Factor 2 fields. Each condition instructs SORT3 to look for a specific test result from the comparison of the two fields. The record is selected or rejected based on the results of the comparison. Columns 17 and 18 are used for defining the results of the comparison. The entries for this field and the restrictions in their use (where applicable) are given in Table 11—5.

Table 11—5. Test Relationships for Factor 1 and 2 Comparisons

Column 17—18 Entry	Test Conducted
EQ	Factor 1 field equal to Factor 2 field
NE	Factor 1 field not equal to Factor 2 field
LT	Factor 1 field less than Factor 2 field*
GT	Factor 1 field greater than Factor 2 field*
LE	Factor 1 field less than Factor 2 field*
GE	Factor 1 field greater than or equal to Factor 2 field*

*These entries are not permitted when the comparison made involves only the zone portions of the data (Z specified in column 8).

The time has arrived to discuss the Factor fields. First, let's approach the Factor 1 field (columns 9–16). SORT3 does not interpret the entry in this field as actual data, but as the location of the data within your input records. As you can see, the Factor 1 field is composed of two parts. The first part (columns 9–12) defines the position at which the data begins in the record. The second part (columns 13–16) defines where the data ends. The number of positions from one point to the other also represents the length of data. Technically, the length of the data defined in the Factor 1 field can be any number of bytes from 1 to 256. In practice, however, this length cannot exceed the length of the records in the file. In addition, the length specified in the Factor 1 field is restricted by how the data is interpreted (column 8 entry) and whenever the Factor 2 field defines a constant or keyword. The allowable field lengths and the restrictions other field specifications place on this length are defined in Table 11–6.

Table 11—6. Factor 1 Field Length Requirements

Column 8 Entry	Maximum Factor 1 Field Length (in bytes)	
C	256	When Factor 2 defines a constant, the length defined must not exceed 20. When Factor 2 defines a keyword, the length must not exceed 6.
Z	1	
D	16	
P*	8	Because the field is packed, it can actually represent 15 decimal digits and a sign.
U*	16	

*Do not use a packed or unpacked Factor 1 field if an alternate collating sequence is specified in the header specification (S in column 26).

A few rules to keep in mind when coding the Factor 1 field are:

- All entries must be right-justified (the From Location must end in column 12 and the To Location must end in column 16)
- You need not enter anything in the From Location when a Factor 1 field length of one byte is defined.

In the brief description of how SORT3 compares the Factor fields, it was stated that the data defined by the Factor 1 field was compared against a constant, keyword, or the data in another field of the record. This constant or field location is identified in the Factor 2 field (columns 20–39). Because the Factor 2 field can specify either of two types of information, you must tell the SORT3 program how to interpret the entry in the Factor 2 field. If a C is entered in column 19, the Factor 2 field contains a constant. If it contains a field position, enter an F in the column.

When the Factor 2 entry defines a field, only columns 20—27 are used. The length of the field defined must be the same as that specified for Factor 1. It must also be in the same record type as Factor 1. The purpose of the From Location and the To Location, and the rules for coding, are the same as for the Factor 1 field.

All of the columns (20—39) are used when Factor 2 is a constant. However, the rules for coding your entry depend on whether the constant is a packed or unpacked number, an alphanumeric constant, a numeric constant, or a signed constant. In general, the constant must be the same length as the Factor 1 field.

For example, if Factor 1 is a 4-position field, the constant field must take up four positions. If the constant is the number 6, enter the 6 in column 23, and either leave columns 20—22 blank or fill them with zeros.

If the Factor 1 field contains a packed number, the length of the constant (including the sign) must be twice the length of the Factor 1 field. The reason is that Factor 1 data is in packed form, and the constant is in unpacked form. When alphanumeric constants are specified (column 8 entry is C, Z, or D), the constant must be the same length as the factor 1 field and must always begin in column 20. When numeric constants are specified (column 8 entry is P, U, or D), they must be right-justified within the field length defined in Factor 1 (within twice the field length if Factor 1 is a packed number). For example, assume that Factor 1 defines a 6-position field in the input record, and that Factor 2 is the numeric constant 456. To right-justify the constant within six positions, put the constant in columns 23—25. Because leading zeros are not required (to SORT3, blanks and zeros look the same), columns 20—25 could contain either 000456 or 456 with three leading blanks.

For signed constants, the last character in the constant must be its sign (+ or —) when Factor 1 is a packed number. If Factor 1 is an unpacked number, and the constant is a negative number, the last digit in the constant must be a character that indicates both the numeric value of the last digit and the negative sign for the entire constant.

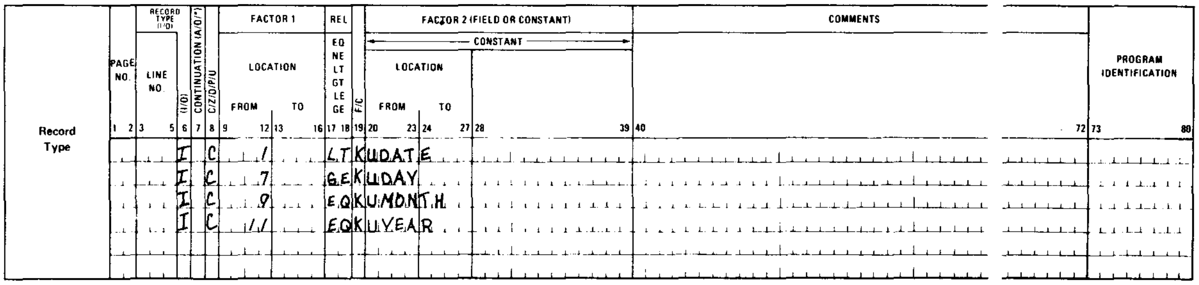
The following example shows the entries you make for records that have a packed —1 in positions 1 and 2 of the record, an unpacked —24 in positions 5 through 8 of the record, and an unpacked —10 in positions 11 through 16 of the record.

		RECORD TYPE (I/O)				FACTOR 1		REL		FACTOR 2 (FIELD OR CONSTANT)			COMMENTS		
										CONSTANT					
	PAGE NO.	LINE NO.	(I/O)	CONTINUATION (A/O/*)	C/Z/D/P/U	LOCATION FROM	LOCATION TO	EQ NE LT GT LE GE	F/C	LOCATION FROM	LOCATION TO				PROGRAM IDENTIFICATION
Record Type	1 2	3 5	6	7	8	9 12	13 16	17 18	19	20 23	24 27	28 39	40 72		73 80
			I		P	1	2	EQ	C	1-			PACKED-1		
			I	A	U	5	8	EQ	C	2M			UNPACKED-24		
			I	O	U	11	16	EQ	C		1-		UNPACKED-10		

As in the header specification, comments can be written in columns 40 through 80. They have no effect on program functions, and are printed only if the Print Option is specified in the header specification.

When the Factor 2 entry defines a keyword, the column 8 entry must be a C and the column 19 entry must be a K. The permissible Factor 2 entries for a keyword are UDATE, UDAY, UMONTH, and UYEAR. The Factor 1 field length must be 6 if UDATE is specified, and the format of the date in the record field must be the same as UDATE. If UDAY, UMONTH, or UYEAR is specified, the Factor 1 field length must be 2.

The following example shows the entries you make when you are comparing fields with keywords.



Record Type	PAGE NO. 1-2	LINE NO. 3-5	RECORD TYPE (I/O) 6	CONTINUATION (A/O/*) 7	C/Z/D/P/U 8	FACTOR 1 LOCATION FROM 9-12	FACTOR 1 LOCATION TO 13-16	REL EQ NE LT GT LE GE 17-18	F/C 19	FACTOR 2 (FIELD OR CONSTANT) LOCATION FROM 20-23	FACTOR 2 LOCATION TO 24-27	CONSTANT 28-39	COMMENTS 40-72	PROGRAM IDENTIFICATION 73-80
			I		C	1		LT	K	UDAT	E			
			I		C	7		GE	K	UDAY				
			I		C	9		EQ	K	UMON	TH			
			I		C	11		EQ	K	UYEA	R			

As in the header specification, comments can be written in columns 40 through 80. They have no effect on program functions and are printed only if the Print Option is specified in column 27 of the header specification.

A summary of the field entries for the record type specification is provided in Table 11—7.

Table 11—7. Column Summary for Record Type Specification (Part 1 of 2)

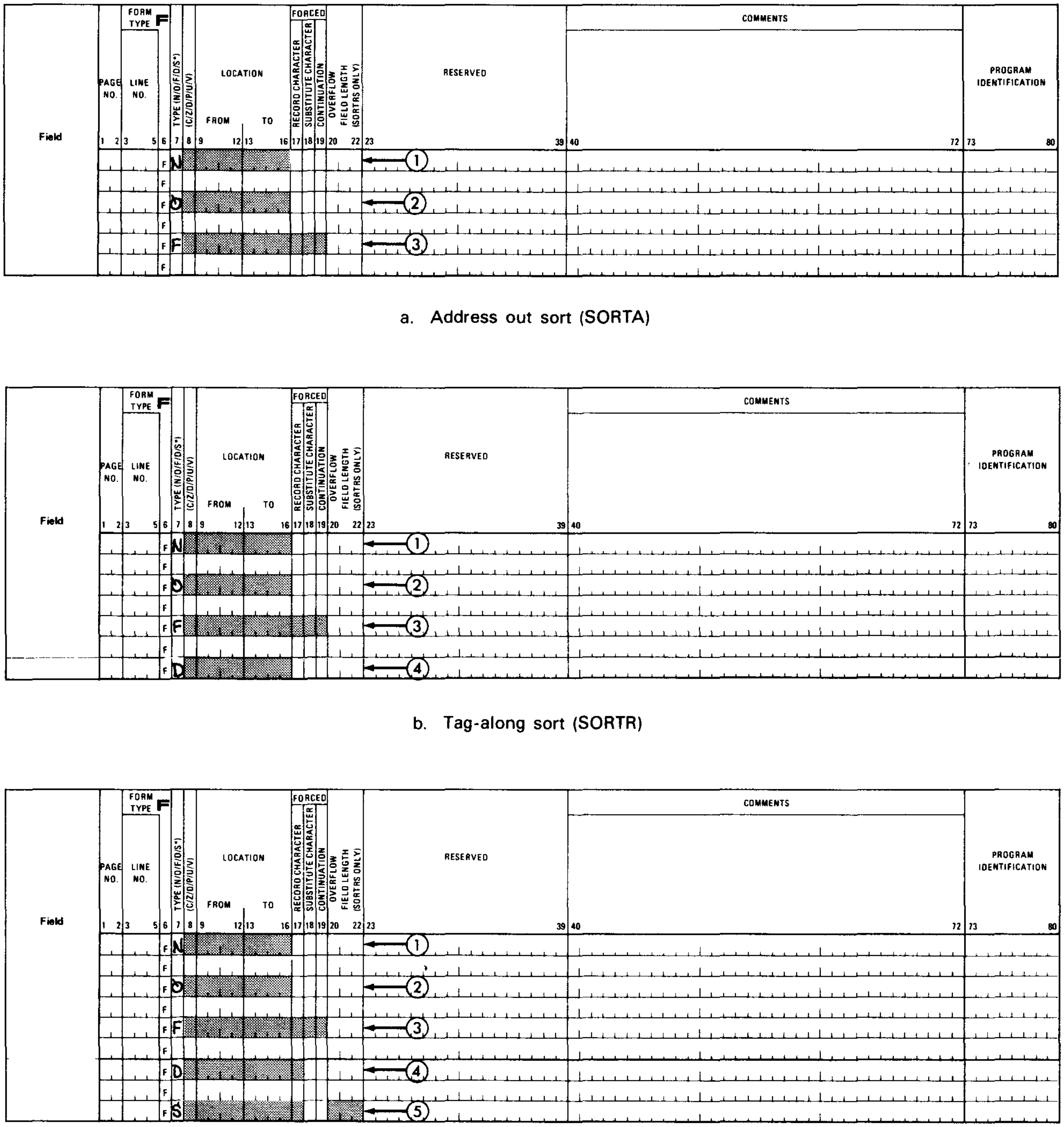
Columns	Entries	Explanation
1-2		Page number
3-5		Line number (Leave column 5 blank or enter any value to keep the specifications in ascending order.)
6	I	Include.
	O	Omit.
7	A	And (These specifications continue the definition of the record described previously.)
	O	Or (These specifications define a different type of record than the previous one.)
	Blank	First of a set of I or O record types
	*	Comment
8	C	Use both zone and digit portions of characters.
	Z	Use only zone portion of 1-character field.
	D	Use only digit portion of characters.
	P	Signed packed decimal data
	U	Signed unpacked decimal data
9-12	1-4096	The input record position in which the Factor 1 field begins (blank if field is only one position long)
13-16	1-4096	The input record position in which the Factor 1 field ends

Table 11—7. Column Summary for Record Type Specification (Part 2 of 2)

Columns	Entries	Explanation
17-18	EQ	Factor 1 must equal Factor 2.
	NE	Factor 1 must not equal Factor 2.
	LT	Factor 1 must be less than Factor 2.
	GT	Factor 1 must be greater than Factor 2.
	LE	Factor 1 must be less than or equal to Factor 2.
	GE	Factor 1 must be greater than or equal to Factor 2.
19	C	Factor 2 is a constant.
	F	Factor 2 is another field in the same input record.
	K	Factor 2 is a keyword: UDATE, UDAY, UMONTH, or UYEAR.
20-23	1-4096	The input record position in which the Factor 2 field begins (blank if field is only one position long)
24-27	1-4096	The input record position in which the Factor 2 field ends
20-39	Any characters	Factor 2 constant
40-80	Any characters	Comments or program identification

11.3.3.3. Field Description Specification

The last sort specification to be prepared is the field description specification. This specification instructs SORT3 how to format the records in the output file. For address out sorts (SORTA), the field parameters in each line describe the control fields used to sort record addresses. For tag-along sorts (SORTR and SORTT) and summary sorts (SORTRS), the field parameters in each line define the fields SORT3 uses to create the output records. Figure 11—7 shows the format of the field description specification and the fields that must be considered for each type of sort performed by SORT3. When writing records to a disk file, SORT3 normally blocks the file by a factor of eight unless this block size exceeds 1024 bytes; in which case, the block size is made as close as possible to 1024 without exceeding this value. If any record size exceeds 512 bytes, then the output file is unblocked. Records written to diskette files are always unblocked, and the block size of records written to tape files is specified by the user.



a. Address out sort (SORTA)

b. Tag-along sort (SORTR)

c. Summary sort (SORTRS)

LEGEND:

① Defining normal control fields

② Defining opposite control fields

③ Defining forced control fields

④ Defining data control fields

⑤ Defining summary data fields

Figure 11—7. Field Description Specification Formats

In reviewing the specification formats, note that columns 7—16 must be specified for all sort jobs. Columns 17—19 are applicable only when forced control fields are used, and columns 20—22 apply only to summary sorts.

To make certain SORT3 properly interprets the contents of the field description specification, you must define whether a control field, data field, or comment is being described in each specification line. Defining the field type is a function of the entry in column 7 of the specification. Data field descriptions are indicated by a D entered in the column, comments are indicated by an asterisk (*) in the column, and summary data are indicated by an S in the column. Control field descriptions are indicated by column entries of N, O, or F depending on the type of control field described (normal, opposite, or forced).

Data field descriptions are applicable only to the tag-along sorts (SORTR, SORTRS) performed by SORT3. They define the fields you want SORT3 to include in your sorted output records. They are not used for defining the fields used in sorting the records. When your input file contains more than one type of record, it is not necessary that the total length of data fields defined be the same for all record types, or that all record types contain the same number of data fields.

The SORT3 program blank-fills short data fields to maintain a uniform length for all data fields. It is necessary that within each set of included record type and field description lines, the data field description lines follow the control field description lines in your control stream. (SORT3, when building work records, requires control records to appear ahead of data records.)

Do not include data field descriptions in your control stream for address out sorts (SORTA) because SORT3 will process the description line as a comment. In mentioning comments, it should be pointed out that comment lines serve no function other than helping you document the sort. When properly identified (* in column 7), they can be coded anywhere in the specification. However, it is preferred that they be coded in columns 40—80 to avoid confusion. Comments are printed only when the Print Option (0 or blank in column 27) is specified in the header specification.

Summary data field descriptions (S in column 7) can be defined for all three sort jobs performed by SORT3. However, the data fields are summarized (added together) only in the summary tag-along sort (SORTRS). For tag-along sorts (SORTR and SORTT), summary data fields are processed as normal data fields. For address out sorts (SORTA), the summary data fields are processed as comments.

It is important, when performing a summary sort, that the summary data fields in the work and output records of the individual record types be located in the same position on each record. This is not a consideration for the input records. No more than 24 data fields can be summarized for each record type included in the sort.

The format for summary fields is defined by the first summary field description specification processed for an included record type. It is suggested that a summary specification be specified for each record type included in the sort. The advantage is that SORT3 will issue a warning whenever summary fields are not aligned; you can then make the changes necessary. If summary field specifications are not provided, the data field specifications should align the data for summarization. If summary specifications are not provided when a summary tag-along sort (SORTRS) is specified, the output of the job produced a file consisting of records with unlike control fields. This is due to the fact that SORT3 eliminates all but one copy of each record having common control fields.

Another consideration when defining summary data fields is the possibility of overflow. To allow for the possibility of an anticipated overflow condition, you should complete the overflow field length entry in columns 20—22. These columns are used only by a summary tag sort to eliminate the possibility of an overflow condition. (An entry in the overflow field length columns is ignored for forced fields because they are only 1-byte in length.) The entry made in columns 20—22 effectively increases the length of the summary data field and should reflect the sum of the summary data field length and the anticipated overflow length. Entries in columns 20—22 must also be right-justified and must not exceed the maximum field length determined by your entry in column 8.

To illustrate the coding for columns 20—22, assume you want to summarize an unpacked field in positions 8—11 of the input record. You know that the output will exceed the 4-position summary field by 1 position. To allow for the expected overflow; specific an entry of 5 in column 22.

If packed fields are summarized, columns 20—22 should specify the number of bytes of packed data. For example, to summarize a packed field in positions 3—6 of the input record, knowing that the output will exceed the 4-position packed summary field (7 numbers plus sign) by 1 position, specify a 5 in column 22 (9 numbers plus sign).

Normal control field descriptions are used to sort records in the normal sequence as specified by the sequence field (column 18) in the header specification for the job.

Opposite control field descriptions are also used to sort records. However, they instruct SORT3 to sequence the records opposite to that specified by the sequence field in the header specification.

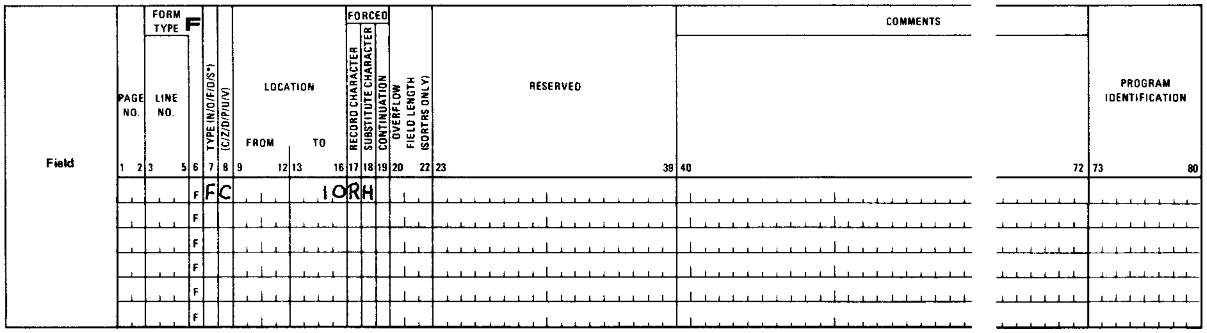
By defining normal and opposite control fields for your job, records can be sorted so some control fields are in ascending order and other control fields are in descending order.

Force control field descriptions are used to modify the contents of control fields in the work records constructed for the sort. Forced control field descriptions affect work records and output records, but not your input records.

When a line of the specification is identified as a forced control field line (F in column 7), SORT3 looks at columns 9—16 to identify the position of the control field in the input record, then it checks columns 17—19 to see how the control field is affected. That is, the entries in columns 17—19 tell sort how to modify the control field when it is placed into the work record. The column 17 entry specifies which character in the control field (identified in columns 13—16) SORT3 is to replace. The entry in column 18 gives the replacement character for the field or, when column 17 is not used, adds a new character to the control field identified. The column 19 entry shows a continuation in the force field description (relates the specification to the preceding specification line). To review the various conditions under which you may use forced control fields, three examples are provided as follows:

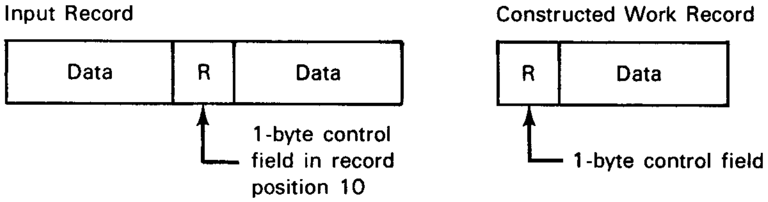
Example 1:

Example 1 illustrates a conditional force. Assume that each record in the file to be sorted has a 1-byte control field in record position 10. If the byte contains the charcter R, SORT3 replaces it with the character H before sorting the records. The field description specification is coded as follows:



		FORM TYPE F						FORCED					COMMENTS		PROGRAM IDENTIFICATION
	PAGE NO.	LINE NO.		TYPE (N/O/F/D/S*)	(C/Z/D/P/U/V)	LOCATION FROM	TO	RECORD CHARACTER	SUBSTITUTE CHARACTER	CONTINUATION	OVERFLOW FIELD LENGTH (SORTRS ONLY)	RESERVED			
Field	1 2	3 5	6	7	8	9 12	13 16	17	18	19	20 22	23 39	40	72	73 80
			F	F	C		10	R	H						
			F												
			F												
			F												
			F												
			F												

The SORT3 program constructs a work record for the selected input record.



The content of the control field is checked for the character R. If it contains an R, the control field content is changed to H in the work record.

Original Work Record

R	Data

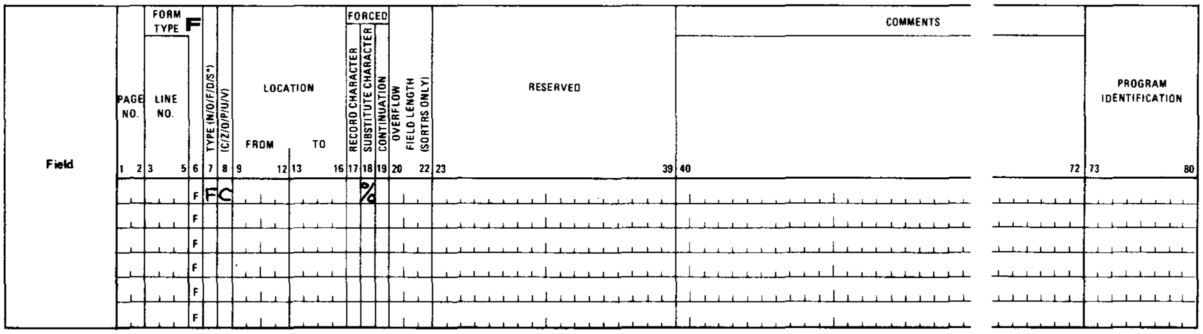
Modified Work Record

H	Data

Example 2:

Example 2 illustrates an unconditional force. An unconditional force allows you to add (force) a new character into your output records. This is done without basing the force on a specific control field of the input record as you would for a conditional force. Therefore, you do not code columns 9—16 of the specification. However, you must define the character that is being forced. You define this character in column 18.

The position that the forced character occupies in the output record is determined by the sequence in which it appears in your control stream. That is, if it is the first control field specification defined in your control stream, it will be the first control field in the output record. This is shown in the following coding:



	PAGE NO.	LINE NO.	FORM TYPE F	TYPE (N/O/F/D/S*)	(C/Z/D/P/U/V)	LOCATION FROM	LOCATION TO	FORCED RECORD CHARACTER	FORCED SUBSTITUTE CHARACTER	CONTINUATION	OVERFLOW FIELD LENGTH (SORTRS ONLY)	RESERVED	COMMENTS	PROGRAM IDENTIFICATION
Field	1 2	3 5	6	7	8	9 12	13 16	17	18	19	20 22	23 39	40 72	73 80
			F	F	C				%					
			F											
			F											
			F											
			F											
			F											

When processed, this specification instructs SORT3 to place a percent sign in the first control field position of the work and output records. All other control fields are positioned after the percent sign. For example, if the input record format appears as:

Data	Control Field A	Data	Control Field B	Data

The work record constructed from the field specification appears as follows:

%	Control Field A	Control Field B	Data

If the forced control field is defined after the field descriptions for A and B, then the percent sign (%) occupies the third control field position (first undefined control field available) in the record. The work record constructed from the specification would appear as:

Control Field A	Control Field B	%	Data

Example 3:

Example 3 illustrates a force-all condition. Force-all is a special form of conditional force (Example 1) that can only occur when a control field in the input record does not contain a particular field entry. If, for example, a specific control field in the input record does not contain a particular character, you can direct SORT3 to change the contents of the control field in the work record. Force-all specifications usually follow a series of conditional force specifications. In the specifications shown, SORT3 checks the 1-byte control field in position 1 to see if it contains the characters A, B, or #. If it does, A is replaced with the character 2, B is replaced with a 4, and # with a $. If the control field does not contain an A, B, or # entry, SORT3 places + in the control field.

		FORM TYPE F							FORCED					COMMENTS	
	PAGE NO.	LINE NO.		TYPE (N/O/F/D/S*)	(C/Z/D/P/U/V)	LOCATION FROM	TO	RECORD CHARACTER	SUBSTITUTE CHARACTER	CONTINUATION	OVERFLOW FIELD LENGTH (SORTRS ONLY)	RESERVED			PROGRAM IDENTIFICATION
Field	1 2	3 5	6	7	8	9 12	13 16	17	18	19	20 22	23 39	40	72	73 80
			F	F	C		1	A	2						
			F	F	C		1	B	4	X					
			F	F	C		1	#	$	X					
			F	F	C				*	X					
			F												
			F												

Note the use of the X in column 19. This shows that each specification is a continuation of the preceding line.

SORT3 replaces the specified control fields with hexadecimal FF or OO if (depending if ascending or descending sequence is specified in the header specification):

- the force-all specification did not follow the conditional force specifications; and
- SORT3 cannot locate any of the characters specified in the conditional force specification.

The following list summarizes the rules for coding the field description specification for forced control fields:

Defining a Conditional Force Character

— Fill in columns 1—6 as for any control field.

— Put an F in column 7.

— Define the position of the control field in the input record in columns 13—16.

— Enter the character to be replaced in column 17.

— Enter the character it is to be replaced with in column 18.

Defining a Force-All Character

— Fill in columns 1—6 as for any control field.

— Put an F in column 7.

— Enter the character which replaces the control field in column 18.

— Put any character in column 19. (The character in column 19 tells SORT3 that the line is a continuation of the preceding line.)

— Leave columns 9—17 blank.

NOTE:

If a force-all line is not placed after conditional force and SORT3 does not find the specified characters in the control field of the input record, then SORT3:

1. *replaces the control field character with hex FF (if ascending sequence is specified in the header); or*

2. *replaces the control field character with hex 00 (if descending sequence is specified in the header).*

Defining an Unconditional Force Character

— Fill in column 1—6 as for any control field.

— Put an F in column 7.

— Put the character to be forced in column 18.

— Leave columns 9—17 blank.

After defining the type of field being described, you must indicate to SORT3 what portion of the input record it must use to build and sort work records. This definition is the function of the entry made in column 8. If the data to be used is alphanumeric, you can enter the characters C, Z, or D in column 8. A C entry tells SORT3 to use both zone and digit portions of the data bytes in the fields defined. If you only want SORT3 to use the zoned portion of each byte, enter a Z in column 8. The D entry limits SORT3 to use of the digit portion of each byte. When numeric data is used for building and sorting work files, you must define to SORT3 whether the data being used are signed packed decimal numbers or signed unpacked decimal numbers. This is accomplished by entering a P or U, respectively, in column 8.

In a situation where SORT3 is to force characters into a data field of the work record, enter a V in column 8 and define the character to be forced in column 18.

To illustrate how the column 8 entry functions, assume that a 1-byte control field in the input record can contain any one of the following characters:

Character	Zone	Digit
$	0101	1011
A	1100	0001
B	1100	0010
C	1100	0011

If you wanted the records sorted into ascending order using the digit portion of the control field characters, put a D in column 8. The characters will appear in the following order in the output record:

A

B

C

$

If the records are to be sorted into ascending order using both the zone and digit portions, enter a C in column 8. The order of the characters in the output records will be:

$

A

B

C

If you had a Z entered in column 8, and specified ascending sequence in the header specification, the records with an $ control field precede records with an A, B, or C control field. Because A, B, and C have identical zone portions, records with any of these characters as a control field will not be any special order after the sort.

If you want to sort records so that some control fields are in ascending order and other control fields are in descending order, opposite control fields should be used. An opposite field is sorted in ascending order if descending order is specified on the header specification, or in descending order if ascending order is specified on the header specification. If the file contains different record types, all of which have an opposite control field in the same record position, the column 8 entries for all these control fields must be D, C, Z, or any combination of C and Z. With any other combination, the results of the sort will be unpredictable.

NOTE:

When using opposite control fields, SORT3 changes them into meaningless control field information when building the work record. Therefore, information is usually dropped by coding an X in column 28 of the header specification for tag-along or summary sorts. To retain the original control field data in the output record, repeat the description for the information as a data field. The same holds true when using packed or unpacked control fields.

If you specified packed or unpacked control fields (normal or opposite), SORT3 changes the control fields while building the work record. Therefore, the control field information must be dropped by coding an X in column 28 of the header specification. To retain the original control field data in the output record, redefine the information as a data field.

When using control fields to sequence information in the sorted records, the following rules must be followed:

- Only one character is allowed in a forced control field.
- Either a conditional or an unconditional force can be indicated.
- A force-all must be preceded by a conditional force.
- A forced control field can be defined by placing an F in column 7 of the field specifications.

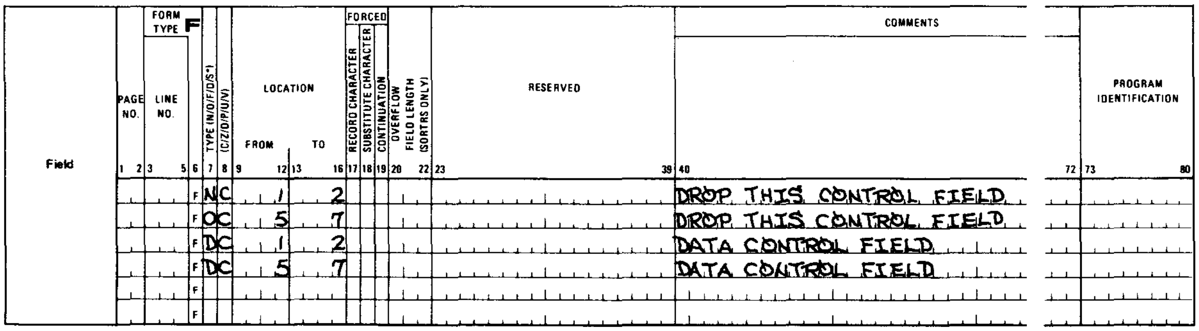
The order in which control fields are described in the field specification lines determines the sequence of the records (tag-along sort) or the record addresses (addrout sort) in the sorted file.

Suppose a file is to be sorted in ascending order (A in column 18 of the Header line) and each record in that file has a normal control field in positions 1—2 and an opposite control field in positions 5—7. Each record represents one customer's order for a separate item. The part number is in position 1—2; the number of parts ordered is in positions 5—7. The unsorted file appears as:

Input Record Name	Input Record Position						
	1	2	3	4	5	6	7
0	2	0			3	0	0
1	4	0			2	5	0
2	3	0			6	0	0
3	1	4			4	5	0
4	6	0				7	0
5	3	0				5	9
6	6	0			1	3	0

Part Number (positions 1—2)   Number Ordered (positions 5—7)

The first control field can be used to sort the records in ascending order according to the part number. The second control field is then used to sort the number of parts ordered in descending order within each group of parts. The field specification would be coded as follows:



Field	PAGE NO. 1 2	LINE NO. 3 5	FORM TYPE F 6	TYPE (N/O/F/D/S*) 7	(C/Z/D/P/U/V) 8	LOCATION FROM 9 12	LOCATION TO 13 16	FORCED RECORD CHARACTER 17	FORCED SUBSTITUTE CHARACTER 18	CONTINUATION 19	OVERFLOW FIELD LENGTH (SORTRS ONLY) 20 22	RESERVED 23 39	COMMENTS 40 72	PROGRAM IDENTIFICATION 73 80
			F	N	C	1	2						DROP THIS CONTROL FIELD	
			F	O	C	5	7						DROP THIS CONTROL FIELD	
			F	D	C	1	2						DATA CONTROL FIELD	
			F	D	C	5	7						DATA CONTROL FIELD	
			F											
			F											

The sorted output file is formatted as follows:

Output Record Number	Output Record Position				
	1	2	3	4	5
0	1	4	4	5	0
1	2	0	3	0	0
2	3	0	6	0	0
3	3	0		5	9
4	4	0	2	5	0
5	6	0	1	3	0
6	6	0		7	0
	Part Number		Number Ordered		

After completing the column 7 and 8 entries, you must identify where the record field being used starts and ends in the input record. The position at which the field begins in the record is entered (right-justified) in columns 9—12 (From columns). The position at which the field ends in the record is entered (right-justified) in column 13—16 (To columns). The order in which the fields are described in the specification determines the order they appear in the sort output records. For example, suppose you have an input record that looks as follows:

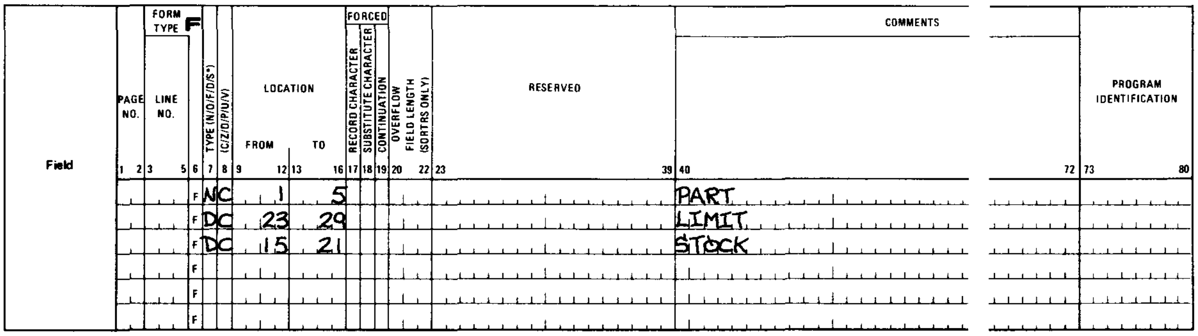
Record Field Positions

Part (part number)		Cost (price per item)		Stock (balance in stock)		Limit (reorder limit)
1 5	6	7 12	13 14	15 21	22	23 29

But you want your output (sorted) record to look like:

Part	Limit	Stock
1 5	6 12	13 19

To format the output record, columns 9 through 16 would have to be coded as follows:



Field	PAGE NO.	LINE NO.	FORM TYPE F	TYPE (N/O/F/D/S*)	(C/Z/D/P/U/V)	LOCATION FROM	LOCATION TO	FORCED RECORD CHARACTER	FORCED SUBSTITUTE CHARACTER	CONTINUATION	OVERFLOW FIELD LENGTH (SORTRS ONLY)	RESERVED	COMMENTS	PROGRAM IDENTIFICATION
	1 2	3 5	6	7	8	9 12	13 16	17	18	19	20 22	23 39	40 72	73 80
			F	N	C	1	5						PART	
			F	D	C	23	29						LIMIT	
			F	D	C	15	21						STOCK	
			F											
			F											
			F											

The limitation to the maximum length of the field described in columns 9—16 is determined by the entry in column 8.

Column 8 Entry	Maximum Allowable Field Length (bytes)
C	256
Z	1
D	16
P	8
U	16
V	1

When fields 1-byte in length are being described, leave columns 9—12 (From) blank, and enter the record position of byte right-justified in column 13—16.

Comments pertaining to field description specification are coded in columns 40—80 and have no effect on the sort function. Comments are printed only if specified in Print option of the header specification.

A summary of the field entries for the field description specification is provided in Table 11—8.

Table 11—8. Column Summary for Field Description Specification

Columns	Contents	Explanation
1—2		Page number
3—5		Line number (Leave column 5 blank, or enter any value to keep the specifications in ascending order.)
6	F	Field specification line
7	N	Normal control field
	O	Opposite control field
	F	Forced control field
	D	Data field
	S	Summary data field
	*	Comment
8	P	Signed packed decimal data
	U	Signed unpacked decimal data
	C	Use both zone and digit portions of bytes in the field.
	Z	Use only zone portion of 1-byte field.
	D	Use only digit portion of bytes in the field.
	V	Force a data character into the data field.
9—12	1—4096	Starting position of field in the record (blank if field is one byte long)
13—16	1—4096	End position of field in the record
17	Any character	Forced control fields only (the character the sort is to change)
18	Any character	Forced control fields only (the character to substitute)
19	Blank	Forced control field line is not a continuation of the preceding line.
	Any character other than blank	Forced control field is a continuation of the preceding line.
20—22	1-256	Summary tag-along sort only (overflow field length entry)
23—39	Not used	Not used
40—80	Any character	Comments or program identification

11.3.4. Defining an Alternate Collating Sequence

If you elect to use a collating sequence other than the standard collating sequence provided by the SORT3 program, you are required to define the alternate collating sequence to be used in its place. To do this, you must prepare alternate collation (ALTSEQ) statements and include them in your job control stream. ALTSEQ statements are prepared in 80-column punch card format and are positioned immediately after the header specification in the control stream. A punch card with double asterisks (** in columns 1 and 2) immediately follows the ALTSEQ statements to mark their ending in the job stream. When inserted into your job stream, ALTSEQ statements should appear as follows:

```
Job control statements
     .
     .
     .
// EXEC SORT3
/$
Header specification
ALTSEQ statements
**
Record type/field description specifications as required
/*
.
.
.
Job control statements
```

You may include as many ALTSEQ statements as needed to define the alternate collating sequence. Each new statement, however, must begin in column 1 and must begin with ALTSEQ.

The rules for preparing ALTSEQ statements are as follows:

1. Enter ALTSEQ in columns 1 through 6.

2. Leave columns 7 and 8 blank.

3. Enter, into columns 9 and 10, the hexadecimal equivalent of the character being moved from its normal position in the collating sequence.

4. Enter, into column 11 and 12, the hexadecimal equivalent of the character whose position in the collating sequence is to be assumed by the character specified in step 3.

5. Repeat steps 3 and 4 for as many pairs as required to define the characters that must be taken out of the normal sequence. Do *not* leave spaces between sets of hexadecimal entries.

6. End the series of statements by placing a card with double asterisks (**) in columns 1 and 2 after the last ALTSEQ statement.

Although ALTSEQ statements do not affect data fields or forced control field characters, they do affect Factor 1 and Factor 2 fields, normal and opposite control fields, and control field characters before they are replaced or added to by forced fields.

You should consider what effect an alternate collating sequence will have on these fields for your particular job. In addition, packed and unpacked Factor 1 and 2 fields must not be specified when an alternate collating sequence is used.

Another consideration when using an alternate collating sequence is whether the characters moved in the sequence are considered equal or unequal. That is, when a character is moved into the sequence position normally assigned to another character, both the new and the original character occupy the same position. They are considered equal. If they are not to be considered equal, the character that originally occupied the position must be moved to another position. To illustrate this point, two examples are provided. The first example shows the coding required to change one character in the sequence (characters are considered equal). The second example applies to changing several characters where they are unequal.

Example 1:

```
ALTSEQ    505B
**
```

The character defined by hexadecimal 50 (&) is moved to the position defined by hexadecimal 5B ($). The ampersand and the dollar sign both occupy the same position and are therefore considered equal.

Example 2:

```
ALTSEQ    4EF3F3F4F4F5
**
```

The characters represented by the hexadecimal values shown in the ALTSEQ format are as follows:

≠	4E
3	F3
4	F4
5	F5

The format shown moves the character ≠ into the position occupied by the character 3. Because you do not want them to be considered equal, you must move the character 3 to another position. To maintain the proper sequence in the collation, the character 3 is moved into the character 4 position, 4 is moved into character 5 position, and so on. Basically, you have altered the collating sequence so that ≠ is inserted between 2 and 3.

12. System/3, 32, and 34 Compatible Sort Program and Control Stream Examples

12.1. GENERAL

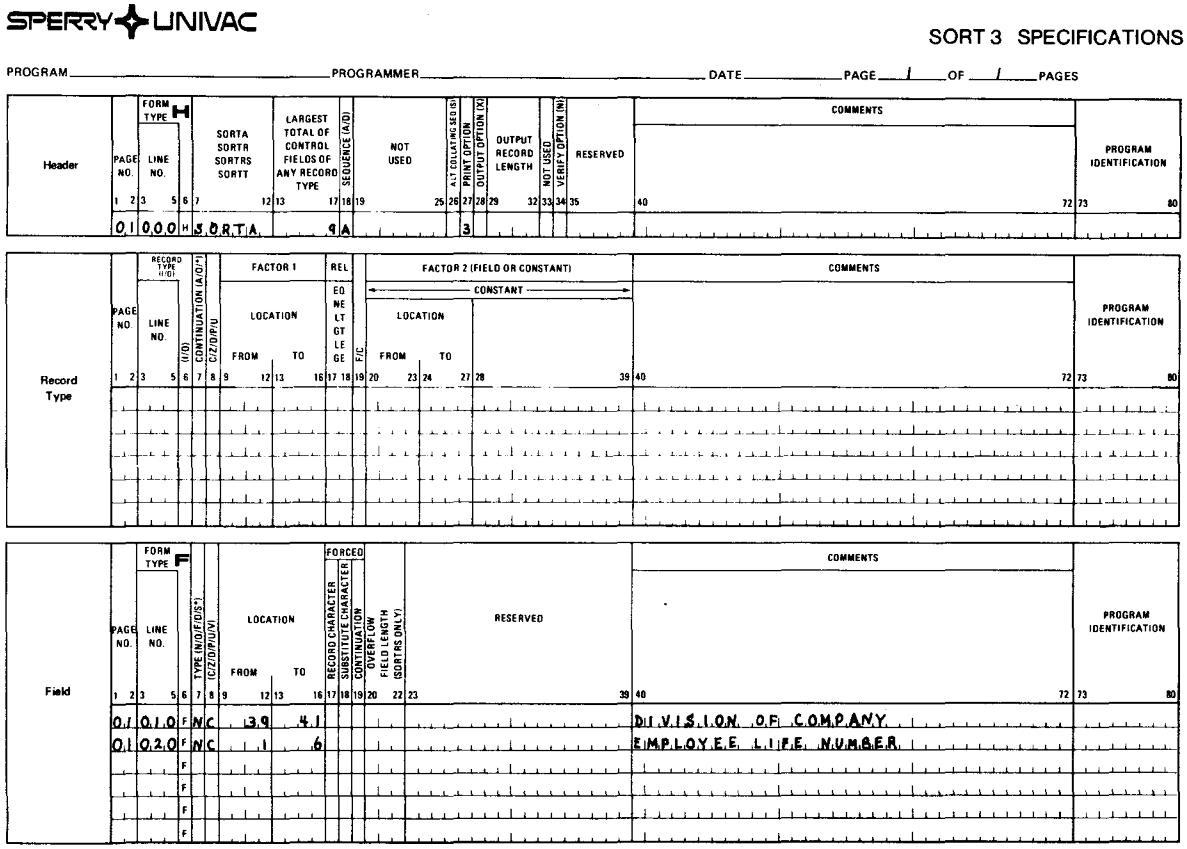
This section contains examples that illustrate program coding and job control streams for the System/3, 32, and 34 compatible sort operation. The first three examples illustrate only the sort specifications, and the four remaining examples show complete job streams with the sort statements included.

12.2. SORT PROGRAM CONTROL SPECIFICATION EXAMPLES

The following three examples illustrate only the sort specifications used for an address out sort, a tag-along sort, and a summary sort.

Example 1 shows the sort specifications for an address out sort. The purpose of the job is to produce an output file containing the 10-byte relative addresses of all the records in the input file. The records are sorted in ascending order by company division number (control field position 39—41 of the input record), and then by employee life number (control field position 1—6) within each division. You would code the sort specifications for this job as follows:

Example 1:



SPERRY UNIVAC

SORT 3 SPECIFICATIONS

PROGRAM ______ PROGRAMMER ______ DATE ______ PAGE 1 OF 1 PAGES

	PAGE NO. 1 2	FORM TYPE H LINE NO. 3 5	6	SORTA SORTR SORTRS SORTT 7 12	LARGEST TOTAL OF CONTROL FIELDS OF ANY RECORD TYPE 13 17	SEQUENCE (A/D) 18	NOT USED 19 25	ALT COLLATING SEQ (S) 26	PRINT OPTION 27	OUTPUT OPTION (X) 28	OUTPUT RECORD LENGTH 29 32	NOT USED 33	VERIFY OPTION (N) 34	RESERVED 35	COMMENTS 40 72	PROGRAM IDENTIFICATION 73 80
Header	01	000	H	SORTA	9	A			3							

	PAGE NO. 1 2	RECORD TYPE (I/O) LINE NO. 3 5	(I/O) 6	CONTINUATION (A/O/*) 7	C/Z/D/P/U 8	FACTOR 1 LOCATION FROM 9 12	FACTOR 1 LOCATION TO 13 16	REL EQ NE LT GT LE GE 17 18	F/C 19	FACTOR 2 (FIELD OR CONSTANT) LOCATION FROM 20 23	FACTOR 2 LOCATION TO 24 27	CONSTANT 28 39	COMMENTS 40 72	PROGRAM IDENTIFICATION 73 80
Record Type														

	PAGE NO. 1 2	FORM TYPE F LINE NO. 3 5	6	TYPE (N/O/F/D/S*) 7	(C/Z/D/P/U/V) 8	LOCATION FROM 9 12	LOCATION TO 13 16	FORCED RECORD CHARACTER 17	FORCED SUBSTITUTE CHARACTER 18	CONTINUATION 19	OVERFLOW FIELD LENGTH (SORTRS ONLY) 20 22	RESERVED 23 39	COMMENTS 40 72	PROGRAM IDENTIFICATION 73 80
Field	01	010	F	N	C	39	41						DIVISION OF COMPANY	
	01	020	F	N	C	1	6						EMPLOYEE LIFE NUMBER	
			F											
			F											
			F											
			F											

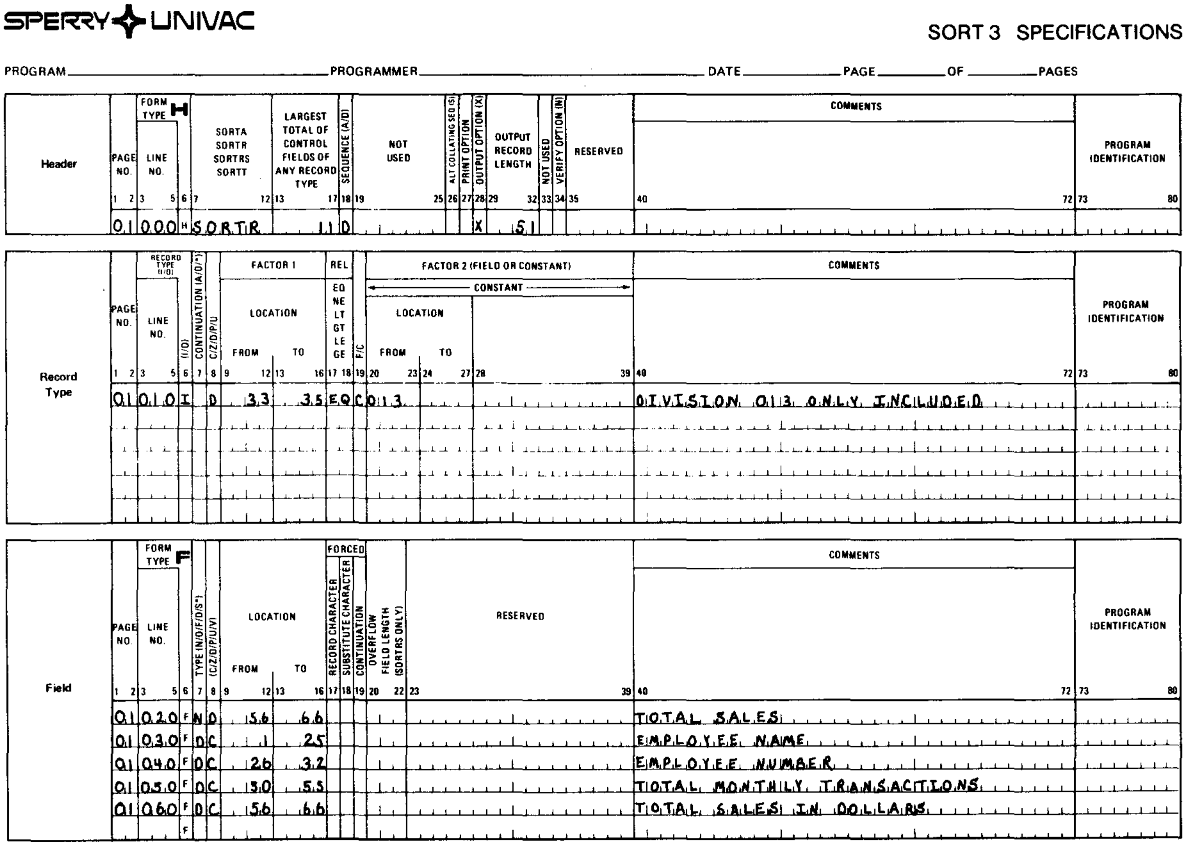
In this example, the header statement defines the job as an address out sort by the entry SORTA in columns 17—12 and specifies the longest total length of the control fields used for the sort as 9. (The two control fields, division and life number, are 3 and 6 respectively and are contained on the same record type.) The entry A in column 18 indicates ascending order for the sort. Because no alternate collating sequence is specified, SORT3 will use the standard collating sequence. The printing of all messages is inhibited by the 3 entry in column 27.

Because all of the input records are involved in the sort, and they all have the same format, it is not necessary to prepare a record type specification for this job. SORT3 assumes an include-all situation.

Both control fields used for sorting the records are identified as normal control fields by the N entry in column 7 of the field description specification. The entry C in column 8 indicates that both the zone and the digit portion of the characters in the two control fields are used for the sort. Because the division number is the first field used in the sort, its position in the input record is defined first. It occupies three positions in the record, beginning at position 39 and ending at position 41. The next field used for sorting (life number) occupies six positions beginning at position 1 and ending at position 6.

Example 2 shows the sort specifications for a tag-along sort. The output file produced is to contain the records of only those salespersons working for division 013 of the company. Records are to be sorted in descending sequence by total sales. Each output record is to contain the employee name and number, total monthly transactions, and total sales in dollars. The division number is to be dropped from the output. You would code the sort specifications for this job as follows:

Example 2:



SPERRY UNIVAC

SORT 3 SPECIFICATIONS

PROGRAM ________ PROGRAMMER ________ DATE ________ PAGE ____ OF ____ PAGES

Header	PAGE NO. 1-2	LINE NO. 3-5	FORM TYPE H 6	SORTA SORTR SORTRS SORTT 7-12	LARGEST TOTAL OF CONTROL FIELDS OF ANY RECORD TYPE 13-17	SEQUENCE (A/D) 18	NOT USED 19-25	ALT COLLATING SEQ (S) 26	PRINT OPTION 27	OUTPUT OPTION (X) 28	OUTPUT RECORD LENGTH 29-32	NOT USED 33	VERIFY OPTION (N) 34	RESERVED 35	COMMENTS 40-72	PROGRAM IDENTIFICATION 73-80
	01	000	H	SORTR	11	D				X	51					

Record Type	PAGE NO. 1-2	RECORD TYPE (I/O) LINE NO. 3-5	(I/O) 6	CONTINUATION (A/O/*) 7	C/Z/D/P/U 8	FACTOR 1 LOCATION FROM 9-12	FACTOR 1 LOCATION TO 13-16	REL EQ NE LT GT LE GE 17-18	F/C 19	FACTOR 2 (FIELD OR CONSTANT) CONSTANT LOCATION FROM 20-23	TO 24-27	28-39	COMMENTS 40-72	PROGRAM IDENTIFICATION 73-80
	01	010	I		D	33	35	EQ	C	013			DIVISION 013 ONLY INCLUDED	

Field	PAGE NO. 1-2	LINE NO. 3-5	FORM TYPE F 6	TYPE (N/O/F/D/S*) 7	(C/Z/D/P/U/V) 8	LOCATION FROM 9-12	LOCATION TO 13-16	FORCED RECORD CHARACTER 17	FORCED SUBSTITUTE CHARACTER 18	CONTINUATION 19	OVERFLOW FIELD LENGTH (SORTRS ONLY) 20-22	RESERVED 23-39	COMMENTS 40-72	PROGRAM IDENTIFICATION 73-80
	01	020	F	N	D	56	66						TOTAL SALES	
	01	030	F	D	C	1	25						EMPLOYEE NAME	
	01	040	F	D	C	26	32						EMPLOYEE NUMBER	
	01	050	F	D	C	30	55						TOTAL MONTHLY TRANSACTIONS	
	01	060	F	D	C	56	66						TOTAL SALES IN DOLLARS	
			F											

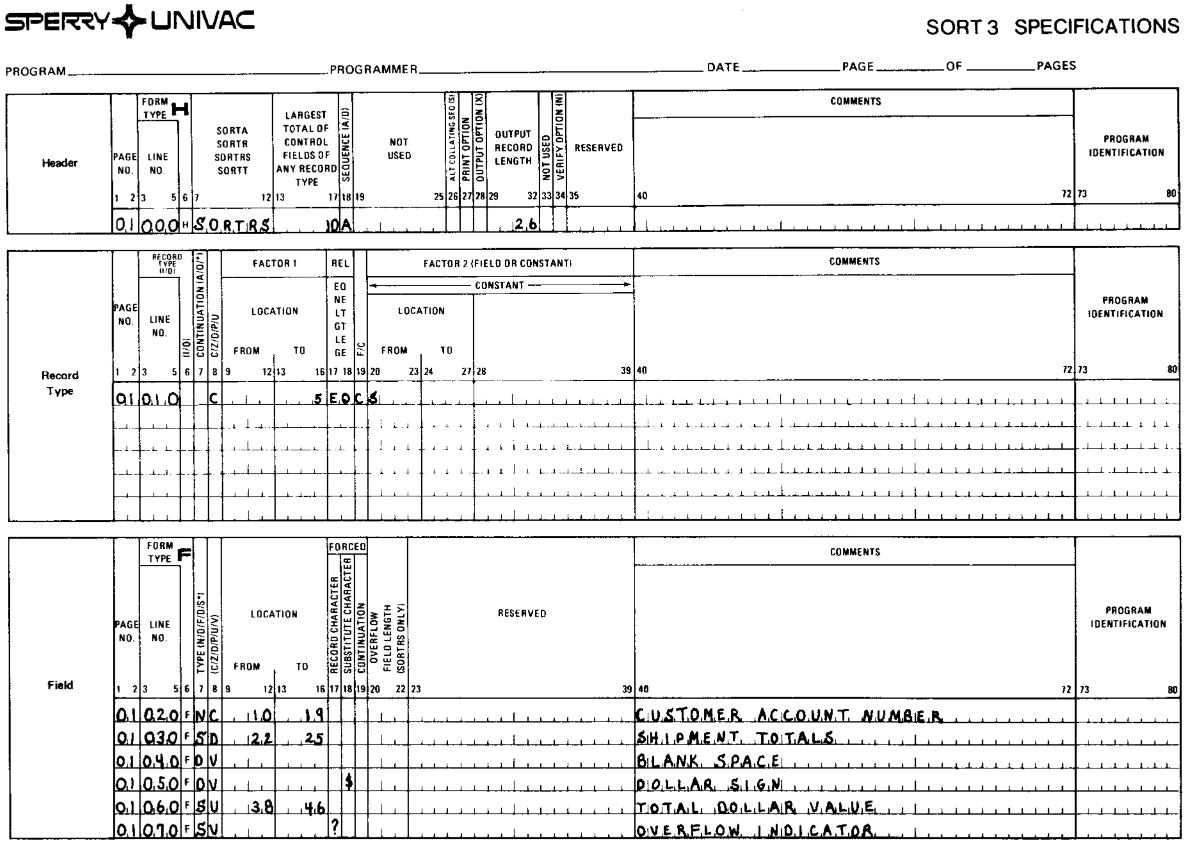
The header statement defines the job as a tag-along sort (SORTR) and defines the total length of the control field used for the sort as 11. (Total sales field extends from position 56 to 66 in the record — a length of 11.) The entry D in column 18 specifies descending order for the sort and the X in column 28 instructs SORT3 to drop control fields from the output records. The total length of the output record, as determined by the data fields described in the field description specification, is 51. (Control field lengths are not included when they are dropped from the output records.) The standard collation sequence is used and all messages are printed.

To select only those records applicable to the salespersons employed at division 13, the record statement sets up a comparison (EQ in columns 17—18) between the division number field of the input record (33—35) and the constant 013 (columns 20—23). If the comparison proves equal, the record is included (I in column 6) in the sort.

Because the selected records are sorted according to the value in the total sales field (56—66), this field must be the first described in the field description specification. The entry N defines the field as a normal control field. Only the data portion of the characters in this field are to be used for the sort as indicated by the entry D in column 8. The control field begins at position 56 and ends at position 66. The remaining field description specifications define the data fields (D in column 7) that are to be included in the output records. The position (entries in columns 9—16) of each field is also identified for the SORT3 program. Take note that the total sales field is described twice, once as a control field and once as a data field. This was purposely done so this field would appear in the output record. Remember SORT3 was instructed to drop control fields from the output records.

Example 3 shows the sort specifications for a summary sort. The purpose of this job is to produce an output file that lists, by customer account, the total number of shipments made to each customer and the total dollar value of those shipment. The program, therefore, must be capable of selecting only those records of customers to whom shipments were made, and to summarize the data in the shipment and dollar value fields of each record to produce the totals required for the output.

Example 3:



SPERRY UNIVAC SORT3 SPECIFICATIONS

PROGRAM ______ PROGRAMMER ______ DATE ______ PAGE ______ OF ______ PAGES

Header	PAGE NO. 1 2	FORM TYPE H LINE NO. 3 5	6	SORTA SORTR SORTRS SORTT 7 12	LARGEST TOTAL OF CONTROL FIELDS OF ANY RECORD TYPE 13 17	SEQUENCE (A/D) 18	NOT USED 19 25	ALT COLLATING SEQ (S) 26	PRINT OPTION 27	OUTPUT OPTION (X) 28	OUTPUT RECORD LENGTH 29 32	NOT USED 33	VERIFY OPTION (N) 34	RESERVED 35	COMMENTS 40 72	PROGRAM IDENTIFICATION 73 80
	01	000	H	SORTRS	10	A					26					

Record Type	PAGE NO. 1 2	RECORD TYPE (I/O) LINE NO. 3 5	(I/O) 6	CONTINUATION (A/O/*) 7	C/Z/D/P/U 8	FACTOR 1 LOCATION FROM 9 12	FACTOR 1 LOCATION TO 13 16	REL EQ NE LT GT LE GE 17 18	F/C 19	FACTOR 2 (FIELD OR CONSTANT) CONSTANT LOCATION FROM 20 23	LOCATION TO 24 27	28 39	COMMENTS 40 72	PROGRAM IDENTIFICATION 73 80
	01	010			C		5	EQ	C	S				

Field	PAGE NO. 1 2	FORM TYPE F LINE NO. 3 5	6	TYPE (N/D/F/D/S*) 7	(C/Z/D/P/U/V) 8	LOCATION FROM 9 12	LOCATION TO 13 16	FORCED RECORD CHARACTER 17	FORCED SUBSTITUTE CHARACTER 18	CONTINUATION 19	OVERFLOW FIELD LENGTH (SORTRS ONLY) 20 22	RESERVED 23 39	COMMENTS 40 72	PROGRAM IDENTIFICATION 73 80
	01	020	F	N	C	10	19						CUSTOMER ACCOUNT NUMBER	
	01	030	F	S	D	22	25						SHIPMENT TOTALS	
	01	040	F	D	V								BLANK SPACE	
	01	050	F	D	V				$				DOLLAR SIGN	
	01	060	F	S	U	38	46						TOTAL DOLLAR VALUE	
	01	070	F	S	V			?					OVERFLOW INDICATOR	

The SORTRS entry in columns 7—12 of the header specification defines the job as a summary sort. The largest control field for any record is 10 and the records are to be sorted into ascending order (A in column 18) by customer account number. Standard collation is used, all messages are printed, and control fields are not to be dropped from the output records (columns 26—28 blank). The output record length specified in column 26 is a total of the control field lengths and data field lengths specified in the field description specification.

The record type specification identifies the records to be included (I in column 6) in the sort as those with the character S in position 5 of the record. The records are sorted by customer account number as defined by columns 9—16 of coding line 020 of the field description. As the individual records are sorted, the two data fields identified in coding lines 030 and 060 of the field description are summarized for identical customer account numbers. The output records, therefore, reflect the total number of shipments made to each customer and the total dollar value of those shipments. The output record format will also contain a dollar sign preceding the total dollar value field as specified by the use of the force field entry in coding line 050. An occurrence of an overflow condition while summarzing the summary data fields is indicated by a question mark (?) in the last field of the output record.

12.3. SORT PROGRAM CONTROL STREAM EXAMPLES

Sort control stream examples are provided to show how an address out sort and a tag-along sort would be set up for execution under OS/3 job control or the OCL processor. The data sets used for these examples are the same as those described in examples 1 and 2 respectively.

Example 4 shows the control stream and sort specifictions to perform an address out sort under OS/3 job control execution.

Example 4:

```
1        10        20                                                     72
________________________________________________________________________________
// JOB EXAMPLE,,8ØØØ,8ØØØ
// DVC 2Ø // LFD PRNTR
// DVC 65 // VOL SYS2ØØ // LBL DISKOUT // LFD INPUT
// DVC 66 // VOL SCR2ØØ // LBL $SCR1 // DMØ1
// DVC 65 // VOL SYS2ØØ
// EXT ...
// LBL CARDIN // LFD OUTPUT
// EXEC SORT3
/$
     HSORTA     9A      3
     FNC  39  41                                      DIVISION OF COMPANY
     FNC   1   6                                      EMPLOYEE LIFE NUMBER
/*
/&
// FIN
```

Example 5 shows the same sort program as example 4 but set up for execution under the OCL processor.

Example 5:

```
// LOAD $DSORT,F1
// FILE NAME-INPUT,UNIT-D1,PACK-SYS2ØØ,LABEL-DISKOUT,RETAIN-S
// FILE NAME-WORK,UNIT-D2,PACK-SCR2ØØ,RECORDS-7ØØ,RETAIN-S
// FILE NAME-OUTPUT,UNIT-D1,PACK-SYS2ØØ,LABEL-CARDIN,RECORDS-6ØØ
// RUN
     HSORTA      9A     3
     FNC  39  41                                DIVISION OF COMPANY
     FNC   1   6                                EMPLOYEE LIFE NUMBER
// END
/&
// FIN
```

Example 6 shows the control stream and sort specifications data set to perform a tag-along sort number OS/3 job control execution.

Example 6:

```
1        10        20
______________________________________________________________________________
// JOB EXAMP6,,8ØØØ,8ØØØ
// DVC 2Ø // LFD PRNTR
// DVC 65 // VOL SYS2ØØ // LBL HONPROD // LFD INPUT
// DVC 65 // VOL QC1111
// EXT ...
// LBL NAME // LFD OUTPUT
// DVC 66 // VOL DØØ921 // LBL $SCR1 // LFD DMØ1
// EXEC SORT3
/$
     HSORTR   11D          X  51
     I D  33  35EQC13
     FND  56  66                               TOTAL SALES
     FNC   1  25                               EMPLOYEE NAME
     FDC  26  32                               EMPLOYEE NUMBER
     FDC  5Ø  55                               TOTAL MONTHLY TRANSACTIONS
     FDC  56  66                               TOTAL SALES IN DOLLARS
/*
/&
// FIN
```

Example 7 shows the same sort program as example 6 but set up for execution under the OCL processor.

Example 7:

```
// LOAD $DSORT,F1
// FILE NAME-INPUT,UNIT-D1,PACK-SYS2ØØ,LABEL-HONPROD,RETAIN-S
// FILE NAME-OUTPUT,LABEL-NAME,UNIT-D2,PACK-QC1111,RECORDS-2ØØØ
// RUN
     HSORTR   11D          X  51
     I D  33  35EQC13
     FND  56  66                          TOTAL SALES
     FDC   1  25                          EMPLOYEE NAME
     FDC  26  32                          EMPLOYEE NUMBER
     FDC  5Ø  55                          TOTAL MONTHLY TRANSACTIONS
     FDC  56  66                          TOTAL SALES IN DOLLARS
// END
/&
// FIN
```

PART 5. APPENDIXES

Appendix A. Statement Conventions

A.1. GENERAL FORMAT RULES FOR SUBROUTINE AND INDEPENDENT SORT/MERGE

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

- Lowercase letters and words are generic terms representing information that must be supplied by you (Figure A—1, lines 2 and 4). Such lowercase terms may contain hyphens and acronyms (for readability).

  Examples:

  — Independent Sort/Merge

  ```
           ⎧ ([strt-pos-1][,lgth-1][,form-1][,seq-1]                 ⎫
  FIELDS=  ⎨ [,...,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                                 ⎭
  ```

  — Subroutine Sort/Merge

  [,USEQ=(to-address,from-address)]

- Capital letters, commas, equal signs, and parentheses must be coded exactly as shown. For example: The letter D on the ADDROUT parameter means return both address field and record key fields to the program in a subroutine tag sort (Figure A—1, line 1), and the letter A on the DATA parameter means input data is recorded in ASCII in an independent sort/merge (Figure A—1, line 5).

```
     LABEL     ΔOPERATIONΔ                 OPERAND
     1          10        16
1.              MR$PRM     ADDROUT=D
2.              MR$PRM     RESUME=(PASS,Ø33)
3.              MR$PRM     MERGE=YES
4.              MR$PRM     FIELD=(1,6)
5.              INPFIL     DATA=A
```

Figure A—1. Statement Conventions Example

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

— Independent Sort/Merge

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

— Subroutine Sort/Merge

```
⎧FIELD= (strt-pos-1,lgth-1 [,form-1] [,seq-1] [,order-1]                  ⎫
⎨              [,...,strt-pos-n,lgth-n[,form-n] [,seq-n] [,order-n]])     ⎬
⎩RSOC=symbol                                                              ⎭
```

- 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:

— Independent Sort/Merge

Braces within brackets:

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

Brackets:

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

— Subroutine Sort/Merge

Braces within brackets:

```
⎡,DROC= ⎧DELETE⎫⎤
⎣       ⎩symbol⎭⎦
```

Brackets:

```
[,SIZE=number]
```

- 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.

Examples:

—Independent Sort/Merge

$$\left[\text{TYPE=}\begin{Bmatrix}\text{D}\\ \text{F}\\ \text{V}\end{Bmatrix}\right]$$

— Subroutine Sort/Merge

$$\left[\text{,MERGE=}\begin{Bmatrix}\text{NO}\\ \text{YES}\end{Bmatrix}\right]$$

$$\left[\text{CSPRAM=}\begin{Bmatrix}\text{NO}\\ \text{YES}\end{Bmatrix}\right]$$

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

Examples:

— Independent Sort/Merge

$$\left[\text{,BLKSIZE=}\begin{Bmatrix}\text{bytes}\\ \text{(bytes-1[,...,bytes-8])}\end{Bmatrix}\right]$$

— Subroutine Sort/Merge

```
FIELD=     (strt-pos-1,lgth-1[,form-1] [,seq-1] [,order-1]
           [,...,strt-pos-n,lgth-n[,form-n] [,seq-n] [,order-n]])
```

- 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:

  — Independent Sort/Merge

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

```
LABEL     ΔOPERATIONΔ            OPERAND                              Δ
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
```

  — Subroutine Sort/Merge

  Assume that MR$PRM, the sort macro instruction, is specifying three of the required keyword parameters: FIN, IN, and OUT.

```
SORT1      MR$PRM    FIN=SORTFIN,IN=SORTIN,OUT=SORTOUT
SORT2      MR$PRM    FIN=SORTFIN,OUT=SORTOUT,IN=SORTIN
SORT3      MR$PRM    IN=SORTIN,FIN=SORTFIN,OUT=SORTOUT
SORT4      MR$PRM    OUT=SORTOUT,IN=SORTIN,FIN=SORTFIN
```

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

  Example:

  $\text{MR\$OPN}=\left\{\begin{matrix}\text{parameter-table-name}\\ 1\end{matrix}\right\}$

- 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. They must be coded in the order shown, and the comma must be retained when a subparameter is omitted, except for trailing commas.

  Examples:

  — Independent Sort/Merge

```
LABEL   ΔOPERATIONΔ                 OPERAND                              Δ
1        10    16
--------------------------------------------------------------------------
         SORT  FIELDS=(1,4,CH,A)
         SORT  FIELDS=(1,4,,A)
         SORT  FIELDS=(1,4)
```

  — Subroutine Sort/Merge

```
         MR$PRM FIELD=(Ø,1,AC,D,3)
         MR$PRM FIELD=(Ø,1,,D)
         MR$PRM FIELD=(Ø,1)
```

You can apply the following rules to the coding of both independent and subroutine sort/merge control statements and macro instructions.

- Sort/merge control statements and macro instructions are coded in the operation field, which may begin in any column except column 1.
- Parameters are specified in the operand field. This field is separated from the operation field by one or more blanks and must begin on the same line of the card.
- At least one space must separate the operand field from the comments field, if included.
- Embedded blanks are not allowed. Anything after a blank is regarded as a comment.
- Values can be written with up to eight alphanumeric characters.
- Commas, equal signs, parentheses, and blanks are used only as field delimiters; they may not be used in values.
- Periods are used to separate byte-bit specifications in the FIELD and FIELDS parameters and input and output file-partition-numbers in the COPY parameter.
- Any nonblank character in column 72 indicates that the statement is continued on the following card. The continuation of an operand starts in column 16; the continuation of comments starts in column 17. A continuation statement may not begin with a comma in column 16.

A.2. INDEPENDENT SORT/MERGE CONTROL STATEMENT FORMAT RULES

Independent 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.

- 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.

<table>
<tr><th>1 15</th><th>16</th><th>17 STATEMENT CONTINUATION 71</th><th>72</th><th>73 80</th></tr>
<tr><td rowspan="2">blank</td><td colspan="2">operand</td><td rowspan="2">continuation character</td><td rowspan="2">sequence-id</td></tr>
<tr><td></td><td>comment</td></tr>
</table>

- 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.

A.3. SUBROUTINE SORT/MERGE MACRO FORMAT RULES

Macro instructions provide an interface between subroutine sort/merge and your program. By using these macro instructions, you define the sort run; control the function, structure, and execution of the subroutine by building a sort parameter table; and link the various functional modules of the subroutine with your program. The conventions illustrated in A.1 show how to use the subroutine sort/merge macro instructions presented in this manual.

Appendix B. Contents of Sort Parameter Table

The sort parameter table is the primary interface between your program and the modules of the independent or subroutine sort/merge. Through this table, you define the requirements that sort/merge uses to sequence your input files and produce an output file ordered to your specifications.

Although a sort parameter table is needed by both independent and subroutine sort/merge, you can never see or modify it when using the independent sort/merge. Nevertheless, your sort control statements and job control statements used with independent sort/merge supply information to the sort parameter table.

If you use the subroutine sort/merge, the sort parameter table is generated inline in your program by execution of the MR$PRM macro instruction (6.4). Parameters can be modified via entries you submit on sort parameter statements issued in your job control stream for either independent or subroutine sort/merge. (See CSPRAM discussion, 3.2.6, and PARAM discussion, 6.12.) Table B—1 is the resulting parameter table generated inline after the MR$PRM macro is executed in your subroutine sort/merge program.

Each sort keyword generates a code. When you want to modify the location of certain sort keyword parameters, you find their code in your program printout. The value next to each code specifies the value you want to change. You may want to change some, all, or none of these parameters.

Lowercased letters in the value column of Table B—1 represent variable information which you supply via your sort keyword parameters on the MR$PRM sort macro. Their meaning is explained under the description of values in Table B—1. The codes, values, and keyword parameters are the only information actually generated inline in your subroutine sort/merge program.

Table B—1. Sort Parameter Table (Part 1 of 3)

Code	Value	Keyword Parameter	Description of Values
00	000000		000000 — Indicates the end of a parameter table
00	aaaaaa	ADTABL	aaaaaa — Is the address of an additional parameter table containing information which applies to this sort
01	aaaaaa	IN	aaaaaa — The address specified by IN keyword. This address identifies the location to which control returns following the opening of the sort/merge.
02	aaaaaa	OUT	aaaaaa — The address specified by the OUT keyword. This address specifies the location to which control returns when the sequenced records are ready to be returned.
03	aaaaaa	FIN	aaaaaa — Specifies the location to which control returns after the last record has been returned to the user
04	aaaaaa	RSOC	aaaaaa — Specifies the address at which the user own-code record sequencing routine is located
05	aaaaaa	DROC	aaaaaa — Specifies the address at which the user own-code, data-reduction routine is located
07	aaaaaa	STOR	aaaaaa — The address of the first byte of the work area reserved for the sort
FF	nnnnnn		nnnnnn — A binary value indicating the number of bytes available for sort usage in the work area. This value is zero if the number of bytes is absent.
08	00 nnnn	RCSZ	nnnn — A binary value specifying size of the record to be sorted. This specifies the maximum record size for variable-length records and includes the 4-byte record length field.
09	000000	MERGE	09 — Indicates a merge-only application
0A	00 nnnn	BIN (form 1)	nnnn — A binary number specifying the BIN size for variable-length records
0A	00 nnnn	BIN (form 2)	nnnn — A binary number specifying the minimum BIN size for variable-length records
FF	ssssss		ssssss — A binary number specifying a record size within the file to be sorted
FF	00 v v v v . . .		v v v v — A binary number specifying the number of times this record size occurs in the file or percentage of occurrences
FF FF	ssssss 00 v v v v		Each size-n and freq-n subparameter pair requires two BIN continuation words in the parameter table.

Table B—1. Sort Parameter Table (Part 2 of 3)

Code	Value	Keyword Parameter	Description of Values		
0B FF FF	ii pppp cc qq rr 00 ll bb	FIELD	ii	–	A binary number specifying the length of the field in bytes that are represented as length-n for all fields that are in the byte-bit format. For binary fields, it is defined as true length (0 ≤ ii ≤ 255).
			pppp	–	The location of the first byte of the key field relative to thc start of the record (0 ≤ pppp ≤ 32767)
			cc	–	Binary code of the FIELD form parameter 00 = CH – character 01 = BI – unsigned binary 02 = FI – fixed pointer integer 03 = PD – packed decimal 04 = ZD – zoned decimal 05 = FL – floating point 06 = MC – multiple character, user-specified collating sequence 07 = AC – EBCDIC data in ASCII collation sequence 08 = CSL – leading sign numeric 09 = CST – trailing sign numeric 0A = CLO – numeric data overpunched leading sign 0B = CTO – numeric data overpunched trailing sign 0C = ASL – ASCII numeric data leading sign 0D = AST – ASCII numeric data trailing sign 0E = USQ – user specified collation sequence
			qq	–	Binary code of the field sequence parameter 00 = ascending sequence, A 01 = descending sequence, D
			rr	–	Binary value specifying the order of significance of this field (01 ≤ rr ≤ 255)
			ll	–	A binary value, used only when BI is specified. Specifies a number of bits when the length of a 'BI' field is not an even multiple of bytes (00 ≤ ii ≤ 07)
			bb	–	A binary value used only in B1 fields to specify the first bit location of a B1 field within the byte location specified by pppp (00 ≤ bb ≤ 07)
0C FF	0000 nn aaaaaa	DISC	nn	–	A binary value specifying the maximum number of disk file names that may be assigned (0 ≤ nn ≤ 8)
			aaaaaa	–	The address of the list of user-supplied disk file names
0D	00 nn xx	TAPE	nn	–	A binary value specifying the maximum number of tape file names that may be assigned (0 ≤ nn ≤ 6)
			xx	–	A binary code indicating the tapes label parameter 00 = standard labels, STD 01 = no labels, NO
0E	0000 bb	SHARE	bb	–	Two unsigned decimal digits representing the last two characters of the file name of the tape unit to be shared (SM06, bb = 6 or less)
0F	0000 bb	RESERV	bb	–	Two unsigned decimal digits representing the last two characters of the file name of the tape unit to be reserved (SM06, bb = 6 or less)
14	rrrrrr	RESUME= (PASS,..)	rrrrrr	–	Three-character pass recovery-number parameter

Table B—1. Sort Parameter Table (Part 3 of 3)

Code	Value	Keyword Parameter	Description of Values

1A	00 gg hh	NOCKSM	gg—hh — Binary code indicating checksum to be omitted
			gg = 01 — omit tape checksum
			hh = 01 — omit disk checksum

1B	aaaaaa	USEQ	aaaaaa — The address of a 256-byte translation table specifying the desired collation sequence
FF	dddddd		dddddd — The address of a 256-byte translation table specifying the inverse of the first table
20	000000	PAD	The null PAD entry is used to reserve space in the parameter table. The entry is repeated (bytes+3)/4 times, where bytes in (bytes+3) represents the PAD 'bytes' parameter.
21	0000 jj	CSPRAM	jj — Binary code indicating the CSPRAM option 01 — OPTION 02 — YES 03 — NO
22	0000 nn	ADDROUT	nn — A binary code specifying the tag-sort option 00 — A — return only the direct access address of record 01 — D — return the disk address and the record key fields
25	0000 nn	CALC	nn — A binary code signifying that sort optimization information is to be calculated and displayed. The sort may be executed or terminated. 00 — NO — no execution 01 — YES — execution
26	nnnnnn	SIZE	nnnnnn — A binary number indicating the approximate number of records to be sorted
29	0000 nn	PRINT	nn — A binary code indicating the type of messages to be displayed 00 — ALL 01 — CRITICAL 02 — NONE

Appendix C. Subroutine Sort/Merge Interface Requirements for the COBOL Programmer

The SPERRY UNIVAC Operating System/3 (OS/3) COBOL sort facility uses the capabilities of the subroutine sort/merge for sorting the data fields involved in your COBOL program. The method of defining the requirements for the initialization and execution of the subroutine sort/merge from your COBOL program, however, varies from that described in Section 3 of this manual in that the requirements are specified according to OS/3 COBOL conventions. During compilation of your COBOL program, the OS/3 Extended COBOL compiler expands the COBOL program sort statements into the parameter information required by the subroutine sort/merge for defining the various sort/merge entries of the sort parameter table and for executing the modular elements that make up the subroutine sort/merge routine. If it is necessary to change the requirements of the sort prior to its execution, you have the option to redefine some, but not all, of the compiler-generated parameter information by inserting parameter control statements in the job control stream for your program. You may also introduce certain additional parameters to the subroutine sort/merge in the same manner.

During program execution, a message that questions the presence of parameter statements in the control stream for your program is displayed on the system console. (This display will not occur if you have specified the compiler output option //△PARAM△OUT=S in your program to disable the display feature.) Your response to the question determines whether or not the parameter statements are accessed by the subroutine sort/merge during execution of the sort. The parameters that you may insert into the job control stream allow you to designate the type of sort desired, to reserve or share the use of magnetic tape devices assigned as auxiliary working storage to the sort, to replace the arbitrary BIN size provided by the compiler, to resume an interrupted tape sort, or to inhibit the calculation of a summation check for files written to tape or disk.

Table C—1 summarizes the parameters supplied by the Extended COBOL compiler and indicates those which may be inserted in the job control stream. Figure C—1 depicts a typical job deck for a COBOL program executing a sort and shows the placement of sort parameter statements in the job control stream. Further details of the use of subroutine sort/merge through the OS/3 COBOL sort facility are given in the Extended COBOL supplementary reference manual, UP-8059 (current version).

Table C—1. Extended COBOL Interface with OS/3 Subroutine Sort/Merge (Part 1 of 2)

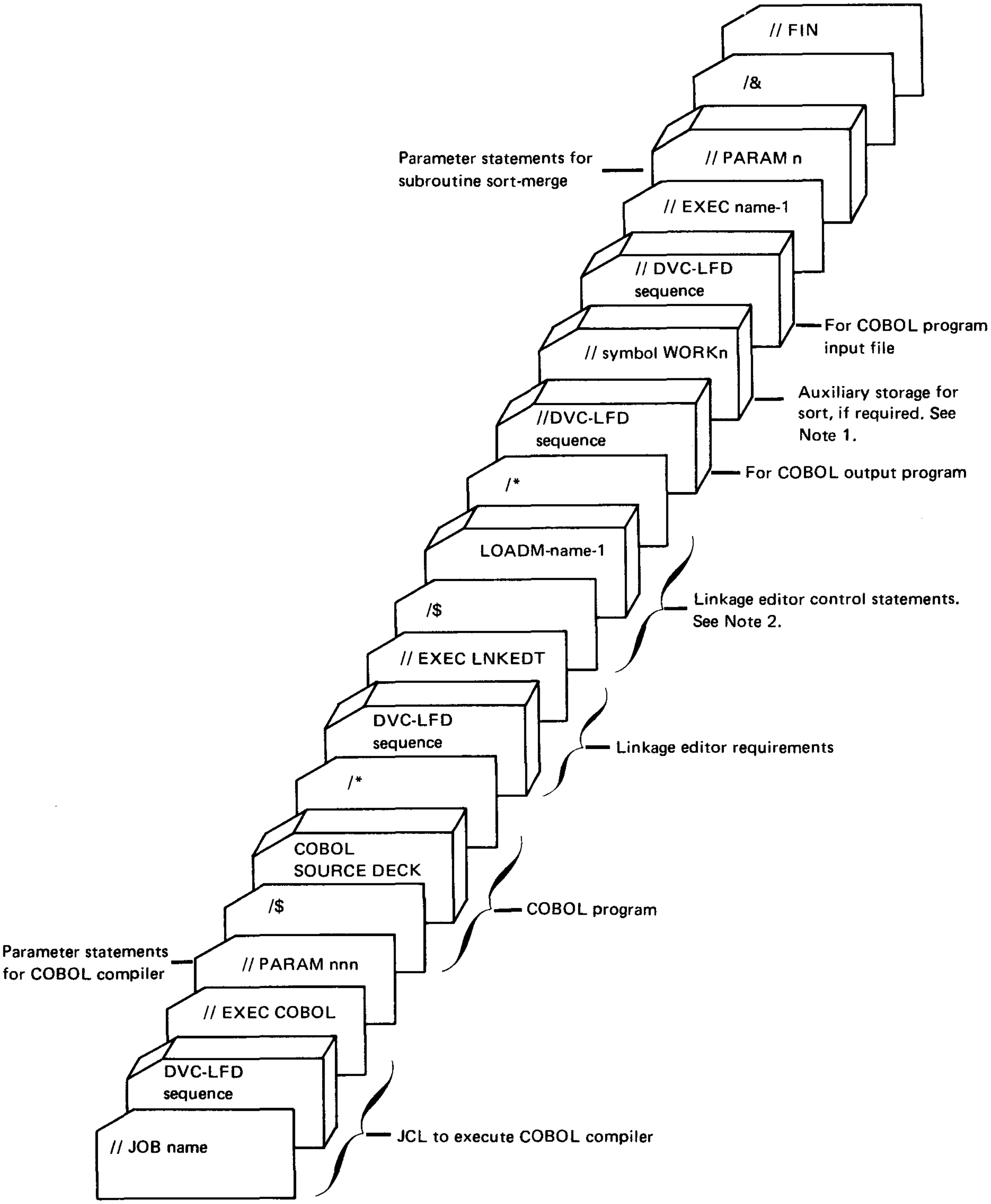
Use of Parameter	Parameter	Execution Requirement for Sort/Merge*	Parameter May Be Passed in Job Control Stream	Compiler Action and COBOL Programmer's Options
Normal sort linkage	DROC	O	No	None. Neither automatic nor user own-code data reduction (DROC) is available to the COBOL programmer.
	FIN	X	No	Provides FIN from the AT END imperative statement of COBOL program RETURN statement
	IN	X	No	Provides address to which sort/merge returns control to the COBOL program after being initialized; this is the address immediately before the file named in the COBOL USING statement that has been opened, or immediately before the PERFORM statement resulting from the INPUT procedure of the SORT statement.
	OUT	X	No	Provides address immediately before the opening of the file named in the GIVING statement, or the PERFORM statement resulting from the OUTPUT procedure in the COBOL SORT statement
	RSOC	O	No	None. User own-code exit for record sequencing (RSOC) is not available to the COBOL programmer.
Facility assignment	DISC	X	Yes	Enters the maximum (DISC=8). May be overridden by the COBOL programmer. Must be overridden by setting DISC=0 when an internal-only sort is specified
	TAPE	X	Yes	Enters the maximum (TAPE=6). May be overridden by the COBOL programmer. Must be overridden by setting TAPE=0 when an internal-only sort is specified
	STOR	X	No	Provides externally defined address KE$ALP, which is defined by linkage editor as the end of the longest phase in the load module. Also provides the amount of main storage available for the sort to use; the sort uses main storage from KE$ALP to the end of job region.
	RESERV	O	Yes	None. COBOL programmer may pass in job control stream.
	SHARE	O	Yes	None. COBOL programmer may pass in job control stream.
Record definition	ADDROUT	O	No	None. Tag sorting is not available to a COBOL program.
	BIN	O	Yes	Provides the length of shortest record described in the sort file description (SD). COBOL programmer may override when the lengths and positions of the key fields in his records justify using a different BIN length to improve the efficiency of the sort.
	FIELD	X	No	Provides FIELD from the information on sort key fields contained in SORT statement in COBOL program
	RCSZ	X	No	Provides RCSZ from the maximum record size described in 01-level entries following sort file description (SD)
	USEQ	O	No	None. To sort data in other than ASCII or EBCDIC, the COBOL programmer will need to use format 3 of the TRANSFORM verb before presenting his data to the sort and afterwards to retranslate it.

*X = Required for execution
O = Optional

Table C—1. Extended COBOL Interface with OS/3 Subroutine Sort/Merge (Part 2 of 2)

Use of Parameter	Parameter	Execution Requirement for Sort/Merge*	Parameter May Be Passed in Job Control Stream	Compiler Action and COBOL Programmer's Options
Tape sort restart	RESUME	O	Yes	None. COBOL programmer specifies when required.
Miscel-laneous	ADTABLE	O	No	Uses ADTABLE to dynamically link parameter table to program
	CALCAREA	O	No	None. The CALCAREA option is not available to the COBOL programmer.
	CSPRAM	O	No	Enters CSPRAM=OPTION, thus permitting the COBOL programmer to make the decision at execution time whether to pass parameters to sort/merge in job control stream
	NOCKSUM	O	Yes	None. COBOL programmer may pass in job control stream
	PAD	O	No	Uses PAD to reserve space for parameter table dynamically linked by compiler
	PRINT	O	No	None. Default value (PRINT=ALL) is assumed by subroutine sort/merge, and all messages generated by the sort are displayed on the system printer.
	SIZE	O	No	None. No means available to COBOL programmer to specify

*X = Required for execution
O = Optional


// FIN
/&
Parameter statements for subroutine sort-merge
// PARAM n
// EXEC name-1
// DVC-LFD sequence
For COBOL program input file
// symbol WORKn
Auxiliary storage for sort, if required. See Note 1.
//DVC-LFD sequence
For COBOL output program
/*
LOADM-name-1
Linkage editor control statements. See Note 2.
/$
// EXEC LNKEDT
DVC-LFD sequence
Linkage editor requirements
/*
COBOL SOURCE DECK
COBOL program
/$
Parameter statements for COBOL compiler
// PARAM nnn
// EXEC COBOL
DVC-LFD sequence
// JOB name
JCL to execute COBOL compiler

NOTES:

1. Symbol specified is either DMnn for disk or SMnn for tape auxiliary storage. If you specify the // WORK jproc, the DVC/LFD sequence set may be omitted. The // WORK jproc or the DVC/LFD sequence set is omitted when an internal sort is specified by // PARAM DISK=0 or // PARAM TAPE=0.

2. Linkage editor control statements prepared by COBOL program should not include a RES statement (as this statement is generated by OS/3 extended COBOL compiler) unless the programmer has specified the compiler output option // △ PARAM △ OUT=L to inhibit generation of linkage editor control statements in the object module.

Figure C—1. Typical Job Deck for OS/3 COBOL Program Executing a Sort via OS/3 Subroutine Sort/Merge

Appendix D. Standard EBCDIC and ASCII Collating Sequences

D.1. GENERAL

Appendix D provides three useful tables containing collating sequences. The first (Table D—1) presents a cross-reference table that enables you to compare the following standard codes commonly used in data processing and in the SPERRY UNIVAC Operating System/3 (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 D—2 provides a convenient chart of OS/3 EBCDIC graphics only, and Table D—3 lists OS/3 ASCII graphics only.

D.2. EBCDIC/ASCII/HOLLERITH CORRESPONDENCE

Table D—1 is a cross-reference table depicting the correspondences among the Hollerith punch 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 holer 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 D—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 D—1).

D.2.1. Hollerith Punch Card Code

The Standard Hollerith punch 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.

D.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 D—1 shows the EBCDIC graphic characters only; the EBCDIC control characters are not indicated.

D.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 D—1.

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

EBCDIC					ASCII	
Decimal	Hexa-decimal	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 D—1. Cross-Reference Table: EBCDIC/ASCII/Hollerith (Part 2 of 5)

EBCDIC					ASCII	
Decimal	Hexa-deci-mal	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 D—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 D—1. Cross-Reference Table: EBCDIC/ASCII/Hollerith (Part 4 of 5)

EBCDIC					ASCII	
Decimal	Hexa-decimal	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 D—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

D.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 (0000000) to high (11111111). 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 collating algebraically; i.e., each quantity is interpreted as having a sign.

Table D—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 D—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	

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

Table D—3 shows the OS/3 collating sequence for ASCII characters and unsigned decimal data. The collating sequence ranges from low (00000000) to high (01111111). 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 D—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 D—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

Index

SPERRY UNIVAC

USER COMMENT SHEET

Your comments concerning this document will be welcomed by Sperry Univac for use in improving subsequent editions.

Please note: This form is not intended to be used 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 along line.

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 UNIVAC

ATTN.: SYSTEMS PUBLICATIONS

P.O. BOX 500
BLUE BELL, PENNSYLVANIA 19424

FOLD

CUT