SPERRY

PUBLICATIONS UPDATE

Operating System/3 (OS/3)

Sort/Merge Macroinstructions

User Guide/Programmer Reference

UP-9072 Rev. 1-A

This Library Memo announces the release and availability of Updating package A to "SPERRY® Operating System/3 (OS/3) Sort/Merge Macroinstructions User Guide/Programmer Reference", UP-9072 Rev. 1.

This update corrects Table 1–2, Comparison of Transfer Rates for Magnetic Tape Devices. It adds two columns to this table.

Copies of Updating Package A are now available for requisitioning. Either the updating package only or the complete manual with the updating package may be requisitioned by your local Sperry representative. To receive only the updating package, order UP-9072 Rev. 1–A. To receive the complete manual, order UP-9072 Rev. 1.

LIBRARY MEMO ONLY	LIBRARY MEMO AND ATTACHMENTS	THIS SHEET IS
Mailing Lists BZ, CZ, and MZ	Mailing Lists B00, B18, 28U, and 29U (Package A to UP-9072 Rev. 1, 9 pages plus Memo)	Library Memo for UP-9072 Rev. 1–A RELEASE DATE: January, 1985

UNISYS

OS/3 Sort/Merge Macroinstructions

Programming Guide

Relative to Release Level 9.0

Priced Item

August 1987

Printed in U S America
UP–9072 Rev. 1

UNISYS

OS/3 Sort/Merge Macroinstructions

Programming Guide

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

Relative to Release
Level 9.0

Priced Item

August 1987

Printed in U S America
UP-9072 Rev. 1

PAGE STATUS SUMMARY

ISSUE: Update B – UP-9072 Rev. 1
RELEASE LEVEL: 9.0 Forward

Part/Section	Page Number	Update Level
Cover		B
Title Page/Disclaimer		B*
PSS	1	B
Preface	1, 2	Orig.
Contents	1 2 thru 4	A Orig.
1	1, 2 3 thru 5 6 thru 11	Orig. A Orig.
2	1 thru 49	Orig.
3	1 thru 6	Orig.
4	1 thru 14	Orig.
5	1 thru 14	Orig.
Appendix A	1 thru 3	Orig.
Appendix B	1 thru 4	Orig.
Appendix C	1 thru 11	Orig.
Appendix D	1 thru 13	Orig.
Index	1 thru 7	Orig.
User Comment Form		

Part/Section	Page Number	Update Level

Part/Section	Page Number	Update Level

*New pages

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

Preface

This manual is one of a series designed to instruct and guide the programmer in the use of the SPERRY Operating System/3 (OS/3). It specifically describes the use of the subroutine sort/merge program available to the System 80 user of OS/3. The intended audience is the experienced programmer with a thorough knowledge of data processing and programming experience in basic assembler language (BAL).

An introductory manual, the introduction to sort/merge, UP-8835, also is available. It briefly describes the general characteristics and facilities offered by the sort/merge programs for OS/3.

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

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

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

- Consolidated data management concepts and facilities, UP-9978

  Describes the organization and record formats of various file types.

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

  Describes the data management macroinstructions

- Basic data management user guide, UP-8068

  Describes the effective use of OS/3 basic data management

- Job control user guide, UP-9986

  Describes the job control language used under OS/3.

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

  Describes the OS/3 general editor (EDT).

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

- Section 1. Basic Concepts

  Introduces you to subroutine sort/merge, gives you some program restrictions and considerations, tells you how to structure your data, and describes the operational phases.

- Section 2. Sort/Merge Requirements You Supply

  Describes what you must do to run your sort/merge program, including:

  - initiating the operation;
  - defining files;
  - explaining run requirements (MR$PRM macro);
  - getting data into and out of the sort process;
  - assembling, linking, and executing your program; and
  - submitting sort parameter table entries via the job control stream.

- Section 3. User Own-Code Routines

  Describes the record sequence own-code routine (RSOC) and the data reduction own-code routine (DROC).

- Section 4. Special Applications

  Describes tag sorts, restart facilities, and the merge-only function.

- Section 5. Program Examples

  Provides program examples for a tape sort, a tape sort with restart, a tape sort using an own-code routine, and an internal sort.

The following appendixes are also included:

- Appendix A. Statement Conventions
- Appendix B. Contents of Sort Parameter Table
- Appendix C. Standard EBCDIC and ASCII Collating Sequences
- Appendix D. Summary of Sort/Merge Macroinstructions

Contents

5. PROGRAM EXAMPLES

APPENDIXES

A. STATEMENT CONVENTIONS

B. CONTENTS OF SORT PARAMETER TABLE

C. STANDARD EBCDIC AND ASCII COLLATING SEQUENCES

D. SUMMARY OF SORT/MERGE MACROINSTRUCTIONS

INDEX

USER COMMENT SHEET

FIGURES

TABLES

1. Basic Concepts

1.1. GENERAL

Subroutine sort/merge is a modular sort program that requires a good working knowledge of basic assembler language (BAL) and data management macroinstructions. It is not a *canned* service program; i.e., you will have to assemble, link, and execute subroutine sort/merge in addition to programming many of the other activities that are automatically performed by canned programs. However, this allows you to write the sort program you want using the modules provided by subroutine sort/merge and gives you greater control over the sort process, including flexibility in specifying:

- external input record formats;
- sources of input records;
- external output record formats; and
- disposition of final output records.

When using subroutine sort/merge, you get directly involved with job control, data management, and the assembler. Your responsibilities include:

- providing the routines for inputting and outputting data;
- establishing the necessary communication links between your program and subroutine sort/merge;
- opening and closing files;
- defining files;
- reserving buffer areas;
- manipulating register addresses;
- delivering data to and retrieving data from the sort; and
- 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. BAL serves as the medium through which you establish the communications link to subroutine sort/merge.

When we refer to subroutine sort/merge as a *modular* program, we mean that rather than writing separate sort programs for every conceivable type of sort, we have broken the sort 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 (YLOD).

The implementation of load modules 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 of the advantages 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.

In addition to the sort modules, the subroutine sort/merge has a call module to interface each sort module or routine with the system. This call module is the sort common module (SG$ORT), which resides in the system object library file (YOBJ).

If you wish to copy subroutine sort/merge into your own user library file, you can do so by means of the librarian, as described in the system service programs (SSP) user guide, UP-8841 (current version). Be sure to include the following modules:

- Sort common module (SG$ORT) from the system object library file (YOBJ)
- Five sort/merge macroinstructions beginning with MR$ and two merge-only macroinstructions beginning with MG$ from the system macro library file (YMAC)

1.2. WHAT SORT/MERGE DOES FOR YOU

Subroutine sort/merge assists you in producing an ordered, tailored output file from your existing input data files. Through the sorting and merging techniques employed in subroutine sort/merge, you can:

- sort records in ascending or descending sequence;
- sort fixed-length or variable-length records;
- sort records with noncontiguous key fields;
- recognize key fields in the following formats:
  - character
  - binary (signed or unsigned)

- decimal (signed zoned and unsigned zoned)
- packed decimal
- 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 successful execution of your job results in a terminated normally indication 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 from the sort.

1.3. PROGRAM RESTRICTIONS AND CONSIDERATIONS

Variation in program design, capability, and implementation sometimes restricts the use of a subroutine sort program for specific applications or for specific system configurations. Consideration should be given to the following:

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

■ Input and output files can be diskette, disk, card, or tape.

■ Auxiliary storage work areas can be either disk or tape, but not both. Subroutine sort/merge is limited to eight disk files or six tape files.

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

■ User own-code routines can be substituted for the sort routines provided only if they satisfy the requirements of the program and OS/3 programming conventions.

■ Subroutine sort/merge can only be executed in a batch environment; i.e., it cannot be initiated from a workstation.

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

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

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

1.3.1. 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 to main storage requirements reduces processing time and increases program efficiency.

Subroutine sort/merge requires a minimum of 12,400 bytes of main storage. 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 entirely in main storage) requires sufficient main storage to hold the entire input file, plus eight bytes for each record, in addition to the preceding requirements.

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

1.3.2. Auxiliary Storage Work Area Assignments

Work areas may be assigned as auxiliary storage on either tape or disk, but not both. If disk storage is used, all work area disks must be 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 subroutine sort/merge programs. The amount of disk space requested must be sufficient to hold the entire volume of data to be sorted, plus 10 to 20 percent additional space for overhead requirements. (An additional 10 to 20 percent space should be requested if data involves variable-length records.) In addition, all disk files used in the sort operation must be the same type; i.e., mixed disk types are unacceptable. Table 1–1 contains data capacities and access speeds of the direct access storage devices.

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

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

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

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

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

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

1.3.3. 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.

1.3.4. Sort Options

When using subroutine sort/merge, there are two optional parameters in the MR$PRM macroinstruction that can affect sort/merge performance time:

- NOCKSM
- 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 again increase sort/merge execution time.

1.4. STRUCTURING YOUR INPUT/OUTPUT DATA

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

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

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

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

Figure 1—1. Input Data Records before Sort

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

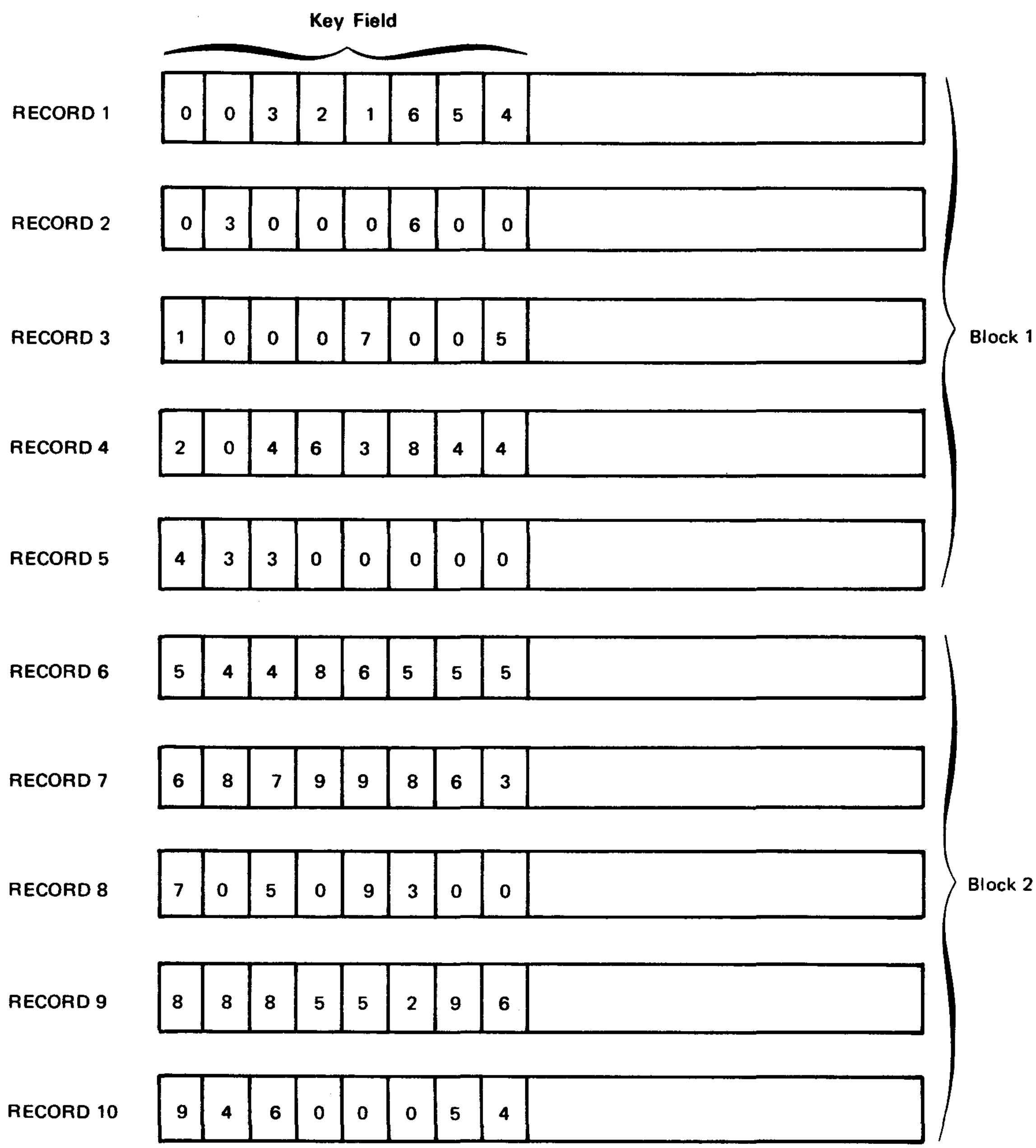


Figure 1—2. Data Records after Sort

1.5. ELEMENTS AFFECTING PERFORMANCE OF A SORT PROGRAM

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

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

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

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

As a rule, simplfication 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.

1.6. SORT/MERGE OPERATION

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 (YLOD) located on the SYSRES volume. When your program activates the subroutine sort/mege, it calls the appropriate sort/merge modules into main storage as they are needed.

To call subroutine sort/merge, you must first link the SG$ORT object module to your program. This module resides in YOBJ and is automatically linked to your program when you specify the label MR$ORT as an EXTRN. SG$ORT initiates subroutine sort/merge when the MR$OPN macroinstruction is executed.

Before sorting can begin, your program must open the input file and read the input file records from the appropriate input device. Your program reads input records, block by block, 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. This increase occurs because we can read records into one buffer while we empty the other buffer into a work area for further processing.

As the input records are read, they are passed to subroutine sort/merge, where they are sorted into sequenced strings of data and stored on disk work files. These strings are then repeatedly merged to produce a single string of sorted data.

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.

1.7. SORT/MERGE OPERATIONAL PHASES

The subroutine sort/merge operation consists of two or 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 modules needed during that phase are loaded into main storage and executed. The phases and the functions they perform are discussed in the following subsections.

1.7.1. Sort Initialization and Assignment

The sort initialization and assignment phase is always executed first. It collects and analyzes all information required by the phases that follow it to determine the overall sort/merge requirements. It extracts this information from the data your program provides via parameter statements either in the MR$PRM macroinstruction or the PARAM job control statement. When the assignment function is completed, control normally passes to the initial sort phase, except in a merge-only procedure, in which case, control passes to the final merge phase.

1.7.2. Initial Sort

In this phase, 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). The records are accumulated in sequential lists called *record strings*. These strings are then written out to *disk* or *tape*. If you assign insufficient auxiliary storage to your program, the sort job step terminates.

1.7.3. Preliminary Merge

The preliminary merge phase is initiated by the release of the last record to subroutine sort/merge, which then repeatedly merges the record strings produced during the intitial sort 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, subroutine sort/merge passes control to the final merge phase.

If the record strings produced during the initial sort phase can be sequenced in one merge pass, preliminary merge is unnecessary and it is bypassed. This occurs when input to the initial sort 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, the preliminary merge is skipped and control passes from the initial sort phase to the final merge phase.

1.7.4. Final Merge

This phase performs the final merge of the sequenced record strings and produces a single string of sorted records. These records are then made 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.

2. Sort/Merge Requirements You Supply

2.1. GENERAL

Subroutine sort/merge (which we'll call simply *sort/merge*) provides greater control over the sort/merge process than other sort programs available with OS/3. Naturally, the benefits you receive for this control cost something – an increase in the programming you must do. You will have to program many of the activities that are done automatically by other *canned* sort/merge programs.

Writing your own routines requires a good working knowledge of basic assembler language (BAL) and data management macroinstructions.

In this section, we'll discuss a disk sort/merge program showing the use of BAL instructions and data management macros.

To activiate sort/merge, you use sort/merge macroinstructions, which you code as part of your program. These instructions bring the sort/merge into your program as they are needed. You will also need a job control stream consisting of control statements that:

- name the devices used by your program and the sort/merge modules;
- describe label and space allocations; and
- call for the execution of assembly and linkage editor routines as they are needed.

An easy way to remember the 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 sort/merge:

I Initiate the operation.

D Define files.

E Explain sort/merge run requirements.

A Activate sort/merge services.

S Stop or end the sort/merge process.

Before coding any program, a flowchart is helpful. The flowchart for a disk sort program might look like Figure 2-1.

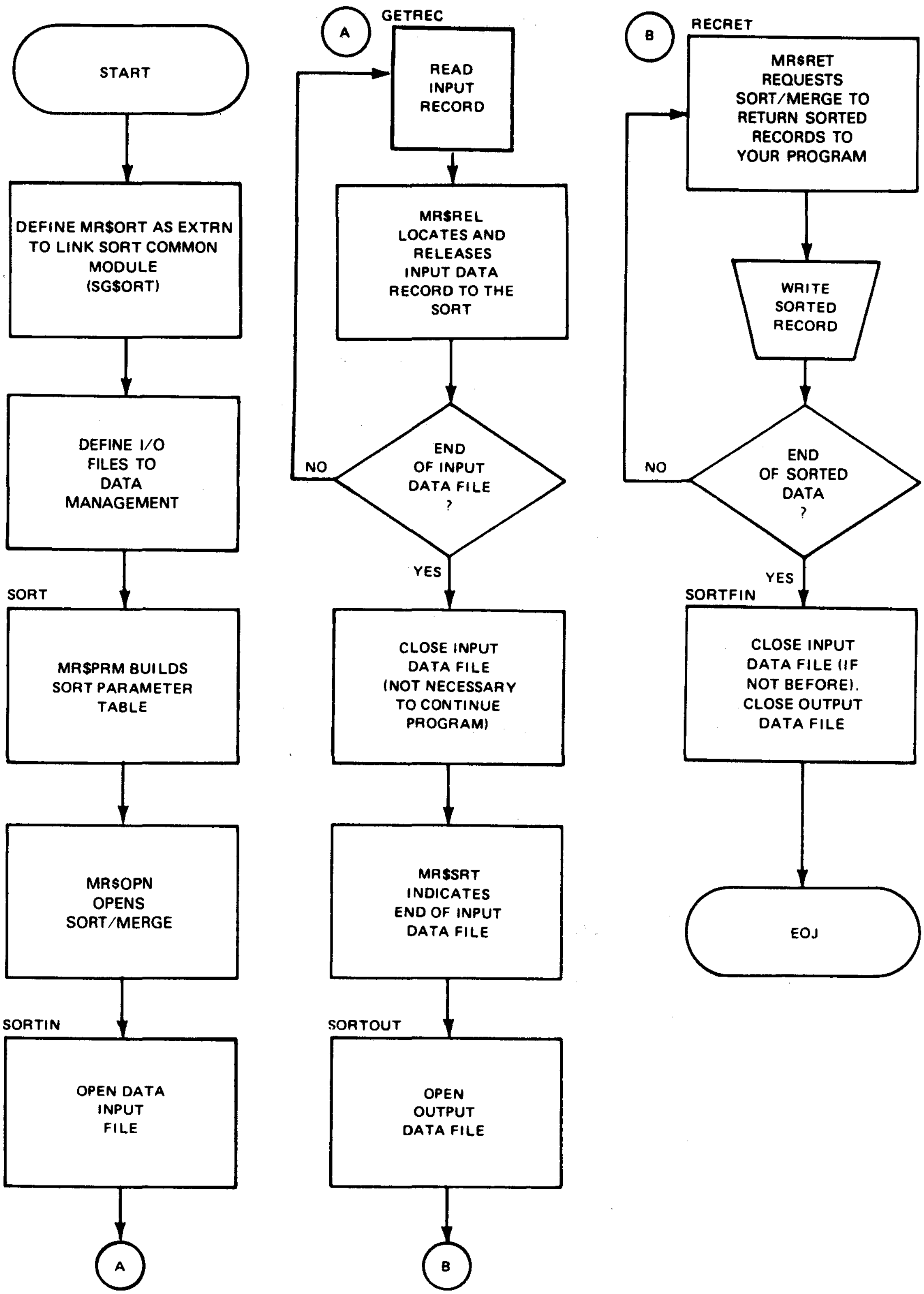


Figure 2—1. Disk Sort Program Flowchart

2.2. 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.

```
1        10    16
SRTEXMPL START 0
```

Part of initiating sort/merge is to establish a communications interface between your program and the 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 (YOBJ). To establish the communications 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 YOBJ 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 0
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 that 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 (YRUN). 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 the sort initialization and assignment phase into main storage, and the sort common module remains there for the duration of sort/merge processing and provides a link between your program and the sort.

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:

```
BALR  4,0
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:

```
1        10     16
-----------------------------------------------------------------
         B      START
```

START is the label of the first instruction in our 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
```

2.3. DEFINING FILES

For model 8 users, software supplied by SPERRY includes consolidated data management (CDM) and basic data management (BDM). These are groups of routines that handle several types of data functions, such as sequential or random processing. Features such as CDI (common data interface), used in CDM, and DTF (define the files), used in BDM, are applicable to such processing. For additional information, refer to the current versions of consolidated data management concepts and facilities, UP-9978, or basic data management user guide, UP-8068. When using subroutine sort/merge, you must provide your own I/O routines. Each record is read in order of its physical location on tape or disk (subroutine tape sorts are shown in Section 5). To operate properly, data management needs specific information defining your program's data files.

File definitions such as record size, record format, and buffer size, in addition to other file information, must appear in your program in the form of resource information blocks (RIBs). You specify an RIB with the RIB macroinstruction. Only those RIB parameters that cannot assume default values need be specified. Take note, however, that one RIB does not necessarily define one file; this is the function of the common data interface block (CDIB), one of which you specify for each file used in your program.

The CDIB is specified by a CDIB macroinstruction. You also use macroinstructions to link the CDIB of a file with the attributes of that file (record size, record format, etc) as defined in an RIB. Data management permits you to link two or more CDIBs to a single RIB that correctly defines all the files involved. Data management uses the information contained in your CDIBs and RIBs to supply file information to the system when your program requires it. You can find the descriptions and formats of the RIB and CDIB macroinstructions in the consolidated data management macro language user guide/programmer reference, UP-9979 (current version).

The coding for the input and output file definitions for our sample disk sort program is illustrated in Figure 2-2.

```
1        10    16                                                    72
SORTRIB  RIB   BFSZ=512,RCSZ=80,IOA1=BUFF1,IOA2=BUFF2,WORK=YES,       X
               RCFM=FIX,OPTN=YES,MODE=SEQ
INPUT    CDIB
OUTPUT   CDIB
```

Figure 2—2. Data Management Macroinstruction Specifications

In this disk sort, we're specifying two files named INPUT and OUTPUT, each with a CDIB macroinstruction. In addition, we specify an RIB labeled SORTRIB with the following characteristics:

- 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;
- a record format of fixed blocks; and
- sequential access.

In addition, we're specifying the optional parameters WORK and OPTN. WORK indicates that all input and output operations take place using data contained in a work area. As you will see, you must specify the symbolic address of that work area in every input or output macroinstruction in the program. OPTN=YES indicates that all files associated with SORTRIB are optional; i.e., you won't always use them. We're coding only one RIB because it defines the characteristics of both files INPUT and OUTPUT.

```
1        10    16                                                    72
         USING CD$CDIB,5
         VTOC  CDIB=YES
```

For I/O processing later in the program we will need to attach labels to certain areas within the two CDIBs. Using their standard names, these labels are:

- CD$CDIB - the address of the first byte within a CDIB.
- CD$ISUCC - a CDIB bit set after an I/O operation to indicate whether the operation was successful (set to 1) or unsuccessful (set to 0).
- CD$IEOF - a CDIB bit set after an input operation to indicate that data management has reached the end of the file (set to 1) or that more records remain (set to 0).

You won't have to concern yourself with the exact location of these indicators if you use the VTOC macroinstruction. When coded with the keyword parameter CDIB=YES, it generates a DSECT, which in effect is a map of the CDIB. The USING directive coded just above VTOC associates register 5 with the address of the first CDIB byte, CD$CDIB. Statements generated by VTOC then fix the indicators CD$ISUCC and CD$IEOF as offsets from register 5. All you need to do, as a result, is to load register 5 with the address of any actual CDIB, and you can then test the bit indicators within the CDIB as symbols rather than having to know where exactly within the CDIB they lie. The use of register 5 with VTOC does not affect register 4 since the remainder of the program continues to use register 4 as its base register.

When you specify IOA1 and IOA2, you must also define how much main storage is required to handle the block size you indicated on the BFSZ parameter. Thus, somewhere in your program you write the *define storage* statements as illustrated in lines 2 and 3 of the following coding:

```
    1        10    16
1.           DS    0H
2.  BUFF1    DS    CL512
3.  BUFF2    DS    CL512
```

Not only does data management associate IOA1 and IOA2 with their names, BUFF1 and BUFF2, but it also looks for the needed space, indicated by character length 512 (CL512). This data management space accommodates alternate input and output block processing. If your input files are on 8417 or 8419 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. I/O buffers must be half-word aligned (line 1 in the preceding coding). Up to this point, our coding looks like Figure 2-3.

```
     1        10     16                                                      72
1.   SRTEXMPL START  Ø
2.            EXTRN  MR$ORT                       DEFINES MR$ORT AS AN EXTRN
     .                                            LINKS SORT COMMON MODULE TO
     .                                            YOUR PROGRAM
3.            BALR   4,Ø
4.            USING  *,4
5.            B      START
6.   SORTRIB  RIB    BFSZ=512,RCSZ=8Ø,IOA1=BUFF1,IOA2=BUFF2,WORK=YES,        X
7.                   RCFM=FIXBLK,OPTN=YES,MODE=SEQ
8.   INPUT    CDIB
9.   OUTPUT   CDIB
10.  USING    CD$CDIB,5
11.  VTOC     CDIB=YES
19.           DS     0H
20.  BUFF1    DS     CL512
21.  BUFF2    DS     CL512
22.  INOUTBUF DS     CL8Ø
```

Figure 2—3. Sort/Merge Disk Sort Coding

2.4. EXPLAINING RUN REQUIREMENTS TO SORT/MERGE

Your program must describe its requirements for sorting to sort/merge. You use the MR$PRM macroinstruction to do this. Sort/merge uses the information specified by MR$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$PRM macroinstruction has many parameters, we are going to show its format in two parts. The first part illustrates only the required parameters. After discussing the use of these parameters, the second part of the format shows the optional parameters followed by a discussion of their use. To complete the explanation, 2.4.3 explains the MR$PRM parameters for our sample disk sort program. Finally, 2.10 provides the entire MR$PRM macroinstruction format and Table 2–2 summarizes the use of the parameters.

2.4.1. Required MR$PRM Parameters

The MR$PRM format for the required parameters is:

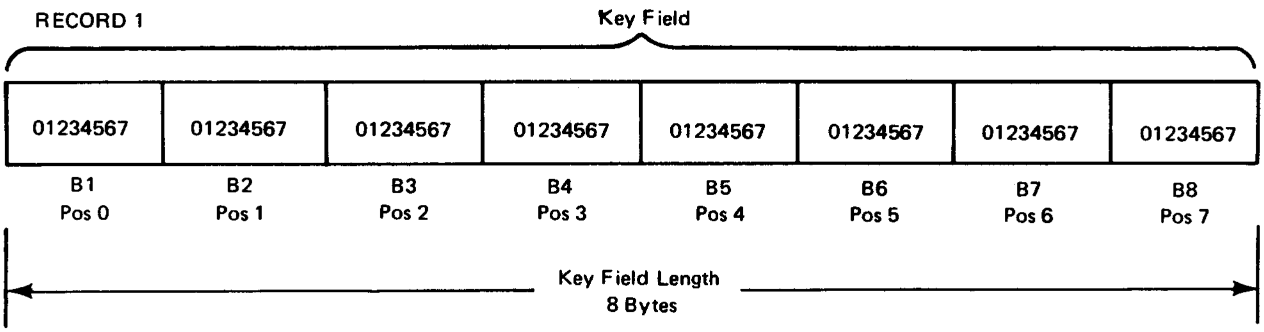
LABEL	△OPERATION△	OPERAND
[symbol]	MR$PRM	{FIELD=(strt-pos-1,lgth-1[,form-1][,seq-1] [,order-1][,...,strt-pos-n,lgth-n [,form-n][,seq-n][,order-n]]) / RSOC=symbol},
		FIN=symbol,
		IN=symbol,
		OUT=symbol,
		RCSZ=max-bytes,
		STOR={symbol / (symbol,number-of-bytes)}

- Defining sort key fields (FIELD)

  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 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 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 the record layout in Figure 1-1 as an example, 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 2-4).

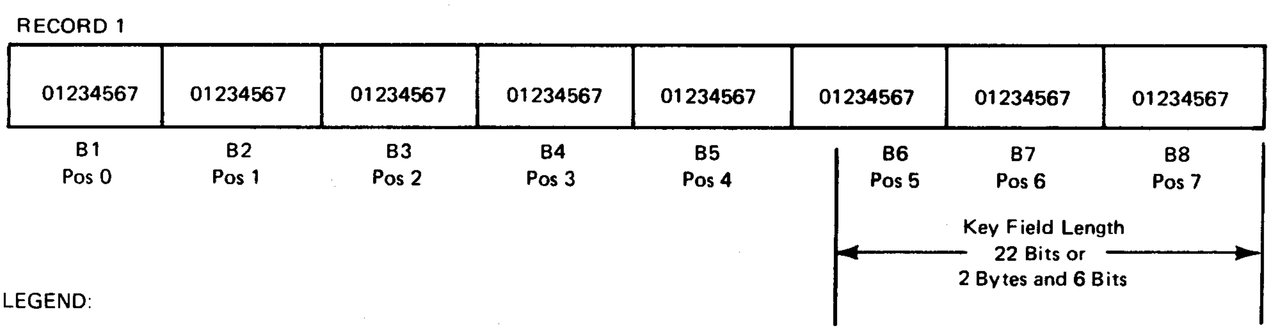


LEGEND:

B - Byte
Pos - Position

Figure 2—4. 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 2-5).



LEGEND:

B - Byte
Pos - Position

Figure 2—5. 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 2-4). 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:

```
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 2-5, 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 2-1, the default would be CH for character code (*form*).

Table 2—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 2—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 12 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 12 key fields. If you omit the *order* subparameter, 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. Line 3 shows three key fields with varying data formats to be sorted in ascending sequence.

```
    1       10    16
1.          MR$PRM FIELD=(Ø,7,AC,D)
2.          MR$PRM FIELD=(5,8,,,2,16,3,,,1,58,1Ø,,,3)
3.          MR$PRM FIELD=(85,3,PD,,88,3,PD,,8,9,CH)
```

- Writing your own record sequencing routine (RSOC)

  If you elect to write your own routine for record sequencing, you would choose the record sequencing own-code parameter (RSOC) instead of specifying the FIELD parameter. You would code RSOC and the symbolic name of your own-code routine. For example:

```
1       10    16
        MR$PRM RSOC=MYROUT
```

  This parameter overrides the FIELD parameter if you specify both FIELD and RSOC.

- Naming your output end-of-data routine (FIN)

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

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

- Specifying your program's entry address (IN)

  The macro that initializes sort/merge is discussed in 2.5. Once initialization is complete, 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 macroinstruction:

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

- Specifying the return address (OUT)

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

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

- Specifying record size (RCSZ)

  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.

```
        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.

- Allocating main storage (STOR)

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 sort/merge modules and operations.

Using the STOR parameter, you can indicate either:

1. the symbolic name of the first main storage location available for sort/merge; or

2. 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, 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 2-6).

```
1           10     16
-------------------------------------------
            MR$PRM  STOR=WORK
```

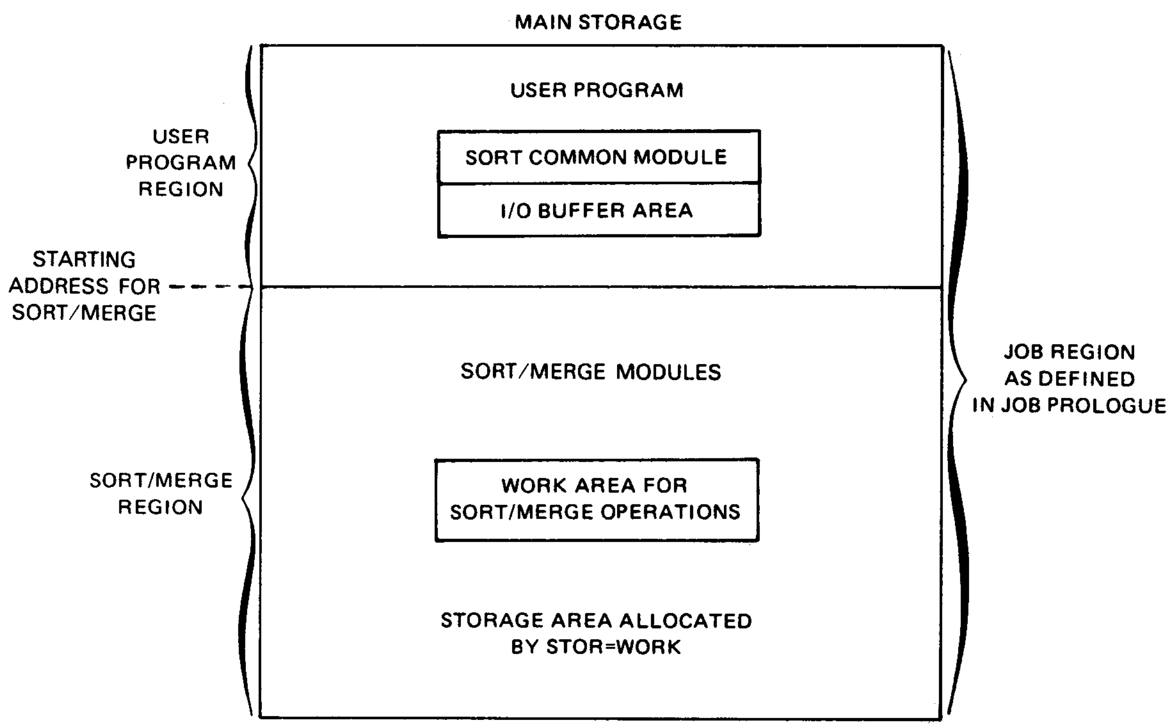


Figure 2—6. Main Storage Area Allotted by STOR without Number of Bytes Specified

If, for example, you specify a maximum number of main storage bytes by writing STOR=(WORK,15000), the main storage area allocated for sort/merge would extend 15,000 bytes from your starting address of WORK. Main storage space allocation would look like Figure 2-7.

```
1            10      16
             MR$PRM  STOR=(WORK,15000)
```

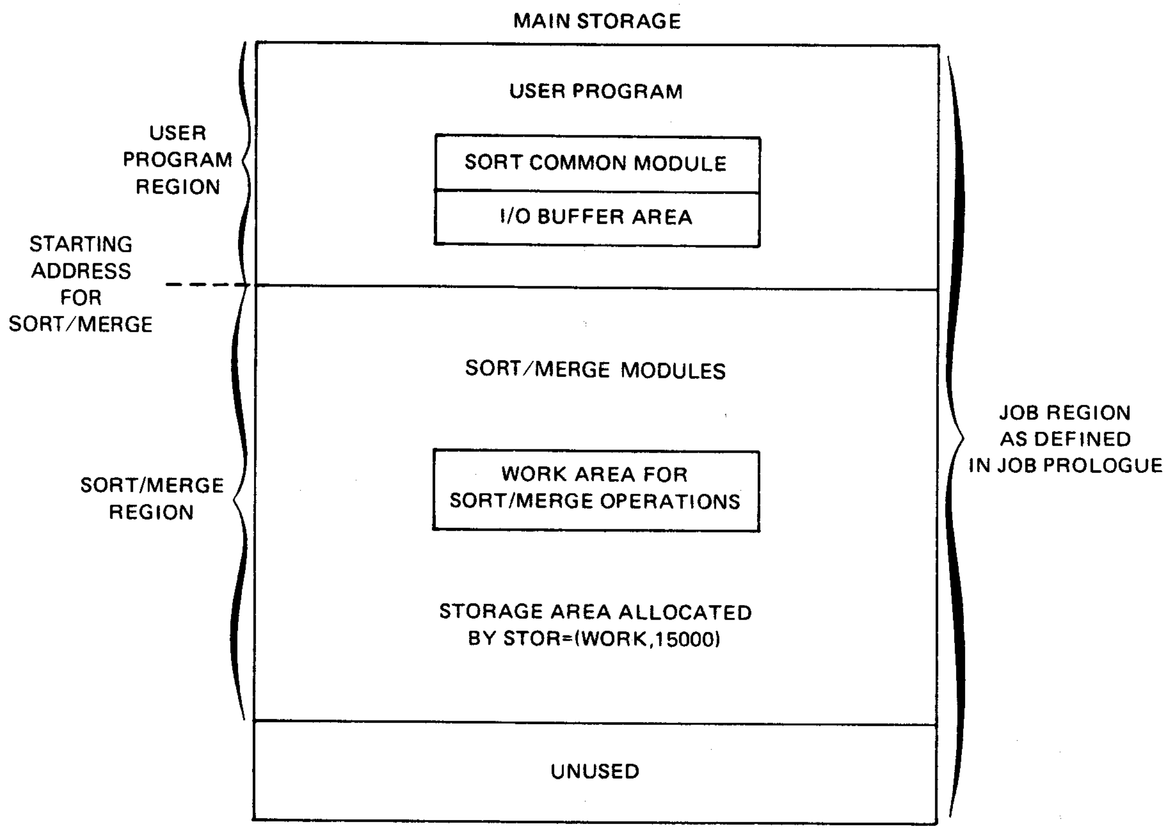


Figure 2—7. Main Storage Area Allotted by STOR Specifying Maximum Number of Bytes

If you use the STOR parameter to specify the amount of main storage available to sort/merge, be sure to allocate a sufficient amount. See 1.3.1 for minimum main storage requirements.

2.4.2. 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	[,ADDROUT={A,D}] [,ADTABL=symbol] [,BIN={bytes, (min-bytes,size-1,freq-1[,...,size-n,freq-n])}] [,CALC={NO, YES}] [,CSPRAM={[redacted], YES}] [{,DISC={(address,max-disk-file-number), max-disk-file-number}, ,TAPE={label-type, (label-type,max-file-number)}}] [,DROC={DELETE, symbol}] [,MERGE={[redacted], YES}] [,NOCKSM={D, T}] [,PAD=bytes] [,PRINT={[redacted], CRITICAL, NONE}] [,RESERV=sort-filename] [,RESUME=(PASS,recovery-number)] [,SHARE=sort-filename] [,SIZE-number] [,USEQ=(to-address,from-address)]

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.

2.4.2.1. Device Assignment Parameters

Parameters used to define devices include:

LABEL	△OPERATION△	OPERAND
[symbol]	MR$PRM	[{ DISC={ (address,max-disk-file-number) / max-disk-file-number } / TAPE={ label-type / (label-type,max-file-number) } }] [,RESERV=sort-filename] [,SHARE=sort-filename]

- Identifying the storage medium (DISC and TAPE)

  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 8. 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 sort/merge. Line 2 of the following coding shows a maximum of seven disks to be used for work files.

```
   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 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.

```
   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, 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.

- Reserving a tape unit (RESERV)

  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 the initialization phase, the initial sort phase, and the preliminary merge phase. At the beginning of the final merge phase when sort/merge transfers control to your program at the address specified in the OUT parameter, the work file is closed and rewound to the unload point. After you demount it and mount your data output file, the reserved tape unit accepts your output file on the same device. You might specify a standard tape sort file name (SM01,...,SM06) as follows:

  ```
  MR$PRM RESERV=SM04
  ```

- Sharing a tape unit (SHARE)

  The SHARE parameter allows a tape unit assigned to sort/merge to be used (shared) as a device for an input file during the initial sort phase 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:

  ```
  MR$PRM SHARE=SM01
  ```

  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.

2.4.2.2. Record Definition Parameters

The following parameters define records:

LABEL	△OPERATION△	OPERAND
[symbol]	MR$PRM	[,ADDROUT={A D}] [,BIN={bytes (min-bytes,size-1,freq-1[,...,size-n, freq-n])}] [,USEQ=(to-address,from-address)]

- Performing a tag sort (ADDROUT)

The ADDROUT parameter is related to a special sort application called a tag sort. The tag sort is a method of sorting in which the output file contains only the direct access addresses, or the addressses 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. Multiple input files cannot be tag sorted. When you want to perform a tag sort, you tell sort/merge via the ADDROUT parameter:

```
1       10    16
---------------------------------
        MR$PRM ADDROUT=A
        MR$PRM ADDROUT=D
```

The ADDROUT parameter has two options:

D

Specifies that both the address field and the record key fields are returned to your program in the sorted record.

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. 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 orginal record at a later time. Figures 2-8 and 2-9 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 these figures, but only to illustrate the concept of record sorting by key fields and the outputs produced by a tag sort operation.

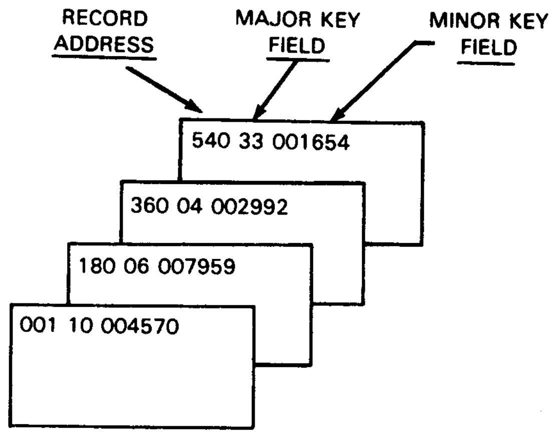


Figure 2—8. Input File, Unsorted Records (Additional Data Fields Not Shown)

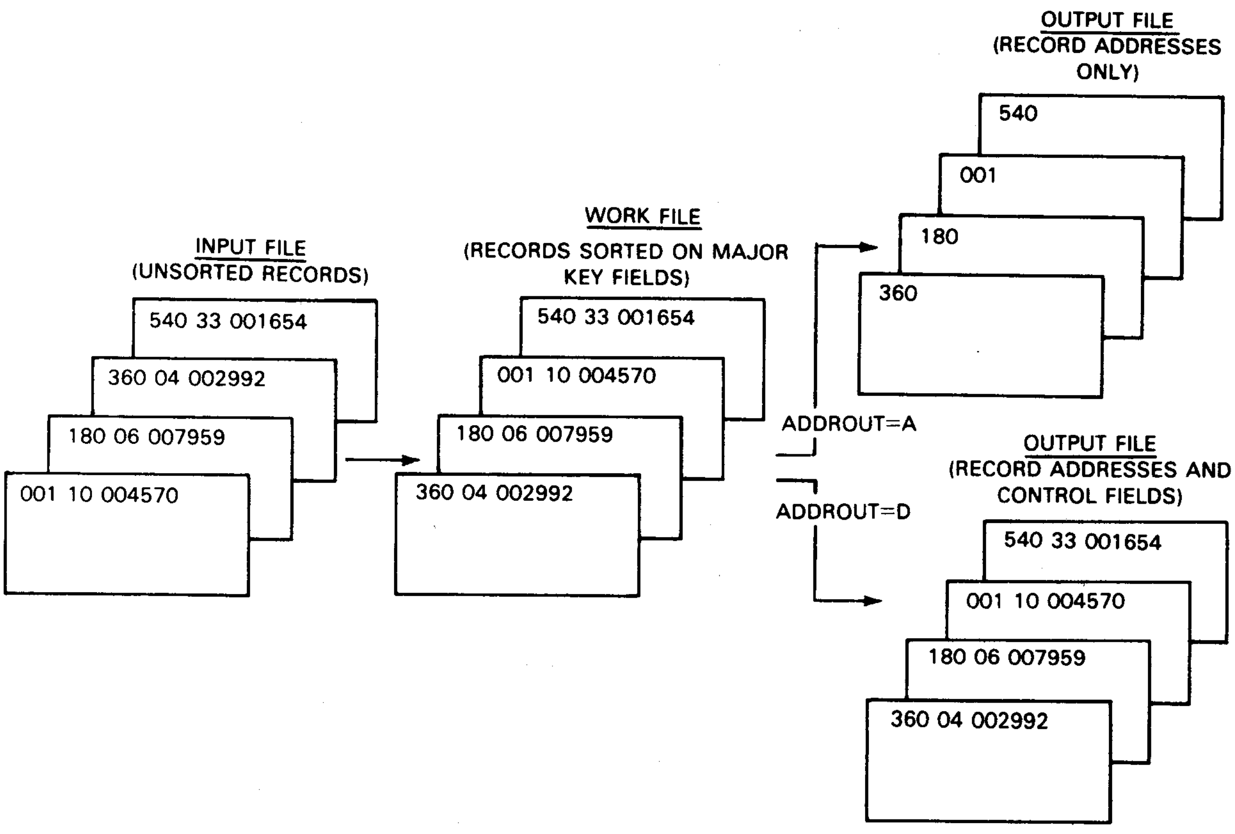
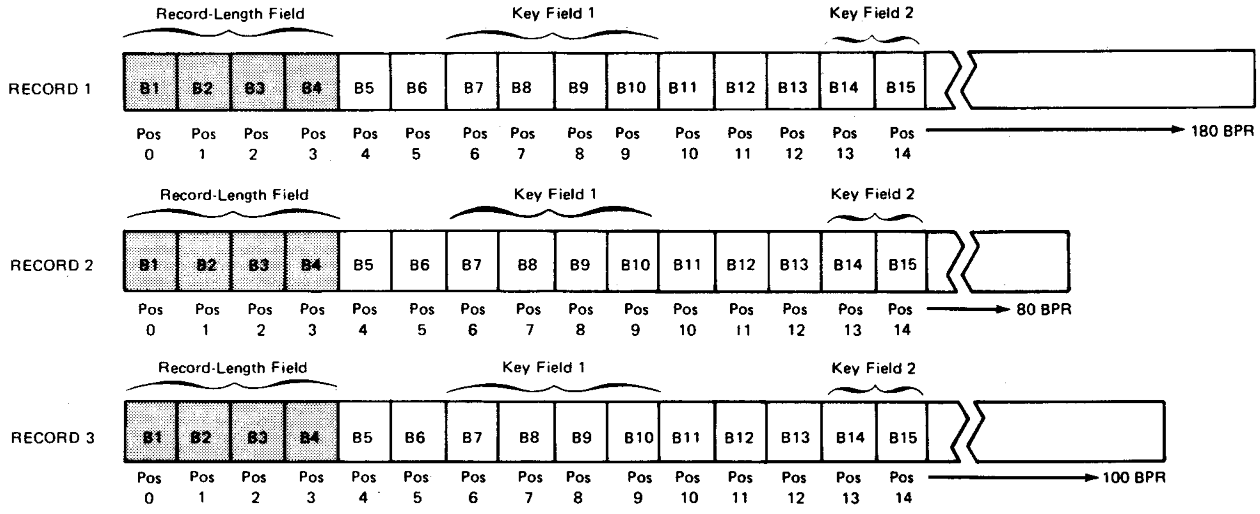


Figure 2—9. Tag-Sorted Output Files

- Specifying fixed-length subrecord size (BIN)

  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, sort/merge divides variable-length records into fixed-length subrecords called *bins*. Remembering that a 4-byte *record-length* field is considered as part of variable-length records, several of them with key fields might look like Figure 2–10.



LEGEND:

B - Byte
BPR - Bytes per record
Pos - Position

Figure 2—10. Variable-Length Records 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 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 2–10 to determine the number of bytes for the format 1 BIN parameter, notice that each record contains two key fields that 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.

```
1       10    16
----------------------------
        MR$PRM  BIN=20
```

Suppose you have the same record information from Figure 2–10 but you decide to let sort/merge calculate the bin size. To calculate this number, sort/merge needs:

- the minimum number of bytes that 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 of percentage of *size-1* records in the input file *(freq-1)*. If the number specified is less than 100, sort/merge assumes it to be 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.

```
1        10      16
         MR$PRM  BIN=(15,180,50,80,200,100,300)
```

- Specifying your own collation sequence (USEQ)

In our discussion of the FIELD parameter, we learned that there are many format codes used to perform collation sequences (Table 2-1). 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 macroinstruction.

```
         MR$PRM  FIELD=(0,8,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=(0,8,USQ),
                 .
                 .
                 .
                 USEQ=(MYCODE,MYCODE)
```

2.4.2.3. 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	△OPERATION△	OPERAND
[symbol]	MR$PRM	[,RESUME=(PASS,recovery-number)]

Example:

```
1          10      16
           MR$PRM  RESUME=(PASS,Ø53)
```

Instead of coding RESUME on the MR$PRM macroinstruction and having to reassemble your program, you can enter it from the job control stream by submitting a PARAM job control statement (2.12), 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 macroinstruction (2.4.2.4).

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

2.4.2.4. Miscellaneous Parameters

The remaining optional parameters are:

LABEL	△OPERATION△	OPERAND
[symbol]	MR$PRM	[,ADTABL=symbol]
		[,CALC={NO | YES}]
		[,CSPRAM={NO | YES}]
		[,DROC={DELETE | symbol}]
		[,MERGE={NO | YES}]
		[,NOCKSM={D | T}]
		[,PAD=bytes]
		[,PRINT={ALL | CRITICAL | NONE}]
		[,SIZE=number]

- Generating and linking additional parameter tables (ADTABL)

  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 sort/merge ignores all parameter entries following the ADTABL parameter (Figure 2-11). 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$PRM macroinstruction, you must create another sort parameter table in the current program or reference a sort parameter table from another program. To link tables within the same program later in the program, you indicate the symbolic label specified on the ADTABL and write a MR$PRM there as follows (line 13).

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

Figure 2—11. 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 2-12). If duplicate fields exist in the two parameter tables, the first occurrence is used.

```
        1        10      16                                                72
        SORT     MR$PRM  FIELD=(Ø,8),                                      C
                         IN=SORTIN,                                        C
                         OUT=SORTOUT,                                      C
                         FIN=SORTFIN,       FIRST PROGRAM SORT             C
                         RCSZ=8Ø,           PARAMETER TABLE                C
                         STOR=WORK,                                        C
                         PAD=16,                                           C
                         ADTABL=MYTABL
                 EXTRN   MYTABL
ADDED    SRTAB   MR$PRM  FIELD=(12,4),                                     C
AFTER                    IN=SORTIN,                                        C
FIRST                    OUT=SORTOUT,                                      C
TABLE                    FIN=SORTFIN,       SECOND PROGRAM SORT            C
        MYTABL           RCSZ=12Ø,          PARAMETER TABLE                C
                         DISC=4,                                           C
                         ADDROUT=D
                 ENTRY   MYTABL
```

Figure 2—12. ADTABL Parameter Referencing Table in Previous Program

- Calculating optimum working storage (CALC)

Another very useful optional parameter is the CALC parameter. This parameter may be specified only for disk sorts. If you want 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        10      16
1.               MR$PRM  CALC=NO
2.               MR$PRM  CALC=YES
```

The YES specification (line 2) causes 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.

```
MR$PRM SIZE=2ØØØ,CALC=YES
```

- Entering parameters in the sort parameter table (CSPRAM)

Sort/merge also accepts sort/merge parameters from the job control stream by means of the PARAM job control statement. See 2.12 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 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), sort/merge will not look for PARAM statements in the job control stream.

```
      1        10    16
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 macroinstruction 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.

- Eliminating or combining records with equal key fields (DROC)

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 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), sort/merge performs data reduction automatically (line 1). 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 3.3.

```
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 macroinstruction.

- Performing a merge-only operation (MERGE)

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

```
    1        10     16
1.           MR$PRM MERGE=YES
2.           MR$PRM MERGE=NO
```

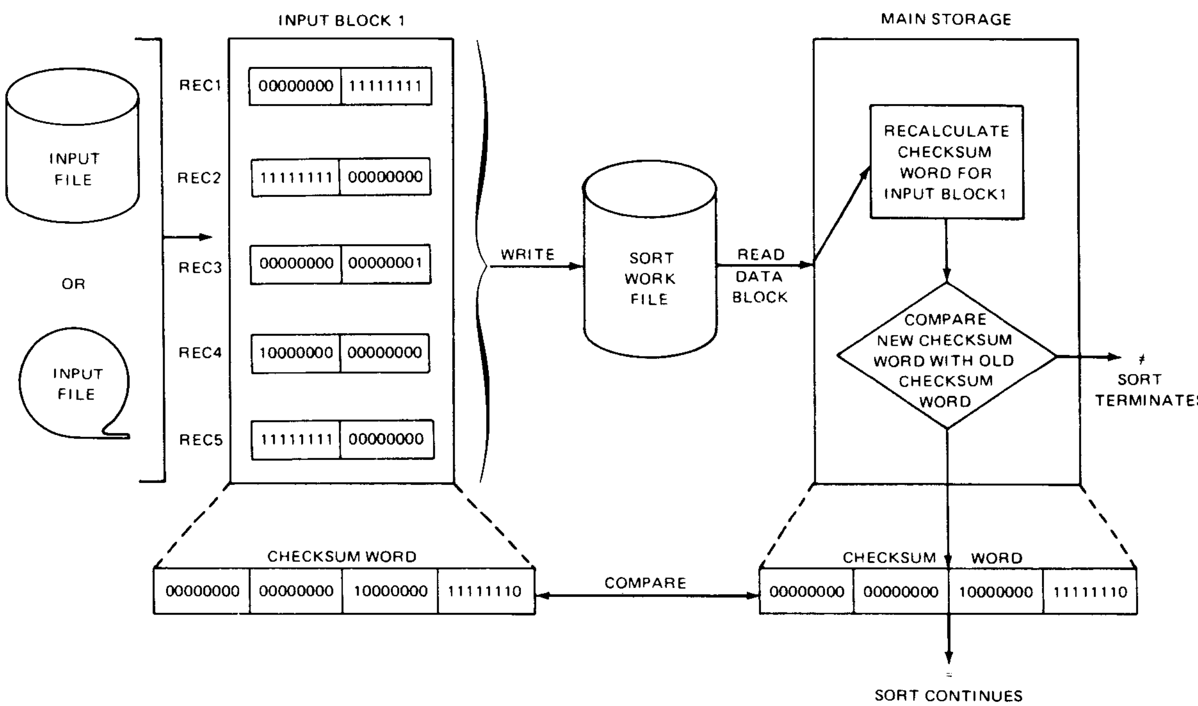
  If you omit this parameter, sort/merge assumes NO by default, a merge-only operation is not performed. You can also specify that this is *not* a merge-only operation by coding NO (line 2).

- Suppressing calculation of the checksum word (NOCKSM)

  Normally, 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.

```
   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.

- Adding parameters to the sort parameter table (PAD)

  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 4.

  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 2-11, line 7 and Figure 2-12, line 7.

- Writing messages in the job log (PRINT)

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

- Indicating the number of records (SIZE)

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

2.4.3. MR$PRM for the Disk Sort Program

Let's consider the specifications you might make in the parameter table for a disk sort program. Specifying the required parameters and other parameters pertinent to your disk sort (Figure 2-13), you might write:

```
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 2—13. Disk Sort Parameters

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 sort/merge can return control after it has sorted the records and it is ready to return them to your program.

When sort/merge returns the last sorted record to your program, it looks for the name of your output end-of-data 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 2-6).

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 sort/merge. The 4 specifies that four file names are assigned to sort/merge.

2.5. ACTIVATING SORT/MERGE (MR$OPN)

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

A label on the MR$OPN macroinstruction is optional, but for the operand you must indicate either the symbolic label (address) of the sort parameter table or the number 1 indicating register 1 where you have previously loaded the address of your sort parameter table. A blank operand field will also indicate that register 1 was loaded with the parameter table address. In our disk program, we indicate the symbolic label of the sort parameter table on the MR$OPN macroinstruction. Continuing the disk sort program coding from the last coding examples of Figure 2-3 and Figure 2-13, you would write:

```
1        10     16
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$ISUCC    SUCCESSFUL OPERATION?
         BZ     IOERROR                IF NOT, BRANCH TO IOERROR.
GETREC   EQU    *
         DMINP  INPUT,INOUTBUFF        GET RECORD FROM INPUT FILE.
         TM     CD$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.
```

When the MR$OPN has opened sort/merge, it passes control to your program at the address you specified in the IN keyword parameter of the MR$PRM macroinstruction. 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 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 sort/merge at that point.

Before doing any I/O operation you will want to link bit indicators CD$ISUCC and CD$IEOF to the file whose condition they are to test. As explained in 2.3, 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 26) you associate it with the RIB named SORTRIB, thus giving INPUT all the attributes specified in SORTRIB. Note that lines 27 and 28 contain a pair of instructions that recur throughout the program. The *test under mask* (TM) instruction at line 27 tests the CDI-successful-operation indicator CD$ISUCC that is set during the preceding OPEN operation. The *branch on zero* (BZ) instruction at line 28 causes a branch to routine IOERROR only if the CD$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 macroinstruction in your program.

With the file open, you can read the input file by designating the DMINP imperative macroinstruction (line 30). 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 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 our case, the work area INOUTBUF). In this example, a *load address* (LA) instruction is used.

2.6. GETTING DATA INTO THE SORT PROCESS

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

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

```
    1        10    16
36.          MR$REL                          RELEASE RECORD TO THE SORT
37.          B     GETREC                    GET NEXT RECORD
```

After the transfer occurs, sort/merge returns control to your program at the instruction immediately following the MR$REL macroinstruction. Now you want to read the next record, so you branch back to your DMINP macroinstruction labeled GETREC. Reading records, setting their address in register 1, and releasing records to the sort procedure are repeated until the end of the input file is reached. At this point, the CDIB end-of-file indicator CD$IEOF is set on (it has previously remained off). The TM and *branch on ones* (BO) instructions at lines 31 and 32, which before have passed control to the next instruction, now cause a branch to the routine beginning at 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 sort/merge. When control is returned to your program immediately following the MR$REL macroinstruction, 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 (2.9, line 61); or write a special routine to handle this condition in some other way. In the example, we have labeled this routine NOROOM.

```
     1        10     16
37a.          LTR    1,1                               LOAD R1 AND CHECK FOR NEGATIVE VALUE
37b.          BM     NOROOM                            GO TO 'NOROOM' ROUTINE
```

2.7. 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 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.

```
38.  EOF      EQU    *                                 THIS LOCATION IS SPECIFIED
39.  *                                                 AS THE END OF FILE ADDRESS.
40.           CLOSE  INPUT                             CLOSE INPUT FILE.
41.           TM     CD$ISUCC,L'CD$ISUCC
42.           BZ     IOERROR
43.           MR$SRT                                   TELLS THE SORT THAT THE END-OF-FILE
     *                                                 HAS BEEN REACHED.
```

After the sort receives all input data records, it completes a preliminary merge of record strings. Sort/merge may skip this phase if your input file is small. The final merge always occurs, and 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, sort/merge returns control to that label address, which is the beginning of your output routine.

2.8. DRAWING DATA FROM THE SORT PROCESS

Your output routine coding might continue as follows:

```
     1        10     16
44.  SORTOUT  EQU    *                     OUT ADDRESS
45.           LA     5,OUTPUT
46.           OPEN   OUTPUT,(SORTRIB)      OPEN OUTPUT FILE.
47.           TM     CD$ISUCC,L'CD$ISUCC
48.           BZ     IOERROR
49.  RECRET   MR$RET                       REQUEST A RECORD RETURNED.
50.           LA     2,INOUTBUF            LOAD R2 WITH BUFFER ADDRESS.
51.           MVC    Ø(8Ø,2),Ø(1)          MOVE THE SORTED RECORD TO THE
     *                                     OUTPUT BUFFER AREA.
52.           DMOUT  OUTPUT,INOUTBUF       OUTPUT THE RECORD RETURNED.
53.           TM     CD$ISUCC,L'CD$ISUCC
54.           BZ     IOERROR
55.           B      RECRET
```

To begin your output routine, you load register 5 with the address of the OUTPUT file CDIB. This action causes bit indicator CD$ISUCC to reflect the condition of file OUTPUT, the program having finished processing file INPUT. You then open the output file (line 46) and request sort/merge to return sorted records to your program via the MR$RET sort macro (line 49). 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, assign a symbolic label to the MR$RET macro to develop your output record processing loop (line 49). MR$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.

When you open the output file and specify WORK=YES in its RIB, you must specify for each DMOUT the symbolic address of the work area from which the data is written to the file. When control returns to your program (following the MR$RET), the record to which register 1 points must be moved to the output work area INOUTBUF (lines 50 and 51). When you issue a DMOUT macroinstruction specifying INOUTBUF as the work area, data management moves the contents of INOUTBUF to the buffer, 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 DMOUT imperative macro (line 52) and issue an unconditional branch (line 55) to MR$RET, which you labeled RECRET (line 49). This loop repeats until it reaches the end of data, indicating that all sorted records have been returned to your program.

When 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 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.

2.9. 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.

```
    1        10     16
56. SORTFIN  EQU    *                        FIN ADDRESS
57.          CLOSE  OUTPUT                   CLOSE THE OUTPUT FILE.
58.          TM     CD$ISUCC,L'CD$ISUCC
59.          BZ     IOERROR
60.          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. As described in 2.5, you address the error routine by a BO instruction that branches to the routine if indicator CD$ISUCC has not been set (an unsuccessful operation).

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 62) to halt the current job run.

```
61. IOERROR  EQU    *
62.          CANCEL                          CANCEL THE JOB.
63.          LTORG                           DEFINE ALL LITERALS HERE.
64. WORK     EQU    *                        START OF SORT WORK AREA.
    *                                        THIS SET UP ALLOWS THE SORT
    *                                        TO USE ALL MEMORY FROM
    *                                        THIS LOCATION TO THE END OF
    *                                        THE JOB REGION.
65.          END    SRTEXMPL
```

Finally, to lead into the necessary job control statements for your disk sort program, you would write the LTORG assembler directive (line 63), which generates into your source module all previously defined literals.

In the STOR parameter of the MR$PRM macroinstruction, you specified the symbolic label WORK. That name indicated the starting address of the main storage area available to sort/merge. Here, at the end of your program, you place your designation of that area (line 65). 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 2-14 shows the coding of your disk sort program to this point and the diagram following it, Figure 2-15, illustrates your program's interface with 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.

```
     1        10     16                                                      72
 1. SRTEXMPL START 0                          SETS LOCATION COUNTER TO ZERO.
 2.          EXTRN MR$ORT                     MR$ORT DEFINES AN EXTRN,
    *                                         LINKS COMMON SORT MODULE
    *                                         TO YOUR PROGRAM.
 3.          BALR  4,0
 4.          USING *,4
 5.          B     START
 6. SORTRIB  RIB   BFSZ=512,RCSZ=80,IOA1=BUFF1,IOA2=BUFF2,WORK=YES,        C
 7.                RCFM=FIX,OPTN=YES,MODE=SEQ
 8. INPUT    CDIB
 9. OUTPUT   CDIB
10. USING    CD$CDIB,5
11. VTOC     CDIB=YES
    *
    *
12. SORT     MR$PRM FIELD=(0,7,CH),                                          C
                   IN=SORTIN,                                                C
14.                OUT=SORTOUT,                                              C
15.                FIN=SORTFIN,                                              C
16.                RCSZ=80,                                                  C
17.                STOR=WORK,                                                C
18.                DISC=4
    *
    *              DATA MANAGEMENT WORK AREA
19.          DS    0H
20. BUFF1    DS    CL512
21. BUFF2    DS    CL512
22. INOUTBUF DS    CL80
    *
    *
23. START    EQU   *
24.          MR$OPN SORT                      OPEN THE SORT/MERGE SUBROUTINE
25. SORTIN   LA    5,INPUT
26.          OPEN  INPUT,(SORTRIB)            OPEN THE INPUT FILE
27.          TM    CD$ISUCC,L'CD$ISUCC        SUCCESSFUL OPERATION?
28.          BZ    IOERROR                    IF NOT, BRANCH TO IOERROR.
29. GETREC   EQU   *
30.          DMINP INPUT,INOUTBUF             GET RECORD FROM INPUT FILE
31.          TM    CD$IEOF,L'CD$IEOF          INPUT FILE EMPTY?
32.          BO    EOF                        IF EMPTY, BRANCH TO EOF.
33.          TM    CD$ISUCC,L'CD$ISUCC
34.          BZ    IOERROR
35.          LA    1,INOUTBUF                 LOAD R1 WITH RECORD ADDRESS.
36.          MR$REL                           RELEASE RECORD TO THE SORT.
37.          B     GETREC                     GET NEXT RECORD.
    *
38. EOF:     EQU   *                          THIS LOCATION IS SPECIFIED
39. *                                         AS THE END OF FILE ADDRESS.
40.          CLOSE INPUT                      CLOSE THE INPUT FILE.
41.          TM    CD$ISUCC,L'CD$ISUCC
42.          BZ    IOERROR
    *
43.          MR$SRT                           TELLS THE SORT THAT THE END
    *                                         OF FILE HAS BEEN REACHED.
44. SORTOUT  EQU   *                          OUT ADDRESS.
```

Figure 2—14. Disk Sort Program Coding (Part 1 of 2)

```
1       10     16
                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.
       *
       *               ERROR ADDRESS FOR DATA MANAGEMENT
       *
       IOERROR  EQU    *
                CANCEL                          CANCEL THE JOB.
                LTORG                           DEFINE ALL LITERALS HERE.
       WORK     EQU    *                        START OF SORT WORK AREA.
       *                                        THIS SETUP ALLOWS THE SORT
       *                                        TO USE ALL MEMORY FROM
       *                                        THIS LOCATION TO THE END OF
       *                                        THE JOB REGION.
                END    SRTEXMPL
```

Figure 2—14. Disk Sort Program Coding (Part 2 of 2)

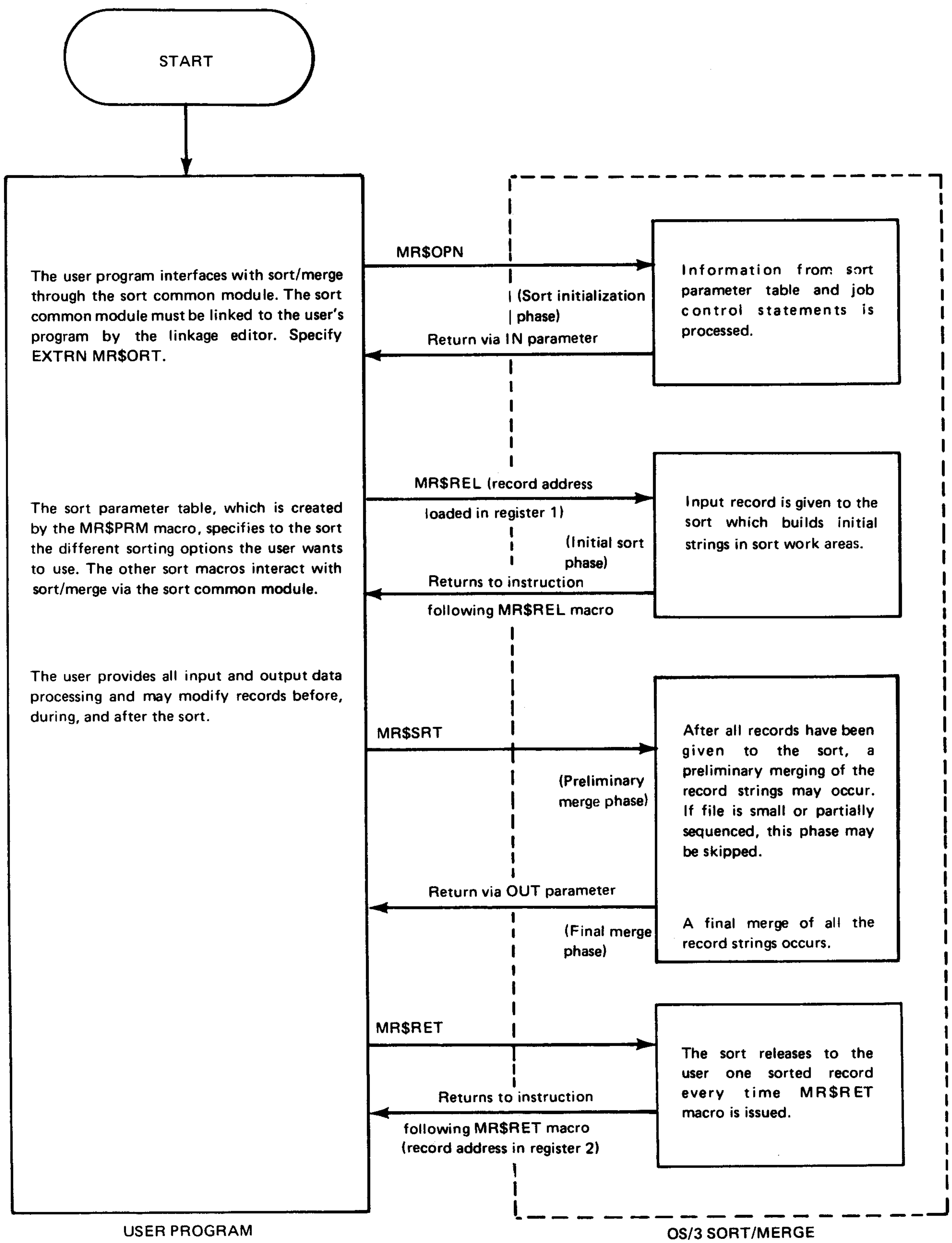


Figure 2—15. User Program Interface with Sort/Merge

2.10. SORT/MERGE MACROINSTRUCTION PARAMETERS

We've examined the required and optional MR$PRM macroinstruction parameters and a typical disk sort program MR$PRM specification. The entire MR$PRM macro format is:

LABEL	△OPERATION△	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] [,PRINT={ALL CRITICAL NONE}] [,RESERV=sort-filename] [,RESUME=(PASS,recovery-number)] [,SHARE=sort-filename] [,SIZE=number] [,USEQ=(to-address,from-address)]

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

2.11. 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 statements that name devices used by your program and sort/merge; describe labels and space allocations; and assemble, link, and execute your program.

2.11.1. 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 (YRUN), in the object library file (YOBJ), or in some other library. 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 (YOBJ).

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.

Table 2—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

2.11.2. Link Editing the Program

You now have an object module representing your source program in YRUN. 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 2-19, 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 2-17, line 8.) The linkage editor also scans your program object module for external references and finds MR$ORT. It looks for MR$ORT in YOBJ, 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 (YRUN) unless you specify that the load modules be placed in your user load library (a file separate from the system-resident library files).

2.11.3. 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 2-17, 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 YRUN and the execution begins. In the execution run, the load modules for sort/merge are called from YLOD 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 2-17, lines 13 through 16.)

Figure 2-16 illustrates the assembly, linkage edit, and execution runs for a disk sort program.

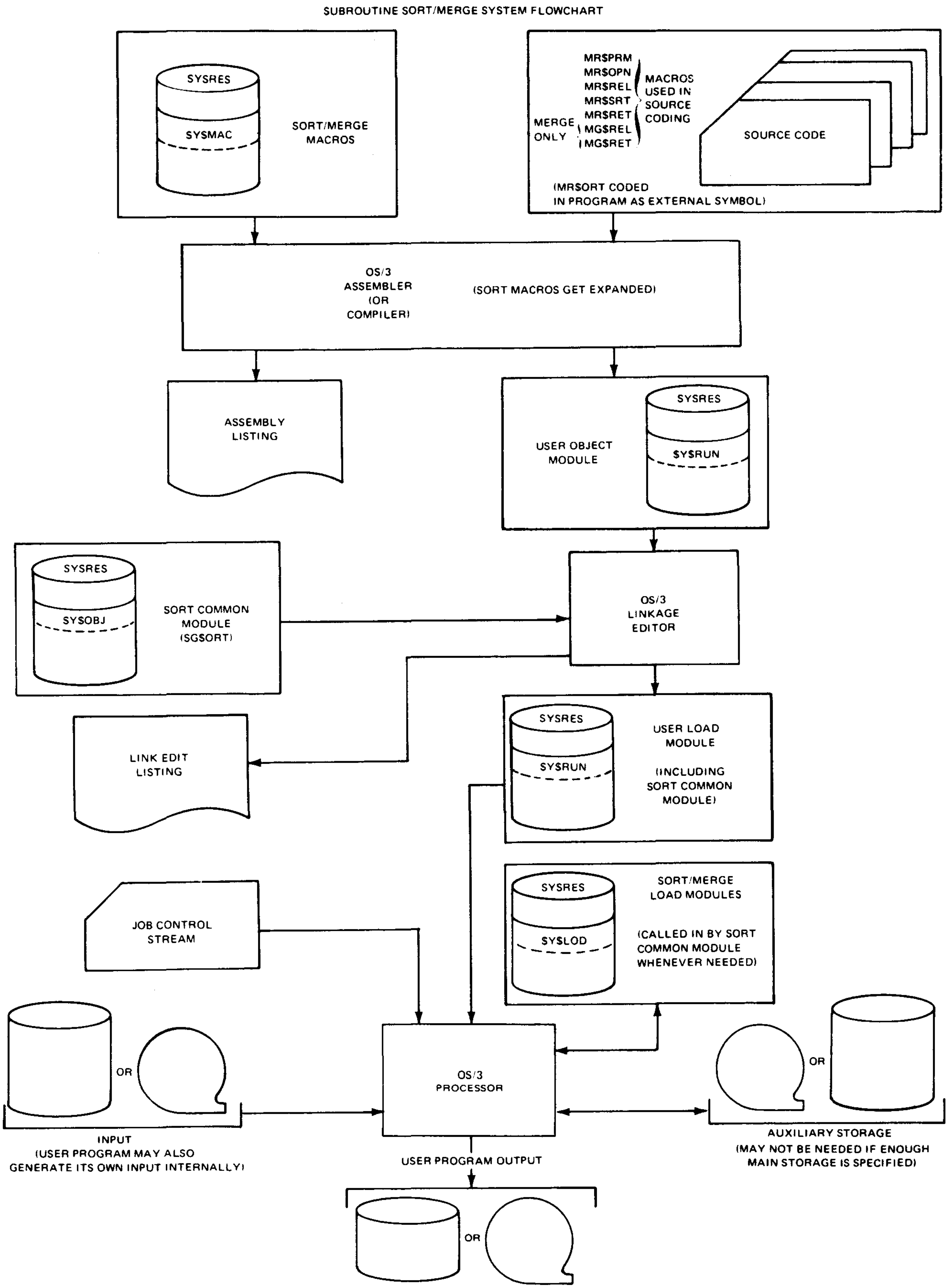


Figure 2—16. Assembly, Linkage Edit, and Execution Run System Flowchart

2.11.4. 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 2-17 shows the entire job control stream required for our disk sort program. Each line is explained in detail following the figure.

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     16
 1.  // JOB SRTEXMPL,,7000,9000,2
 2.  // DVC 20 // LFD PRNTR
 3.  // WORK1
 4.  // WORK2
 5.  // EXEC ASM
 6.  /$
       .
       .  } Your program coding
       .
 7.  /*
 8.  //SRTEXM LINK SRTEXMPL
 9.  // DVC 50
10.  // VOL DSP028
11.  // LBL MYFILE1
12.  // LFD INPUT
13.  // DVC 50
14.  // VOL DSP028
15.  // LBL MYFILE2
16.  // LFD OUTPUT,,INIT
17.  // DVC 50
18.  // VOL DSP028
19.  // EXT ST,C,,CYL,5
20.  // LBL $SCR1
21.  // LFD DM01
22.  // EXEC SRTEXM,$Y$RUN
23.  /&
24.  // FIN
```

Figure 2—17. Disk Sort Program Job Control Stream

- Line 1

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

- Lines 2–6

  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 (line 2). The two following job control statements, // WORK1 and // WORK2 (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-date (/$) control statement (line 6).

  Finally, the // EXEC ASM statement (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.

- Line 7

  At the end of your source coding, you code a /* delimiter statement to indicate the end of data (your program) to the assembler.

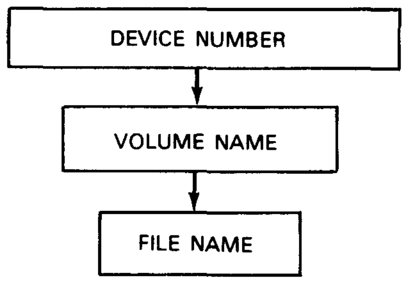
- Line 8

  So far, we have generated an object module called SRTEXMPL (label of the START assembler directive) and it is in YRUN. 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 (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 YRUN, 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-9986 (current version).

When you execute the load module SRTEXM (line 22), you tell the supervisor to retrieve it from YRUN. Otherwise, the supervisor searches for the SRTEXMPL load module in the YLOD first before going to YRUN. Thus, by specifying YRUN, you save processing time.

- Lines 9-21

Your next series of job control statements (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 (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-9986 (current version).

Your first DVC statement assigns device number 50 to your input file named MYFILE1 (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 CDIB 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 CDIB macros, lines 12 and 16 must be the same as the file names in the labels of corresponding CDIB 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 (YRUN). 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-9986 (current version).

- Line 22

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 2–18 shows the job control stream required to assemble, link, and execute a disk sort program.

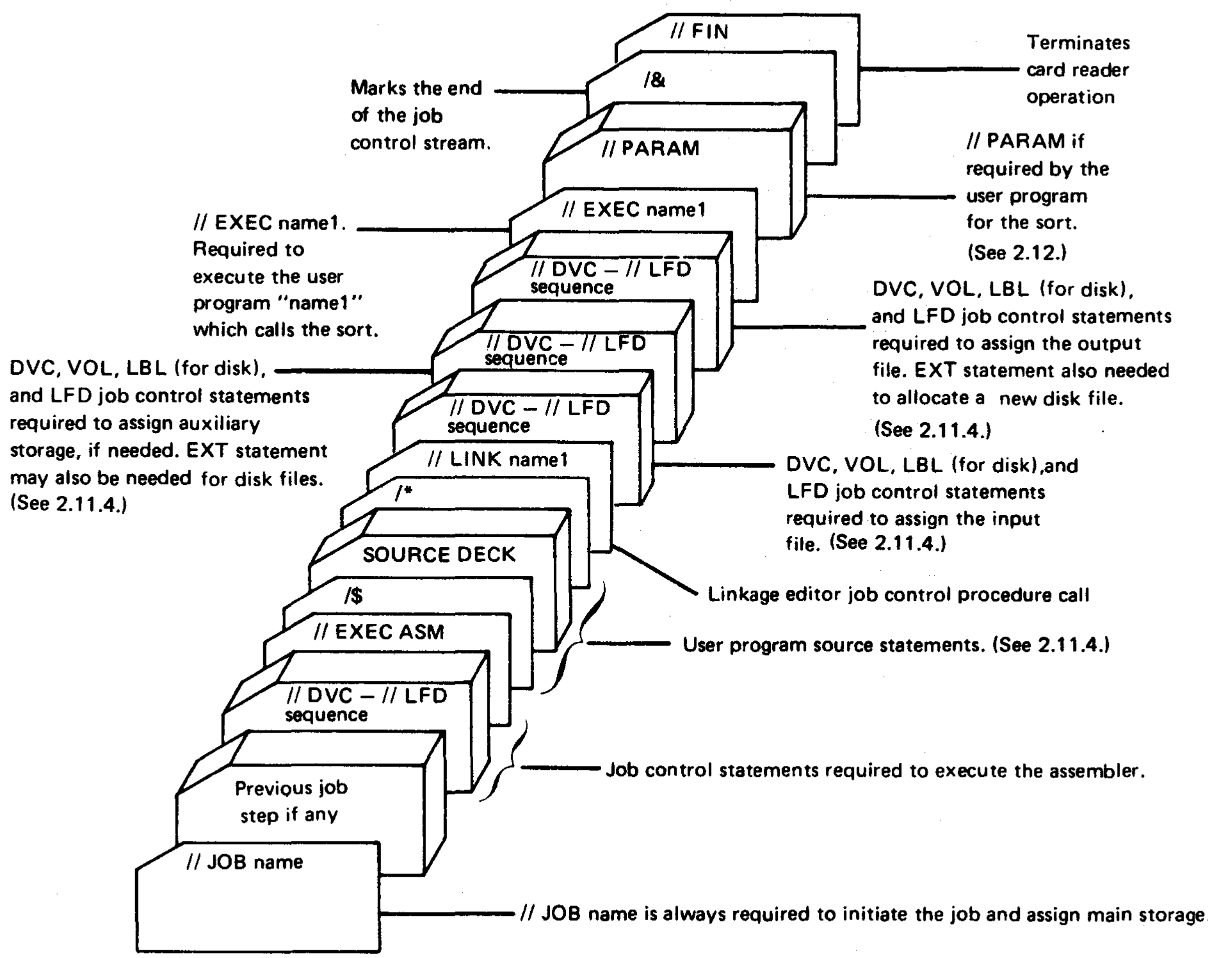


Figure 2—18. Typical Job Control Stream for a Sort/Merge Application

2.11.4.1. Alternate Job Control Stream

The job control stream shown in Figure 2-17 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 2-19); 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 2-19.

```
    1         10     16
1.  // JOB SRTEXMPL,,7ØØØ,9ØØØ,2
2.  // DVC 2Ø // LFD PRNTR
3.  // DVC RES
4.  // EXT ST,,1,BLK,(256,4ØØØ)
5.  // LBL $SCR1
6.  // LFD $SCR1                         // WORK1
7.  // DVC RUN                           // WORK2
8.  // EXT ST,,1,BLK,(256,4ØØØ)
9.  // LBL $SCR2
10. // LFD $SCR2
11. // EXEC ASM
12. /$
13.       .
14.       .      Your program coding
15.       .
16. /*
17. // WORK1
18. // EXEC LNKEDT
19. /$                                   // LINK SRTEXMPL
20.  LOADM   SRTEXM
21.  INCLUDE SRTEXMPL
22. /*
23. // DVC 5Ø
24. // VOL DSP111                        Device assignment set 1
25. // LBL MYFILE1
26. // LFD INPUT
27. // DVC 5Ø
28. // VOL DSP111                        Device assignment set 2
29. // LBL MYFILE2
30. // LFD OUTPUT,,INIT
31. // DVC 51
32. // VOL DSP12Ø
33. // EXT ST,C,,CYL,2Ø                  Device assignment set 3
34. // LBL SRTWK1
35. // LFD DMØ1
36. // EXEC SRTEXM
37. /&
38. // FIN
```

Figure 2—19. Alternate Job Control Stream for A Disk Sort Program

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.

2.11.5. 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     16
1. // DVC 90
2. // VOL TAP150
3. // LBL SRTWK1
4. // LFD SM03
```

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.

2.12. 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 macroinstruction (2.10) and begin writing your PARAM statement keyword parameters in column 10. Separate each parameter by a comma, 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.

```
// PARAM TAPE=(STD,6),SHARE=SM01
// 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        10     16
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 5.3, line 28 and line 114.

2.13. RUNNING YOUR SORT JOB FROM A WORKSTATION

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

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

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

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

If you are not familiar with job control, use the job control dialog for assistance. The job control dialog is an interactive facility of OS/3 that allows you to describe your job's requirements to it in English, in response to a series of questions, and then produces as its output, the job control stream needed by OS/3 to run your job. The control stream produced by the job control dialog is virtually identical to the control stream that you would have to produce if you were running your job in a batch environment. Only now, you do not have to be concerned with the intricacies of the job control language. The job control dialog eliminates this requirement on your part.

After you have answered all the questions presented to you by the job control dialog, it builds a control stream and stores it in a permanent library file for you. 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 by using the general editor.

The procedures for activating the general editor are detailed in the general editor user guide/programmer reference, UP-9976 (current version).

The procedures for activating the job control dialog and initiating the running of a job are detailed in the job control user guide, UP-9986 (current version).

3. User Own-Code Routines

3.1. GENERAL

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 macroinstruction, 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 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.

3.2. 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, sort/merge transfers control to your own-code routine at the address (symbolic label name) you specified on your RSOC keyword parameter. Sort/merge places the RSOC address in register 15 and stores the sort/merge return address in register 14.

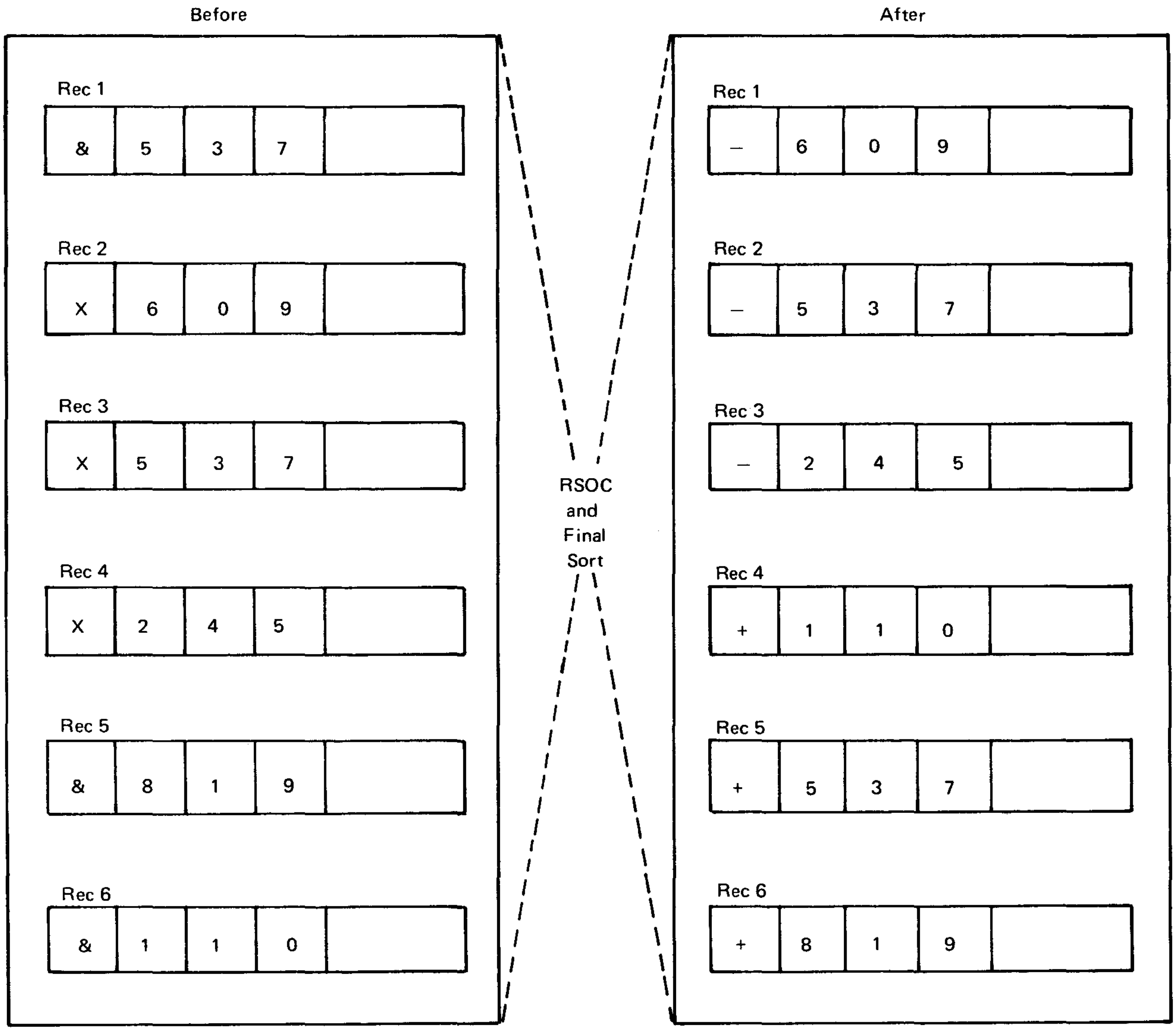
The first instruction in your own-code routine must be the USING assembler directive. It must assign register 15 for use as the base register of your RSOC routine. Your RSOC routine automatically receives the addresses of the two records to be compared in registers 11 and 12. For variable-length records, 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 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 sort/merge via a branch to register 14. A sketch of the key instructions needed for using RSOC follows:

```
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 SORT/MERGE
  .
```

For a complete program example illustrating a user own-code routine for a sort, see 5.4.

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:



LEGEND:

X = –
& = +

3.3. DATA REDUCTION OWN-CODE ROUTINE (DROC)

DROC routines concern final disposition of fixed-length or variable-length records with equal key field values. When you specify the DROC keyword parameter on the MR$PRM sort macroinstruction, 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;

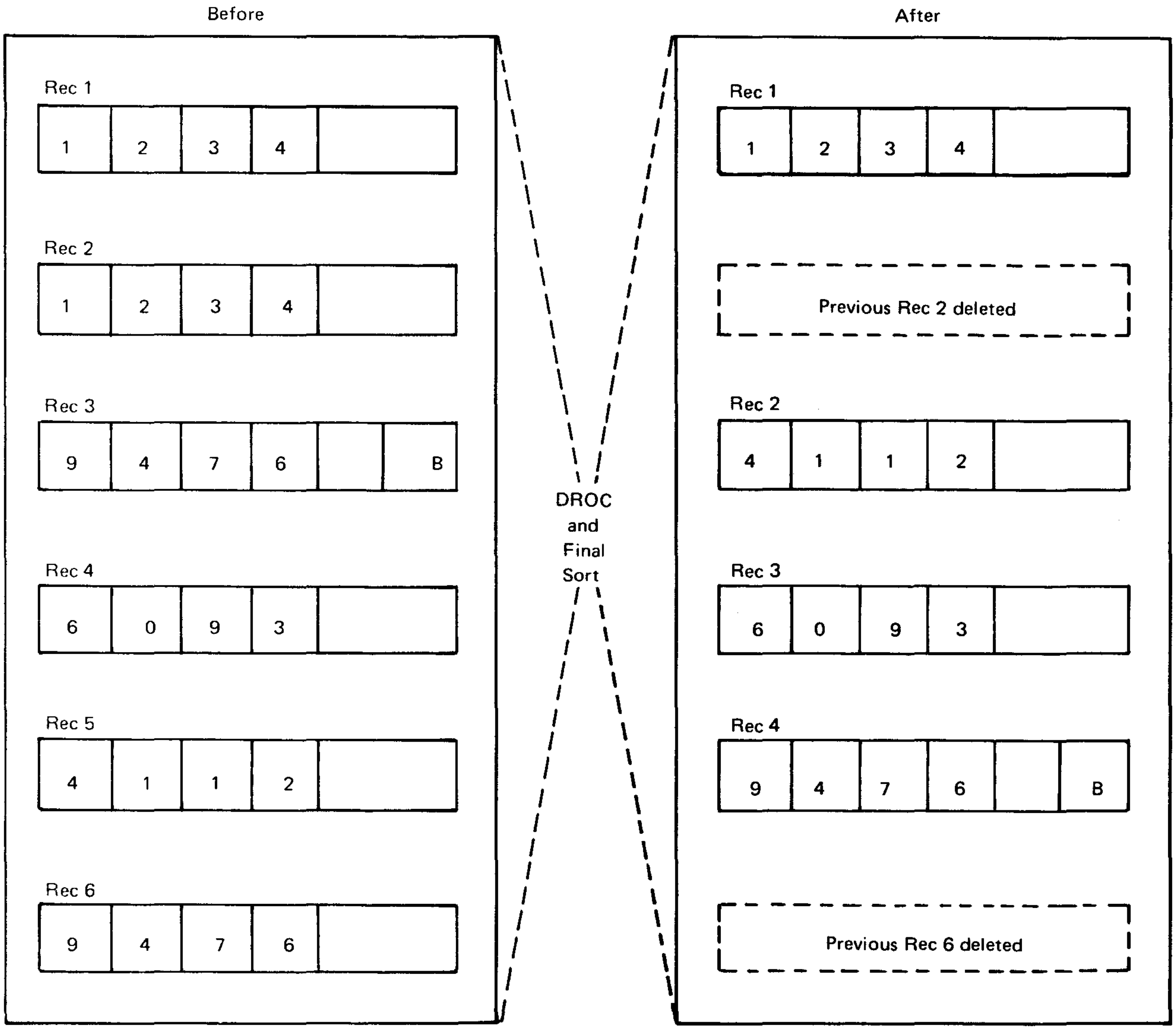
  2. by combining data contained in the two records to create a new record; and

  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 sort/merge four bytes beyond the sort/merge return address specified in register 14. If you want to retain both records, control must return to sort/merge at the address specified in register 14. Because register 14 contains the sort/merge return address, take great care not to change its contents.

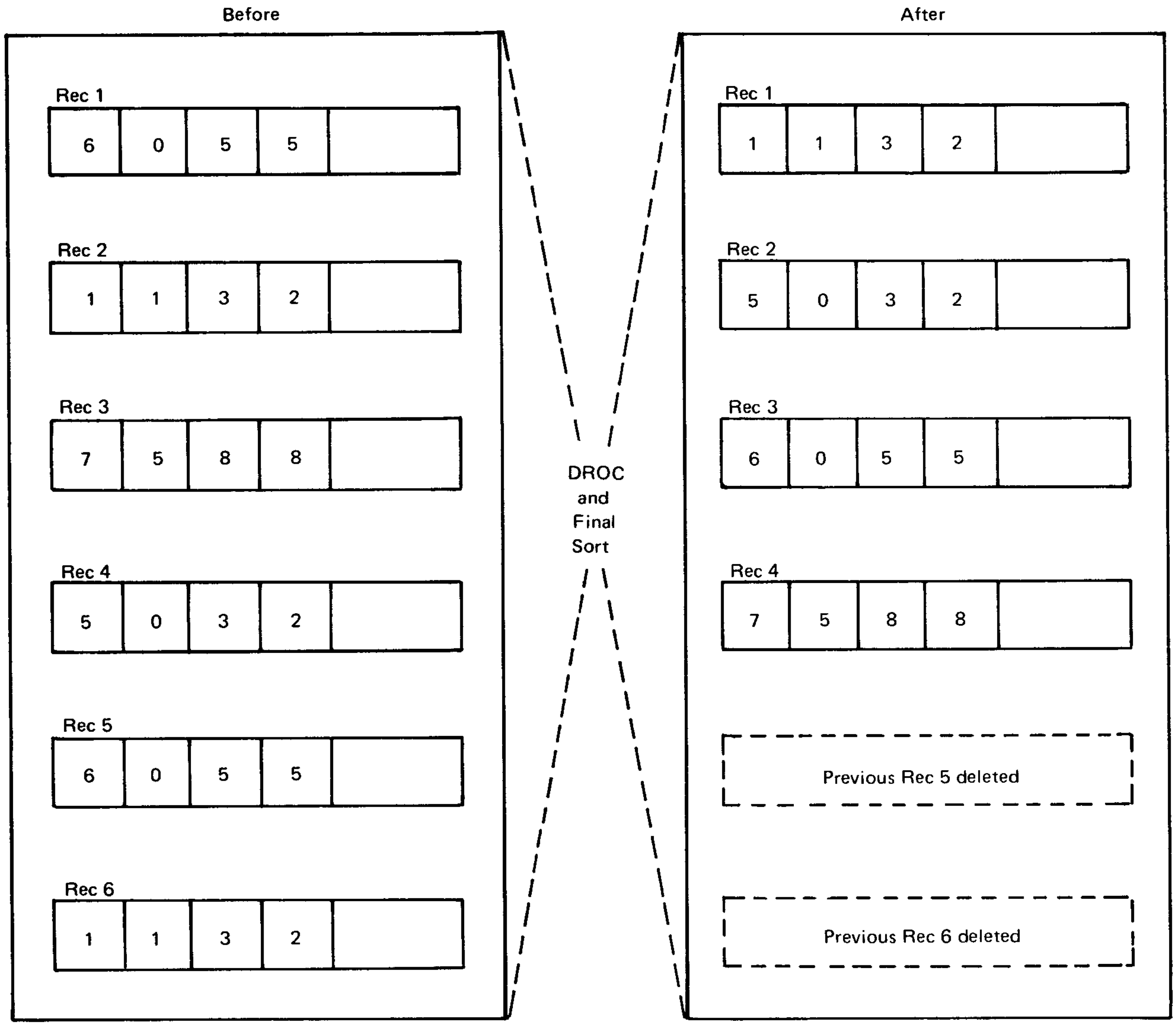
To end your DROC routine, you return control to sort/merge at the address specified in register 14. You may optionally include the DROP assembler directive to disengage register 15 from use as your routine's base register. The following shows the coding required to specify your own DROC routine. Notice 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.

```
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 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 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.

```
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

4. Special Applications

4.1. 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 MIRAM disk files.

By specifying the ADDROUT parameter on the MR$PRM macroinstruction, 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 2.4.2.2.)

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 with the RIB data management macroinstruction, using the SKAD keyword parameter to specify a location in your program that is to contain the 4-byte relative disk address of the record. Then, when you issue a DMINP macroinstruction, data management places the relative disk address of the input record at the SKAD location. You can then move the disk address to the address field of the tag sort record, and the input record key to the key field of the tag sort record. After loading register 1 with the address of the tag sort record, you issue a MR$REL macroinstruction to release that record to the sort.

The following coding example illustrates the key instructions needed for a tag sort that returns only the address field to your program.

```
1          10     16                                                        72
           MR$PRM ADDROUT=A,                                                C
                  FIELD=(0,80)
   .
   .
   .
INPUT      CDIB
MYRIB      RIB    RCSZ=100,WORK=YES,SKAD=INPUTB,...
INBUF1     DS     CL100
   .
   .
   .
           OPEN   INPUT,(MYRIB)
   .
   .
   .
GETREC     DMINP  INPUT,INBUF                       READS RECORD
           TM     CD$IEOF,L'CD$IEOF
           BO     EOF                               IF END OF FILE, GO TO EOF.
           TM     CD$ISUCC,L'CD$ISUCC
           BZ     IOERROR                           IF ERROR, GO TO IOERROR.
           MVC    TAGREC+4(4),INPUTB                PLACES INPUT REC ADDR IN TAG
*                                                   SORT REC ADDR AREA
           MVC    TAGREC+10(80),INBUF               PLACES KEY FIELD IN TAG SORT
*                                                   RECORD
           LA     1,TAGREC                          PLACES TAG SORT REC ADDR IN R1
           MR$REL                                   RELEASES RECORD TO THE SORT
           B      GETREC                            GETS NEXT RECORD
*
INPUTB     DS     F                                 AREA IN WHICH DATA MANAGEMENT
*                                                   PLACES INPUT RECORD ADDRESS
TAGREC     DC     XL10'00'                          TAG SORT REC ADDR
           DS     CL80                              TAG SORT KEY FIELD
```

4.2. 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 macroinstruction, 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 2.4.2.3.) For additional program examples, see 5.3.

4.3. 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 sort/merge, you must indicate to sort/merge that you want merge-only processing by writing the MERGE=YES parameter on your MR$PRM macroinstruction. This places the merge-only indication in your sort parameter table.

4.3.1. What Merge-Only Does for You

The 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, the final merge phase 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, 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 sort/merge for comparison by pairs. 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.

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 sort/merge processes all records from each input file and returns them to your program.

4.3.2. 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 4–1) and the job description that follows.

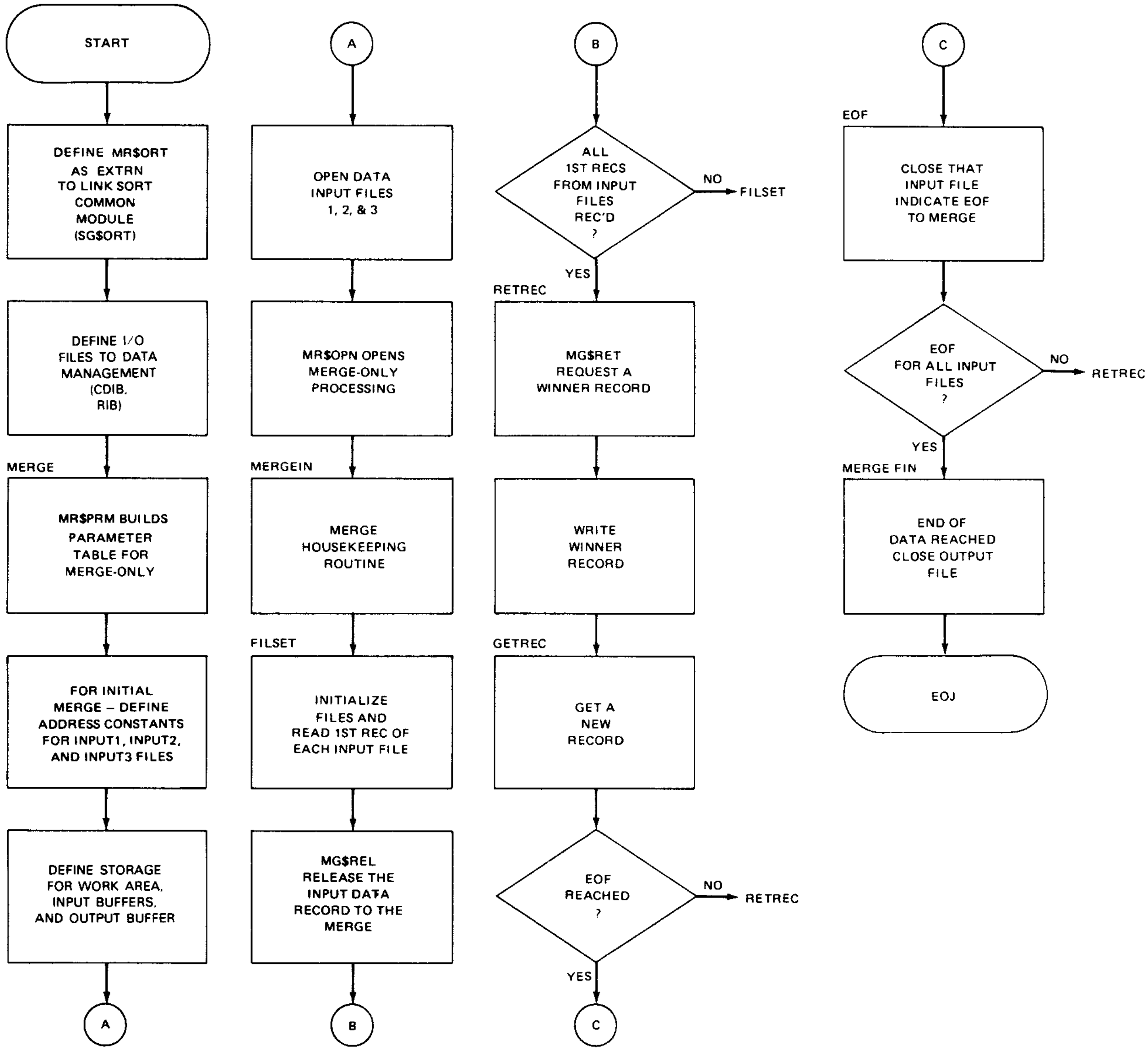


Figure 4—1. Subroutine Merge-Only Program Flowchart

This merge-only program has the following characteristics:

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.

4. This program needs a table of file addresses to help locate input files for initiation of the first record merge from each file.

5. Buffer and output processing areas must be reserved in main storage for input and output file processing.

6. All three input files are assigned to disk device 51.

7. Both input and output files use fixed-length, blocked records.

8. Each record contains 80 bytes.

9. The first input file contains 1024 bytes per block, the second input file contains 512 bytes, and the third input file contains 2048 bytes.

10. The program produces a single output file of records merged in ascending order from the 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 10), you issue the EXTRN assembler directive, which links the sort common module from YOBJ to your program (line 11), and you define your input and output files to data management (lines 14 through 25).

```
     1        10     16                                                      72
 1.  // JOB MRGEXMPL,,7ØØØ,9ØØØ,2
 2.  // DVC 2Ø    // LFD PRNTR
 3.  // WORK1
 4.  // WORK2                                     }=// ASM
 5.  // EXEC ASM
 6.  /$
 7.  MRGEXMPL START Ø
 8.           BALR  4,Ø
 9.           USING *,4
10.           B     START
11.           EXTRN MR$ORT                        DEFINES THE SORT COMMON MODULE
12.  *                                            TO BE INCLUDED BY THE LINKAGE
13.  *                                            EDITOR.
14.  INPUT1   CDIB
15.  MERGRIB1 RIB   BFSZ=1Ø24,RCSZ=8Ø,IOA1=BUFF1,IORG=(2),RCFM=FIX,      C
16.                 OPTN=YES
17.  INPUT2   CDIB
18.  MERGRIB2 RIB   BFSZ=512,RCSZ=8Ø,IOA1=BUFF2,IORG=(2),RCFM=FIX,
19.                 OPTN=YES
20.  INPUT3   CDIB
21.  MERGRIB3 RIB   BFSZ=2Ø48,RCSZ=8Ø,IOA1=BUFF3,IORG=(2),RCFM=FIX,      C
22.                 OPTN=YES
23.  OUTPUT   CDIB
24.  OUTRIB   RIB   BFSZ=512,RCSZ=8Ø,IOA1=OUTØ1,IOA2=OUTØ2,WORK=YES,     C
25.                 RCFM=FIX,OPTN=YES
```

Your MR$PRM macroinstruction, which creates the parameter table for your program, supplies all the information needed by sort/merge to perform the merge. All the following parameters coded for the merge-only program example are required (lines 26 through 31). For other optional merge-only parameters that may be included in the sort parameter table generated by MR$PRM, see 2.10.

```
    1        10      16                                                          72
26. MERGE    MR$PRM  IN=MERGEIN,                                                 C
27.                  FIN=MERGEFIN,                                               C
28.                  STOR=WORK,                                                  C
29.                  RCSZ=80,                                                    C
30.                  FIELD=(0,8,CST),                                            C
31.                  MERGE=YES
```

Initially, sort/merge needs a way to locate the first record of each input file. Lines 32–34 show the way to provide that information to your program when it begins the initial merge comparison.

```
32. FILTABL  DC      A(INPUT1)                  ADDRESS OF IN1 CDIB
33.          DC      A(INPUT2)                  ADDRESS OF IN2 CDIB
34.          DC      A(INPUT3)                  ADDRESS OF IN3 CDIB
```

To test the status of your input and output files after performing I/O operations on them, you use the bit indicators CD$ISUCC and CD$IEOF as described in 2.3; as you recall, they indicate, respecitively, a successful operation and an end-of-file condition. To link these with the specific file whose status you want to test, you first set up register 1 as a base register for the DSECT, called by the VTOC macroinstruction, that maps the CDIB.

```
35.          USING   CD$CDIB,1
36.          VTOC    CDIB=YES
```

The second step in testing the status of a file is to load the address of the CDIB for that file in register 1. Because every imperative I/O macroinstruction uses register 1 to hold the CDIB address of the file being operated on (loading the address in register 1 if necessary), and because that address remains unchanged after the operation finishes, you can at that point test CD$ISUCC and CD$IEOF with the assurance that they reflect the status of the file just operated on. This assumption underlies every instruction in the program testing CD$ISUCC and CD$IEOF.

To begin your program, you open all the input files and the output file (lines 38 through 49). By issuing the MR$OPN (line 50) and refencing the table from your MR$PRM sort parameter table specifications (line 26), you open the subroutine merge (lines 37 through 50).

```
     1        10      16
37. START     EQU     *
38.           OPEN    INPUT1,(MERGERIB1)            OPEN INPUT FILE.
39.           TM      CD$ISUCC,L'CD$ISUCC
40.           BZ      IOERROR
41.           OPEN    INPUT2,(MERGERIB2)            OPEN INPUT FILE.
42.           TM      CD$ISUCC,L'CD$ISUCC
43.           BZ      IOERROR
44.           OPEN    INPUT3,(MERGERIB3)            OPEN INPUT FILE.
45.           TM      CD$ISUCC,L'CD$ISUCC
46.           BZ      IOERROR
47.           OPEN    OUTPUT,(OUTRIB)               OPEN OUTPUT FILE
48.           TM      CD$ISUCC,L'CD$ISUCC
49.           BZ      IOERROR
50.           MR$OPN MERGE                          OPEN SORT/MERGE SUBROUTINE
    *                                               REFERENCING MR$PRM MACRO
    *                                               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 56). This means that you must increment the full length of your input file to get to the second file (line 57). This is the value of setting up your file table of address constants earlier in lines 32 through 34.

```
51. MERGEIN   EQU     *                             IN ADDRESS
52.           LA      5,3                           LOAD R5 WITH THE NUMBER OF
    *                                               INPUT FILES
53.           LA      6,FILTABL                     GET FILE TABLE ADDRESS
54. FILSET    L       1Ø,Ø(6)                       LOAD CDIB ADDRESS IN R10 AND
    *                                               USE AS AN INDEX TO IDENTIFY
    *                                               INPUT FILE TO MERGE AND TO
    *                                               YOUR PROGRAM.
55.           BAL     7,GETREC                      GET FIRST RECORD FOR EACH FILE
56.           MG$REL                                RELEASE RECORD TO MERGE.
57.           LA      6,4(6)                        INCREMENT TO NEXT CDIB ADDR.
58.           BCT     5,FILSET                      TEST FOR LAST INPUT FILE: IF
    *                                               YES, CONTINUE. IF NO, GET FIRST
    *                                               RECORD OF NEXT FILE.
```

Before continuing, let's examine the function of the MG$REL macroinstruction. The MG$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 sort/merge and your program (EXTRN MR$ORT);
- defined merge-only processing (MR$PRM);
- opened input and output files; and
- initiated sort/merge for merge-only processing (MR$OPN).

Two registers, R1 and R10, play important roles in receiving and storing addresses used by sort/merge. Before releasing the initial record of an input file to 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 71, GETREC routine) and the address or identifier of the file into register 10 (line 54, FILSET routine).

```
     1        10     16                                          72
59.  RETREC   EQU    *
60.           MG$RET                            REQUEST WINNER RECORD.
61.           BAL    7,PUTREC                   WRITE RECORD TO OUTPUT FILE.
62.           BAL    7,GETREC                   GET NEW RECORD FROM INPUT FILE.
63.           B      RETREC                     GET NEW WINNER RECORD.
64.  GETREC   EQU    *                          GET A RECORD ROUTINE
65.           LR     1,10                       POINT TO INPUT FILE CDIB.
66.           DMINP  (1)                        INPUT RECORD.
67.           TM     CD$IEOF,L'CD$IEOF
68.           BO     EOF                        IF END OF FILE, GO TO EOF.
69.           TM     CD$ISUCC,L'CD$ISUCC
70.           BZ     IOERROR                    IF ERROR, GO TO IOERROR.
71.           LR     1,2                        POINT TO NEW RECORD.
72.           BR     7                          RETURN
```

Your record and file identification coding must precede the MG$REL macro (line 55 branches out to the GETREC routine, which points to the next record in line 71 before branching back to the MG$REL macro). After you release the initial record of the first input file, sort/merge returns control to your program at the instruction immediately after the MG$REL macro (line 56). Your program must then point to (identify) the next input file, where you must retrieve the first record for release to the merge (line 56). You create a processing loop from the initial record accessing to its release. When the initial records from each input file have been released, your program can request sort/merge to compare them. Select one winner record that fulfills the output sequence requirements you specified, and return it to your program. To return the winner record, you issue the MG$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 sort/merge does not move records accessed by your program during the merge-only processing. 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 71, 72, 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 73 through 78) before placing the address of the next record to be released into register 1 (line 71). The following coding shows how to avoid overlaying the winner record in our merge-only program:

```
     1         10    16                                                      72
59. RETREC    EQU   *
60.           MG$RET                                  REQUEST WINNER RECORD.
61.           BAL   7,PUTREC                          WRITE RECORD TO OUTPUT FILE.
62.           BAL   7,GETREC                          GET NEW WINNER RECORD.
63.           B     RETREC                            GET NEW WINNER RECORD.
64. GETREC    EQU   *                                 GET A RECORD ROUTINE
65.           LR    1,1Ø                              POINT TO INPUT FILE CDIB.
66.           DMINP (1)                               INPUT RECORD.
67.           TM    CD$IEOF,L'CD$IEOF
68.           BO    EOF                               IF END OF FILE, GO TO EOF.
69.           TM    CD$ISUCC,L'CD$ISUCC
70.           BZ    IOERROR                           IF ERROR, GO TO IOERROR.
71.           LR    1,2                               POINT TO NEW RECORD.
72.           BR    7                                 RETURN
73. PUTREC    EQU   *                                 OUTPUT RECORD ROUTINE
74.           MVC   WORKAREA,Ø(1)                     MOVE WINNER RECORD TO WORKAREA
75.           DMOUT OUTPUT,WORKAREA                   OUTPUT THE RECORD
76.           TM    CD$ISUCC,L'CD$ISUCC
77.           BZ    IOERROR                           IF ERROR, GO TO IOERROR.
78.           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 72). 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 71). Execution of the MG$RET macroinstruction 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 sort/merge before releasing additional records to the merge (EOF routine, lines 79 through 85).

```
79. EOF       EQU   *                                 END-OF-FILE ADDRESS
80.           LR    1,1Ø                              LOAD R1 WITH CDIB ADDRESS
81.           CLOSE (1)                               CLOSE THAT INPUT FILE
82.           TM    CD$ISUCC,L'CD$ISUCC
83.           BZ    IOERROR
84.           XR    1,1                               INDICATE EOF CONDITION TO MERGE
85.           B     RETREC                            REQUEST ANOTHER WINNER.
```

By loading binary 0's into register 1 and executing the MG$RET macroinstruction, your program can indicate end-of-file status to sort/merge (lines 59, 60, 84, and 85).

The merge is complete when all input files have been closed and the last winner record has been returned to your program. Sort/merge looks for the symbolic label specified on the FIN parameter of your sort parameter table (lines 27 and 79).

```
     1        10    16                                                    72
86. MERGEFIN EQU   *                              FIN ADDRESS
87.          CLOSE OUTPUT                         CLOSE OUTPUT FILE
88.          TM    CD$ISUCC,L'CD$ISUCC
89.          BZ    IOERROR
90.          EOJ
91. IOERROR  EQU   *
92.          CANCEL                               CANCEL JOB.
93.          LTORG                                DEFINE LITERALS HERE.
```

In this routine, you close the output file and indicate an end-of-job condition to job control (lines 87 and 88 through 89). Finally, you add the error routine named IOERROR as specified in your BZ instructions (lines 40, 43, 46, 49, 70, 77, 83, and 89). The LTORG assembler directive in line 93 defines all literals from your program and line. Line 94 defines the work area specified in your output RIB (line 24). All I/O buffers must be half-word aligned. Lines 98 and 99 indicate 400-byte buffer areas for each output buffer, and lines 95–97 define the three 400-byte input buffers.

```
94.  WORKAREA DS    CL8Ø
95.  BUFF1    DS    CL1Ø24                        INPUT AREA 1
96.  BUFF2    DS    CL512                         INPUT AREA 2
97.  BUFF3    DS    CL2Ø48                        INPUT AREA 3
98.  OUTØ1    DS    CL512                         OUTPUT AREA 1
99.  OUTØ2    DS    CL512                         OUTPUT AREA 2
100. WORK     EQU   *
101.          END   MRGEXMPL
```

Line 100 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 28). The END assembler control directive indicates the end of your source program. Figure 4-2 illustrates a printout of the entire source program.

```
     1        10      16                                                      72
 1.  // JOB MRGEXMPL,,7000,9000,2
 2.  // DVC 20    // LFD PRNTR
 3.  // WORK1
 4.  // WORK2                                       =//ASM
 5.  // EXEC ASM
 6.  /$
 7.  MRGEXMPL START 0
 8.           BALR  4,0
 9.           USING *,4
10.           B     START
11.           EXTRN MR$ORT                      DEFINES THE SORT COMMON MODULE
12.  *                                          TO BE INCLUDED BY THE LINKAGE
13.  *                                          EDITOR.
14.  INPUT1   CDIB
15.  MERGRIB1 RIB   BFSZ=1024,RCSZ=80,IOA1=BUFF1,IORG=(2),RCFM=FIX,             C
16.                 OPTN=YES
17.  INPUT2   CDIB
18.  MERGRIB2 RIB   BFSZ=512,RCSZ=80,IOA1=BUFF2,IORG=(2),RCFM=FIX,
19.                 OPTN=YES
20.  INPUT3   CDIB
21.  MERGRIB3 RIB   BFSZ=2048,RCSZ=80,IOA1=BUFF3,IORG=(2),RCFM=FIX,             C
22.                 OPTN=YES
23.  OUTPUT   CDIB
24.  OUTRIB   RIB   BFSZ=512,RCSZ=80,IOA1=OUT01,IOA2=OUT02,WORK=YES,            C
25.                 RCFM=FIX,OPTN=YES
26.  MERGE    MR$PRM IN=MERGEIN,                                                C
27.                 FIN=MERGEFIN,                                               C
28.                 STOR=WORK,                                                  C
29.                 RCSZ=80,                                                    C
30.                 FIELD=(0,8,CST),                                            C
31.                 MERGE=YES
32.  FILTABL  DC    A(INPUT1)                   ADDRESS OF IN1 CDIB
33.           DC    A(INPUT2)                   ADDRESS OF IN2 CDIB
34.           DC    A(INPUT3)                   ADDRESS OF IN3 CDIB
35.           USING CD$CDIB,1
36.           VTOC  CDIB=YES
37.  START    EQU   *
38.           OPEN  INPUT1,(MERGERIB1)
39.           TM    CD$ISUCC,L'CD$ISUCC
40.           BZ    IOERROR                     OPEN INPUT FILES.
41.           OPEN  INPUT2,(MERGERIB2)
42.           TM    CD$ISUCC,L'CD$ISUCC
43.           BZ    IOERROR
44.           OPEN  INPUT3,(MERGERIB3)
45.           TM    CD$ISUCC,L'CD$ISUCC
46.           BZ    IOERROR
47.           OPEN  OUTPUT,(OUTRIB)             OPEN OUTPUT FILE.
48.           TM    CD$ISUCC,L'CD$ISUCC
49.           BZ    IOERROR
50.           MR$OPN MERGE                      OPEN SORT/MERGE SUBROUTINE
     *                                          REFERENCING MR$PRM MACRO
     *                                          GENERATED AT MERGE.
51.  MERGEIN  EQU   *                           IN ADDRESS
52.           LA    5,3                         LOAD R5 WITH THE NUMBER OF
     *                                          OUTPUT FILES
53.           LA    6,FILTABL                   GET FILE TABLE ADDRESS
```

Figure 4—2. Merge-Only Program Coding (Part 1 of 3)

```
      1        10     16
 54.  FILSET   L      1Ø,Ø(6)                     LOAD CDIB ADDRESS IN R10 AND
      *                                           USE AS AN INDEX TO IDENTIFY
      *                                           INPUT FILE TO MERGE AND TO
      *                                           YOUR PROGRAM.
 55.           BAL    7,GETREC                    GET FIRST RECORD FOR EACH FILE
 56.           MG$REL                             RELEASE RECORD TO MERGE.
 57.           LA     6,4(6)                      INCREMENT TO NEXT CDIB ADDR.
 58.           BCT    5,FILSET                    TEST FOR LAST INPUT FILE: IF
      *                                           YES, CONTINUE. IF NO, GET FIRST
      *                                           RECORD OF NEXT FILE.
 59.  RETREC   EQU    *
 60.           MG$RET                             REQUEST WINNER RECORD.
 61.           BAL    7,PUTREC                    WRITE RECORD TO OUTPUT FILE.
 62.           BAL    7,GETREC                    GET NEW WINNER RECORD.
 63.           B      RETREC                      GET NEW WINNER RECORD.
 64.  GETREC   EQU    *                           GET A RECORD ROUTINE
 65.           LR     1,1Ø                        POINT TO INPUT FILE CDIB.
 66.           DMINP  (1)                         INPUT RECORD.
 67.           TM     CD$IEOF,L'CD$IEOF
 68.           BO     EOF                         IF END OF FILE, GO TO EOF.
 69.           TM     CD$ISUCC,L'CD$ISUCC
 70.           BZ     IOERROR                     IF ERROR, GO TO IOERROR.
 71.           LR     1,2                         POINT TO NEW RECORD.
 72.           BR     7                           RETURN
 73.  PUTREC   EQU    *                           OUTPUT RECORD ROUTINE
 74.           MVC    WORKAREA,Ø(1)               MOVE WINNER RECORD TO WORKAREA
 75.           DMOUT  OUTPUT,WORKAREA             OUTPUT THE RECORD
 76.           TM     CD$ISUCC,L'CD$ISUCC
 77.           BZ     IOERROR                     IF ERROR, GO TO IOERROR.
 78.           BR     7                           RETURN
 79.  EOF      EQU    *                           END-OF-FILE ADDRESS
 80.           LR     1,1Ø                        LOAD R1 WITH CDIB ADDRESS
 81.           CLOSE  (1)                         CLOSE THAT INPUT FILE
 82.           TM     CD$ISUCC,L'CD$ISUCC
 83.           BZ     IOERROR
 84.           XR     1,1                         INDICATE EOF CONDITION TO MERGE
 85.           B      RETREC                      REQUEST ANOTHER WINNER.
 86.  MERGEFIN EQU    *                           FIN ADDRESS
 87.           CLOSE  OUTPUT                      CLOSE OUTPUT FILE
 88.           TM     CD$ISUCC,L'CD$ISUCC
 89.           BZ     IOERROR
 90.           EOJ
 91.  IOERROR  EQU    *
 92.           CANCEL                             CANCEL JOB.
 93.           LTORG                              DEFINE LITERALS HERE.
 94.  WORKAREA DS     CL8Ø
 95.  BUFF1    DS     CL1Ø24                      INPUT AREA 1
 96.  BUFF2    DS     CL512                       INPUT AREA 2
 97.  BUFF3    DS     CL2Ø48                      INPUT AREA 3
 98.  OUTØ1    DS     CL512                       OUTPUT AREA 1
 99.  OUTØ2    DS     CL512                       OUTPUT AREA 2
100.  WORK     EQU    *
101.           END    MRGEXMPL
102.  /*
```

Figure 4—2. Merge-Only Program Coding (Part 2 of 3)

```
     1         10      16
103. // WORK1
104. // EXEC LNKEDT
105. /$                                        =//MERØ1 LINK MRGEXMPL
106.  LOADM MERØ1
107.  INCLUDE MRGEXMPL
108. /*
109. // DVC 51
110. // VOL DSPØ28
111. // LBL MYLIB1
112. // LFD INPUT1
113. // DVC 51
114. // VOL DSPØ28
115. // VOL MYLIB2
116. // LFD INPUT2
117. // DVC 1
118. // VOL DSPØ28
119. // LBL MYLIB3
120. // LFD INPUT3
121. // DVC 51
122. // VOL DSPØ28
123. // LBL MYLIB4
124. // LFD OUTPUT,,INIT
125. // EXEC MERØ1,$Y$RUN
126. /&
127. // FIN
```

Figure 4—2. 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 103 through 108 may be replaced by the single jproc call //MER01 LINK MRGEXMPL.

4.3.3. Assembling, Link Editing, and Executing a Merge-Only Program

The process of assembling, link editing, and executing the merge-only program is basically the same as our sort/merge disk sort program (2.11.4). Job control statements precede and follow the 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 109 through 127). 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 YRUN library. Refer to the system flowchart (Figure 2-16) which depicts assembly, linkage edit, and execution runs. Figure 4-3 illustrates your program's interface with merge-only.

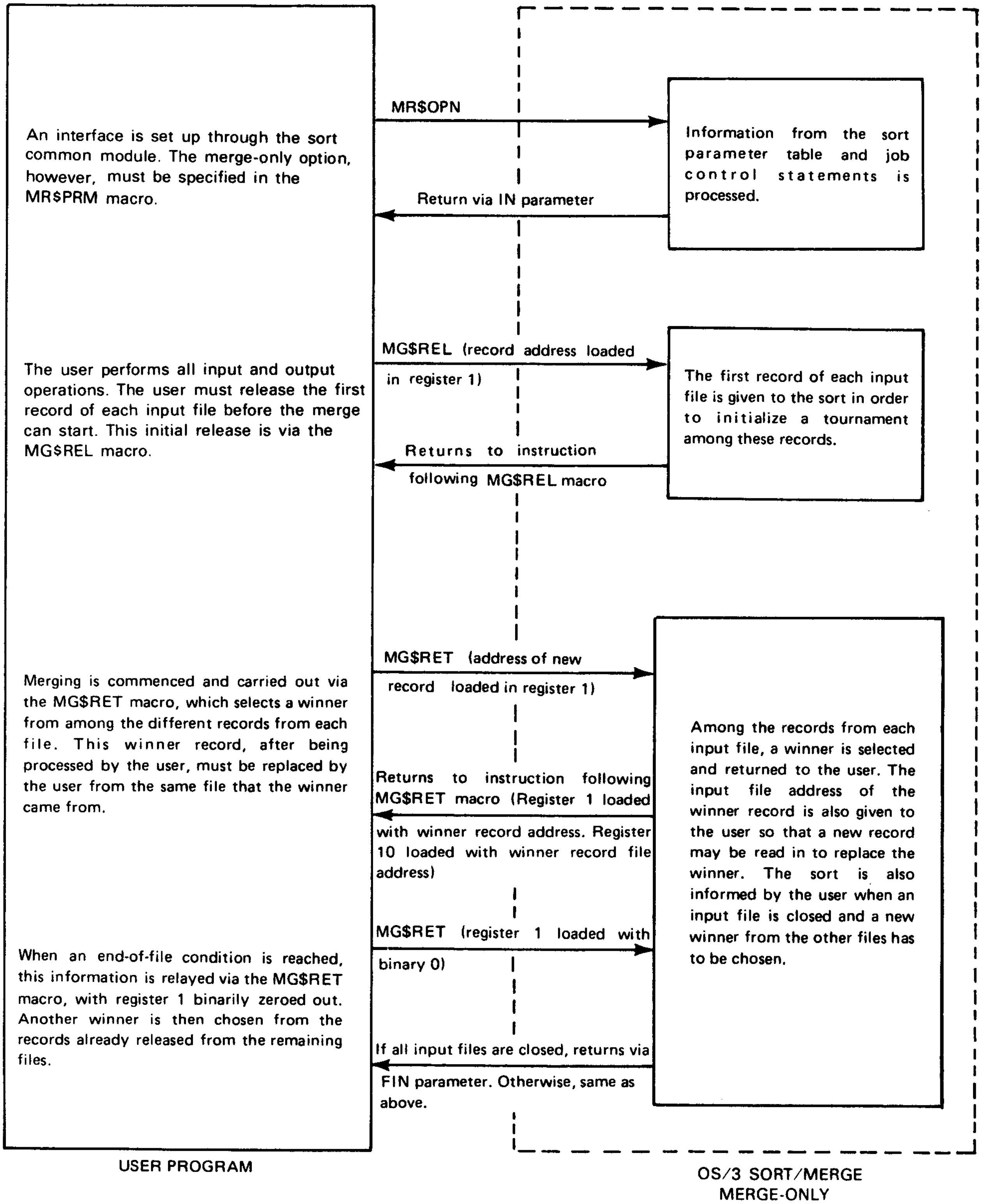


Figure 4—3. User Program Interface with Merge-Only

5. Program Examples

5.1. GENERAL

This section contains complete program coding examples and explanations of:

- A 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

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.

5.2. TAPE SORT

The following example illustrates the general requirements for performing a typical sort operation using tape work files and disk input and output files.

```
1        10     16
// JOB SRTEXMP2,,7ØØØ,9ØØØ,2
// 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    CDIB
OUTPUT   CDIB
```

```
1        10     16                                                      72
MYRIB1   RIB    BFSZ=4Ø96,RCSZ=256,IOA1=BUFF1,RCFM=FIXBLK,              C
                OPTN=YES,WORK=YES,TYPEFLE=INPUT
MYRIB2   RIB    BFSZ=4Ø96,RCSZ=256,IOA1=BUFF1,RCFM=FIXBLK,              C
                OPTN=YES,TYPEFLE=OUTPUT
         USING CD$CDIB,1
         VTOC   CDIB=YES
SORT     MR$PRM IN=SORTIN,                                              C
                OUT=SORTOUT,                                            C
                FIN=SORTFIN,                                            C
                STOR=WORK.                                              C
                TAPE=(NO,3),                                            C
                RCSZ=256,                                               C
                FIELD=(Ø,8,CH)
*
*               DATA MANAGEMENT WORK AREAS
*
         DS     OH
BUFF1    DS     CL4Ø96                    IOAREA
TAPEREC  DS     CL256                     WORK AREA
*
*
START    EQU *
         MR$OPN SORT                      OPEN SORT/MERGE SUBROUTINE.
SORTIN   OPEN   INPUT,(MYRIB1)            OPEN THE INPUT FILE.
         TM     CD$ISUCC,L'CD$ISUCC
         BZ     IOERROR
*
*               INPUT AND RECORD RELEASE ROUTINE
*
GETREC   EQU    *
         DMINP  INPUT,TAPEREC             GET RECORD FROM INPUT FILE.
         TM     CD$IEOF,L'CD$IEOF
         BO     EOF
         TM     CD$ISUCC,L'CD$ISUCC
         BZ     IOERROR
         LA     1,TAPEREC                 LOAD R1 WITH ADDR OF RECORD
*                                         TO BE RELEASED.
         MR$REL                           RELEASE RECORD TO THE SORT.
         B      GETREC
*
*               EOF ROUTINE
*
EOF      EQU    *
CLOSE    CLOSE  INPUT                     CLOSE THE INPUT FILE.
         TM     CD$ISUCC,L'CD$ISUCC
         BZ     IOERROR
         MR$SRT                           INFORM THE SORT OF THE
*                                         EOF CONDITION.
*
*               OUTPUT AND RECORD RETURN ROUTINE.
*
SORTOUT  EQU    *
         OPEN   OUTPUT,(MYRIB2)           OPEN THE OUTPUT FILE.
         TM     CD$ISUCC,L'CD$ISUCC
         BZ     IOERROR
*
```

```
1            10      16
REQREC     MR$RET                             REQUEST THE RETURN OF A
*                                             RECORD.
           MVC     TAPEREC,Ø(1)               MOVE THE SORTED RECORD TO
*                                             OUTPUT BUFFER AREA.
*
           DMOUT   OUTPUT,TAPEREC             OUTPUT THE RECORD.
           TM      CD$ISUCC,L'CD$ISUCC
           BZ      IOERROR
           B       REQREC                     REQUEST NEXT RECORD.
*
*          END OF SORT ROUTINE.
*
SORTFIN    EQU     *
           CLOSE   OUTPUT                     CLOSE THE OUTPUT FILE.
           TM      CD$ISUCC,L'CD$ISUCC
           BZ      IOERROR
           EOJ
*
*                  ERROR ADDRESS FOR DATA MANAGEMENT
*
IOERROR    EQU     *
           CANCEL                             CANCEL THE JOB
           LTORG                              DEFINE ALL LITERALS HERE
WORK       EQU     *                          SORT WORK AREA.
           END     SRTEXMPL
/*
// WORK1
// EXEC LNKEDT
/$
 LOADM   SORTØ2
 INCLUDE SRTEXMPL
/*
// DVC 5Ø
// VOL DSPØØ1
// LBL SORTIN
// LFD INPUT
// DVC 5Ø
// VOL DSP001
// LBL SORTOUT
// LFD OUTPUT
// DVC 90
// VOL SCRCH1
// LFD SMØ1
// DVC 91
// VOL SCRCH2
// LFD SM02
// DVC 92
// VOL SCRCH3
// LFD SMØ3
// EXEC SORT02,$Y$RUN
/&
// FIN
```

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 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.
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 CDIB and RIB macroinstructions describe input and output files to data management.
21-22	The VTOC macroinstruction sets up a map of the CDIB, associated by the USING directive to base register 1.
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 2.4.
33	This DS statement half-word aligns the I/O buffer.
34	This define storage statement sets up 4096 bytes for input/output buffer 1.
35	This DS statement defines the 256-byte work area that all files use for input and output.
38-39	The MR$OPN macro opens sort/merge.
40–42	The sort input routine opens the input data file.
46-55	The input and record release routine reads records and releases them to the sort.
59-64	The end-of-file routine closes the input data file and tells the sort that it has reached the end-of-file (MR$SRT).
68-71	The sort output routine opens the output data file.

Line Number	Explanation
73–81	This routine returns records from the sort/merge, writes the record, and requests the next record.
85–89	The end-of-sort routine closes the output file and notifies job control that the end-of-job condition was reached.
93–94	The IOERROR routine is the error handling routine for data management.
95	LTORG assembler control directive defines all literals at this point in your program.
96–98	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.
99	Sets up a temporary work file for the link edit step.
100	This statement executes the linkage editor.
101–104	These statements indicate the data set to control the building of the load module named SORT02.
105–108	Assigns the input file named SORTIN to volume DSP001, on device 50.
109–112	Assigns the output file named SORTOUT to volume DSP001 on device 50.
113–121	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.
122	Executes your program named SORT02, which is found in YRUN library.
123	Marks the end of the job stream.
124	Marks the end of reader operations.

5.3. TAPE SORT WITH RESTART USING PARAM STATEMENT

The following example illustrates requirements to perform a restart after a sort/merge program is interrupted. This example includes the RESUME parameter via the job control PARAM statement and the CSPRAM parameter indication.

```
1        10     16                                                      72
// JOB SRTEXM13,,7ØØØ,9ØØØ
// DVC 2Ø
// LFD PRNTR
// WORK1
// WORK2
// EXEC ASM
/$
SRTEXMPL START Ø
         BALR  4,Ø
         USING *,4
         B     START
         EXTRN MR$ORT                    THIS DEFINES THE COMMON SORT
                                         MODULE FOR INCLUSION BY THE
                                         LINKAGE EDITOR.
*
INPUT    CDIB
OUTPUT   CDIB
*
MYRIB    RIB   BFSZ=4Ø96,RCSZ=256,IOA1=BUFF1,RCFM=FIXBLK,              C
               OPTN=YES,WORK=YES
MYRIB2   RIB   BFSZ=4Ø96,RCSZ=256,IOA1=BUFF1,RCFM=FIXBLK,              C
               OPTN=YES,WORK=YES,TYPEFLE=OUTPUT
         USING CD$CDIB,1
         VTOC  CDIB=YES
*
SORT     MR$PRM IN=SORTIN,                                             C
               OUT=SORTOUT,                                            C
               FIN=SORTFIN,                                            C
               STOR=WORK,                                              C
               TAPES=(NO,3),                                           C
               RCSZ=256,                                               C
               CSPRAM=YES,                                             C
               FIELD=(Ø,8,CH)
*
*              DATA MANAGEMENT WORK AREA
         DS    OH
BUFF1    DS    CL4Ø96                    IOAREA
INOUTREC DS    CL256
*
*
START    EQU   *
         MR$OPN SORT                     OPEN SORT/MERGE SUBROUTINE
SORTIN   OPEN  INPUT,(MYRIB1)            OPEN THE INPUT FILE.
         TM    CD$ISUCC,L'CD$ISUCC
         BZ    IOERROR
*
*              INPUT AND RECORD RELEASE ROUTINE
*
GETREC   EQU   *
         DMINP INPUT INOUTREC            GET RECORD FROM INPUT FILE.
         TM    CD$IEOF,L'CD$IEOF
         BO    EOF
```

```
1        10    16
         TM    CD$ISUCC,L'CD$ISUCC
         BZ    IOERROR
         LA    1,INOUTREC                LOAD R1 WITH ADDR OF
*                                        RECORD TO BE RELEASED.
         MR$REL                          RELEASE RECORD TO THE SORT.
         B     GETREC                    GET THE NEXT RECORD.
*
*              EOF ROUTINE
*
EOF      EQU   *
         CLOSE INPUT                     CLOSE THE INPUT FILE
         TM    CD$ISUCC,L'CD$ISUCC
         BZ    IOERROR
         MR$SRT                          INFORM THE SORT OF THE END-
*                                        OF-DATA CONDITION.
*
*              OUTPUT AND RECORD RELEASE ROUTINE
*
SORTOUT  EQU   *
         OPEN  OUTPUT,(MYRIB2)           OPEN THE OUTPUT FILE.
         TM    CD$ISUCC,L'CD$ISUCC
         BZ    IOERROR
*
REQREC   MR$RET                          REQUEST THE RETURN OF A
*                                        RECORD.
         MVC   INOUTREC,Ø(1)             MOVE THE SORTED RECORD TO THE
*                                        OUTPUT BUFFER AREA
         DMOUT OUTPUT,INOUTREC           OUTPUT THE RECORD.
         TM    CD$ISUCC,L'CD$ISUCC
         BZ    IOERROR
         B     REQREC                    REQUEST THE NEXT RECORD.
*
*              END OF SORT ROUTINE
*
SORTFIN  EQU   *
         CLOSE OUTPUT                    CLOSE THE OUTPUT FILE.
         TM    CD$ISUCC,L'CD$ISUCC
         BZ    IOERROR
         EOJ
*
*              ERROR  ADDRESS FOR DATA MANAGEMENT
*
IOERROR  EQU   *
         CANCEL                          CANCEL THE JOB.
         LTORG                           DEFINE ALL LITERALS HERE.
WORK     EQU   *                         SORT WORK AREA
         END   SRTEXMPL
/*
// WORK1
// EXEC LNKEDT
/$
 LOADM   SORTØ3
 INCLUDE SRTEXMPL
/*
// DVC 5Ø
// VOL DSPØØ1
// LBL SORTIN
// LFD INPUT
// DVC 5Ø
```

```
      1        10     16
112. // VOL DSPØØ1
113. // LBL SORTOUT
114. // LFD OUTPUT
115. // DVC 9Ø
116. // VOL SCRCH1
117. // LFD SMØ1
118. // DVC 91
119. // VOL SCRCH2
120. // LFD SMØ2
121. // DVC 92
122. // VOL SCRCH3
123. // LFD SM03
124. // EXEC SORTØ3,$Y$RUN
125. // PARAM RESUME=(PASS,233)
126. /&
127. // FIN
```

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.
16-22	The CDIB and RIB macros describe input and output files to data management.
23-24	The VTOC macro sets up a map of the CDIB, and the USING directive associates it with base register 1.
26-33	SORT is the label of the MR$PRM macro that specifies sort parameter table entries. (See 2.4 for details.) Line 32 must be specified if you intend to enter the RESUME parameter via a PARAM statement in line 125.
36	This DS statement half-word aligns the I/O buffer.
37	This DS statement defines the storage area for the I/O buffer.
38	This DS statement defines the 256-byte work area used to hold input/output records.
41-42	MR$OPN opens sort/merge by specifying the name of the sort parameter table (line 26).

Line Number	Explanation
43-45	This sort input routine opens the input file.
49-58	This sort input routine reads records and releases them to the sort.
62-67	This end-of-file routine closes the input file and indicates the end-of-data (MR$SRT).
71-83	This output sort routine opens the output file, requests the return of sorted records, and writes sorted records.
87-91	The end-of-sort routine, SORTFIN, closes the output file and tells job control that the end-of-job condition was reached.
95-96	IOERROR is the error routine for data management.
97	LTORG defines all literals.
98-100	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.
101-102	WORK1 provides a temporary work file to the linkage editor and EXEC executes the linkage editor.
103-106	This is the data set to the linkage editor (the load module SORT03).
107-114	Disk input and output data files named SORTIN and SORTOUT are assigned to volume DSP001, on device 50.
115-123	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.
124	Your program named SORT03 is executed from the YRUN library.
125-126	The PARAM statement includes the RESUME parameter to provide the restart capability. (See 2.4.2.3 for more details.) The /& delimiter statement indicates the end-of-job to job control.
127	Marks the end of reader operations.

5.4. TAPE SORT USING OWN-CODE ROUTINE

The following example shows the use of a record sequence own-code routine (RSOC).

```
1        10    16                                                      72
// JOB SRTEXM15,,7ØØØ,9ØØØ
// DVC 2Ø
// LFD PRNTR
// WORK1
// WORK2
// EXEC ASM
/$
SRTEXMPL START Ø
         BALR  4,Ø
         USING *,4
         B     START
         EXTRN MR$ORT                     THIS DEFINES THE COMMON SORT
*                                         MODULE FOR INCLUSION BY THE
*                                         LINKAGE EDITOR.
INPUT    CDIB
OUTPUT   CDIB
INRIB    RIB   BFSZ=48Ø,RCSZ=8Ø,IOA1=BUFF1,                            C
               WORK=YES,OPTN=YES,TYPEFLE=INPUT
OUTRIB   RIB   BFSZ=48Ø,RCSZ=8Ø,IOA1=BUFF1,WORK=YES,                   C
               OPTN=YES,TYPEFLE=OUTPUT
         USING CD$CDIB,1
         VTOC  CDIB=YES
*
*
SORT     MR$PRM IN=SORTIN                                              C
               OUT=SORTOUT,                                            C
               FIN=SORTFIN,                                            C
               STOR=WORK,                                              C
               TAPES=(NO,3)                                            C
               RCSZ=8Ø,                                                C
               RSOC=RECCMPR
*
*
*              DATA MANAGEMENT WORK AREAS
         DS    OH
BUFF1    DS    CL4ØØ                      IOAREA
INOUTBUF DS    CL8Ø                       WORK AREA
*
*
START    EQU   *
         MR$OPN SORT                      OPEN SORT/MERGE SUBROUTINE
SORTIN   OPEN  INPUT,(INOUTRIB)           OPEN THE INPUT FILE.
         TM    CD$ISUCC,L'CD$ISUCC
         BZ    IOERROR
*
*              INPUT AND RECORD RELEASE ROUTINE
*
GETREC   EQU   *
         DMINP INPUT,INOUTBUF             GET RECORD FROM INPUT FILE.
         TM    CD$IEOF,L'CD$IEOF
         BO    EOF
         TM    CD$ISUCC,L'CD$ISUCC
         BZ    IOERROR
```

```
1        10    16                                                      72
         LA    1,INOUTBUF                LOAD REG 1 WITH ADDR OF
*                                        RECORD TO BE RELEASED.
         MR$REL                          RELEASE RECORD TO THE SORT.
         B     GETREC                    GET THE NEXT RECORD.
*
*              EOF ROUTINE
*
EOF      EQU   *
         CLOSE INPUT                     CLOSE THE INPUT FILE
         TM    CD$ISUCC,L'CD$ISUCC
         BZ    IOERROR
         MR$SRT                          INFORM THE SORT OF THE END-
*                                        OF-DATA CONDITION.
*              RECORD RETURN AND OUTPUT ROUTINE
*
SORTOUT  EQU   *
         OPEN  OUTPUT,(OUTRIB)           OPEN THE OUTPUT FILE.
         TM    CD$ISUCC,L'CD$ISUCC
         BZ    IOERROR
*
REQREC   MR$RET                          REQUEST THE RETURN OF A
*                                        RECORD.
*        MVC   INOUTBUF,Ø(1)             MOVE SORTED REC TO OUTPUT
*                                        BUFFER AREA.
         DMOUT OUTPUT,INOUTBUF           OUTPUT THE RECORD.
         TM    CD$ISUCC,L'CD$ISUCC
         BZ    IOERROR
         B     REQREC                    REQUEST THE NEXT RECORD.
*
*              END OF SORT ROUTINE
*
SORTFIN  EQU   *
         CLOSE OUTPUT                    CLOSE THE OUTPUT FILE.
         TM    CD$ISUCC,L'CD$ISUCC
         BZ    IOERROR
         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.
*
*
```

```
1        10     16
         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 Ø(8,12),Ø(11)
         DROP   15                       DISENGAGE USE OF R15 AS RSOC BASE RG
         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 LITERLAS HERE TO
*                                        FREE THE WORK AREA.
WORK     EQU    *                        SORT WORK AREA
         END    SRTEXMPL
/*
// WORK1
// EXEC LNKEDT
/$
 LOADM   SORTØ3
 INCLUDE SRTEXMPL
/*
// DVC 13Ø
// VOL DSPØØ1
// LBL SORTIN
// LFD INPUT
// DVC 13Ø
// VOL DSPØØ1
// LBL SORTOUT
// LFD OUTPUT
// DVC 9Ø
// VOL SCRCH1
// LFD SMØ1
// DVC 91
// VOL SCRCH2
// LFD SMØ2
// DVC 92
// VOL SCRCH3
// LFD SMØ3
// EXEC SORTØ3,$Y$RUN
/&
// FIN
```

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$ORT).
15-20	CDIB and RIB macroinstructions define the input and output files to data management.
21-22	The VTOC macro maps out a CDIB, and the USING directive associates it with base register 1.
25-31	MR$PRM defines the sort parameter table. Notice the name of the record sequence own-code routine is RECCMPR (line 31). For more details, see 2.4.1.
35	This DS statement half-word aligns the I/O buffer.
36-37	These instructions define storage for the half-word aligned I/O buffer area and the work area.
40-41	The MR$OPN macro opens sort/merge.
42-44	The sort input routine opens the input file.
48-57	This input routine reads input records and releases them to the sort.
61-66	The end-of-file routine closes the output file and informs the sort of the end-of-data condition.
69-72	The sort output routine opens the output file.
74-81	This routine requests the return of records from the sort and writes the record.
85-89	The end-of-sort routine closes the output file and informs job control that end-of-job was reached.
93-117	RECCMPR is the name of the user's own-code routine for record sequencing.
120-121	IOERROR is the data management error handling routine.

Line Number	Explanation
125	LTORG assembler directive defines all literals.
127-129	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.
130-135	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.
136-143	Both diskette input file SORTIN and diskette output file SORTOUT on volume DSP001 use the same device 130.
144-152	Tape work files named SM01, 02, and 03, on volumes SCRCH1, 2, and 3 reside on devices 90, 91, and 92, respectively.
153-154	Execute the program SORT03 from YRUN library and indicate end-of-job to job control (/&).
155	Marks end of reader operation.

5.5. 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 disk or tape sort (Figure 2-14 and 5.2) except for the following modifications:

- The DISC and TAPE keyword parameters specified in the MR$PRM macroinstruction 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, since 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 1.4.1 for minimum main storage requirements.

Appendix A. Statement Conventions

A.1. GENERAL FORMAT RULES

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

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

  Example:

  ```
  [,USEQ=(to-address,from-address)]
  ```

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

  Example:

  ```
  [,RESUME=(PASS,recovery=number)]
  ```

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

  Example:

  ```
  {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:

  Brackets:

  ```
  [,SIZE=number]
  ```

  Braces within brackets:

  ```
  [,DROC={DELETE}]
         {symbol}
  ```

- 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:

  ```
  [,MERGE={NO }]
          {YES}

  [,CSPRAM={NO }]
           {YES}
  ```

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

  Example:

  ```
  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.

  Example:

  Assume that MR$PRM, the sort macroinstruction, is specifying three of the required keyword parameters: FIN, IN, and OUT.

  ```
  1        10     16
  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:

  ```
  MR$OPN={parameter-table-name}
         {1                   }
  ```

- 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:

  ```
  1       10    16
  ------------------------------------
          MR$PRM FIELD=(Ø,1,AC,D,3)
          MR$PRM FIELD=(Ø,1,,D)
          MR$PRM FIELD=(Ø,1)
  ```

A.2. MACROINSTRUCTION FORMAT RULES

Macroinstructions provide an interface between sort/merge and your program. By using these macroinstructions, you define the sort run; control the function, structure, and execution of sort/merge by building a sort parameter table; and link the various functional modules of sort/merge with your program. The following rules apply to the use of the sort/merge macroinstructions presented in this manual.

1. Sort/merge control statements and macroinstructions are coded in the operation field, which may begin in any column except column 1.
2. 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.
3. At least one space must separate the operand field from the comments field, if included.
4. Embedded blanks are not allowed. Anything after a blank is regarded as a comment.
5. Values can be written with up to eight alphanumeric characters.
6. Commas, equal signs, parentheses, and blanks are used only as field delimiters; they may not be used in values.
7. 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.
8. 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.

Appendix B. Contents of Sort Parameter Table

The sort parameter table is the primary interface between your program and the modules of the sort/merge program. 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.

The sort parameter table is generated inline in your program by execution of the MR$PRM macroinstruction (2.4). Parameters can be modified via entries you submit on sort parameter statements issued in your job control stream. (See 2.12.) Table B–1 is the resulting parameter table generated inline after the MR$PRM macro is executed.

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 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 vvvv . . .		vvvv	–	A binary number specifying the number of times this record size occurs in the file or percentage of occurrences
FF FF	ssssss 00 vvvv		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 represented as length–1 for all but binary fields that are in the byte bit format where it is defined as true length. ($0 \leqslant ii \leqslant 255$)
			pppp – The location of the first byte of the key field relative to the start of the record. ($0 \leqslant pppp \leqslant 32767$)
			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 ($01 \leqslant rr \leqslant 255$)
			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 ($00 \leqslant ii \leqslant 07$)
			bb – A binary value used only in BI fields to specify the first bit location of a BI field within the byte location specified by pppp. ($00 \leqslant bb \leqslant 07$)
0C FF	0000 nn aaaaaa	DISC	nn – A binary value specifying the maximum number of disk file names which may be assigned ($0 \leqslant nn \leqslant 8$)
			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 which may be assigned ($0 \leqslant nn \leqslant 6$)
			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. Standard EBCDIC and ASCII Collating Sequences

C.1. GENERAL

Appendix C provides three useful tables containing collating sequences. The first (Table C–1) presents a cross-reference table that enables you to compare the following standard codes commonly used in data processing and in 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 C–2 provides a convenient chart of OS/3 EBCDIC graphics only, and Table C–3 lists OS/3 ASCII graphics only.

C.2. EBCDIC/ASCII/HOLLERITH CORRESPONDENCE

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

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

The column headed *Dec.* also represents the collating sequence for the ASCII graphic characters shown in the fourth column of the table; the fifth column, *Hollerith Code,* contains the hole patterns assigned to these ASCII graphics. The ASCII space character is represented in the fourth column by the conventional notation SP at decimal position 32, and the corresponding card code is *no punches.* 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 C-1).

The EBCDIC graphic characters, listed in the sixth column of Table C-1, are also in their collating sequence, and the hole patterns in the seventh column correspond to the EBCDIC graphics. The EBCDIC space character is represented by the notation *SP* in the sixth column at decimal position 64; the corresponding card code is, again, *no punches*. The empty space in the sixth column represents the positions of the EBCDIC control characters.

C.2.1. Hollerith Punched Card Code

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

C.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 C-1 shows the EBCDIC graphic characters only; the EBCDIC control characters are not indicated.

C.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 C-1.

Table C-1. Cross-Reference Table: EBCDIC/ASCII/Hollerith (Part 1 of 6)

Numeric Values			ASCII		EBCDIC	
Dec.	Hex.	Binary	ASCII Char.	Hollerith Code	EBCDIC Char.	Hollerith Code
0	00	0000 0000	NUL	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		9-7		12-9-4
5	05	0000 0101		0-9-8-5		12-9-5
6	06	0000 0110		0-9-8-6		12-9-6
7	07	0000 0111		0-9-8-7		12-9-7
8	08	0000 1000		11-9-6		12-9-8
9	09	0000 1001		12-9-5		12-9-8-1

Table C–1. Cross-Reference Table: EBCDIC/ASCII/Hollerith (Part 2 of 6)

Numeric Values			ASCII		EBCDIC	
Dec.	Hex.	Binary	ASCII Char.	Hollerith Code	EBCDIC Char.	Hollerith Code
10	0A	0000 1010		0-9-5		12-9-8-2
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		9-8-4		11-9-4
21	15	0001 0101		9-8-5		11-9-5
22	16	0001 0110		9-2		11-9-6
23	17	0001 0111		0-9-6		11-9-7
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		9-8-7		11-9-8-2
27	1B	0001 1011		0-9-7		11-9-8-3
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	SP	No punches		11-0-9-8-1
33	21	0010 0001	!	12-8-7		0-9-1
34	22	0010 0010	"	8-7		0-9-2
35	23	0010 0011	=	8-3		0-9-3
36	24	0010 0100	$	11-8-3		0-9-4
37	25	0010 0101	%	0-8-4		0-9-5
38	26	0010 0110	&	12		0-9-6
39	27	0010 0111	'	8-5		0-9-7
40	28	0010 1000	(	12-8-5		0-9-8
41	29	0010 1001	)	11-8-5		0-9-8-1
42	2A	0010 1010	*	11-8-4		0-9-8-2
43	2B	0010 1011	+	12-8-6		0-9-8-3
44	2C	0010 1100	'	0-8-3		0-9-8-4
45	2D	0010 1101	-	11		0-9-8-5
46	2E	0010 1110	.	12-8-3		0-9-8-6
47	2F	0010 1111	/	0-1		0-9-8-7
48	30	0011 0000	0	0		12-11-0-9-8-1
49	31	0011 0001	1	1		9-1
50	32	0011 0010	2	2		9-2
51	33	0011 0011	3	3		9-3
52	34	0011 0100	4	4		9-4
53	35	0011 0101	5	5		9-5
54	36	0011 0110	6	6		9-6
55	37	0011 0111	7	7		9-7
56	38	0011 1000	8	8		9-8
57	39	0011 1001	9	9		9-8-1
58	3A	0011 1010	:	8-2		9-8-2
59	3B	0011 1011	;	11-8-6		9-8-3

Table C–1. Cross-Reference Table: EBCDIC/ASCII/Hollerith (Part 3 of 6)

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

Table C–1. Cross-Reference Table: EBCDIC/ASCII/Hollerith (Part 4 of 6)

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

Table C-1. Cross-Reference Table: EBCDIC/ASCII/Hollerith (Part 5 of 6)

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

Table C–1. Cross-Reference Table: EBCDIC/ASCII/Hollerith (Part 6 of 6)

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

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

Table C-2 shows the OS/3 collating sequence for EBCDIC characters and unsigned decimal data. The collating sequence ranges from low (0000 0000) to high (1111 1111). The bit configurations that do not correspond to symbols (e.g., 0-73, 81-89, etc) are not shown. Some of these correspond to control commands for printers and other devices.

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

Table C—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 C—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	

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

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

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

Table C—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 C—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	Λ	Circumflex
95	0101 1111	_	Underscore
96	0110 0000	`	Grave accent
97	0110 0001	a	
98	0110 0010	b	
99	0110 0011	c	
100	0110 0100	d	
101	0110 0101	e	
102	0110 0110	f	
103	0110 0111	g	
104	0110 1000	h	
105	0110 1001	i	
106	0110 1010	j	
107	0110 1011	k	
108	0110 1100	l	
109	0110 1101	m	
110	0110 1110	n	
111	0110 1111	o	
112	0111 0000	p	
113	0111 0001	q	
114	0111 0010	r	
115	0111 0011	s	
116	0111 0100	t	
117	0111 0101	u	
118	0111 0110	v	
119	0111 0111	w	
120	0111 1000	x	
121	0111 1001	y	
122	0111 1010	z	
123	0111 1011	{	Opening brace
124	0111 1100	|	Vertical line
125	0111 1101	}	Closing brace
126	0111 1110	~	Tilde

Appendix D. Summary of Sort/Merge Macroinstructions

D.1. GENERAL

This appendix summarizes the sort/merge macroinstructions and is provided for quick reference only. Section 2 describes the macroinstructions in more detail.

D.2. MG$REL

Function:

Releases the initial record of a previously sequenced data file to sort/merge for merge-only processing. Since two or more input files are required for a merge-only process, the MG$REL macroinstruction must be executed for each input file involved in the merge. After the initial record of each input file has been released and the merge process begun, the MG$REL is not to be used for releasing subsequent records to sort/merge.

Prior to issuing the MG$REL macroinstruction, the user program must:

- define the input and output files and assign the devices on which they are located;
- establish the interface (EXTRN MR$ORT) between sort/merge and the user program;
- define the conditions (MR$PRM) under which sort/merge is to perform the merge-only processing;
- open the input and output files; and
- initiate (MR$OPN) sort/merge for merge-only processing.

Before releasing the initial record of an input file to sort/merge, the user program must identify both the record to be released and the file to which it belongs. This is done by loading the address of the first byte of the record into register 1 and the identifier for the file into register 10. Upon the release of the initial record of the first file involved in the merge, sort/merge returns control to the user program at the line of code immediately following the MG$REL macroinstruction. The user program must point to the next input file from which the initial record is to be accessed and released to sort/merge for merge-only processing. The procedure is repeated until the initial record of each input file involved in the merge is released to sort/merge.

Format:

LABEL	△OPERATION△	OPERAND
[symbol]	MG$REL	

No parameters are associated with the MG$REL macroinstruction.

D.3. MG$RET

Function:

Requires the return of a record processed by sort/merge in a merge-only application. The execution of this macroinstruction also initiates the process for merging data records and, with the exception of the initial record for each input file involved in the merge, releases subsequent records from these files to sort/merge for merge-only processing.

Prior to executing the MG$RET macroinstruction, the user program must release the initial record of each input file to sort/merge by execution of the MG$REL macroinstruction.

After the initial record of each input file has been released for merging, the MG$RET macroinstruction is issued, initiating the merging process. The record which meets the sequencing requirements of the program is declared the "winner". The address of the winner record is placed into register 1 and control is returned to the user program at the line of code immediately following the MG$RET macroinstruction. Make certain that the winner record is not overlayed before it is returned to your program. This can occur since register 1 is the same register in which the user program identifies the address of the next record to be released to the merge. To avoid this error, place the winner record into the output file or work area before placing the address of the next record to be released into register 1.

After the winner record is written to the output file, it must be replaced in the merge with another record from the same input file. Sort/merge places the identifier of the winner record input file into register 10 at the same time it returns the winner record address to the user program. This file identifier is used as a pointer to locate the file from which the next record is to be released to the merge. It is important not to alter the contents of register 10; otherwise, the merge will be in error. To obtain the next record from the selected file, the user program must load the address of the record into register 1. Execution of the MG$RET macroinstruction releases the record to the merge for processing.

The entire cycle repeats itself until an end-of-file condition is encountered for one of the input files (identified by the file identifier in register 10). The user program must close this file and inform sort/merge of the end-of-file condition by loading a binary 0 into register 1 before releasing additional records to the merge.

Format:

LABEL	△OPERATION△	OPERAND
[symbol]	MG$RET	

No parameters are associated with the MG$RET macroinstruction.

D.4. MR$OPN

Function:

Generates the linkage necessary to activate the sort initialization module. This module performs the actual initialization procedures required prior to sort/merge execution. Sort/merge must be opened before data records are delivered to it for sorting and merging. Once the subroutine is opened, control is returned to the user program at the address specified by the IN keyword parameter of the MR$PRM macroinstruction. Before executing the MR$OPN macroinstruction, the user program must:

- define the input and output files;
- establish a communications interface (EXTRN MR$ORT) for modules of sort/merge;
- generate the sort parameter table (MR$PRM) to define the requirements of the sort;
- establish input and output data areas; and
- open the input data files for processing. The input file can also be opened after sort/merge has been opened; however, no attempt must be made to release records to the sort until the input data files are opened.

Format:

LABEL	△OPERATION△	OPERAND
[symbol]	MR$OPN	{parameter-table-name [illegible]}

Positional Parameter:

parameter-table-name

Specifies the symbolic label of the address of the sort parameter table (MR$PRM macroinstruction).

(1)

Indicates that register 1 has been loaded with the address of the parameter table.

D.5. MR$PRM

Function:

Generates a sort parameter table that defines the requirements of a particular sort/merge run. The specifications governing the operations to be performed are defined by the keyword parameters associated with this macroinstruction. Each keyword parameters specified becomes an entry in the sort parameter table to be used by sort/merge during program execution. All table entries pertaining to a particular sort/merge operation must be properly defined before attempting program execution.

Format:

LABEL	△OPERATION△	OPERAND
[symbol]	MR$PRM	see below

```
{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]
[,PRINT={ALL     }]
[       {CRITICAL}]
[       {NONE    }]
[,RESERV=sort-filename]
[,RESUME=(PASS,recovery-number)]
[,SHARE=sort-filename]
[,SIZE=number]
[,USEQ=(to-address,from-address)]
```

Keyword Parameters:

```
FIELD=(strt-pos-1,lgth-1[,form-1][,seq-1][,order-1][,...,strt-pos-n,lgth-n
    [,form-n][,seq-n][,order-n]])
```

Defines the sort key fields to sort/merge. Key field definition includes starting position, length, data format, sorting sequence, and order of significance. A maximum number of 12 key fields can be specified. When variable-length records are involved, the 4-byte record length field is considered a part of the record. All key fields of the record must be contained within the first bin of the record.

This keyword parameter should not be coded if an own-code routine is supplied for record sequencing; in this case, RSOC must be specified.

Positional Subparameters:

`strt-pos-n`

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

Key fields, with the exception of binary key fields, start at a full byte boundary, and their starting point is defined by specifying their byte position within the record. Byte positions are numbered from 0 while byte numbers begin at 1, so the byte position of a key field is always 1 less than its byte number. For example, if the first key field begins at byte 10 of a record, the *strt-pos-1* subparameter would be specified as 9.

The starting position for a binary key field is not limited to a byte boundary but can start at a bit position within a byte. In this case, the key field starting position is defined in a byte-bit form of reference. For example, if a key field starts at bit position 2 of byte 10 of the record, its starting position would be defined as 9.2.

`lgth-n`

A decimal number specifying the length of the key field. Key field length is defined in full bytes beginning and ending on a byte boundary except for binary key fields. A binary key field may be specified in the byte bit format, based upon the number of bits that the field occupies. For example, assume that the binary key field extends from bit position 2 of byte 10 through bit position 5 of byte 12, a total of 20 bits. This would be specified as 2.4.

`form-n`

A 2- or 3-character code specifying the data format of the key field. If omitted, the format is assumed to be character (CH). The format codes and their maximum allowable field lengths are shown in Table 2-1.

`seq-n`

Defines the sorting sequence for the key fields, A for ascending and D for descending sequence. If omitted, ascending sequence is assumed.

order-n

A decimal number specifying the significance of the record key fields from major to minor. The major key field is numbered 1, the next most significant is 2, and so on. The maximum number of key fields that can be specified is 12.

If omitted, sort/merge assumes the order of key field definition as the order of significance; that is, the first key field defined is the major key field, etc. However, if this subparameter is specified for one key field, it must be specified for all key fields.

RSOC=symbol

Required when key field comparisons for record sequencing are to be performed through a user own-code routine. This keyword parameter specifies the symbolic address of the own-code routine and overrides the specifications of the FIELD keyword parameter if both are coded.

FIN=symbol

Defines the symbolic address in the user program to which control is returned when the output end-of-data has been reached.

IN=symbol

Defines the location within the user program to which control is returned after sort/merge has been initiated.

OUT=symbol

Defines the location within the user program to which control is returned when sort/merge is ready to return records to the program.

RCSZ=max-bytes

Defines the size of fixed-length records or the maximum size of variable-length records of the data to be sorted. The size specified for variable-length records must include the 4-byte record length field that precedes each record. If tag sort has been specified (ADDROUT keyword parameter), the record size must equal the combined length of all key fields specified plus the 10-byte record access address field. The maximum allowable record size specified by this keyword parameter is dependent upon the hardware configuration.

STOR=symbol

Specifies the symbolic address of the main storage area available to sort/merge. When this form is used, the sort will utilize all main storage in the job region from the location specified by the symbol to the end of the job region as defined in the job prologue.

STOR=(symbol,number-of-bytes)

Positional Subparameters:

symbol

Specifies the symbolic address of the available main storage.

number-of-bytes

A decimal number indicating the maximum number of bytes available in main storage starting at the address specified.

ADDROUT={A
D}

Specifies that a tag sort is to be performed. The first 10 bytes of the record to be released must contain the record access address field.

ADDROUT=A

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

ADDROUT=D

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

ADTABL=symbol

Specifies the symbolic address of an additional sort parameter table within the user program which is to be linked to the existing sort parameter table. Any number of tables may be linked in this manner. ADTABL should be coded as the last parameter on the MR$PRM macro; any subsequent parameter entries are ignored.

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

Required when variable-length records are to be sorted. Defines the size of fixed-length subrecords (bin size), or provides the information from which sort/merge can calculate the bin size.

BIN=bytes

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

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

Positional Subparameters:

min-bytes

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

size-1

Defines the record length (in bytes) that appears most frequently in the file.

freq-1

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

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

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

CALC=NO
Specifies that sort/merge is to calculate the optimum working-storage area in a disk sort, display the estimated sort time in minutes and the number of cylinders required for work space, and then terminate the job step.

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

If the CALC parameter is used, the SIZE keyword parameter and all record description keyword parameters must be specified.

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

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

DISC=(address,max-disk-file-number)
Indicates the symbolic address of a list of user-supplied file names for disk work files and the maximum number of file names in the list.

DISK=max-disk-file-number
Indicates the maximum number of standard disk files (DM01 through DM08 or $SCR1 through $SCR8) assigned to sort/merge as working storage.

TAPE=label-type
Identifies tape work files as unlabeled or standard labeled. For unlabeled tapes, specify NO; for standard labeled tapes, specify STD.

TAPE=(label-type,max-file-number)
Identifies tape work files as unlabeled (NO) or standard labeled (STD) and specifies the maximum number of tape files assigned to sort/merge as working storage. Three to six tape files may be assigned.

If the DISC and TAPE keyword parameters are omitted, sort/merge determines the type and number of work files from job control. If work files are not assigned, an internal (main storage) sort is performed.

DROC=DELETE

Specifies that sort/merge is to perform automatic data reduction; i.e., eliminate or combine records with equal key fields.

DROC=symbol

Specifies the symbolic address of a user own-code routine for data reduction.

MERGE=YES

Specifies a merge-only application. If omitted, a sort/merge operation is assumed.

NOCKSM=D

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

NOCKSM=T

Suppresses the checksum calculation for tape work files.

PAD=bytes

Augments the parameter table beyond its generated length, allowing the user to enter additional parameters into the table at run time. The number of additional bytes must be specified in multiples of 4.

PRINT=ALL

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

PRINT=CRITICAL

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

PRINT=NONE

Specifies that no error messages are to be written to the job log.

NOTE:

Regardless of PRINT option, all error messages are displayed on the system console.

RESERV=sort-filename

Allows a tape unit to function as a work file device during the input and intermediate phases of sort/merge operation and as the device for the output data file during the output phase. The reserved tape file is identified by a standard sort work file name (SMnn) and is associated with this name thorugh an LFD job control statement. Messages instructing the operator when to unload the scratch tape and mount the output tape are displayed on the system console. The same device cannot be used for both RESERV and SHARE.

RESUME=(PASS,recovery-number)

Restarts an interrupted tape sort/merge operation. The interrupted sort can be resumed at a tape collation pass.

Positional Subparameters:

PASS
Specifies resumption of a sort that was interrupted during tape collation.

recovery-number
Specifies the most recent collation pass number displayed on the system console for recovery of a tape sort.

SHARE=sort-filename
Allows a tape unit assigned to sort/merge as an input device to be used as a sort work file during the intermediate and output phases. The shared tape file is identified by a standard sort tape file name (SMnn) and is associated with this name through an LFD job control statement. Messages instructing the operator when to unload the input tape and mount the scratch tape are displayed on the system console. The same device cannot be used for RESERV and SHARE.

SIZE=number
Specifies the approximate number of records in the input file. This information is used for calculating optimum working-storage area when the CALC parameter is specified. If SIZE is omitted, a file of 25,000 records is assumed.

USEQ=(to-address,from-address)
Required to perform a collation sequence for 8-bit character data differing from EBCDIC or ASCII representation. Both positional subparameters must be specified regardless of whether or not two translation tables are used. Usually one table is sufficient to perform the necessary translations. In such cases, both subparameters are coded with the same address. This keyword must be included if USEQ is speciifed in the *form* subparameter of the FIELD keyword parameter.

Positional Subparameters:

to-address
Specifies the address of a 256-byte translation table that translates the record field or fields into the user collation character format for the desired sequence.

from-address
Specifies the address of a 256-byte translation table that translates the field or fields back to the original data format for output.

D.6. MR$REL

Function:

Releases unsorted records to sort/merge for processing. The user program must identify each record before it is released. To do this, precede the MR$REL macroinstruction with an instruction to load register 1 (R1) with the address of the first byte of the record to be transferred. Execution of the MR$REL macroinstruction generates the linkage required to release the record identified in R1. After the transfer has taken place, sort/merge returns control to the user program at the line of coding immediately following the MR$REL macroinstruction. The user program may now get the next record to be released to the sort. The process of releasing records to the sort repeats itself until an end-of-file condition is detected, indicating that all records are released to the sort; the input may now be closed. The user program must then execute the MR$SRT macroinstruction to request sort/merge to complete the sort.

If work files are on disk, a routine can be included which will check on the availability of work space before each record is passed to sort/merge. When control is returned to the user program after the MR$REL instruction, R1 will be set to a positive value if more records can be accepted, or to a negative value if work space is insufficient to complete the sort. The user program can check R1 and branch to end-of-file or an error routine.

Format:

LABEL	△OPERATION△	OPERAND
[symbol]	MG$REL	

No parameters are associated with the MR$REL macroinstruction.

D.7. MR$RET

Function:

Requests the return of sorted records to the user program. Sort/merge notifies the user program that is is ready to return the sorted records by returning control at the address specified by the OUT keyword parameter in the sort parameter table. At this time, the user program may open its output data file and then request the return records by executing the MR$RET macroinstruction. Since the records are released one at a time, the user program must request the return of each record.

One method of accomplishing this is to set up a loop within the program that requests the return of a record and handles the disposition of that record to the output data file. The loop functions in the following manner: The MR$RET macroinstruction requests the return of a record from sort/merge. After MR$RET is executed, sort/merge places the address of the record being returned into register 1 and returns control to the user program at the line of code immediately following the MR$RET macroinstruction. The user program, at this point, must move the record into the user output buffer area; final disposition of the records in the output buffer area is the responsibility of the user. The program can then branch back to the MR$RET macroinstruction and request sort/merge to return the next record. This loop process repeats itself until an end-of-data condition is encountered indicating that all of the sorted records have been returned to the user program. Control at this point is returned to the user program at the address specified by the FIN keyword parameter in the sort parameter table. This address is the line of code following the record request loop and usually pertains to closing the output data file.

Format:

LABEL	△OPERATION△	OPERAND
[symbol]	MR$RET	

No parameters are associated with the MR$RET macroinstruction.

D.8. MR$SRT

Function:

Notifies sort/merge that the end of input data has been reached and that it may now complete the processing of the input data records.

The MR$SRT macroinstruction should not be executed until all of the input data records have been released to sort/merge. The user program may then close the input data file, but this is not required for continuing the program. When all the records are sequenced, sort/merge returns control to the user program at the address specified by the OUT keyword parameter in the sort parameter table. The output data file can now be opened, and return of the sorted records to the program can be requested. The request for the return of the sorted data records is made by use of the MR$RET macroinstruction.

Format:

LABEL	△OPERATION△	OPERAND
[symbol]	MR$SRT	

No parameters are associated with the MR$SRT macroinstruction.

Index

CUT

SPERRY

USER COMMENT SHEET

We will use your comments to improve subsequent editions.

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

(Document Title)

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

Comments:

From:

(Name of User)

(Business Address)

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

CUT

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 21 BLUE BELL, PA.

POSTAGE WILL BE PAID BY ADDRESSEE

SPERRY CORPORATION

ATTN.: SOFTWARE SYSTEMS PUBLICATIONS

P.O. BOX 500
BLUE BELL, PENNSYLVANIA 19424

FOLD

SPERRY

USER COMMENTS

We will use your comments to improve subsequent editions.

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

(Document Title)

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

Comments:

From:

(Name of User)

(Business Address)

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

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 21 BLUE BELL, PA.

POSTAGE WILL BE PAID BY ADDRESSEE

SPERRY CORPORATION

ATTN: SYSTEM PUBLICATIONS

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

FOLD

UNISYS

USER COMMENTS

We will use your comments to improve subsequent editions.

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

(Document Title)

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

Comments:

From:

(Name of User)

(Business Address)

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

CUT

FOLD

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 21 BLUE BELL, PA.

POSTAGE WILL BE PAID BY ADDRESSEE

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

FOLD