MRX/OS Disc Sort

Reference Manual

2200.009

MEMOREX

Computer System Products

November 1972 Edition

Requests for copies of Memorex publications should be made to your Memorex representative or to the Memorex branch office serving your locality.

A reader's comment form is provided at the back of this publication. If the form has been removed, comments may be addressed to the Memorex Corporation, Publications Dept., 8941 – 10th Ave. No. (Golden Valley) Minneapolis, Minnesota 55427.

PREFACE

This manual is a reference for programmers using the MRX/OS Disc Sort program (SORT). Chapters 1-3 describe the function of the SORT program and the Control Language statements by which a sort run is defined. Chapters 4-5 describe the internal phases of SORT, and how the user may write modification routines for the SORT program. A sample program appears in Appendix D.

Additional information on Control Language can be found in the publication **MRX/OS Control Language Services, Extended Reference.** File organization is fully discussed in the publication **MRX/OS Control Program and Data Management, Basic Reference.**

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

FUNCTIONS OF DISC SORT

The MRX/OS Disc Sort program (SORT) provides the user with the capability to sort a randomly ordered file or merge two or more presorted files. A file may be sorted in ascending or descending order according to key fields within the record. The key fields are defined by the user according to position within the record, length, and type of field (EBCDIC, ANSI, packed decimal, zoned, binary, user, tag-along, select, reject, and force).

SORT accepts a variety of data file organizations (sequential, relative, and indexed) as input files from disc, tape, or cards. From these input files, records are sorted in the specified sequence and written as tag files, address files, or data files. The sort may be performed as a full sort or a tag sort. With the use of the ACTION=RETRIEVE option full records may be retrieved from the input file for a full record output file after a tag sort was performed. The user may write his own subroutines with the usage of the User Modification Routines. A predetermined value may be forced into a record by using TYPE=FORCE.

OUTLINE OF CONTROL LANGUAGE FOR SORT

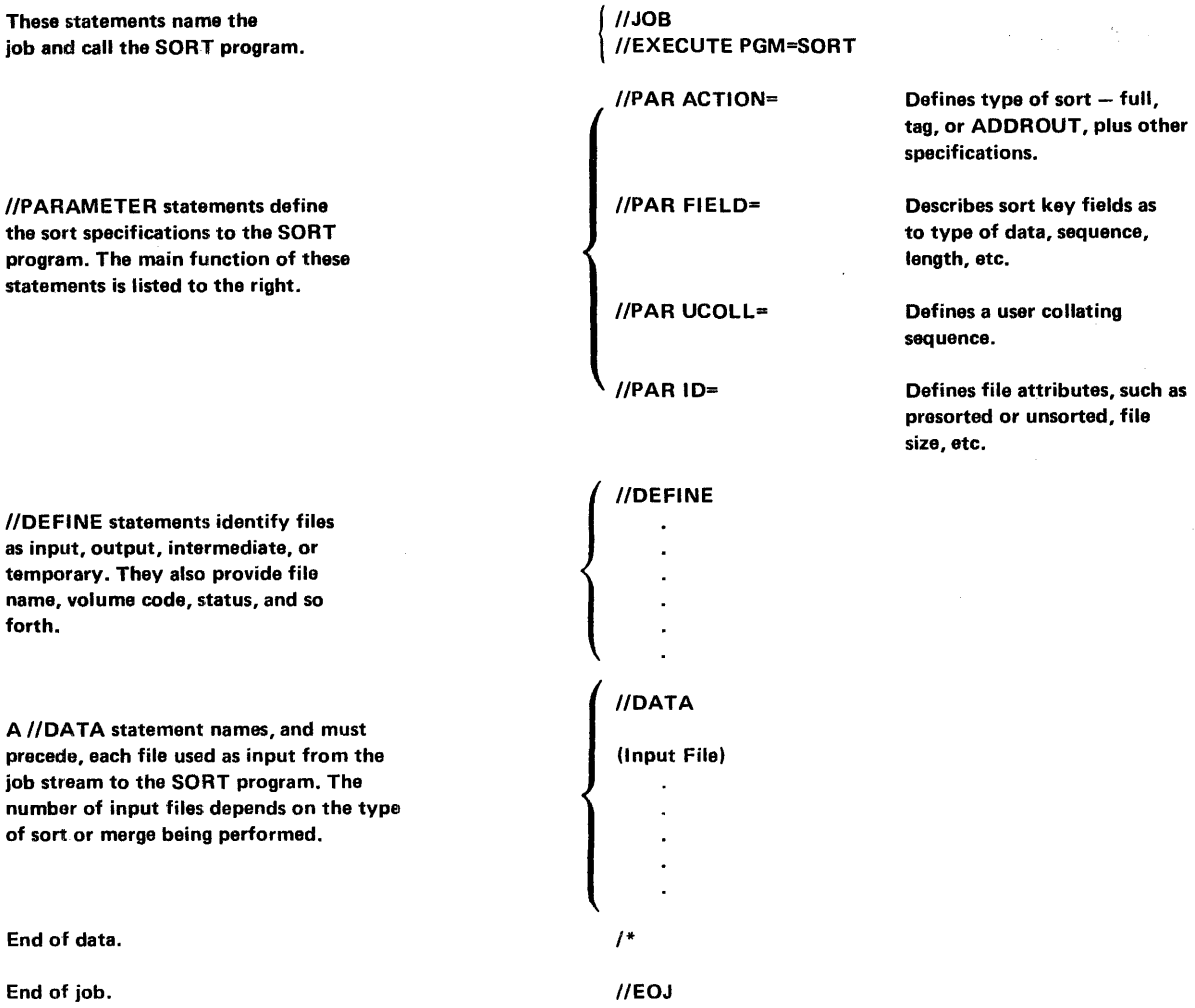
A sort job is specified to the SORT program through the use of Control Language statements. Specifications that are unique to SORT are defined by //PAR statements. Figure 1-1 illustrates the use of Control Language for a sort job.

RELATIONSHIP TO SYSTEM

SORT uses the input/output facilities of MRX/OS for file assignment and release and the input/output facilities at both the logical I/O and the block I/O level.

SYSTEM REQUIREMENTS

In addition to the minimum hardware requirements for MRX/OS, a user partition of at least 8000 bytes and at least one disc drive must be available for execution of the SORT program. A larger user partition and/or additional disc drives will lead to a more efficient sort of a given file.



NOTES ON CONTROL LANGUAGE

1. The following statements must be used once, and only once: //JOB, /*, and //EOF.

2. //DEFINE is required and may be used more than once, depending on the number of files in the sort run.

3. //PAR statements have both mandatory and optional specifications. The default values are provided in the text describing these statements.

Figure 1-1. Input to the Disc Sort Program

2. FILES

The SORT program accepts a variety of file organizations and file assignments. From these various input files, records are sorted in the specified sequence and written as tag files, address files, or data files.

FILE ORGANIZATION

SORT accepts any data file organization supported by the Data Management system. Sequential, relative, and indexed files are described in the publication **Control Program and Data Management Services, Basic Reference.**

FILE STRUCTURE

The output files of the SORT program may be data files, tag files, or address files (ADDROUT). Each of these file structures are discussed in the following paragraphs.

DATA FILES

Data files contain entire records, as opposed to address records. These files may be disc, tape, or unit record files. Disc files or magnetic tape files may have either fixed or variable length records. Maximum record size is 32K bytes. Records can be blocked up to 255 records per block. The minimum block size is 18 bytes.

The data records are stored in a standard data format as described in the publication **Control Program and Data Management Services, Basic Reference.**

TAG FILES

If the TAGSORT option from the ACTION statement is selected, SORT creates a tag file composed of data blocks containing tag records. The tag record consists of a record address and optional sort, tag-along, or forced fields.

The data blocks contain fixed length tag records, the length of which is determined by the sum of the sort key fields* plus four bytes. Within the tag record, the first four bytes contain the corresponding data record address. The remainder of the tag record contains sort key fields, optional tag-along fields, and optional forced fields (Figure 2-1). The order of these fields following the record address is determined by the order specified in the FIELD statement.

Figure 2-1. Tag Record

The tag records may have one of two formats depending upon the type of record address desired. The RECADD parameter of the ACTION statement determines whether Format A (containing the logical record number) or Format B (containing the logical block number and record number) will be used. These formats are shown in Figures 2-2 and 2-3.

If RECADD=LOGREC, Format A is used for the tag record in the data block. In this format, the 4-byte logical record number corresponds to the appropriate record in the input data file. The sort key fields and their accompanying tag-along fields and forced fields follow the logical record number with each field beginning on an even byte boundary. If these fields have an odd number of bytes, padding with blanks is performed.

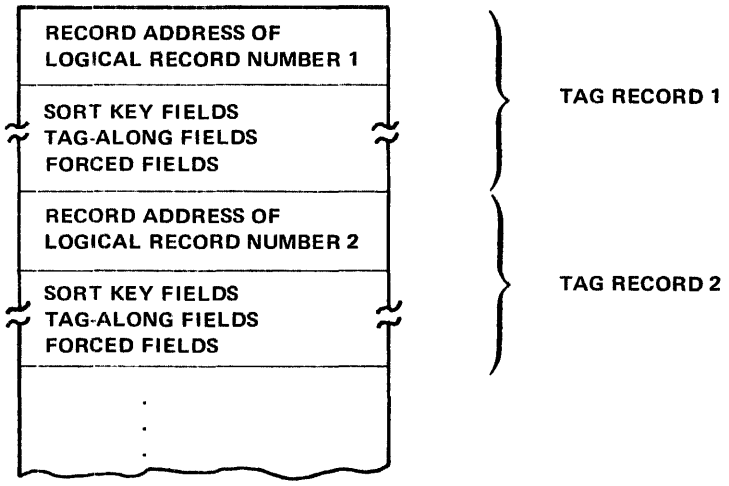


Figure 2-2. Format A for Tag Files

*Each field is adjusted to an even number of bytes. The block is padded to 18 bytes in length if the block is too short.

However, if RECADD=BLKREC, Format B is used.

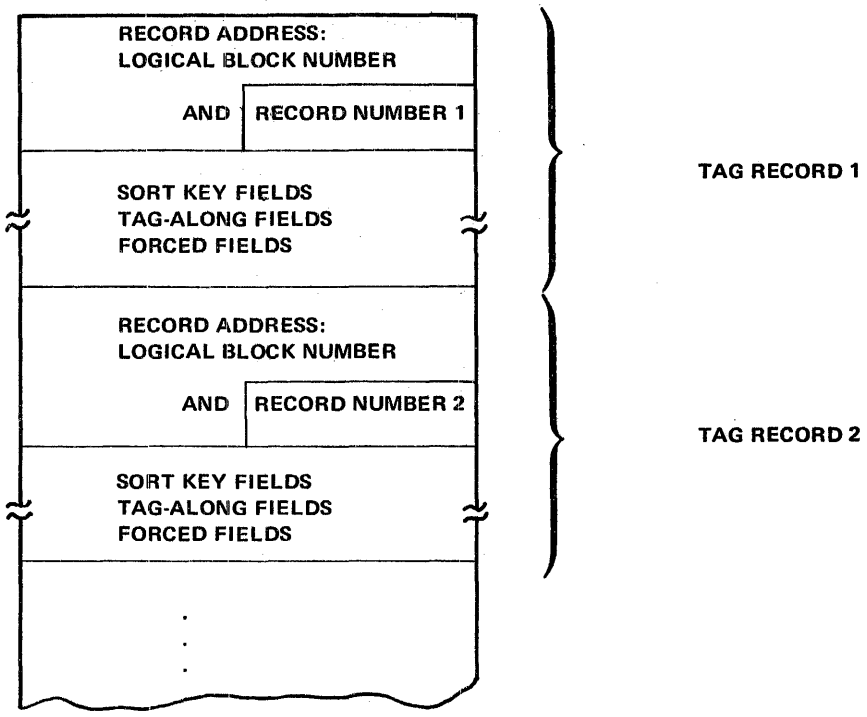


Figure 2-3. Format B for Tag Files

The 3-byte logical block number in Format B locates the block in the input data file, and the 1-byte record number locates the corresponding data record in that block. The sort key fields, tag-along fields and forced fields are the same as in Format A.

ADDROUT FILES

A special case of tag file output is the ADDROUT file, in which the sort keys have been dropped, leaving only data record addresses. The ADDROUT record has two formats depending upon the record address type.

If RECADD=LOGREC (in the ACTION statement), the 4-byte logical record number appears as illustrated by Format A (Figure 2-4).

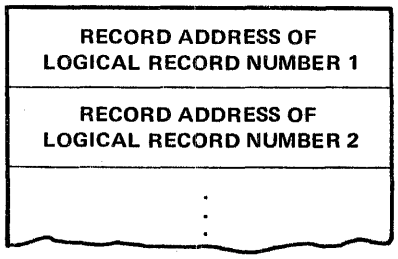


Figure 2-4. Format A for ADDROUT Files

If RECADD=BLKREC, Format B (Figure 2-5) is used with a 3-byte logical block number and a 1-byte record number within that block.

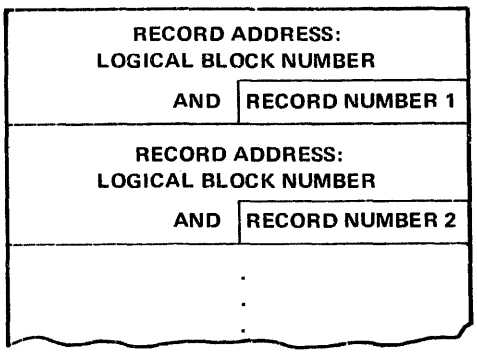


Figure 2-5. Format B for ADDROUT Files

FILE ASSIGNMENT

The SORT program uses five basic file assignments.

1. Input files
2. Output file
3. Intermediate files
4. Scratch file
5. List file

File assignments are specified by the identifier parameter (ID=) of the //DEFINE statement. (For a complete description of the //DEFINE statement refer to the **MRX/OS Control Language Services, Extended Reference.**) The file identifiers are also used to identify the file in the ID parameter for the SORT file attributes. (File attributes are described in Chapter 3 of this manual.) The following paragraphs define and describe the file identifiers.

INPUT FILES

Input to the SORT run may be a single unsorted file or two to seven presorted files. An input file may be on any input device supported by Data Management. A maximum of seven presorted input files may be merged in one run.

The file identifiers are defined as follows:

Identifier	Description
SRTINP	Unsorted input file
SRTPI1	First presorted input file
SRTPI2	Second presorted input file
SRTPI3	Third presorted input file
SRTPI4	Fourth presorted input file
SRTPI5	Fifth presorted input file
SRTPI6	Sixth presorted input file
SRTPI7	Seventh presorted input file

The user must assign files by using the Control Language //DEFINE statement. For example, the following statements specify the presorted input files named TUESDAY and SUNDAY.

```
//DEF   ID=SRTPI2,FILE=TUESDAY,STA=(P,I),VOL=ABC003,SIZ=128
//DEF   ID=SRTPI1,FILE=SUNDAY,STA=(P,I),VOL=ABC001,SIZ=128
```

OUTPUT FILES

The user designates an output file by using a Control Language //DEFINE statement with SRTOUT as the file identifier. An output file may be assigned to any output device supported by Data Management. Output files assigned to unit record devices include a control character which is supplied by the SORT program. This control character is the first byte of each record.

The following Control Language //DEFINE statement specifies an output file named WEEKLY.

```
//DEF   ID=SRTOUT,FILE=WEEKLY,STA=(P,O),VOL=ABC008,SIZ=128
```

INTERMEDIATE FILES

SORT normally assigns two intermediate files, SRTWKA and SRTWKB to on-line shared devices. The intermediate files, which are scratch files, have block and record sizes which are dependent upon the input record file size and the SORT action required. The number of blocks allocated to the intermediate file is approximately 1.5 times the number of blocks in the input file.

To improve efficiency of the sort run, the user may force assignment of intermediate files to nonshared disc packs by using the following Control Language statements, where xxxxxx and zzzzzz are the volume identifiers of the nonshared packs.

```
//DEFINE    ID=SRTWKA,FIL=DUMMY,VOL=xxxxxx,DEV=DISC
//DEFINE    ID=SRTWKB,FIL=DUMMY,VOL=zzzzzz,DEV=DISC
```

SCRATCH FILES

SORT allocates a scratch file with eighteen 256-byte blocks. This file contains interphase information that is necessary for the sort run.

LIST FILES

A LIST file must be assigned in a //DEFINE statement when ACTION=DUMP is specified. The following Control Language statement specifies a printer output to accommodate the specified dump option.

```
//DEFINE    ID=LIST,DEV=PRINTER
```

3. SORT CONTROL LANGUAGE

INTRODUCTION

The SORT program is called into execution by the //EXECUTE PGM=SORT Control Language statement. Following the EXECUTE statement is a series of control statements (//PAR) which define the characteristics of the particular SORT run.

The SORT run is defined by the following identifier parameters:

Identifier Parameter	Characteristic
ACTION	Sort actions
FIELD	Sort fields
UCOLL	User collating sequence
ID	File attributes

The associated parameters follow the identifier parameter which is necessary for associated parameters. At least one FIELD statement is always required. The other identifier parameter should be given once; if more than one parameter appears, the final value overrides the preceding values.

In most cases (exceptions in the ID parameters), the SORT program interrogates only the first character of each parameter. Thus, the example:

```
//PAR    FIELD=7,LENGTH=2,TYPE=BINARY,SEQUENCE=DESCEND
```

may be abbreviated as:

```
//PAR    F=7,L=2,T=B,S=D
```

In the //PAR statement, the keywords are separated by commas. All the identifier parameters may appear on a single //PAR card; for example:

```
//PAR    ACTION=TAGSORT,FIELD=6,LENGTH=5,ID=SRTINP
```

However, if more than one //PAR card is necessary or desired, a new //PAR card follows without any ending punctuation on the first card; for example:

```
//PAR   ACTION=TAGSORT
//PAR   FIELD=6,LENGTH=5
//PAR   ID=SRTINP
```

In the following text, optional entries are denoted by brackets,[ ] ; parameters with a choice of specification are denoted by braces, { } , with the default case being underlined.

SORT ACTIONS

The ACTION identifier parameter statement defines the different sort actions. If ACTION=FULLSORT and there are no associated parameters (MESSAGE, ERROR, etc.), the ACTION statement is not required. The ACTION statement must be included if associated parameters are desired. The format of the ACTION statement is as follows:

```
[ ACTION= { FULLSORT }                ]
          { TAGSORT  }
          { ADDROUT* }
          { RETRIEVE*}
          { MERGE    }
          { SEQUENCE }
          { ESTIMATE }
          { GENERATE }
          { DUMP     }
[ MESSAGE= { BRIEF    }               ]
           { DETAILED }
           { NONE     }
[ ERROR= { STOP }                     ]
         { DROP }
         { GO   }
[ OVERFLOW= { STOP }                  ]
            { GO   }
[ VERIFY= { NO  }                     ]
          { YES }
[ START=n]
[ QUIT=n ]
[ PRESORT=n]
[ RECADD= { LOGREC }                  ]
          { BLKREC }
```

*Limited to random access devices

ACTION= { **FULLSORT** | **TAGSORT** | **ADDROUT*** | **RETRIEVE*** | **MERGE** | **SEQUENCE** | **ESTIMATE** | **GENERATE** | **DUMP** } **(Optional)**

The ACTION parameter specifies the sorting process to be performed.

If ACTION=FULLSORT, SORT produces, from an unsorted data file, an output file having ordered full data records. If the ACTION statement is omitted, FULLSORT is the default case.

If ACTION=TAGSORT, an output tag file is produced from an unsorted disc file. The tag records are carried throughout the sort run.

If ACTION=ADDROUT, SORT performs a tag sort from an unsorted disc file, but drops all the sort fields from the final output records. Consequently, a file of data file record addresses is produced.

If ACTION=RETRIEVE, SORT performs a tag sort from an unsorted disc file, and carries the tag records throughout the intermediate stages of the sort. When the tag records are in order, SORT retrieves the data file records from the input file and produces the final output file in the same order as the tag file. The RETRIEVE option is not valid for indexed files.

If ACTION=MERGE, two or more presorted data files are merged into one output file.

If ACTION=SEQUENCE, a sequence check of one or more input files is performed and records out of sequence are reported.

If ACTION=ESTIMATE, an estimate of the amount of time required to sort the input data file (SRTINP) is produced. (Time estimates are automatically supplied for both FULLSORT and RETRIEVE options.)

If ACTION=GENERATE, a test file described by FIELD statements is generated on file SRTOUT. The sort process is not begun, as this option is for checkout and demonstration purposes.

If ACTION=DUMP, a formatted listing of sort fields of each record of SRTINP described by FIELD statements is produced. This option is for checkout and demonstration purposes. A LIST file must be specified in a //DEFINE statement for use with the DUMP option.

*Limited to random access devices

MESSAGE= {**BRIEF** / **DETAILED** / **NONE**} **(Optional)**

The MESSAGE option determines what kind of message format will be used. Appendix C illustrates the difference between brief and detailed listings. If MESSAGE=BRIEF, messages are written in brief listing format; this is the default case. If MESSAGE=DETAILED, messages are written in detailed listing format. If no messages are desired, MESSAGE=NONE suppresses all messages.

ERROR= {**STOP** / **DROP** / **GO**} **(Optional)**

The ERROR option enables continuance of processing if an I/O error is detected. If ERROR=STOP, the sort run is terminated when an I/O error is detected; this is the default case. If a read or write error is detected, the record block in question is dropped if ERROR=DROP. If ERROR=GO, the error is ignored, and the block is accepted as correct. (Appendix E explains error procedures.)

OVERFLOW= {**STOP** / **GO**} **(Optional)**

The OVERFLOW option allows continuance of sorting when intermediate file storage is insufficient. If there is insufficient intermediate file storage to accommodate all input records, the sort run is terminated when OVERFLOW=STOP (this is the default case). Whereas if OVERFLOW=GO, as many records as possible are sorted. (Appendix E explains error procedures.)

VERIFY= {**NO** / **YES**} **(Optional)**

The VERIFY option specifies write verification for SRTOUT files. VERIFY=YES indicates write verification. If VERIFY=NO or if omitted, no write verification is performed.

START=n **(Optional)**

This optional parameter gives the logical record number of the first record to be sorted. The default value is one. This option is not available for indexed files.

QUIT=n **(Optional)**

This optional parameter gives the logical record number of the last record to be sorted. The default is the last record in the input file. This option is not available for indexed files.

PRESORT=n **(Optional)**

This parameter, which can be used only in conjunction with ACTION=MERGE, gives the number of presorted input files.

RECADD= $\left\{\begin{matrix}\textbf{LOGREC}\\\textbf{BLKREC}\end{matrix}\right\}$ **(Optional)**

This optional parameter allows the user to specify the type of record address format to be used in tag or ADDROUT files. (The formats are described under *File Structure* in Chapter 2.) The ACTION statement must equal TAGSORT or ADDROUT if the RECADD parameter is used.

If RECADD=LOGREC, Format A containing the logical record number is used. This is the default case for input files with either relative file organization, or sequential file organization and fixed length records.

NOTE

RECADD=LOGREC is invalid for input files with sequential file organization and variable length records.

If RECADD=BLKREC, Format B containing the logical block number and record number is used. This is the default case for input files with sequential file organization and variable length records.

EXAMPLES

1. In this example a full sort will be performed according to the sort key fields specified. No messages will be produced.

```
//PAR   ACTION=FULLSORT,MESSAGE=NONE,...
```

2. In this example a tag sort will be performed according to the sort key fields specified. Format B containing the logical block number and record number will be used.

```
//PAR   ACTION=TAGSORT,RECADD=BLKREC,...
```

3. In this example three presorted files will be merged according to the sort key fields.

```
//PAR  ACTION=MERGE,PRESORT=3,...
```

SORT FIELDS

The FIELD identifier parameter statement is used to define sort key fields, but may also be used to define fields that do not influence the record sequence. At least one sort key field must be described; a maximum of 15 fields may be described with a combined length of not more than 256 bytes. Individual key fields must not exceed 255 bytes. Keys are described in declining order of importance (primary key described first, secondary next, etc.). Sort field descriptions apply to all input files in a merge or combine operation. The format of the FIELD statement is as follows:

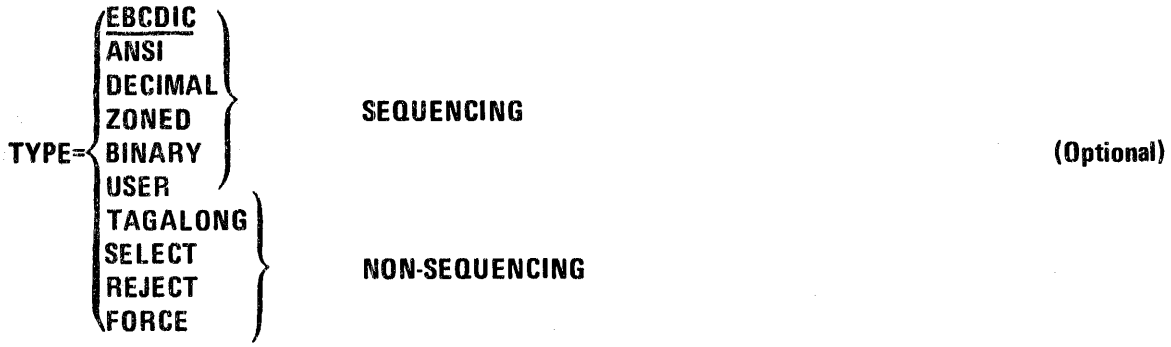
```
FIELD=p
LENGTH=n
[ TYPE={ EBCDIC | ANSI | DECIMAL | ZONED | BINARY | USER | TAGALONG | SELECT | REJECT | FORCE } ]
[ SEQUENCE= {ASCEND | DESCEND} ]
[ VALUE=v]
[ DISPOSITION= {KEEP | DROP} ]
```

FIELD=p

The FIELD parameter gives the leftmost byte position of the sort field relative to the beginning of the record.

LENGTH=n

The LENGTH parameter gives the number of bytes in the field.



The TYPE parameter specifies the type of field that is being used as a sort key. Both sequencing and non-sequencing options are available.

For the sequencing options, the TYPE= value is one of the following types of sequence characters.

Value	Description
EBCDIC	EBCDIC collating sequence; default case
ANSI	ANSI collating sequence
DECIMAL	Packed decimal
ZONED	Zoned decimal
BINARY	Binary value; field must begin on an even-byte boundary and have a length of 2 or 4 bytes.
USER	User collating sequence must be provided

If TYPE=TAGALONG and ACTION=TAGSORT, the field is not included in the sequencing, but is to be carried in the tag record. (See Example 2 in the following text.)

If TYPE=SELECT, the user defines a field whose contents determine whether the record is entered into the sort (Example 3). If the field matches the EBCDIC character or characters specified by the associated VALUE parameter, the record is entered into the sort. If not, the record is bypassed.

If TYPE=REJECT, the user specifies records to be bypassed if the defined field matches the character(s) specified by the VALUE parameter (Example 3). If the values do not match, the record is entered into the sort.

If TYPE=FORCE, a value must be specified in the VALUE parameter. When ACTION=TAGSORT, the character string specified in the VALUE parameter is inserted in the current position of the tag record that is being built (Example 4). The field position has no real meaning in this case. If the LENGTH parameter is not equal to the number of characters in the VALUE parameter, truncation or padding occurs on the right. When ACTION=FULLSORT, the character string given in the VALUE parameter is inserted in the field positions defined by the FIEL D and LENGTH parameters (Example 5).

SEQUENCE={**ASCEND** / **DESCEND**} **(Optional)**

This optional parameter specifies whether the sort key field is sorted in ascending or descending sequence. The default case is ascending sequence.

VALUE=v **(Optional)**

This optional parameter gives the EBCDIC character string to be used as a record selection criterion or to be inserted in the record (Example 3). A value is required when TYPE=SELECT, REJECT, or FORCE.

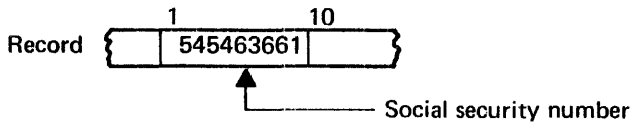
DISPOSITION={**KEEP** / **DROP**} **(Optional)**

The DISPOSITION parameter, valid only for ACTION=TAGSORT, determines whether a field is to be retained or not (Example 6). If DISPOSITION=KEEP, SORT keeps this field in the output file record; this is the default case. If DISPOSITION=DROP, the field is dropped from the output file record.

EXAMPLES

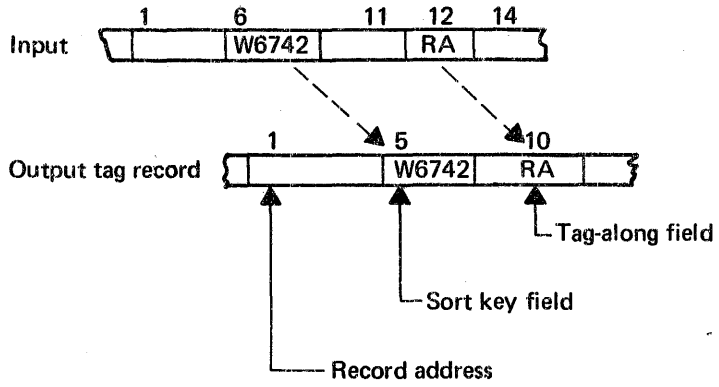
1. The sort key field for this example is the 9-digit social security number starting in position 1.

```
//PAR  FIELD=1,LENGTH=9
```

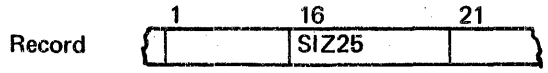


2. When the TAGALONG option is selected, the designated field is taken from the input record and transferred to the output tag record in the order specified by the //PAR card.

```
//PAR  ACTION=TAGSORT,FIELD=6,LENGTH=5,FIELD=12,LENGTH=2,TYPE=TAGALONG
```



3. When the SELECT option is used, only the records with matching values in the VALUE parameter and the selected field are accepted. When the REJECT option is selected, the record is bypassed if these fields match.



Accept Record

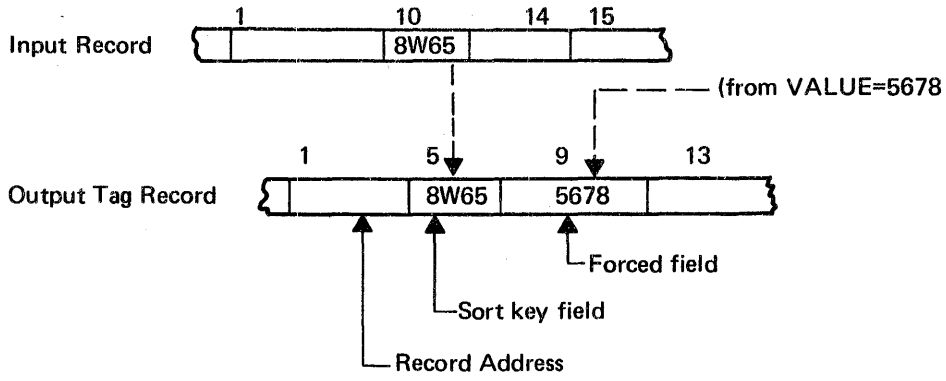
```
//PAR  FIELD=16,LENGTH=5,TYPE=SELECT,VALUE=SIZ25
```

Bypass Record

```
//PAR  FIELD=16,LENGTH=5,TYPE=REJECT,VALUE=SIZ25
```

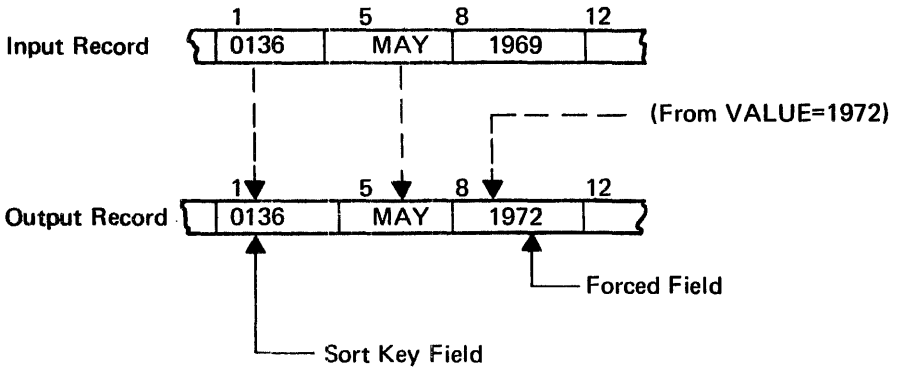
4. When the FORCE option is selected and ACTION=TAGSORT, the characters given in the VALUE parameter are inserted into the output record at the current position of the tag record. The FIELD parameter has no effect; and the LENGTH parameter specifies how many positions in the forced field.

```
//PAR  ACTION=TAGSORT,FIELD=10,LENGTH=4
//PAR  FIELD=15,LENGTH=4,TYPE=FORCE,VALUE=5678
```



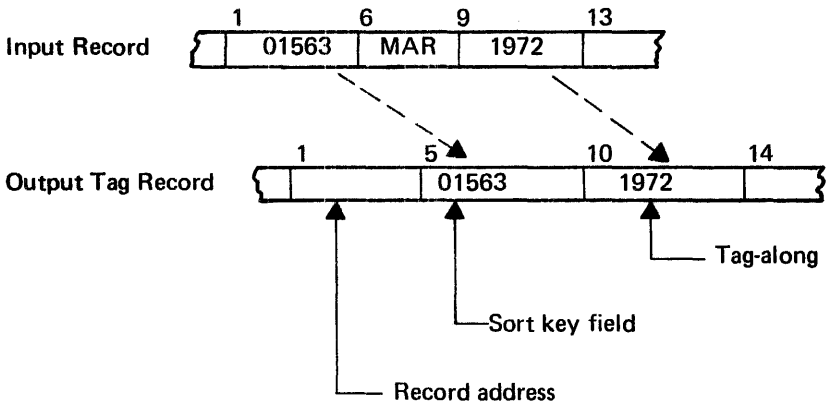
5. When the FORCE option is selected and ACTION=FULLSORT, the characters given by the VALUE parameter are inserted into the output record in the positions specified by FIELD and LENGTH.

```
//PAR  ACTION=FULLSORT,FIELD=1,LENGTH=4
//PAR  FIELD=8,LENGTH=4,TYPE=FORCE,VALUE=1972
```



6. When the DISPOSITION option is selected, a field that is being used for the sort itself may be dropped from the final output tag record.

```
//PAR  ACTION=TAGSORT,FIELD=1,LENGTH=5,FIEL D=6,LENGTH=3,DISPOSITION=DROP
//PAR  FIELD=9,LENGTH=4,TYPE=TAGALONG
```



USER COLLATING SEQUENCE

If TYPE=USER was specified in the FIELD statement, the user must supply a collating sequence. The format of the UCOLL statement is as follows:

[UCOLL=(C1,C2,. . . .,CM)]

UCOLL=(C1,C2,. . . .,CM) **(Optional)**

The UCOLL statement gives the hexadecimal values (ascending sequence) of characters in a user collating sequence. Any code not specified is assigned the highest-order value in the sequence. A pair of values separated by a dash indicates a range of values.

In the following example, the 10-character sort key field beginning in position 20 will be sorted according to a user-specified sequence designated by the UCOLL statement.

```
//PAR    ACTION=MERGE,PRESORT=2,FIELD=10,LENGTH=10
//PAR    FIELD=20,LENGTH=10,TYPE=USER
//PAR    UCOLL=(00-C0,F0-F9,C1-EF,FA-FF)
```

FILE ATTRIBUTES

A file attribute statement must be supplied for each input file except a disc file with fixed length records that is either described in the Central Catalog, or that uses the common stored data format. The file attribute statement is optional for output files.

The file attribute statement must begin with the identifier parameter, ID=SRTxxx. The format of the statement is as follows:

ID= { SRTINP / SRTPI1 / SRTPI2 / . / . / . / SRTPI7 / SRTOUT }

LABEL= { STANDARD / NONSTANDARD / UNLABELED }

REWIND= { YES / NO }

NUMBER=n

TYPE= { FIXED / VARIABLE }

SIZE=n

BLKFAC=n

BLKSIZ=n

CSD= { YES / NO }

Table 3-1 gives the file attribute requirements for files according to device type.

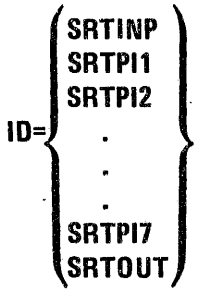
Table 3-1. File Attributes Requirements

Parameter \ File	Disc	Magnetic Tape	Unit Record
ID	X	X	X
LABEL		X	
REWIND		X	
NUMBER	X[1]	X	X
TYPE	X	X	
SIZE		X	
BLKFAC		X[2]	
BLKSIZ		X[3]	
CSD		X	

[1]NUMBER is not required for a file residing on a single disc pack. However, if NUMBER is provided, it supersedes the highest block written in estimating the number of records to provide in the intermediate file allocation.

[2]BLKFAC is not required when TYPE=VARIABLE.

[3]BLKSIZ is required only when TYPE=VARIABLE.



The ID parameter identifies the input file(s) or output file. The file identifiers are defined as follows:

Identifier	Description
SRTINP	Unsorted input file
SRTPI1	First presorted input file
SRTPI2	Second presorted input file
SRTPI3	Third presorted input file
SRTPI4	Fourth presorted input file
SRTPI5	Fifth presorted input file

Identifier	Description
SRTPI6	Sixth presorted input file
SRTPI7	Seventh presorted input file
SRTOUT	Output file

LABEL= {**STANDARD** / **NONSTANDARD** / **UNLABELED**}

The LABEL parameter gives the type of labels for a magnetic tape file. If LABEL=STANDARD, standard tape labels are used; this is the default case. If LABEL=NONSTANDARD, nonstandard tape labels are used. If LABEL=UNLABELED, no tape labels are used.

REWIND= {**YES** / **NO**}

The REWIND parameter specifies whether magnetic tape files are to be rewound or not. REWIND=YES (the default case) specifies the file will be rewound. However, REWIND=NO specifies that no initial rewind be performed for the input files nor any rewind after completion of writing for an output file.

NUMBER=n

The NUMBER parameter gives an estimate of the number of records in a file.

TYPE= {**FIXED** / **VARIABLE**}

TheTYPE parameter specifies the type of record to be used. If TYPE=FIXED, fixed length records are used; this is the default case. If TYPE=VARIABLE, variable length records are used.

SIZE=n

The SIZE parameter gives the record size (in bytes).

BLKFAC=n

The BLKFAC parameter specifies the number of records per block (blocking factor). The default value is 1.

BLKSIZ=n

The BLKSIZ parameter specifies the block size (in bytes) of the file (includes the four bytes for control headers).

CSD= { **YES** / **NO** }

The CSD parameter specifies the use of the common stored data format*. If CSD=YES (the default case), data has the common stored data format. If CSD=NO, it designates IBM format F or U tape file.

EXAMPLE

```
//PAR    ID=SRTINP,NUMBER=500,SIZE=25,BLKFAC=25
```

The SRTINP file contains 500 records of 25 bytes each, blocked 25 records per block.

*Common stored data format is described in the **Control Program and Data Management Services, Basic Reference** manual.

4. SORT PROGRAM — INTERNAL LOGIC

The following material is presented to aid the programmer writing modification routines for SORT. Modification routines are discussed in Chapter 5. The SORT program consists of a Control Section and four phases. The Control Section is main-memory resident for the duration of the sort run; the individual phases — Definition, Internal Sort, Intermediate Merge, and Final Merge — are called into execution as they are needed. Figure 4-1 illustrates the relationships of the different phases and the Control Section.

CONTROL SECTION

The Control Section for SORT consists of calls to the four phases, subroutines common to all phases, and control information such as pointers to file descriptions, to I/O buffers, and to the sort area.

DEFINITION PHASE

The Definition Phase performs the following preliminary functions.

1. Interprets sort parameters designated in the SORT control statements or in a SORT subroutine call.
2. Allocates available storage to I/O buffers, a sort area, and control information.
3. Chooses the intermediate file block size and order of merge depending upon input file characteristics and the amount of available storage.
4. Generates field-dependent code.
5. Assigns intermediate files.
6. Opens input and intermediate files.

INTERNAL SORT PHASE

The Internal Sort phase performs the initial sort of the input file and distributes ordered strings of records onto the intermediate files.

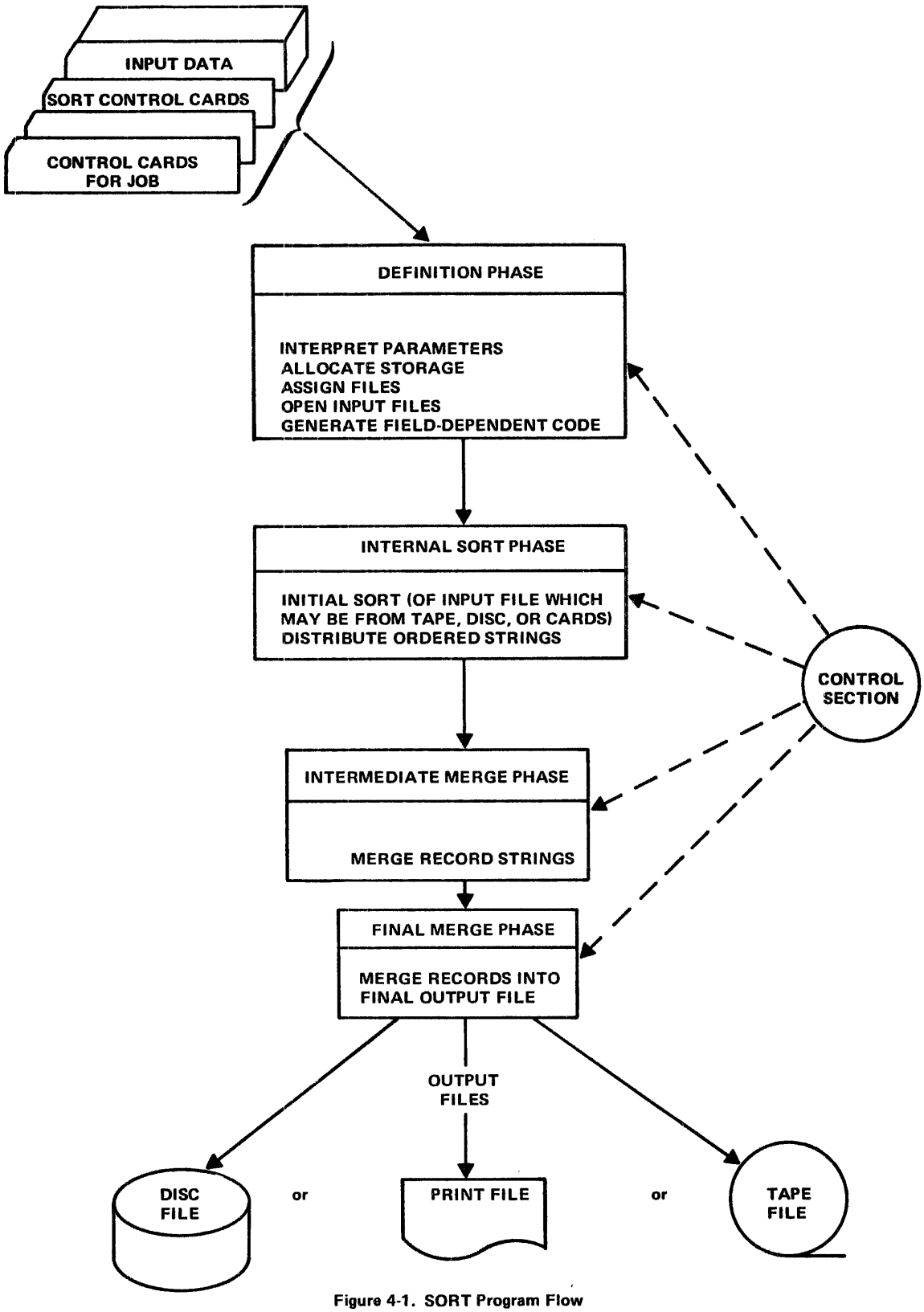


Figure 4-1. SORT Program Flow

A replacement tournament sort method is used in the SORT program. With this method, input records are paired against each other with the winner advancing to the next round of the tournament. (A winner is a record which takes precedence over its opponent in the sorting sequence.) The tournament winner is written on an intermediate file as the next record in an ordered string of records. The next input record replaces the winner in the tournament, and the tournament is repeated. However, it is not necessary to repeat all comparisons of the records in the tournament. Only those comparisons involving the winning record need be repeated, thus holding the number of comparisons to a minimum. Once the tournament has been initialized, only one comparison in each round of the tournament need be performed for each input record. An example of the tournament method can be found in Appendix B.

At the option of the user, either full data records or tag records may be carried throughout the Internal Sort and Intermediate Merge phases. With the use of the ACTION=RETRIEVE parameter, full records can be retrieved from the input file for a full record output after a tag record sort has been performed. Generally, this option is efficient for large sized records.

INTERMEDIATE MERGE PHASE

The Intermediate Merge phase merges strings of records from one section of the intermediate files into longer strings on an alternate section of the intermediate files. A replacement tournament similar to that used in the Internal Sort phase is used. The merge process continues with the intermediate file sections alternating as input and output files until conditions are proper for the Final Merge phase.

FINAL MERGE PHASE

The Final Merge phase produces the final output file. The procedure for this phase varies according to the type of sort specified by the ACTION parameter.

ACTION=TAGSORT, ADDROUT OR RETRIEVE

If tag records were used in the Intermediate Merge phase, the full records may be retrieved from the input file in the sorted sequence and written on the final output file (ACTION=RETRIEVE). The tag records themselves may make up the final output file when ACTION=TAGSORT. A special case of the tag file output occurs when ACTION=ADDROUT; all the sort key fields are dropped leaving only the ordered input file record addresses on the output file.

ACTION=FULLSORT

If full records were retained through the Intermediate Merge phase (ACTION=FULLSORT), strings of records from the intermediate file are merged into one ordered string on the final output file.

5. USER MODIFICATION ROUTINES

INTRODUCTION

One, two, or three modification subroutines may be provided by the user. These routines allow records to be inserted, deleted, or modified at various times during the sort process. Interfaces for user modification subroutines are provided in the internal sort and final merge phases. The first user modification location ($STM1) is in the internal sort phase and allows unsorted input records to be processed before they enter the sort. The second ($STM2) and third ($STM3) user modification locations are in the final merge phase. $STM2 allows records from a presorted merge file to be processed prior to merging, and $STM3 allows sorted records to be processed prior to their being written into the output file.

A cataloged procedure will be provided for the user to link his modification routines with SORT modules. The modified SORT program is then available for the user to execute at will.

The linking procedure call for the modification routines is:

```
//CALL   PRO=SORTMOD,NAME=n,MODLIB=m,LODLIB=i,VOL=v,MSC=c
```

The parameters of this call are defined as:

NAME=n — Name to be given to the modified SORT program; that is, the name the program will be called in an //EXECUTE statement.

MODLIB=m — IDENT of the file containing the user modification routines in object program form. Program name and primary entry point of user mods 1, 2, and 3 must be $STM1, $STM2, and $STM3. (Program name and entry point have the same name.)

LODLIB=i — IDENT of a file which is to contain the linked SORT program.

VOL=v — Volume ID for MODLIB and LODLIB.

MSC=c — Modification security code, if any, for MODLIB

The linked program may then be used exactly as SORT is used except that the name used in the EXECUTE statement must be the name of the modified SORT program.

Record modification or insertion is invalid for ACTION=ADDROUT or RETRIEVE.

GENERAL CALLING SEQUENCE FOR SUBROUTINES

The cataloged procedure used in linking the modification routines has the following general calling sequence for each user subroutine.

```
LODD L-address, AREG

LODD S-address, SREG

BSR  UM-entry,@SREG
```

- The L-address is the address of the argument list, which has the following general format illustrated in Figure 5-1.

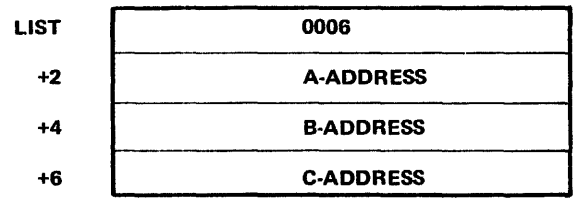


Figure 5-1. Argument List

- A-address, B-address, and C-address are the addresses of parameter values A, B, and C respectively. A is the code describing the entry conditions; B is the data item or field in question; and C is the length (in bytes) of that item.

- AREG is currently defined as General Purpose Register 6 (R6).

- S-address is the address of a save area which is used for program linkage. The format of the save area is illustrated in Figure 5-2.

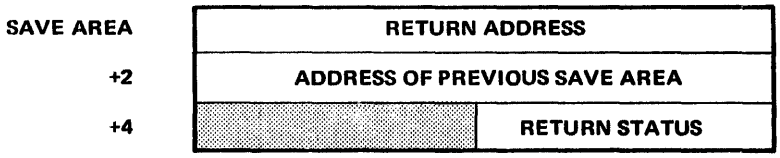


Figure 5-2. Save Area

- SREG is currently defined as General Purpose Register 7 (R7).

- UM-entry is the user modification routine entry point.

$STM1 CALL

The $STM1 call allows the user to exit from the SORT program to his own routine during the input phase of the Internal Sort (just prior to entering the record into the sort tournament). At this point the user may inspect the record and decide whether to accept it, modify it, delete it, or insert another record. Figure 5-3 lists the different entry conditions for the $STM1 call.

A-ENTRY CODE*	B-DATA ITEM	C-ITEM LENGTH	ENTRY CONDITION
0010	–	–	INITIALIZATION
0012	INPUT RECORD	RECORD LENGTH	INPUT RECORD IN STORAGE
0013	–	–	END OF INPUT

*Codes are in hexadecimal format.

Figure 5-3. $STM1 Call

$STM2 CALL

The $STM2 call allows the user to inspect records from the presorted strings of the Intermediate Merge phase. Records may be accepted, modified, deleted, or inserted at this exit point. Figure 5-4 gives the entry conditions for the $STM2 call.

A-ENTRY CODE	B-DATA ITEM	C-ITEM LENGTH	ENTRY CONDITION
0020	NUMBER OF PRESORTED INPUT FILES	–	INITIALIZATION
xx22*	PRESORTED INPUT RECORD	LENGTH OF RECORD	PRESORTED INPUT RECORD IN STORAGE
xx23*	–	–	END OF INPUT ON FILE xx*

*xx is the file ordinal. The presorted input file associated with file identifier SRTPI1 is file ordinal number 01; SRTPI2 is 02; etc.

Figure 5-4. $STM2 Call

$STM3 CALL

The $STM3 call allows the user to inspect records in the sorted strings just prior to output. Records may be accepted, modified, deleted, or inserted at this exit point. Figure 5-5 gives the entry conditions for the $STM3 call.

A-ENTRY CODE	B-DATA ITEM	C-ITEM LENGTH	ENTRY CONDITION
0040	–	–	INITIALIZATION
0042	OUTPUT FILE	LENGTH OF RECORD	RECORD HAS BEEN SELECTED FOR OUTPUT
0043	–	–	OUTPUT RECORDS HAVE ALL BEEN PROCESSED

Figure 5-5. $STM3 Call

RETURN CONDITIONS

The user modification routine returns the status in byte 5 of the save area (Figure 5-2). In addition, the B-address and the C-address (Figure 5-1) may be altered to describe a record to be inserted in the input stream. Or C itself may be altered to reflect a modification of record B; if so, C must not exceed the maximum record length allowed for the file. Status values are described in Figure 5-6.

RETURN STATUS CODE*	RETURN CONDITION
00	NORMAL RETURN; ACCEPT INPUT RECORD AS IS, OR OUTPUT RECORD HAS BEEN PROCESSED WITHOUT CHANGE IN THE RECORD LENGTH
01	RECORD HAS BEEN MODIFIED WITH A RESULTING CHANGE IN RECORD LENGTH. EITHER THE VALUE OF C HAS BEEN MODIFIED OR THE C-ADDRESS WAS CHANGED TO POINT TO THE NEW LENGTH OF THE RECORD. THE NEW LENGTH CANNOT EXCEED THE MAXIMUM RECORD LENGTH FOR THE FILE.
02	INSERT A RECORD IN THE INPUT STREAM; B-ADDRESS AND C-ADDRESS ARE MODIFIED TO DESCRIBE THE RECORD TO BE INSERTED.
04	DELETE THE CURRENT RECORD FROM THE INPUT STREAM OR FROM THE OUTPUT FILE.
10	TERMINATE THE SORT RUN.

*Codes are in hexadecimal format.

Figure 5-6. Return Conditions

6. PERFORMANCE CHARACTERISTICS

SORT performance on a given file improves as additional main storage is made available. Additional mass storage may also improve performance. A partition size larger than the minimum 8000 bytes allows SORT to handle larger record and block sizes. Table 6-1 gives estimates of the minimum partition size required to perform a full record sort on various record sizes and blocking factors.

Table 6-1. Partition Size

Record Size	Blocking Factor	Minimum Partition Size
100	6	8K bytes
	7	9K bytes
200	2	8K bytes
	3	9K bytes
400	2	10K bytes
	3	12K bytes
500	1	10K bytes
	2	12K bytes
1000	1	14K bytes
	2	16K bytes

The estimates do not include memory requirements for the following:

- a large number of sort key fields (8 or more)
- user collating sequence
- user modification routines
- indexed file input or output

A. COLLATING SEQUENCES

STANDARD COLLATING SEQUENCE (EBCDIC)

Code	Character	Meaning	Code	Character	Meaning
00	NUL	Null	1C	CU1	Customer Use 1
01	SOH	Start of Heading	1D	IFS	Interchange File
02	STX	Start of Text	1D	IFS	Interchange File Separator
03	ETX	End of Text	1E	IRS	Interchange Record Separator
04	PF	Punch Off	1F	IUS	Interchange Unit Separator
05	HT	Horizontal Tab	20	DS	Digit Select
06	LC	Lower Case	21	SOS	Start of Significance
07	DEL	Delete	22	FS	Field Separator
0A	SMM	Start of Manual Message	24	BYP	Bypass
0B	VT	Vertical Tab	25	LF	Line Feed
0C	FF	Form Feed	26	ETB	End of Transmission Block
0D	CR	Carriage Return	27	ESC	Escape
0E	SO	Shift Out	2A	SM	Set Mode
0F	SI	Shift In	2B	CU2	Customer Use 2
10	DLE	Data Link Escape	2D	ENQ	Enquiry
12	DC1	Device Control 1	2E	ACK	Acknowledge
13	DC2	Device Control 2	2F	BEL	Bell
14	TM	Tape Mark	32	SYN	Synchronous Idle
15	RES	Restore	34	PN	Punch On
16	NL	New Line	35	RS	Reader Stop
17	BS	Backspace	36	UC	Upper Case
18	IL	Idle	37	EOT	End of Transmission
19	CAN	Cancel	3B	CU3	Customer Use 3
1A	EM	End of Medium	3C	DC4	Device Control 4
1B	CC	Cursor Control	3D	NAK	Negative Acknowledge
			3F	SUB	Substitute

Code	Character
40	Space
4A	¢
4B	. (period)
4C	<
4D	(
4E	+
4F	|
50	&
5A	!
5B	$
5C	*
5D	)
5E	;
5F	¬
60	–
61	/
6B	, (comma)
6C	%
6D	_ (underscore)
6E	>
6F	?
7A	:
7B	#
7C	@
7D	'
72	=
7F	"
81	a
82	b
83	c

Code	Character
84	d
85	e
86	f
87	g
88	h
89	i
91	j
92	k
93	l
94	m
95	n
96	o
97	p
98	q
99	r
A2	s
A3	t
A4	u
A5	v
A6	w
A7	x
A8	y
A9	z
C1	A
C2	B
C3	C
C4	D
C5	E
C6	F

Code	Character
C7	G
C8	H
C9	I
D0	}
D1	J
D2	K
D3	L
D4	M
D5	N
D6	O
D7	P
D8	Q
D9	R
E2	S
E3	T
E4	U
E5	V
E6	W
E7	X
E8	Y
E9	Z
F0	0
F1	1
F2	2
F3	3
F4	4
F5	5
F6	6
F7	7
F8	8
F9	9

ANSI COLLATING SEQUENCE

Code	Character	Meaning	Code	Character	Meaning
00	NUL	Null	11	DC1	Device Control 1
01	SOH	Start of Heading	12	DC2	Device Control 2
02	STX	Start of Text	13	DC3	Device Control 3
03	ETX	End of Text	14	DC4	Device Control 4
04	EOT	End of Transmission	15	NAK	Negative Acknowledge
05	ENQ	Enquiry	16	SYN	Synchronous Idle
06	ACK	Acknowledge	17	ETB	End of Transmission Block
07	BEL	Bell	18	CAN	Cancel
08	BS	Backspace	19	EM	End of Medium
09	HT	Horizontal Tabulation	1A	SUB	Substitute
0A	LF	Line Feed	1B	ESC	Escape
0B	VT	Vertical Tabulation	1C	FS	File Separator
0C	FF	Form Feed	1D	GS	Group Separator
0D	CR	Carriage Return	1E	RS	Record Separator
0E	SO	Shift Out	1F	US	Unit Separator
0F	SI	Shift In			
10	DLE	Data Link Escape			

Code	Character
20	SP (Space
21	!
22	" (Quotation Marks)
23	#
24	$
25	%
26	&
27	' (Apostrophe)
28	(
29	)
2A	*
2B	+
2C	, (Comma)
2D	- (Hyphen)
2E	. (Period)
2F	/
30	0
31	1
32	2
33	3
34	4
35	5
36	6
37	7
38	8
39	9
3A	:
3B	;
3C	<
3D	=
3E	>

Code	Character
3F	?
40	@
41	A
42	B
43	C
44	D
45	E
46	F
47	G
48	H
49	I
4A	J
4B	K
4C	L
4D	M
4E	N
4F	O
50	P
51	Q
52	R
53	S
54	T
55	U
56	V
57	W
58	X
59	Y
5A	Z
5B	[
5C	\
5D	]
5E	^ (Circumflex)

Code	Character
5F	_ (Underline)
60	` (Grave Accent)
61	a
62	b
63	c
64	d
65	e
66	f
67	g
68	h
69	i
6A	j
6B	k
6C	l
6D	m
6E	n
6F	o
70	p
71	q
72	r
73	s
74	t
75	u
76	v
77	w
78	x
79	y
7A	z
7B	{ {
7C	| (Vertical line)
7D	}
7E	~ (Overline)
7F	DEL (Delete)

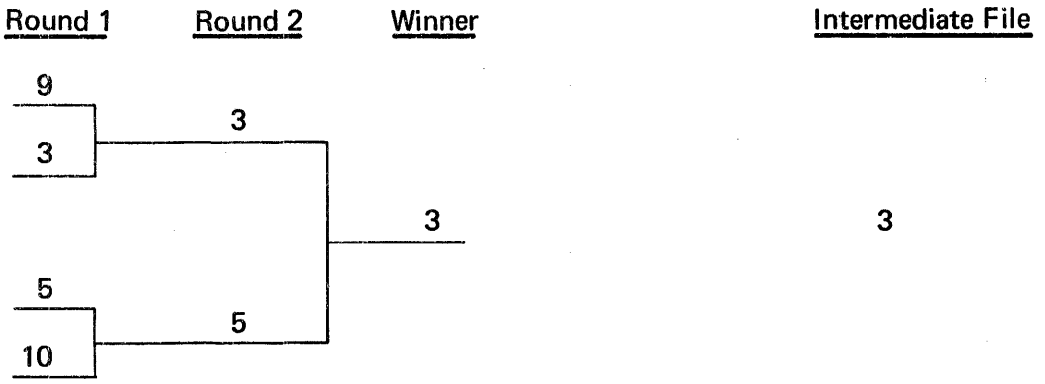
B. EXAMPLE OF TOURNAMENT METHOD

Assume an input file of 10 records, and a sort area which can hold 4 records at a time. In this example each record is represented by a number indicating its rank in the sorting sequence. The records are:

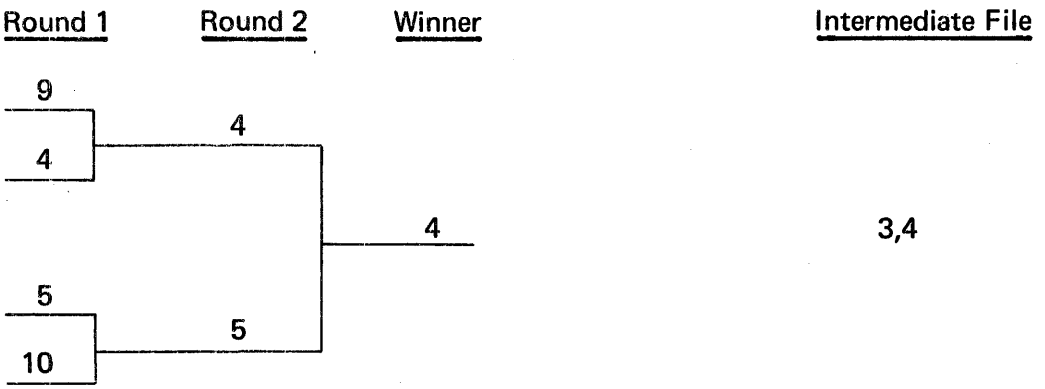
9, 3, 5, 10, 4, 1, 2, 6, 8, 7

To initialize the tournament, the first four records are read into the sort area, all comparisons are made, and the winner (record 3) is written on an intermediate file.

INTERNAL SORT PHASE

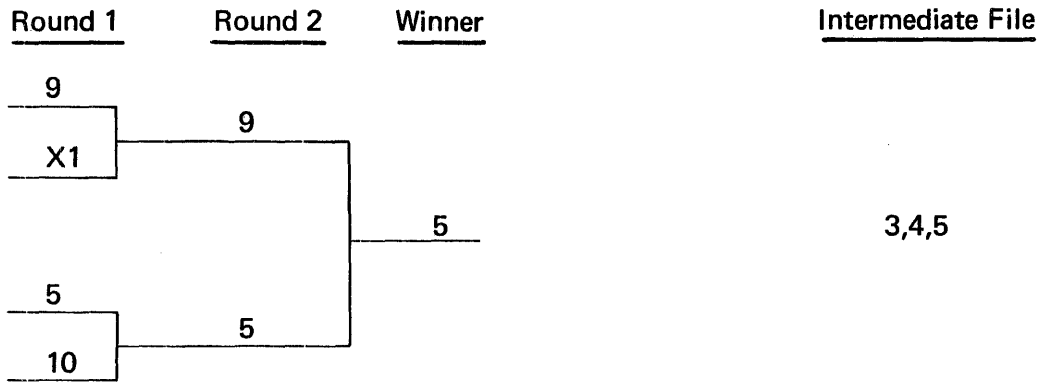


The next input record, 4, replaces record 3 in the tournament. Each input record is compared with the previous winner to see if it precedes or follows the winner in sequence. Since record 4 follows record 3 in the sort sequence it is eligible to complete in the current tournament.

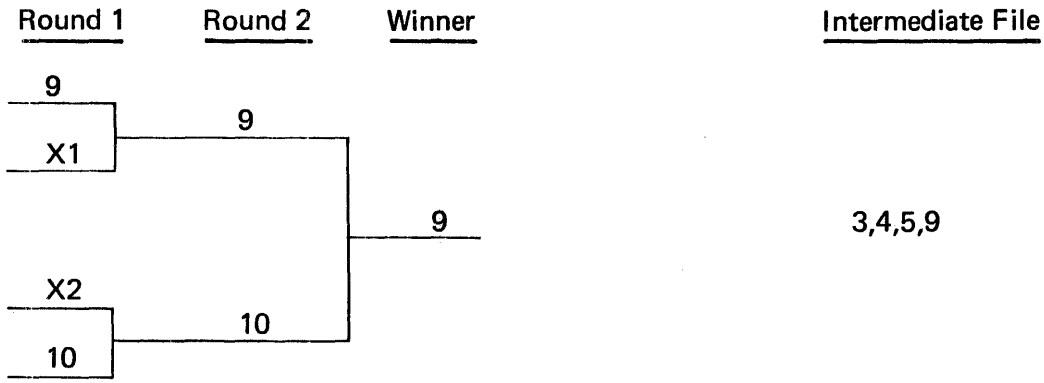


Only comparisons involving the winning record, record 3, were repeated. Record 4 is the winner and follows record 3 on the intermediate file.

Record 1 replaces the record 4 in the tournament. Since record 1 precedes record 4 in the sort sequence, it is marked ineligible for the current tournament, and its opponent (record 9) wins the round by default. In the example record 1 is marked with an X to denote ineligibility, and record 1 remains ineligible until all entries in the tournament are ineligible, at which time a new tournament begins producing a new string.

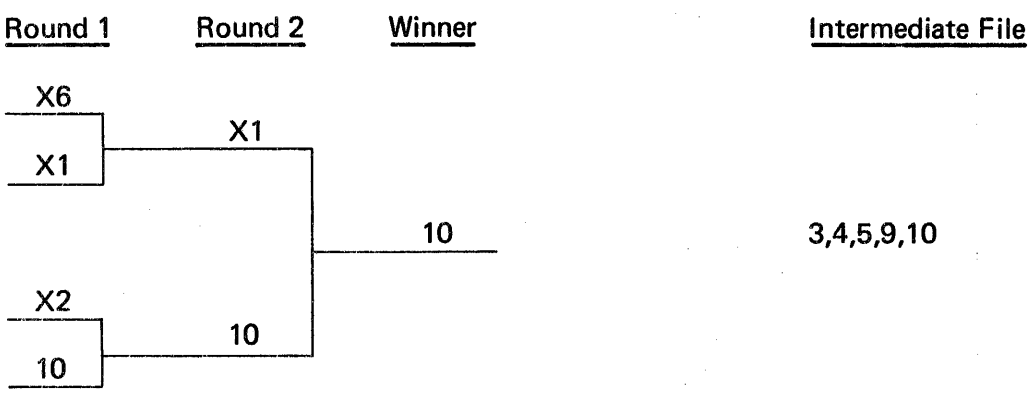


The next input record, 2, is marked ineligible, and record 10 wins by default.

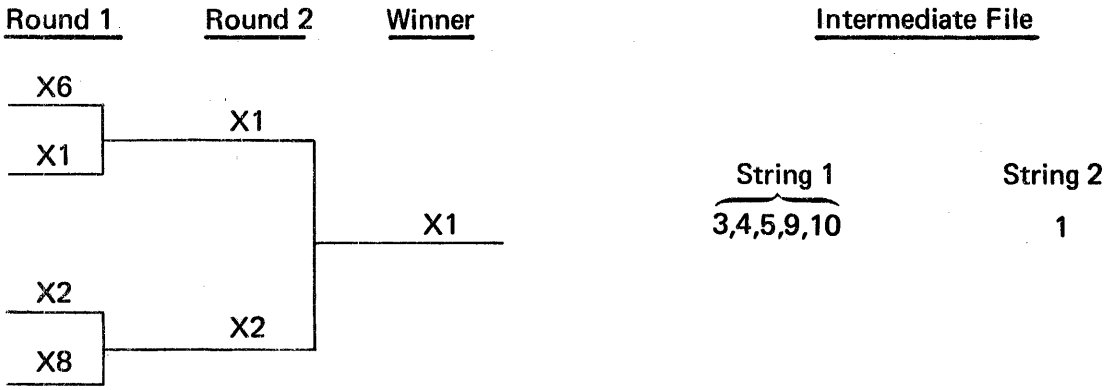


The tournament continues with another ineligible record, 6, entering the tournament. Since both record 6 and record 1 are ineligible, record 10 wins the tournament by default.

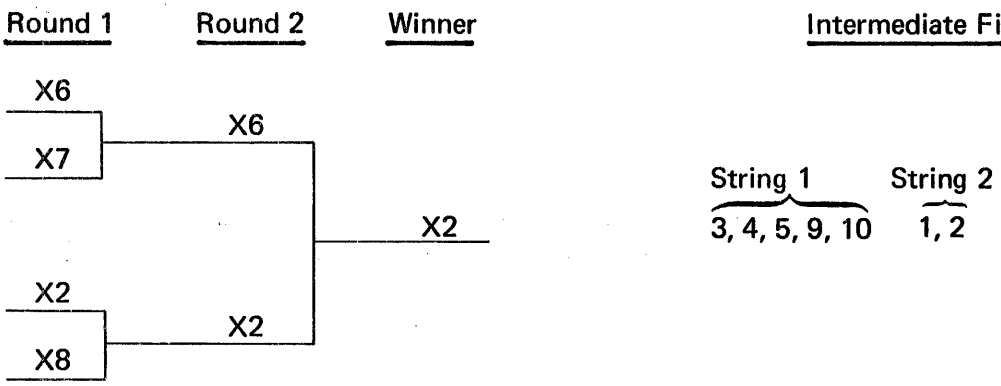
In actual practice, the comparison between record 6 and record 1 is performed at this time, and the winner advances to round 2 where it is marked ineligible. This technique eliminates the need to reinitialize the entire tournament when all records are ineligible.



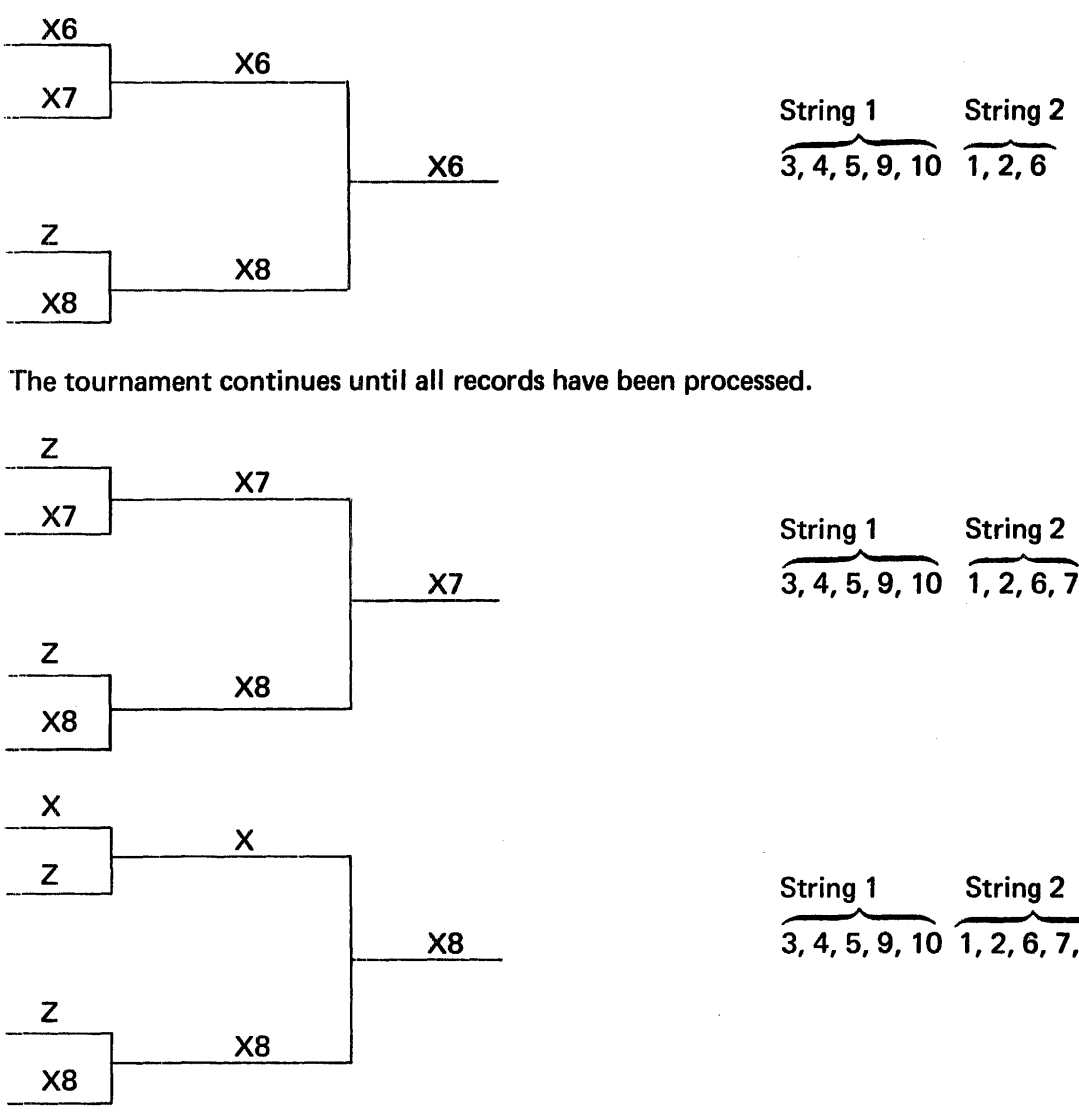
Record 8 replaces record 10 in the tournament. Now all records are ineligible. The first string of records terminates and another string begins.



The last input record, 7, replaces record 1.



Since there are no more input records, the entry for record 2 is marked with a Z to mark it empty. Record 8 wins the record by default.



The tournament continues until all records have been processed.

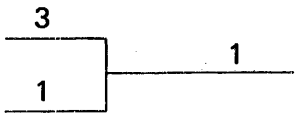
Note that each of the two strings on the intermediate file has more records than the sort area holds at one time. Since there are only two strings, SORT may now enter the final merge phase.

FINAL MERGE PHASE

A two-way merge is used. The tournament is merely a comparison of records from each of two strings.

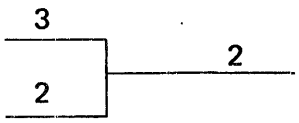
String 1	String 2
3, 4, 5, 9, 10	1, 2, 6, 7, 8

The tournament is initialized with a record from each string.



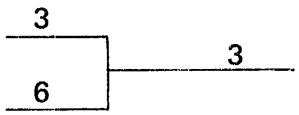
1

The winning record, 1, is written on the final output file. Record 2 replaces record 1 in the tournament.



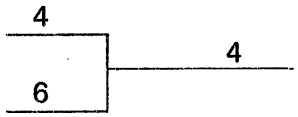
1, 2

Record 2 is replaced by record 6 and the tournament continues:

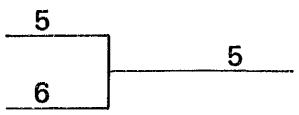


1, 2, 3

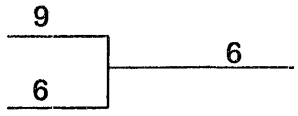
Record 4 replaces record 3, and so forth.



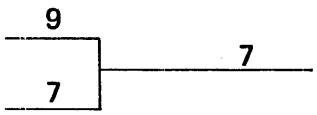
1, 2, 3, 4



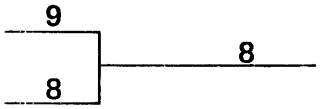
1, 2, 3, 4, 5



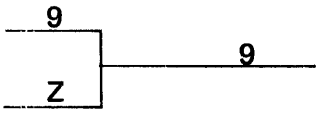
1, 2, 3, 4, 5, 6



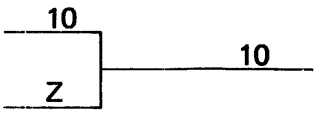
1, 2, 3, 4, 5, 6, 7



1, 2, 3, 4, 5, 6, 7, 8



1, 2, 3, 4, 5, 6, 7, 8, 9



1, 2, 3, 4, 5, 6, 7, 8, 9, 10

C. SAMPLE OUTPUT

DETAILED LISTING

A sample detailed output listing follows:

```
BEGIN SORT RUN 10/28/72                    10:55:20

                        RECORD              BLOCK          NUMBER
FILE ID                 SIZE                SIZE           OF RECORDS

SRTINP                  100                 208            9500
SRTOUT                  100                 208            9500
SRTWKA                  100                 208            5700
SRTWKB                  100                 208            5700

BEGIN INTERNAL SORT                        10:55:35

          RECORDS IN      =     9500
          RECORDS OUT     =     9500
          STRINGS OUT     =      182

BEGIN INTERMEDIATE MERGE                   10:58:02

          PASS 1

               RECORDS IN      =     9500
               RECORDS OUT     =     9500
               STRINGS OUT     =       37

          PASS 2

               RECORDS IN      =     9500
               RECORDS OUT     =     9500
               STRINGS OUT     =        8

          PASS 3

               RECORDS IN      =     9500
               RECORDS OUT     =     9500
               STRINGS OUT     =        2
```

```
BEGIN FINAL MERGE                    11:03:00

          RECORDS IN    =    9500
          RECORDS OUT   =    9500

END SORT RUN    10/28/72             11:05:05
```

BRIEF LISTING

```
BEGIN SORT RUN    10/29/72           13:00:25

RECORDS IN        =    15231
RECORDS OUT       =    11029

END SORT RUN      10/29/72           13:13:01
```

D. SAMPLE SORT JOB

This example sorts a file named ROSEBUD having 9500 records of 100 bytes each blocked 2 records to a block and produces an output file named BLOOM. The sort key fields are:

1. Bytes 4-8, EBCDIC, ascending sequence

2. Bytes 9-10, binary, ascending sequence

3. Bytes 1-3, EBCDIC, descending sequence.

Control statements are:

```
//JOB-----
//EXEC   PGM=SORT,TIME=30
//PAR    ACTION=FULLSORT
//PAR    FIELD=4,LENGTH=5,FIELD=9,LENGTH=2,TYPE=BINARY
//PAR    FIELD=1,LENGTH=3,SEQUENCE=DESCEND
//DEF    ID=SRTINP,FIL=ROSEBUD,STA=(P,I),
//        DEV=DISC,VOL=MSC194,SIZ=100,BLK=2
//DEF    ID=SRTOUT,FIL=BLOOM,STA=(P,O),
//        DEV=DISC,VOL=MSC201,SIZ=100,BLK=2,NUM=9500
```

An abbreviated form of the same parameter set is:

```
//PAR    A=F,F=4,L=5,F=9,L=2,T=B,F=1,L=3,S=D
```

E. ERROR RECOVERY AND DIAGNOSTICS

This appendix gives the error recovery procedures and the diagnostic messages for the SORT program.

OPERATING ERRORS

Three types of errors may occur after the working environment is established by the definition phase and actual sorting has begun; these are:

1. File limit exceeded
2. Irrecoverable I/O error
3. Sequence check error

FILE LIMIT EXCEEDED

If the number of records to be sorted is significantly greater than estimated, the intermediate file allocation will be inadequate. Whenever the intermediate file allocation is exceeded, the SORT program acts according to the OVERFLOW option of the ACTION statement. If STOP is selected, the sort run terminates immediately. However, if GO is selected, the sort process continues on the partial input file. The user may sort the remainder of the input file later and merge the output files from the partial sorts into one sorted file.

IRRECOVERABLE I/O ERROR

The SORT program reacts to I/O errors according to the ERROR option given by the user in the ACTION statement. STOP causes the sort run to terminate immediately; DROP causes SORT to drop the record block in question and proceed to the next block. GO causes SORT to accept the record block as valid and resume normal processing.

SEQUENCE CHECK ERROR

When ACTION=SEQUENCE is specified, SORT sequence checks a file without any sorting being performed. The sort fields of each input record are compared with the corresponding fields of the preceding record. If a record is out of order, the record number is noted on the user's standard output listing, and the sequence check continues.

INPUT PARAMETER ERRORS

The definition phase of the SORT program analyzes input parameters for validity and consistency. Any discrepancies are noted by diagnostic messages. After all parameter input has been analyzed, the sort run is terminated if any error was detected.

DIAGNOSTIC MESSAGES

SORT writes a message on the system output file for each error it detects in the control statements and for each operating error. If a control statement parameter error is detected, the sort run terminates after all control statements have been analyzed for errors. If an operating error, the message type determines the viability of the sort run.

All messages have the following standard form.

STaabbbc dd---d
- aa SORT phase identifier
  - DF Definition
  - IS Internal sort
  - IM Intermediate merge
  - FM Final merge
  - SQ Sequence check
  - SC SORT conclusion
- bbb Message code
- c Message type
  - 0 Information
  - 2 Warning
  - 4 Error, processing continues
  - 8 Error, run terminates
- dd---d Message text

Message	Description
STDF1018 statement id STATEMENT INVALID	Statement identifier is not a valid identifier (ACTION, FIELD, UCOLL, ID)
STDF1048 statement id STATEMENT, parameter name PARAMETER, keyword KEYWORD INVALID	Keyword in the identified parameter and statement is invalid
STDF1058 statement id STATEMENT, parameter name PARAMETER, keyword VALUE INVALID	non-numeric keyword encountered when should have been numeric
STDF1078 TOO MANY FIELD STATEMENTS	More than 15 sort fields specified for input file
STDF1088 FIELD STATEMENT, TOTAL KEY LENGTH EXCEEDS 256	Only 256 bytes may be specified for the sort key fields
STDF1098 FIELD STATEMENT, number POSITION INVALID	Leftmost byte position of sort key field exceeds record size; all or part of key field would be positioned beyond the end of specified record size.
STDF1108 FIELD STATEMENT, number POSITION, LENGTH INVALID	No length parameter given for specified field, or length exceeds record size.
STDF1118 TOTAL VALUE LENGTH LIMIT EXCEEDED	More than 256 value characters present in sort field definitions.
STDF1128 NO SORT KEY FIELD DEFINED	At least one sort key field must be defined with TYPE=EBCDIC, ASCII, USER, DECIMAL, or BINARY.
STDF1138 TOO MANY INPUT FILES	Maximum of eight input files allowed.
STDF1148 USER COLLATING SEQUENCE REQUIRED	Sort field defined with TYPE=USER, but no UCOLL statement specified.
STDF1158 UCOLL STATEMENT INVALID	UCOLL statement not in the form UCOLL=(x1,x2,x3,. . .,xn)
STDF1168 UCOLL STATEMENT, number VALUE INVALID	Number value not in range of 00 to FF
STDF1178 UCOLL STATEMENT, TOO MANY ENTRIES	More than 256 values present in UCOLL statement
STDF1258 INPUT FILE RECORD LENGTH UNDEFINED	Record length is not defined for input file.

Message	Description
STDF1268 file id BLOCKSIZE < BLOCKING FACTOR* RECORD SIZE	File block size is not compatible with record size and blocking factor defined for the file.
STDF1218 ID STATEMENT, ID PARAMETER, file id KEYWORD INVALID	File identifier is not one of the following SRTINP, SRTPI1, SRTPI2, SRTPI3, SRTPI4, SRTPI5, SRTPI6, SRTPI7, or SRTOUT
STDF1228 SORT RUN TERMINATED DUE TO ERROR IN SORT FIELD DESCRIPTION TABLE	Indicates internal operating error.
STDF1238 SORT RUN TERMINATED DUE TO INADEQUATE RUN ENVIRONMENT	There is insufficient storage available is the partition to allocate tables, buffers, and record areas for the sort run.
STDF1248 file id FILE-LABRTN ERROR code	The LABRTN service request is used to get I/O file characteristic. An error code of 6 is returned for nondisc files; this is accepted by SORT. Any other error code implies that the file is not available or that an I/O error occurred while searching for the file definition data.
STDF1318 ALLOCATION ERROR code ON SRTOUT	An error code other than 6 or 8 was returned from an attempt to allocate SRTOUT.
STDF1998 SORT RUN TERMINATED DUE TO ERRORS IN PARAMETER INPUT	One or more parameter errors were detected.
STDF1988 SORT RUN TERMINATED DUE TO ERROR IN FILE CONTROL TABLES	Indicates internal operating error.
STIS201c STIM301c INPUT ERROR ON FILE file id, STATUS=code	Irrecoverable error detected while reading the file; SORT drops record if c=2, accepts it if c=4, or terminates run if c=8 according to the ERROR option specified in the ACTION statement.
STIM302c STFM402c OUTPUT ERROR ON FILE file id, STATUS=code	Irrecoverable error detected while writing the file; SORT drops record if c=2, ignores error if c=4, or terminates run if c=8 according to the ERROR option specified in the ACTION statement.
STIS203c INSUFFICIENT FILE STORAGE, number RECORDS IN SORT	Message gives the number of input records that can be sorted in the intermediately available file storage; SORT sorts these records if c=4 or terminates sort run if c=8 according to the OVERFLOW option of the ACTION statement.

Message	Description
STIS2988 SORT RUN TERMINATED DUE TO USER MOD 1 REPLY	The user modification routine in the internal sort phase returned a run termination code.
STIS2998 SORT RUN TERMINATED DUE TO INTERMEDIATE FILE OVERFLOW	Estimate of input records was inadequate.
STFM4968 SORT RUN TERMINATED – NO INPUT RECORDS	Either the input file was empty or a user modification routine deleted all input records.
STFM4978 SORT RUN TERMINATED DUE TO USER MOD 2 REPLY	The user modification routine for input in the merge phase returned a run termination code.
STFM4988 SORT RUN TERMINATED DUE TO USER MOD 3 REPLY	The user modification routine for output in the final merge phase returned a run termination code.
STFM4998 SORT RUN TERMINATED DUE TO OUTPUT FILE EXCEEDING ALLOCATED AREA	The number of records in the output file exceeded the estimate.
STSQ8012 OUT OF SEQUENCE RECORD, LOGICAL RECORD NUMBER=record number	A sequence error was detected in the sequence check procedure.
STSC9998 SORT RUN TERMINATED DUE TO IRRECOVERABLE I/O ERROR	An I/O error occurred in one of the sort or merge phases.

GLOSSARY

ADDROUT	Sort option giving the disc addresses of the records rather than the records themselves.
Forced Field	Record field that is added to the tag record.
Full Record Sort	A sorting technique in which the entire data record is transferred in all phases of sorting and merging operations.
Merge	Process by which several strings of logical records are collated to form one string.
Sort	Process by which logical records are sequenced according to a given value.
Sort Key	Field in a record which is used as a basis for determining the sequence of records in the output file.
String	A group of sequenced records stored on mass storage.
Tag Record	A subset of a logical record; combination of sort keys, data record addresses, optional tag-along fields, and optional forced fields.
Tag Sort	A sorting technique in which only the tag record is sorted after the initial reading of the input file.
Tag-Along Field	Record field that is not included in sequencing but is included in the tag record.

INDEX

COMMENTS FORM

MRX/OS Disc Sort Reference Manual (2200.009)

Please send us your comments, to help us produce better publications. Use the space below to qualify your responses to the following questions, if you wish, or to comment on other aspects of the publication. Please use specific page and paragraph/line references where appropriate. All comments become the property of the Memorex Corporation.

- Is the material:

	Yes	No
Easy to understand?	☐	☐
Conveniently organized?	☐	☐
Complete?	☐	☐
Well illustrated?	☐	☐
Accurate?	☐	☐
Suitable for its intended audience?	☐	☐
Adequately indexed?	☐	☐

- For what purpose did you use this publication? (reference, general interest, etc.)

  _____

- Please state your department's function: _____

  _____

- Please check specific criticism(s), give page number(s), and explain below:

  ☐ Clarification on page(s) _____

  ☐ Addition on page(s) _____

  ☐ Deletion on page(s) _____

  ☐ Error on page(s) _____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

MEMOREX

First Class

Permit No. 14831
Minneapolis,
Minnesota 55427

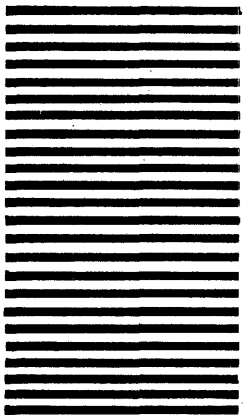
Business Reply Mail

No Postage Necessary if Mailed in the United States

Postage Will Be Paid By

Memorex Corporation

Midwest Operations – Publications
8941 Tenth Avenue North
Minneapolis, Minnesota 55427



. . . . . .

. . . . . .

Thank you for your information. . . . . .

Our goal is to provide better, more useful manuals, and your comments will help us to do so.

. . . . . .Memorex Publications