KAYPRO

SUPERSORT®

SuperSort® 1.6

Operator's Handbook And Programmer's Guide

9/30/81

TABLE OF CONTENTS

I. INTRODUCTION

SuperSort represents the implementation of ideas gathered from many years of data processing experience by the authors. The advent of low cost microprocessing equipment has placed computing power within the reach of many. We at MicroPro are endeavoring to contribute to this new and burgeoning field by providing a commonly required set of facilites in an easy to use yet comprehensive manner without compromising professional quality. As data processing professionals, we have had many experiences of inadequate hardware manufacturer's software documentation. We have attempted to rectify this with an approach to technical manual preparation that addresses readers with differing needs. We hope you like the result. Please feel free to contact us with your comments as we intend to provide the highest quality possible with regard to our products.

A. SUPERSORT 1.6 FEATURE SUMMARY

SuperSort combines high performance and operational flexiblity to perform sorting, merging, and record selection functions on data files compatible with BASIC, FORTRAN, COBOL, and Assembler applications programs, and with text editors, on 8080/8085/Z-80 microcomputers employing CP/M or a similar operating system.

SORT/MERGE OPERATION

* SORTS up to 32 input files into a single output file, automatically using external merge as necessary, depending on amount of data and amount of memory.

* MERGES up to 32 pre-sorted files all of which (memory permitting) are read in parallel for efficient true merge operation.

* Sort and merge input files can be specified IN THE SAME RUN. Thus, sorting new detail records and integrating them into an already-ordered master file is a simple and efficient SuperSort operation.

* Record selection, file conversions, and other features can be used independent of the sort/merge process (by specifying a single input merge).

FLEXIBLE FILE AND RECORD FORMATS

* Handles CP/M diskette files compatible with BASIC, FORTRAN (formatted or binary I/O), COBOL, assembler, and text editors.

* Handles files up to the CP/M file size limit (8Megabytes).

* Handles up to 65K logical records per file.

* Files may contain ASCII, BCD, and/or BINARY data.

* Handles LOGICAL RECORD LENGTHS to 4096 characters.

* Files may contain FIXED LENGTH records, varying length CARRIAGE-RETURN-DELIMITED records, or COBOL-style VARIABLE-LENGTH records with length at beginning; COBOL RELATIVE FILES are also supported.

* FIELDS used as sort keys or in record selection tests may have fixed COLUMN-SPECIFIED position and length, or variable COMMA-DELIMITED position and length.

MULTIPLE SORT KEYS WITH MANY OPTIONS AND DATA TYPES

* One to 32 key fields may be specified, each with an independent ASCENDING/DESCENDING indicator, collating sequence options, and data type attributes.

* Key data may be ASCII STRING text, ASCII NUMERIC, BCD (COBOL packed decimal), or BINARY.

* NUMERIC-ASCII option sorts on free-format numbers (including exponential notation) as typically output by the PRINT statement of BASIC or D, E, F, or G format in FORTRAN.

* BINARY data types handled include FIXED POINT of any length, signed or unsigned, stored low-high or high-low, and MicroSoft FLOATING POINT, single or double precision. This includes the INTEGER, SINGLE PRECISION (REAL), and DOUBLE PRECISION data types of MicroSoft MBASIC and FORTRAN.

* User-command-specified ALTERNATE COLLATING SEQUENCE and EBCDIC collating sequence provided on an individual field basis.

* Additional options include treating lower case as upper case, treating last rather than first character of key as most significant, and ignoring the high order (parity) bit.

POWERFUL RECORD SELECTION

* Desired records can be extracted from a file via SELECT and EXCLUDE commands which act on both sort and merge-only inputs.

* Selection is specified via ANY NUMBER OF CONDITIONAL TESTS of a field against a specified fixed value or range of fixed values or against another field in the same record.

* VALUES to test against are usually specified as TEXT STRINGS, but may also be specified as BCD numbers or as binary values expressed in OCTAL, DECIMAL, or HEXADECIMAL.

* Test OPERATORS include the usual LESS THAN, EQUAL TO, etc, and also BETWEEN and NOT BETWEEN. Tests may optionally be combined with AND, OR, and EXCLUSIVE OR operations.

* Record selection tests can use fixed postion or comma-delimited fields, and all the data types and attributes specified above for keys - ASCII, binary, numeric-ASCII, BCD, Floating point, alternate collating sequences, etc.

ADDITIONAL FEATURES

* EXTRACTION BY RECORD NUMBER: Starting and/or ending record numbers may be specified for each sort input file.

* OUTPUT DISKETTE CHANGE option makes it possible in most cases to sort an entire diskette of data in a two-drive system without writing over the input file.

* TAGSORT option reduces work file diskette space requirements by using pointers to records, rather than full records, during sorting and merging, then retrieving the input records via a high speed random access algorithm while writing the output file.

* CONSOLE PRINTOUTS: user can select five degrees of message print-out including none, a brief message giving number of records processed, or a detailed breakdown showing numbers of records input, sorted, merged, output, inserted, deleted, etc. Printout of disk space usage may also be requested.

* UTILITY FUNCTIONS: SuperSort can perform many useful functions such as converting files to a different record type, changing record lengths, converting records with comma-delimited fields to fixed-position fields, and extracting or rearranging (positional) fields within the record. Data may be simultaneously sorted, or kept in input order.

OPTIONAL OUTPUTS

* KEYS-ONLY OUTPUT: With this option, the output file receives the specified KEYS as extracted from the input records, rather than the sorted input records. This permits building an index to a file, extracting fields in order to print a summary, or selecting and rearranging (positional) fields to form a new data base.

* RECORD NUMBER OUTPUT: The output file receives the record numbers of the input records, in either ASCII or binary format. The extracted keys may accompany the record numbers, or not. This permits building MULTIPLE INDICES into a file, ordered on various keys, without duplicating the data, so that another program may retrieve the records by key value or in key order.

* POINTER OUTPUT: The output file receives the sector number and byte offset at which each record begins, with or without the keys. This provides another method of building indices whereby another program may retrieve the records from the original file.

COMPATIBLE WITH INFOSTAR

* Does File Maintenance on InfoStar Files. (See Appendix)
* Can create an index file for use by InfoStar from files created by other programs.
* Can sort InfoStar files into any desired order.

COMPATIBLE WITH BASIC

* Handles CARRIAGE-RETURN-DELIMITED RECORDS of varying lengths.
* Handles COMMA-DELIMITED FIELDS, with or without spaces surrounding the data. QUOTES may be used to embed commas or leading or trailing blanks.
* FREE FORMAT NUMBERS can be used as keys or in record selection tests. SuperSort correctly interprets the value of variable length signed or unsigned numbers with varying point position (or omitted decimal point) and the optional use of exponential (E) notation.
* It is unnecessary to use special programming techniques to produce fixed record lengths or field positions or to align decimal points in numeric data.
* Variable length records and variable length fields can REDUCE DISK FILE SIZES whether or not the file is to be processed by BASIC.
* Also handles binary INTEGER, SINGLE PRECISION, and DOUBLE PRECISION data as in MicroSoft MBASIC random files.

COMPATIBLE WITH MicroSoft FORTRAN

* Handles FORMATTED files and data in A, D, I, E, F, G and L formats.
* Handles UNFORMATTED (BINARY) files and LOGICAL, INTEGER, REAL, and DOUBLE PRECISION data.
* SuperSort may be called as a subroutine from a FORTRAN program.

COMPATIBLE WITH MicroSoft COBOL

* Handles SEQUENTIAL Files (LINE, VARIABLE-LENGTH, or FIXED-LENGTH) and RELATIVE files.

* Accepts DISPLAY, COMPUTATIONAL, or COMPUTATIONAL-3 data.

* Can convert file types, or convert records with comma-delimited fields to positional fields, to make a file originally intended for use with another language readable by COBOL, or vica versa.

* May be called as a subroutine from a COBOL program.

USER-EXIT ROUTINES

* Users with minimal programming knowledge may optionally install subroutines to inspect, replace, insert, or delete input records and/or output records.

* Once installed, these subroutines are active only when invoked by operator command.

* Typical uses of user-exit routines include:

  specialized record selection during input
  insertion of summary records or headings in the output
  reformatting the output, e.g. to produce a report

EASY OPERATION

* INTERACTIVE KEYWORD COMMAND INPUT for easy operator entry of sort/merge specifications and easy error correction.

* COMMAND FILES may be used to simplify operator invocation of regularly-used SuperSort procedures.

* ERROR MESSAGES contain explicit and understandable text.

FLEXIBLE UTILIZATION

* Supplied as a ready-to-run main program

* RELOCATABLE OBJECT CODE also supplied, permitting:

  * calling as a SUBROUTINE from FORTRAN, COBOL, or Assembler programs;
  * INSTALLATION OF USER-SUPPLIED MODULES, such as user-exit routines and custom collating sequence tables;
  * DELETING UNNEEDED FEATURES to reduce memory requirements.

HIGH PERFORMANCE

* Uses fast HEAPSORT algorithm with "sort-stretch" modification, dynamic merge optimization, and I/O buffering strategies designed to maximize throughput in a floppy disk environment.

* ADJUSTS its internal memory allocation, I/O buffer sizes, and other variables in response to various numbers of inputs, record lengths, options invoked, and amount of RAM memory available. Usable in a minimal system; runs faster and uses less disk work space in systems with additional RAM.

* SPEED: Benchmarked at over 560 records per minute under CP/M with single density diskettes. (48K system; input file containing 1000 80-character records; one 10-character key.)

B. SUPERSORT I, and II

SuperSort is available in two versions, to meet the needs of users with various requirements, as follows:

SuperSort I

SuperSort I is the complete version, with all features and capabilities described in this manual.

SuperSort II

SuperSort II is SuperSort supplied without the relocatable code files. With SuperSort II you cannot use SuperSort as a subroutine, nor invoke load options or install custom modules such as user-exit routines. All capabilities of the main program, as described in this manual, are provided.

C. HOW TO USE THIS MANUAL

Do not let the size of this manual intimidate you. You will find many suggestions in the text as to what not to read. Read the rest of this page, then continue with section II or skip to section III.

If you read the preceding feature summary and thought SuperSort looked complicated, have patience. The "Operator's Handbook" tells how to use SuperSort's basic features in a much more instructional manner.

The rest of this manual consists of three sections: "Installation", "Operators Handbook", and "Programmers Guide".

The "Installation" section is intended primarily for the individual who receives the SuperSort product, ascertains that his computer and system software are suitable, and installs SuperSort by creating backups of the distribution diskette and moving files to be used to working diskettes. Readers who are not performing this function may skip to the "Operators Handbook".

The "Operator's Handbook" defines sorting concepts, describes SuperSort facilities, and gives operating instructions for the SuperSort program. New users should read this section from the beginning, then use it as a reference as required. Programming knowledge, in general, is not assumed by this section.

At the end of the "Operator's Handbook" is a "Utilization Hints" section describing utility uses of SuperSort, application to data produced by various languages, and optimization techniques.

The "Programmer's Guide" gives additional information needed by programmers who wish to install custom modifications in SuperSort or to use SuperSort as a subroutine. Additional detail on file and data formats, that will be needed by some systems designers, is also given. This section is written primarily for those with programming knowledge.

This manual assumes familiarity with the use of the CP/M operating system or whatever similar operating system is in use at their installation. Before attempting to install or use SuperSort, readers without such familiarity should read their systems documentation, particularly with regard to entering console commands and copying files with PIP. Familiarity with the CP/M editor, or an alternate editor such as MicroPro's WORDMASTER, is also useful, for entering (optional) command files and for inspecting and correcting ASCII data files.

II. SYSTEM REQUIREMENTS AND INSTALLATION

This section is intended primarily for the individual who receives and installs the SuperSort product. If another individual has already performed this function for your system, and provided you with a diskette containing the files SORT.COM and SAMPLE.DAT, we suggest you skip to the beginning of section III, "Operator's Handbook".

A. HARDWARE REQUIREMENTS AND OPTIONS

SuperSort runs on microcomputer systems with 8080/8085/Z-80 processors, a console device (CRT or hardcopy terminal), one (preferably two) floppy disk drives, and an adequate amount of RAM memory.

The RAM requirements vary from 26K up:

26K is the minimum to operate SuperSort as distributed.

32K is sufficient to use all features on files of moderate record lengths (up to about 512 bytes), and most features on files of moderate size with longer records (to 2048 bytes).

36K is necessary to load (link) SuperSort (required only by users who wish to use the subroutine version or to install custom modifications).

48K to 64K is necessary for files with extremely long records (to 4096 bytes), and provides maximum speed and minimum disk work space requirement in all cases.

B. SOFTWARE REQUIREMENTS AND OPTIONS

SuperSort requires the CP/M operating system. The operating system must be installed and functional on your hardware before SuperSort can be installed or used.

Before the SuperSort program can be used, the operating system must be relocated (with the "CPM" command or equivalent, see your system documentation) to use at least 26K of RAM. We suggest relocating your system for all of the RAM you have available and recording (with SYSGEN or equivalent) this version of the system image on all system diskettes with which SuperSort will be used.

Most users will use the SuperSort stand-alone program as supplied and no specific other software is needed. However, users with programming knowledge who wish to use the subroutine version or to install custom modifications will also require the Microsoft linking loader, L80. Use of the subroutine version also requires the MicroSoft FORTRAN compiler, COBOL compiler, or assembler to write the calling program.

C. PACKING LIST

Upon Purchasing a SuperSort End User license you should receive:

1 copy of this manual
1 SuperSort distribution diskette
1 copy of the licensing agreement

D. THE DISTRIBUTION DISKETTE

The distribution diskette containing SuperSort is single density, IBM compatible, CP/M format and contains the following files:

Most users will be initially interested primarily in the following two files. These files are supplied with SuperSort I and II.

SORT.COM	SuperSort as a ready-to-run program
SAMPLE.DAT	sample data file, used in examples in this manual

The remaining files are supplied with SuperSort I only. The following are relocatable object files, compatible with the Microsoft loader (L80). Usage is described in the Programmer's Guide:

SORLIB.REL	Sort library, containing SORSUB, SORMSG, SORCNT, and most of SORT main program.
SORT.REL	SORT main program root.
NOERR.REL, NOREPORT.REL, NOSEL.REL, NOCOL.REL	Load options.

Source files, in Microsoft assembler (M80). Usage described in Programmer's Guide:

SUBRDEMO.MAC	Sample program that uses SORSUB
COLTAB.MAC, EBCTAB.MAC	Collating sequence tables
SYSEQA.MAC	defines system addresses

E. INSTALLATION

1. Verify that you can read the distribution diskette

   Turn on your system, cold-start ("boot") your operating system and insert the distribution diskette in a drive. You should be able to list its file directory (with the DIR command), and to TYPE the file SAMPLE.DAT.

2. Make backup copies of the distribution diskette

   Prudent data processing procedure dictates that you should always have one or more extra copies of data that you can not easily regenerate.

   The following procedure is suggested for a two-drive system:

   a. prepare a diskette containing at least the operating system image and PIP.COM. Put this in drive A.

   b. put the distribution diskette in drive B.

   c. enter the command:

      ```
      A>PIP A: = B:*.*[V]
      ```

      This copies all files on drive B to drive A, and verifies them. (With older versions of PIP, it will be necessary to enter a separate command for each file).

   d. put the distribution diskette away safely.

3. Move files to be used to working diskettes

To use the SuperSort main program, and do the demonstration exercises given in the "Operator's Handbook", you need the following files from the SuperSort distribution diskette:

```
SORT.COM
SAMPLE.DAT
```

a. have your working diskette in drive A. This diskette should contain the system image, PIP.COM, and, normally, other useful transients such as STAT, and ED or MicroPro WORDMASTER.

b. insert a copy of the SuperSort distribution diskette in drive B.

c. enter the commands

```
A>PIP A:SORT.COM=B:SORT.COM
A>PIP A:SAMPLE.DAT=B:SAMPLE.DAT
```

d. put the diskette from drive B away safely.

III. O P E R A T O R 'S H A N D B O O K

The "Operator's Handbook" defines sorting and merging concepts, describes SuperSort's capabilities, and gives operating instructions for the SuperSort program. Read from the beginning (initially skipping some sections as suggested in the text), it serves as a tutorial introduction; the later sections are also organized to function as a reference manual.

The novice in data processing, without programming knowledge, can learn how to use most SuperSort functions by reading this Operator's Handbook through from the beginning, skipping sections relating to binary files and some of the more advanced features.

Your attention is called to three other reference aids: the command reference card that is in the back of the manual, the command keywords listed after the relevant section titles in the table of contents at the front of the manual, and the phrases in the upper right corner of each page.

In addition to the stand-alone main program, SORT.COM, SuperSort is also supplied in the form of a subroutine, which a programmer can incorporate directly into an application program. Usage of the subroutine form and a number of options and possible modifications to the main program form are described in the "Programmer's Guide". The Programmer's Guide assumes familiarity with this Operator's Guide at least through the "Concepts and Facilities" section.

A. GETTING STARTED

This section leads you through using SuperSort to sort the sample file supplied. Thus, you can gain an overview of SuperSort operation and function, as well as test your copy of the program.

This example assumes that SuperSort (file SORT.COM) is installed on a diskette: the diskette should contain a 26K or larger CP/M system, and the files SORT.COM and SAMPLE.DAT. Refer to the installation section, preceding, if necessary.

Bring your computer system up and examine the contents of SAMPLE.DAT by typing:

```
A>TYPE SAMPLE.DAT
```

The computer should respond with

```
"DEFG",1,"D","071278",,,"05090",58.29,3800,1
"RBST",1,"D","071278",,,"05091",10.92,3800,1
"NEW1",1,"D","071278",,,"05092",43.28,3800,1
"HAFH",1,"D","071278",,,"05094",106.35,3800,1
"HEFF",1,"D","071278",,,"05095",2,3800,9
"HAFH",1,"D","071278",,,"05096",129.35,3800,9
"HAFH",1,"D","071278",,,"05097",45.5,3800,9
"HAFH",1,"D","071278",,,"05098",24.25,3800,9
...
```

You can terminate the typeout by hitting any key.

Note that each line (commonly called a record) contains 9 commas. The sequences or strings of characters between commas, as well as the beginning and ending strings, are each referred to as a field. Note that some fields are bracketed by quotes while others are not. This is one typical file format for use with SuperSort; other possibilities are described later.

This file might be used in an accounting system, with some of these fields having meanings as follows:

Field 1:	firm number, 1-4 characters (enclosed in quotes)
Field 4:	date, in form YYMMDD (enclosed in quotes)
Field 8:	amount of transaction, up to 14 digits with minus sign and variable decimal point position allowed
Field 9:	general ledger account number, 1-4 digits

We will now use SuperSort to create another file containing the same records as SAMPLE.DAT, but rearranged according to date, with the most recent date first.

First, invoke SuperSort by entering the command:

```
A>SORT
```

followed, of course, with a carriage return. After a few seconds, SORT should type a sign-on message and an *. The * is the prompt, signifying readiness for command entry.

Enter the command:

```
INPUT = 62, CR-DEL
```

This tells SORT that the input records are lines, separated by carriage returns, not longer than 62 characters. Again, terminate the command line with carriage return. Another * should be appear. If an error message occurs, look for a typing error and carefully re-enter the line.

Enter the commands:

```
SORT-FILE = SAMPLE.DAT
OUTPUT-FILE = OUTPUT.DAT
```

These tell sort the names of the input and output files, respectively.

Typing errors during command input may be corrected with your system's regular editing characters, including rubout to delete the preceding character, and control-U to delete the entire line.

Enter the command:

```
KEY = #4,6,DESCEND
```

This tells SuperSort to generate an output file in the descending sequence of field 4, of maximum length 6, so that the most recent date is first. The # indicates reference to a field by field number, rather than by column position.

Our sort is now completely specified. To start it, enter:

```
GO
```

This time no * appears. Instead, sorting begins. After a minute or two, a completion message should appear:

```
645 RECORDS SORTED
OUTPUT FILE SIZE 34K
WORK FILE DISK SPACE USAGE 34K
*** SORT/MERGE COMPLETE ***
```

Check the sorted output file by entering

```
A>TYPE OUTPUT.DAT
```

The computer should respond with:

```
"HAFH",1,"R","780809","    ","     ","NOREF",-7.2,8600,2
"HAFH",1,"R","780809","    ","     ","NOREF",-6.45,8600,1
"HAFH",1,"R","780809","    ","     ","NOREF",-22.89,8600,1
"HAFH",1,"R","780809","    ","     ","NOREF",-48.97,8600,1
"HAFH",1,"R","780809","    ","     ","NOREF",-79.16,8600,1
"HAFH",1,"R","780809","    ","     ","NOREF",-0.6,8600,1
...
```

B. CONCEPTS AND FACILITIES

This section describes the SuperSort functions, and define terms used in the rest of this manual.

1. FILES, RECORDS, FIELDS, AND DATA TYPES

A FILE is a set of data consisting of zero to several million bytes (characters), stored with an associated name on a diskette.

A RECORD is the unit of information at the rearrangement level, i.e. a record is the amount of information which stays together through the sorting and merging process.

In this manual RECORD always means LOGICAL RECORD, or the unit of information meaningful at the user's application level, as opposed to the "physical records" or "sectors" in which the information is recorded on the diskette.

Records processed by SuperSort can be up to 4096 bytes long.

A FIELD is a portion of a record -- normally a single data item. One or more fields are specified as KEYS, that is, the data items which determine the order into which the records are sorted. Fields are also used in record selection tests.

The data in a file may be either ASCII or binary. ASCII data is readable text - each byte contains a character in the ASCII (American Standard Code for Information Interchange) code or some other code, or a control code such as carriage return. Binary data is information (usually numeric) in one of the computer's internal representations (such as fixed point, packed decimal, etc).

SuperSort can process both ASCII and binary data; binary data in several formats can be used for keys or tested for record selection. In general, SuperSort needn't know which type of data the file contains, and there is no provision for overall specification of ASCII versus binary. However, when a field is being used as a sort key or in a record selection test, the user must in many cases specify its data and/or interpretation, via appropriate "test attributes".

ASCII data is simplest to work with and most likely to be encountered first in applying SuperSort; the reader may initially skip portions of this manual relating to binary data.

2. SORTING AND MERGING

SORTING means to rearrange input records to form an ordered output file according to a specified key or keys within the records. From one to 32 files can be sorted into a single output file; selection of records according to user-specified tests can be done at the same time.

The sort process reads the input files one after another. The data is sorted in batches called SORT BLOCKs. Each sort block is written to a WORK FILE on diskette. The sort is then completed by merging the individually ordered sort blocks to the output file. Usually, a single merge run completes the operation, but additional merges will be performed automatically, if required, due to large files and the limitations of the available RAM memory.

MERGING means to combine the records from two or more ordered input files into a single ordered output file. SuperSort can merge up to 32 files in a single run, together with or apart from sorting one or more additional files in the same run. This gives you an efficient way to combine files of unordered records with already ordered files, to produce a new ordered file containing all the records.

Merge input files are read in parallel. In most cases, no work file space is used on diskette and each record is read and written only once, resulting in fast, efficient merge execution.

A merge run with one input file only is a useful way to apply features such as record selection without reordering the data.

SuperSort does not check for sequence errors in merge inputs. If a record in a merge input file contains a key smaller than that of the preceding record in the same file, the out of sequence record is output immediately after the preceding record and merging continues.

In both sorting and merging, SuperSort outputs records with equal keys in an unspecified order. Exception: in a single-input merge the records are always kept in the original order.

3. FILE RECORD TYPES

SuperSort can handle files containing three types of records, called FIXED-LENGTH, CR-DELIMITED, and VARIABLE. In addition, it can handle a special COBOL-compatible type of fixed length record file called a RELATIVE file. Initially, the reader will probably want to concern himself only with FIXED and CR-DELIMITED record files.

FIXED-LENGTH Record Files

Every record in a FIXED-LENGTH record file is of the same length; the user may specify any length between 1 and 4096 bytes. No record separator (delimiter) is assumed for fixed-length records by SuperSort, though any character desired by the user may be present in the file as long as they are allowed for in the record length specification.

Fixed-Length records may contain data of any type.

The length of fixed-length records bears no particular relationship to the physical sector length used on the diskette; data is recorded continuously in the disk file, spanning sector boundaries.

CR-DELIMITED Record Files

CR-DELIMITED records are records of varying length delimited by carriage returns (actually, the file normally contains both an ASCII "carriage return" character and a "line feed" character between records, per CP/M convention).

For a CR-DELIMITED record file the user specifies a maximum record length between 1 and 4096; each record in the file may contain from 1 to this many data characters, plus the record delimiters. An attempt to input a CR-DELIMITED record longer than the specified length causes the record to be split into two records and a warning message printed.

CR-DELIMITED files normally contain ASCII text data only, as binary data isn't necessarily distinct from the codes for carriage return and line feed.

NOTE

Files written by most BASIC programs are CR-DELIMITED.

VARIABLE Record Files

VARIABLE records are records of varying length in which the length is stored in the first two bytes of the record. Usually, such files are created by COBOL programs.

For a VARIABLE record file the user specifies a maximum record length between 3 and 4096. Each record can contain from 1 to the maximum length less 2 bytes of data. Note that the bytes in which the length is stored must be allowed for in specifying the maximum length to SuperSort, although these bytes are not counted in the individual record lengths as stored in the file. An attempt to input an overlong VARIABLE record causes an error.

Since the length of each record is stored separately from the data, VARIABLE records may contain any data type.

CAUTION

> Do not confuse VARIABLE with CR-DELIMITED. In SuperSort files consisting of printable lines of text are always called CR-DELIMITED, NOT VARIABLE.

RELATIVE files

A RELATIVE file is a file type produced by COBOL programs which contains fixed length records and has provisions for non-existent (never written, or deleted since written) records; it also has a "header" recorded on the disk which specifies its record length and maximum active record number. (In FIXED, CR-DELIMITED, and VARIABLE files, there is no header and all records present are stored contiguously from the beginning of the file to the end.)

For a relative file, the user specifies a record length of 1 to 4096 bytes. Each record contains this many bytes of data, and may contain any data type, provided the entire record is not binary zero. For an input file, the user-specified record length must match that with which the file was written.

When SuperSort processes a relative file, all non-existent record spaces are skipped over; the output file will contain only the active records, written contiguously from the beginning.

While it is usual for the input and output records to be of the same length and type, SuperSort allows specification of a record length and file type for the output file that is different than the input file(s). In the case of record length differences, too-long records are truncated, and too-short records are extended with blanks.

If the output type is specified as different from the input, appropriate conversions will be performed. Thus, SuperSort can be used to convert from one file type to another, for instance from CR-DELIMITED to VARIABLE, RELATIVE to FIXED, or FIXED to CR-DELIMITED. Recall that a single-input merge can be used to process a file without reordering the records.

4. END-OF-FILE INDICATION CONSIDERATIONS

Users of FIXED-LENGTH record files containing binary data need to give special consideration to the method of indicating the (logical) end-of-file in the design of such files and in applying SuperSort to such files. Information on this subject is given in the "Programmer's Guide" section entitled "File and Record Formats".

5. FIELDS

There are two ways of specifying the location of a field (data item) within a record to SuperSort:

Positional Fields

For a POSITIONAL FIELD the user specifies the starting and ending positions (columns) for the field. Positional fields have fixed length and use no delimiting characters. The first character in the record is position 1.

NOTE

For VARIABLE records, the length occupies positions 1 and 2; the first data position is position 3.

Comma-Delimited Fields

A COMMA-DELIMITED field is separated from the rest of the record by commas. The user specifies the field number and (for keys) the maximum length to be used. Comma-delimited fields are convenient for use with files written by BASIC programs, as most BASICs write (and can read) values written variable-length with commas between them.

Commas may be imbedded in comma-delimited fields if enclosed in quotes ("). For compatibility with those BASICs that quote data written to files, SuperSort disregards all "'s in a comma-delimited fields when using it as a sort key or for a record selection test. Also, any leading and trailing blanks not enclosed in quotes are disregarded.

For example, field #3 is taken from between the second and third commas in the record; field #1 is from the beginning of the record to the first comma.

NOTE

For VARIABLE records, field #1 begins at position 3, i.e. after the record length bytes.

6. FIELD TEST ATTRIBUTES

SuperSort tests fields in two contexts: when used as sort keys, and when used in record selection tests. When no specific test attributes are specified by the user, fields are compared byte-by-byte, unsigned, left to right.

This default method of comparison sorts ASCII information into ASCII code order (roughly, alphabetical order: see the "Collating Sequences" section and Appendix 1). It will also sort unsigned numbers stored as text into numeric order, but only if the decimal point positions are aligned by means of leading zeroes or blanks (in a comma-delimited field, leading blanks must be quoted), and provided exponential notation is not used. It will sort binary information correctly only if the data is unsigned fixed point stored high order first.

However, if the data in the field is a number represented as text in any but the restricted format specified above, or if the field is to be treated as BCD information or most types of binary information, the proper ATTRIBUTE(s) must be specified.

Specific attributes can also cause right-justified testing, right to left comparison, treatment of lower case as upper case, and masking of the high order bit. To wit:

NUMERIC-ASCII

Determine the numeric value of the text in the field. Text may be a number with or without leading sign, decimal point, or exponential notation. No particular length or point position is required. Formats written by the PRINT statement of BASIC are correctly interpreted, as well as FORTRAN I, E, F, and G formats. Up to 14 significant digits are used; excess digits are disregarded.

Note that NUMERIC-ASCII is only for numbers stored AS TEXT. Other attributes are described below for other (binary) numeric data representations.

UPPER-CASE

Treat any lower case letter as the corresponding upper case letter.

RIGHT-JUSTIFY

Add leading blanks if necessary to extend length of data from a comma-delimited field (normally, left justification is done, by adding trailing blanks). This attribute causes ASCII information to sort with shortest values first, and causes unsigned ASCII numbers with fixed point position to sort in numeric order.

LOHI

Test right to left instead of left to right. Intended primarily for sorting binary data stored with the least significant byte first (most binary data in 8080/8085/Z-80 systems is stored in this backwards manner), but may be used to test any positional field right to left. Not allowed with comma-delimited fields.

MASK-PARITY-BIT

Ignore the high order bit of each byte.

NOTE

CP/M ASCII files normally have this bit clear; this attribute is intended for use if an unusual input source put irrelevant information (such as parity) into the high order bit.

The remaining field test attributes relate to various non-ASCII data types; the reader may wish to skip them initially.

TWOS-COMPLEMENT
COMPUTATIONAL

Either of these words means to interpret the field as SIGNED fixed point binary information. When used alone, the data is assumed to be stored high order first (see LOHI and INTEGER for data stored in normal 8080 backwards format). These attributes may be used with fields of any length; use with a field length of 2 bytes for COBOL COMPUTATIONAL data.

INTEGER

Treat data as signed, twos-complement, fixed point binary data stored low order first. Equivalent to the two attributes LOHI and TWOS-COMPLEMENT used together. Use with a 2-byte field for FORTRAN INTEGER data in an unformatted (binary) file or MBASIC INTEGER data in a random (binary) file.

FLOATING-POINT

Interpret data as MicroSoft floating point format. Use with a 4-byte field for FORTRAN REAL data in an unformatted (binary) file or MBASIC SINGLE PRECISION data in a random (binary) file; use with an 8-byte field for FORTRAN or MBASIC DOUBLE PRECISION data in binary files.

PACKED-BCD
COMPUTATIONAL-3

Treat as packed BCD (Binary Coded Decimal) with two digits stored in each byte and an optional sign in standard IBM format. See the "Programmer's Guide" for details of this data format. Use for COBOL COMPUTATIONAL-3 data.

CAUTION

The normal abbreviation rules for entering SuperSort command keywords (as described below) do not apply to "COMPUTATIONAL-3". Spell it out, or read the details in the description of the KEY command.

In addition, ASCENDING (normal) or DESCENDING (backwards) sequence can be specified for each key field (intermixed sequence indicators).

Two more attributes, EBCDIC and ALTSEQ, are described later in the "Collating Sequences" section.

7. BASIC COMPATIBILITY

When programs are written in BASIC to generate files that will be sorted and/or merged, no special programming is required. In particular, it is not necessary to use PRINT USING nor to generate fixed-length records.

The PRINT statement can generate ASCII files with carriage returns between records, commas between fields, and numbers in free format. Such files use minimal disk space, are easy to program for, and sort quickly because of the small file size.

To process such files with SuperSort, specify CR-DELIMITED records, comma-delimited fields, and use the NUMERIC-ASCII attribute for numeric fields.

In some BASICs, such as CBASIC, the commas between fields are supplied automatically; in others, such as MBASIC, it is necessary to explicitly PRINT the commas. Some BASICs quote string data in files, others do not; SuperSort handles either format.

Further notes on using SuperSort with BASIC and other programming languages are given in the "Utilization Hints" section.

----------------*----------------

At this point, those concepts involved in use of SuperSort's fundamental features have been defined; the reader may wish to examine the first few sections of section III-C, "Operating the SuperSort Program", and do some trial sort operations, before continuing with this section.

----------------*----------------

8. COLLATING SEQUENCES

A COLLATING SEQUENCE is the order into which various characters are sorted. SuperSort's default collating sequence is the unsigned binary value of each byte. For ASCII textual data, this results in sorting in the ASCII code order, which goes, briefly as follows:

space, digits Ø-9, upper case A-Z, lower case a-z

with the punctuation and special characters scattered between the above groups. Refer to appendix 1 for details. The order is, of course, modified by the UPPER-CASE attribute, described previously. Also, the NUMERIC-ASCII, INTEGER, and PACKED-BCD attributes change the whole method of comparing fields.

There are two additional attributes which can be specified on a field-by-field basis to change the collating sequence:

EBCDIC

Sort ASCII data AS THOUGH it were in IBM's "EBCDIC" code. That is, test on the value of the EBCDIC code corresponding to the ASCII code actually in the file. The EBCDIC code goes, briefly,

space, lower case a-z, upper case A-Z, digits Ø-9

As with ASCII, the punctuation and special characters are sprinkled in the gaps, and full details are given in Appendix 1.

NOTE

If your data is in EBCDIC and you want the EBCDIC collating sequence, do NOT specify EBCDIC.

ALTSEQ

Use a USER-SPECIFIED alternate collating sequence. The alternate sequence may be specified to the SuperSort program by command, and/or via a custom-installed table (see the "Programmer's Guide"). For the SuperSort subroutine, the sequence is specified by passing a table as one of the arguments. Note that to use a user-specified collating sequence, it is necessary to both specify the sequence, AND invoke its with the ALTSEQ attribute for EACH applicable key field and/or record selection test.

Since there are two optional collating sequences, plus the default, it is possible to apply THREE different collating sequences in the same run (on separate fields, of course).

9. OUTPUT OPTIONS

Normally, the output file contains the rearranged input records. This is referred to as FULL RECORD OUTPUT. In addition, there are three optional types of output - keys, record numbers, and pointers - and two combination options: keys plus record numbers, and keys plus pointers.

K-OUTPUT (keys only)

The output file receives the only keys as extracted from the input records. The keys are concatenated together, in the order that the key field specifications are given. A carriage return line feed follows the keys if the output file is specified as CR-DELIMITED.

Certain transformations are performed on the keys, including removing quotes from comma-delimited fields and filling them out to their specified length with trailing blanks (leading blanks if

RIGHT-JUSTIFY is specified), ones-complementing DESCENDING keys, and conversion of NUMERIC-ASCII fields to an internal format. The transformations are detailed in the "Programmer's Guide".

Uses of K-OUTPUT include:

Extracting fields from a file, to permit printing a summary - for example, a list of the names of all clients who have records in the file. Be careful not to use any attributes (such as DESCENDING or NUMERIC-ASCII) which produce unprintable results.

Extracting fields from a file, to produce a compact index which may be used by a program to quickly locate records for retrieval from the original file; this is particularly useful if the records are long and the keys are short. The keys only file will be in the same order as the input file if it is produced with a single-input merge.

Rearranging (positional) fields in a file: specify the fields as keys, in the order you wish them to appear in the output file.

Converting a file with comma-delimited fields to positional fields (K-OUTPUT always has a constant length, with quotes and commas removed).

R-OUTPUT (record numbers)

The output file receives only the record numbers (1=first record) of the input file, arranged in order according to the specified keys. This creates a compact file which may be used by a program to process a fixed length record file in the key order, without duplicating the entire file. The record numbers will be in ASCII (text) if the output file is specified as CR-DELIMITED; otherwise, they will be binary. The output formats are detailed in the "Programmer's Guide".

P-OUTPUT (pointers)

Similar to R-OUTPUT output except that the output file receives the sector number and displacement into the sector of the start of each record, instead of the record numbers. Such pointers permit retrieval of records of varying lengths (CR-DELIMITED or VARIABLE), whereas record numbers are generally only useful with fixed length records. On the other hand, record numbers are easier to process in many programming languages.

KR-OUTPUT (keys and record numbers)
KP-OUTPUT (keys and pointers)

These output the keys (see K-OUTPUT) followed by the record numbers (see R-OUTPUT) or pointers (see P-OUTPUT).

For files with records that are long with respect to the keys, A KP-OUTPUT or KR-OUTPUT file produces a compact index which can be searched quickly by a program to find a particular record by key value.

Several KP- or KR-OUTPUT files derived from the same input file can be used to permit retrieving records by various keys. This uses less disk space than multiple copies of the original data, as well as saving search time.

Futhermore, in applications that involve update in place of the non-key portions of the record, use of KR-OUTPUT or KP-OUTPUT indices eliminates the need to update multiple files or to re-sort after update.

The above output options, except K-OUTPUT, may be used only with a single input file; this file may be sorted (to put the output in key order) or merged only (to maintain the input order). K-OUTPUT may be used with multiple sort and/or merge input files.

10. TAGSORT

TAGSORT is an optional METHOD of sorting that has no effect on the contents of the output file. The effect of tagsort is to reduce the amount of work file space needed on disk, traded against longer processing times. Tagsort may be requested when there is one sort input file and no merge input files.

Tagsort is recommended when the input file is relatively large with respect to the space available for the work file, when the input file contains unusually long records, especially if the key length is short relative to the record length, and when there is relatively little RAM in the system (or a relatively small caller-supplied working storage area for the subroutine).

Tagsort is not recommended for small sorts, when there is plenty of space for the work file, lots of RAM available, and when maximum speed is desired.

11. DISKETTE CHANGING

Output Diskette Change

The user may request that the diskette be changed after the input file is read and before the output file is written. This request is made by appending a "/C" to the output file name. When this is requested, SuperSort will type a message at the appropriate time telling the operator which drive's diskette to change, then await a carriage return. NO DISKETTE OTHER THAN THE REQUESTED ONE SHOULD BE CHANGED.

With this option, it is usually possible to sort an entire diskette full of data in a two-drive system, without destroying the input, by using the same drive for input and output and a scratch diskette in the other drive for the work file. We say "usually" because if the amount of RAM in use is relatively small, and/or the records are unusually long, the amount of work file space required will be larger than the input file(s), and thus larger than the capacity of a diskette.

When using output diskette change, the user must place the following files on drives other than the output drive:

all merge-only inputs, if any;
the input file, if tagsort is being used; and
the work file (the work file defaults to the current drive; there are provisions to assign it to any desired drive).

Thus, output diskette change is useful primarily when sorting without tagsort and when there are no merge inputs.

Before-Start Diskette Change

After the SuperSort program is invoked but before sorting/merging has been initiated by operator command, you may change diskettes in any drive. This means the space that would otherwise be taken up by the file SORT.COM (from which SuperSort is loaded) can be made available for data during sorting.

NOTE

When SuperSort is run under MP/M the preceding "Before-Start Diskette Change" rule is not in effect. If it is desired to change disks under MP/M after SuperSort loads, it is necessary to issue a CHANGE command to SuperSort. When the CHANGE command is issued disks may be changed as described above. Note that this command is merely redundant when issued under CP/M.

The user is cautioned that the above options are due to special code in the SuperSort main program, and that in general DISKETTES SHOULD NEVER BE CHANGED UNDER CP/M without a warm-start (control-C). The user is further cautioned that disregarding the above rule, except for the clearly documented exceptions such as SuperSort's output diskette change and before-start diskette change provisions, is likely to destroy the data in his files.

There is no provision for changing diskettes between input files nor during the reading or writing of a file.

12. RECORD SELECTION

There are two methods to specify records to be chosen from the input. In either case, all undesired records are rejected as they are read; they do not enter into the sort/merge process and do not appear in the output.

Extraction by Record Number

A first and last record number (first record=1) may be specified for each sort input file. Only records between these numbers, inclusive, will be included. This feature is not available for merge-only input files.

Selection based on Conditional Tests of the Data

Record selection may also be based on tests of record fields.

There are eight test conditions (comparison operators) available: between, not between, less than, less or equal, not equal, equal, greater or equal, and greater than.

The test (comparison) may be between a field and a constant, or between fields in the same record.

Constants to test against (compare to) may be quoted text (the common case), binary (expressed in hex, octal, or decimal radix), or BCD. There is no practical length limit to quoted texts or other forms of constants.

In the case of the "between" and "not between" operators, one value (usually a field) is tested for being in or outside the range specified by two other values (commonly constants).

All of the field test attributes described previously -- NUMERIC-ASCII, RIGHT-JUSTIFY, PACKED-BCD, FLOATING-POINT, etc. may be applied to the comparisons used in record selection.

Complex conditions can be expressed by using the logical operators AND, OR, XOR, and NOT to combine tests. In the SuperSort program, four levels of nested parentheses may be used to specify grouping of operands.

Furthermore, in the SuperSort program, up to 32 SELECT and/or EXCLUDE commands, each utilizing all of the above features, may be given. (In calls to the subroutine, multiple conditions are expressed as "AND"'s.)

Record Selection by tests of the data acts both on sort input files and on merge-only input files. Thus, if you wish to select records without reordering them, use a one input merge: specify one merge input file and no sort input files.

More exposition on record selection by conditional tests of the data is given in the "Introduction to SELECT/EXCLUDE" section of the "Operating The SuperSort Program" section and following sections.

13. MESSAGE CONTROL

SuperSort provides six levels of status information printout, varying from nothing, through a short printout of number of records processed, through a detailed breakdown of number of records sorted, merged, not selected, etc, to during-execution reporting of the phase of processing (sort, merge, final merge, etc).

The degree of information printout is specified by a number called the PRINT LEVEL. The user can specify print levels from Ø (to print nothing) to 5 (to print the most). The default print level is 2.

At all levels, the printout is kept as brief as practical for the particular run by suppressing zero quantities and redundant quantities (e.g. the number of output records is never printed when it is the same as the number of sorted records).

Print levels Ø through 5 result in messages as follows:

```
     Level Ø: nothing printed

     Level 1:  number of records sorted if non-Ø
               number of records merged if non-Ø
               number of output records if not redundant
               work file space in Kilobytes if non-Ø

     Level 2:  this is the default level.
               prints as above, plus:
               number of sort input records if not redundant
               number of merge input records if not redundant
               number of records rejected by record selection:
                    from sort input if non-Ø
                    from merge input if non-Ø
               User-exit (see below) insertion and deletion counts,
                    for sort input, merge input, and output, if non-Ø
               work file space in Kilobytes if non-Ø

     Level 3: as above, plus:
               number of sort runs (sort blocks)
               number of merge runs

     Level 4: as above, and prints as pertinent:
                    SORTING...
                    MERGING...
                    ADDITIONAL MERGE...
                    FINAL MERGE...
```

Level 5: as above, plus:
number of bytes of working storage
number of input records to each sort run
number of sort blocks input to each merge run

The section entitled "Execution Messages" gives examples and more detailed explanations of these messages.

14. MAIN PROGRAM AND SUBROUTINE FORMS OF SuperSort

The facilities described above are available in SORT, the SuperSort main program, and are also provided as a subroutine with SuperSort I.

The main program is used by entering COMMANDS to request the desired sorting and/or merging operations; these commands are the subject of the next section, "Operating the SuperSort Program". The main program will fill most sorting and merging needs, and can be used without programming or knowledge of programming.

The SuperSort subroutine, SORSUB, is utilized via calls from a user-written program. In addition to SORSUB, subsidiary routines are supplied to print error messages and to return to the calling program the various counters that SuperSort maintains. Calling sequences and loading procedures are given in the "Programmer's Guide".

15. USER-EXIT ROUTINES

USER-EXIT ROUTINES are subroutines WRITTEN BY THE USER and installed in SuperSort (SuperSort I only). There are provisions for two user exit routines: XIT1, which operates on input records, and XIT2, which operates on output records. Each is called for each transfered record. The current record is passed to the routine, which may request that:

* the record be DELETED, or
* ACCEPTED, or
* REPLACED with another record returned by the routine,
* or that another record returned by the routine be INSERTED ahead of the current record, with the current record being transmitted to the exit routine again on the next call.

The latter two possibilities are not allowed for merge input records, nor when using tagsort, R-OUTPUT, P-OUTPUT, KR-OUTPUT, or KP-OUTPUT.

The exit routines are called again, with a flag being passed, at end-of-file. This allows adding one or more records to the end of the sort input and/or output. Such records might contain summary information generated during inspection of the individual records.

Since the user-exit routines allow custom user code to be combined with all of SuperSort's features and options, installing user-exit routines in SuperSort can produce powerful special-purpose application programs with relatively little programming effort. Possible uses of user-exit routines include:

* to select records in a special manner;
* to reformat the records before or after sorting, for example, to make mailing labels from an address file;
* to accumulate summary information;
* to detect control breaks and insert headings or summary records;
* to "pipeline" input and/or output data directly from/to the user's program, without going via a file (by using dummy one-record files).

Once installed (linked), the user-exit routines can be individually invoked or not in a given run of the sort program or subroutine.

The calling sequence for the user-exit routines is such that they can be coded in Microsoft Assembler or FORTRAN. Calling Sequences and Installation procedures for user-exit routines are given in the "Programmer's Guide".

16. LOAD OPTIONS

LOAD OPTIONS are options the user may specify when loading SuperSort (SuperSort I only). The load options provided include:

* User-exit routines (previous section);
* Custom collating sequence tables:
  the EBCDIC table may be changed to any desired sequence;
  the default ALTSEQ table for the main program may also be changed to any desired sequence;
* Options to reduce memory requirements:
  by reducing the amount of message text, or
  by eliminating code for unneeded functions, such as record selection or alternate collating sequences.

There are provisions for reloading the program version in order to specify load options, as well as for invoking load options when loading the subroutine with your program.

See the "Programmer's Guide" for details.

C. OPERATING THE SUPERSORT PROGRAM

This section describes the standard stand-alone program version of SuperSort. Topics covered include invoking the program, command entry in general, entering commands via console or command file, and the specific commands required to invoke all of the SuperSort functions.

Refer back to the "Concepts and Facilities" section as necessary for additional descriptions of the SuperSort functions.

1. INTRODUCTION TO SORT USAGE

The SuperSort program, SORT, performs sorting, merging, and record selection operations as requested in COMMANDS specified by the user. The commands may be interactively input from the console, pre-stored in a command file, and/or included in the CP/M command line that invokes SORT.

In the simplest case, the SuperSort program is invoked with the system command line

```
A>SORT
```

The "A>" shown is the system prompt, which will already be displayed when the system is ready to accept a command. The line must be terminated with a carriage return, as are all lines entered by the user.

If a system message LOAD ERROR or TOO BIG occurs in response to the above command, you are not using a 24K or larger CP/M. To correct this situation, use the "CPM" or equivalent command, as explained in your CP/M system documentation.

When thus invoked, the SuperSort program prints a sign-on message, then prints an * as a prompt to signify that it is ready to accept input. The user then enters the desired commands -- as described in the rest of this section. For an example, see the "Getting Started" section.

When entering command lines, terminate each line with carriage return. Your system's regular editing characters may be used to correct typing errors, including RUBOUT to delete the preceding character, and CTRL-U to delete the entire line.

Some or all of the commands may be entered on the system command line, after the word "SORT". An example, using commands which will be explained later:

```
A>SORT INP=80,CR; SO=A.DAT; OUT=O.DAT; KEY=#1,5; GO
```

If the SuperSort commands on the system command line are sufficient to completely specify and start the sort/merge operation, SORT starts immediately with no sign-on and no console input.

NOTE

It is permissable to change diskettes after SORT is invoked but before execution is started ("GO" command).

2. COMMAND FORMAT AND ABBREVIATION

Each SuperSort command consists of a keyword, identifying the type of command, followed, usually, by parameters. Depending on the command type, the parameters include such things as filenames, numbers, and attribute keywords.

Keywords are formed of letters and -'s and possess both long and short (abbreviated) forms. The letters may be entered in upper case or lower case.

The command descriptions that follow show the keywords in their long, self-documenting form. While this form is easy to read, it can be inconvenient to type in. Hence, abbreviating is allowed as follows: letters may be omitted from the end of the keyword, and/or any -'s can be omitted.

A minimum number of letters at the beginning of the keyword must be typed: ONE, for the most commonly used commands, TWO, in most other cases; THREE if the word begins with "NO". Three or more letters are always accepted. A few exceptions are noted below.

The examples of abbreviated entry shown below are all equivalent:

```
INPUT-ATTRIBUTE
INPUT
IN
I
```

To further simplify entering commands, spaces may be used instead of commas, and in place of the equal sign after the command keyword. Spaces may be used freely between items, but should not be embedded in a keyword or parameter.

Examples of commands abbreviated to various degrees will be given in the command descriptions.

3. ORDER OF COMMAND ENTRY

The several commands necessary to specify a sort/merge operation may be entered in any order. There are two exceptions: keys must be specified in the desired order, and GO, which starts execution, must be last.

4. CHANGING VERSUS ADDING COMMANDS

In general, commands may be repeated, with only the last occurrence effective. This allows you to change your mind and alter previous entries.

However, certain commands which can have multiple entries (e.g. KEY=) may be repeated using a + instead of an = to indicate additional rather than replacement entries. (Note: in this context, a space used instead of an equal sign is treated as the equal sign. The plus sign therefore must be used when specifying additional entries.)

5. COMMAND LINES

SuperSort commands are commonly entered one to a line. However, the following variations are also permitted:

Multiple commands on one line	
	separate the commands with SEMICOLONS.
Continuation of same command on next line	
	type an AMPERSAND (&) before the carriage return.
Comments	a VERTICAL BAR (|) means the rest of the line is a comment.

6. COMMAND DESCRIPTION NOTATION

In defining commands, we will use the following notational convention:

UPPER CASE	enter as shown; keywords may be abbreviated.
lower case	substitute as required: filename, number, etc.
( )	enter one of the choices shown one above another.
[ ]	optional: enter or omit
{ }	enter none or any number of the column of alternatives, in any order, separated by commas or spaces.
...	repeat preceding parameters if desired, as many times as desired.

This notation will be clarified via examples in the following command descriptions.

7. SPECIFYING THE INPUT FILE(S)

Three commands are used to define the input files:

INPUT-ATTRIBUTES	specifies record length, file record type, and other common attributes for all input files. Must always be given
SORT-FILES	names files to be sorted, and optionally gives record number of first and last record to extract from each file
MERGE-FILES	names files to be merged

Either a SORT-FILES or a MERGE-FILES command must always be given; both may be used, if desired, in the same run.

INPUT-ATTRIBUTES command

The form of the INPUT-ATTRIBUTES command is as follows (recall that a column of alternatives enclosed in { }'s means that none, one, or several of them may be entered, in any order, separated by commas or spaces):

```
INPUT-ATTRIBUTES = record-length   { FIXED-LENGTH   }
                                   { CR-DELIMITED   }
                                   { VARIABLE       }
                                   { RELATIVE       }
                                   { NO-SINGLE-Z    }
                                   { NO-ZZZ         }
                                   { FFZZZ          }
```

record-length	is a number between 1 and 4096 specifying the length in bytes of fixed length records, or the maximum length of CR-DELIMITED or VARIABLE records.

One of the following four attributes may be given to specify the file record type for all of the input files. If none is given, the default FIXED-LENGTH is assumed. See the "Concepts and Facilities" section for further description of each type.

FIXED-LENGTH	input file(s) contain fixed-length records.
CR-DELIMITED	input file(s) contain records separated by carriage return and/or line feed characters, usually of varying lengths.
VARIABLE	input file(s) contain "variable length records" with the length in the first two bytes of each record. Note that record length must be specified 2 greater than the maximum number of data types in a record. Don't confuse this with CR-DELIMITED.

RELATIVE	input file(s) are COBOL "relative" files.

We will describe the rest of the attributes after the following examples of valid INPUT-ATTRIBUTES commands:

```
INPUT-ATTRIBUTES = 128             (fixed-length assumed)
INPUT-ATTRIBUTES = 67, RELATIVE
INPUT-ATTRIBUTES = 80, CR-DELIMITED
INPUT-ATTRIBUTES = 293, FIXED-LENGTH
INPUT = 293, FIXED
I 293 FI                           (minimum abbreviation)
```

The last three examples are equivalent; they show full entry, partial abbreviation, and full abbreviation.

The remaining three input attributes relate to end-of-file detection in FIXED-LENGTH record files containing binary data; they need not concern users of other file and data types. See the "Programmer's Guide" for further information.

NO-SINGLE-Z	do NOT end FIXED-LENGTH record input file upon encountering a single 1A hex byte (control-Z).
NO-ZZZ	do NOT end FIXED-LENGTH record input file upon a record length of hex 1A's. Implies NO-SINGLE-Z.
FFZZZ	DO end input file upon reading a record beginning with two FF hex bytes, followed by enough 1A's to fill the record.

Additional examples of INPUT-ATTRIBUTES Commands:

```
INPUT-ATTRIBUTES = 1024, FIXED-LENGTH, NO-SINGLE-Z
INPUT = 200, FIX, NOZZ, FFZZZ
I 200 FI NOZ FF          (abbreviation of preceding)
```

SORT-FILES Command

The SORT-FILES command specifies names of files to be sorted, plus optional start and end record numbers for each. Its form is as follows (recall that a column of alternatives in ( )'s means to enter one of them, and that [ ]'s enclose optional items):

```
SORT-FILES (=) filename [ (start-record, end-record ) ] , ...
           (+)
```

=	means forget any sort input files previously specified. Assumed if neither + nor = given.
+	means use the file(s) specified in this command in addition to any previously specified. Up to 32 sort input files may be used in one run.

filename	is CP/M file name, with optional drive and type fields, in accordance with standard CP/M file naming conventions.
start-record	number from 1 to 65535 indicating first record to include in the sort input; 0 or 1 means begin at the beginning of the file
end-record	number of last record to include; 0, 65535, or omission signifies including all records to end of file.

The start and end record numbers are enclosed in parentheses after the filename they apply to; the record numbers and parentheses may be omitted to sort the entire file.

CP/M file names consist of:

optional drive name:	A, B, C, ... followed by colon. If omitted, file is on current drive
file name:	1 to 8 letters or digits; some punctuation characters also accepted.
optional file type:	period and 1 to 3 letters or digits

Examples of SORT-FILES commands:

```
SORT-FILE = SAMPLE.DAT
SORT-FILE = B:SAMPLE.DAT (100, 200)
SORT-FILES = FILE1.ABC, FILE2.XYZ, FILE3.001
SORT = B:AAA(1,234), C:XXX.Y(100,65535)
SORT-FILES + DDD, EEE(3,73), FFF
S SAMPLE.DAT                  (minimum abbreviation)
```

MERGE-FILES Command

The MERGE-FILES command names files to be merged without being sorted:

```
MERGE-FILES (=) filename, filename, ...
            (+)
```

filename	is a CP/M file name
=	replace any previously specified merge file names with those in this command. Assumed if neither = nor + given.
+	use the merge file names specified in this command in addition to any previously specified. Up to 32 merge files are allowed.

Examples of MERGE-FILES Commands:

```
          MERGE-FILE = OLDATA.DAT
          M OLDATA.DAT                  (minimum abbreviation)
          MERGE + A:FILE1.DAT, B:FILE2.DAT, C:FILE3.DAT
```

When a SORT-FILES or MERGE-FILES command is entered, SuperSort immediately checks that the file(s) exist. An error message is printed if a file is not found. Re-enter the command to correct a typing error.

8. SPECIFYING THE OUTPUT FILE

The OUTPUT-FILE command names the output file, specifies its attributes (if it is desired to specify different attributes from the input files), and output options:

```
      OUTPUT-FILE = filename[/C] [,rec-len]  {FIXED-LENGTH    }
                                             {CR-DELIMITED    }
                                             {VARIABLE        }
                                             {RELATIVE        }
                                             {K-OUTPUT        }
                                             {R-OUTPUT        }
                                             {P-OUTPUT        }
                                             {KR-OUTPUT       }
                                             {KP-OUTPUT       }
                                             {FFZZZ           }
                                             {NO-FFZZZ        }
```

filename	is the name of the CP/M file to receive the sorted and/or merged output. The output file must be different from all merge input files; it may be the same as a sort input file, but outputting to an input file should be used with great caution.
/C	indicates that you wish to change the diskette before the output file is written. SORT will type a message and await a carriage return at the appropriate time.

The following parameters of the OUTPUT-FILE command may be omitted unless you wish to specify values different than those specified in the INPUT-ATTRIBUTES command:

rec-len	number from 1 to 4096 specifying record length in bytes, or maximum for CR-DELIMITED or VARIABLE records. Ignored if an output option (K-OUTPUT, etc) is specified - the optional forms of output have predetermined lengths.

```
     FIXED-LENGTH         file records types.  See
     CR-DELIMITED         "Concepts and Facilities"
     VARIABLE             section and description of
     RELATIVE             INPUT-ATTRIBUTES command.
```

Examples of OUTPUT-FILE commands using the preceding descriptions:

```
OUTPUT-FILE = SAMPLE.SRT
O SAMPLE.SRT                    (minimum abbreviation)
OUTPUT = B:SAMPLE.SRT/C
OUTPUT = FOO, FIXED
OUTPUT = FOO.DAT, 2000, RELATIVE
```

The following notes are significant only when the output file record type and/or record length is specified as different from the input, i.e. when using SuperSort to convert from one file type to another or to a different (maximum) record length:

1. If the output file record type is different and the (maximum) record length is not specified, it defaults to the input record length specification except as follows:

   a. If the input is VARIABLE but the output is not, the output record length defaults to two less than the input record length, because the length bytes are deleted.

   b. If the input is not VARIABLE but the output is, the output record length defaults to two greater than the input, but not greater than 4096.

2. If the output record length is too small to receive a given record, the record is truncated at the right. The length bytes for VARIABLE records are suitably adjusted.

3. If the output is FIXED or RELATIVE, records are extended with blanks if necessary to make up the specified length.

The following output options produce output files with special data in them, as opposed to rearranged input records. The motivations for them are described in the "Concepts and Facilities" section and exact specification of the data formats produced and their record lengths is given in the "Programmer's Guide".

K-OUTPUT	output file is to receive only the keys, as extracted from the input records, with certain attributes already applied. Record length is total key length, with modification for certain attributes.
R-OUTPUT	output file is to receive the sorted record numbers only. If output file is CR-DELIMITED, numbers will be in ASCII text with six digits each; other file types receive 3-byte binary record numbers.
P-OUTPUT	output file is to receive only pointers to the input records. Each pointer consists of the sector number in the file where the record begins and the byte offset to the beginning of the record. The numbers are ASCII, with a comma between them, for a CR-DELIMITED file, or binary, 2 bytes each, for other file types.

KR-OUTPUT	output file receives keys, as above, followed by record numbers, as above.
KP-OUTPUT	output file receives pointers, as above, followed by record numbers, as above.
RECORD-NUMBERS	equivalent to R-OUTPUT, for compatibility with previous releases. Minimum abbreviation is REC.
KEYS-AND-NUMBERS	equivalent to KR-OUTPUT, for compatibility with previous releases. Minimum abbreviation is now KEY-A to avoid confusion with K-OUTPUT.

When any of the above output options except K-OUTPUT is used, only a single input file is allowed. This may be a sort input file or a merge-only input file.

The following output file attributes are of interest only to users of FIXED-LENGTH record files containing binary data; see the "Programmer's Guide" for further information:

FFZZZ	terminate output file by filling excess space in last sector, if any, with two FF hex bytes, then 1A hex (Control-Z)'s. This is the default if FFZZZ was specified for the input.
NO-FFZZZ	terminate output file by filling last sector with hex 1A's (Control-Z's). This is the normal CP/M file termination convention; it is the default UNLESS FFZZZ was specfied for the input.

Additional examples of OUTPUT-FILES commands:

```
        OUTPUT-FILE= ITPOINTS.IDX, FIXED-LENGTH, KR-OUTPUT
        OUTPUT = MASTER.009/C, 2000, FIX, NOFFZ
```

Don't overlook the fact that any previous contents of the output file are lost. SuperSort is capable of writing the output on one of the sort input files, but prudent procedure dictates that you should never create a situation where the input could be lost before creation of the output was complete. Either use a different file for the output, or copy the input file to a backup diskette before sorting.

9. SPECIFYING THE KEY(S)

Use the KEY command to specify one or more sort keys. Up to 32 keys, each with its own attributes, may be used. Multiple keys may be specified in a single command (continued on next line with & if necessary), or in multiple commands (use KEY+). The keys should be specified in the desired priority order.

form of the KEY command is:

```
KEY (=) ( start-posn, end-posn ) { ASCENDING         } , ...
    (+) ( # field-no, max-len )  { DESCENDING        }
                                 { NUMERIC-ASCII     }
                                 { UPPER-CASE        }
                                 { RIGHT-JUSTIFY     }
                                 { LOHI              }
                                 { MASK-PARITY-BIT   }
                                 { EBCDIC            }
                                 { ALTSEQ            }
                                 { TWOS-COMPLEMENT   }
                                 { COMPUTATIONAL     }
                                 { INTEGER           }
                                 { FLOATING-POINT    }
                                 { PACKED-BCD        }
                                 { COMPUTATIONAL-3   }
```

=	means replace previously specified keys
+	means add to previously specified keys

For a positional (columnar) field, give two numbers as follows:

start-posn	first column or position of the field
end-posn	last (inclusive) position

NOTE

For VARIABLE records, the first data character is at position 3, as opposed to position 1.

For a comma-delimited field, enter a # and two numbers. The # indicates field reference rather than column position reference:

field-no	field number, e.g. field #3 is between the second and third commas
max-len	maximum number of characters to take from the field, after deleting quotes, leading blanks, and trailing blanks. Excess characters are ignored; a shortage is padded with blanks.

The following specify the sort sequence; ASCENDING is the default:

ASCENDING	sort into forward (increasing) order on this field
DESCENDING	sort into reverse (decreasing; backwards) order on this field

Here are some examples using KEY features already described:

```
     KEY = 2, 10, ASCENDING              (columns 1 through 10)
     KEY = # 2, 10 ASCENDING             (2nd field, max 10 characters)
     KEY = # 2, 10, DESCENDING           (same, but opposite order)
     K #2 10 DE                          (minimum abbreviation)
```

More examples follow. Note that several keys may be specified in one command, with the # and ASCENDING/DESCENDING repeated as applicable:

```
     KEY = 2, 10, ASC,  20, 45, DESC
     KEY = #11,40,ASC,  #3,5,DESC,  #6,8,ASC
     KEY = 1,10 ASC, # 4,20 DESC
```

Some test attributes which may be specified in the KEY command are:

NUMERIC-ASCII	interpret field as a free-format number stored as text (as opposed to binary). Sorts data as written by the PRINT statement of BASIC into numeric order: most negative number first, then values increasing through zero to the largest positive number. This feature is further discussed in the "Concepts and Facilities" section and the range of number formats accepted is detailed in the "Programmer's Guide".
UPPER-CASE	treat any lower case letters in this field as the corresponding upper case letter.
RIGHT-JUSTIFY	add blanks to beginning, not end, of comma-delimited field if shorter than the specified maximum length.
LOHI	test this field right to left. Intended primarily for binary data which is stored low order first, but may be used with any positional field. Not allowed for comma-delimited fields.
MASK-PARITY-BIT	ignore the high order bit of each byte.
EBCDIC	sort ASCII text in this field as though in the IBM EBCDIC code (see Appendix 1). Ignores the high order bit of each byte.
ALTSEQ	sort this field according to user-specified collating sequence. See the COLLATING-SEQUENCE command below; also see the "Programmer's Guide" re installing custom tables.

Some more examples:

```
     KEY = #17, 10, NUMERIC-ASCII, DESCENDING
     KEY + #5,10 DESC,NUM-ASC,  #8, 20, ASCEND, RIGHT, UPPER
     KEY = 1,10 LOHI,DESCEND, 11,20 ALTSEQ,ASCEND
     KEY = 200,209 EBCDIC
```

The remaining test attributes relate to various non-ASCII data types; the user of ASCII data may initially wish to skip the rest of this description of the KEY command.

TWOS-COMPLEMENT COMPUTATIONAL	Either of these means that the data is signed fixed point binary, with negative numbers stored in twos-complement representation. Used alone, the data is assumed to be stored high order first. (Most binary data in 8080/8085/Z-80 systems is stored low order first; see LOHI above and INTEGER below). May be used with any field length; use with a 2-byte field for COBOL COMPUTATIONAL data.
INTEGER	indicates signed binary fixed point data stored low order first; equivalent to LOHI and TWOS-COMPLEMENT used together. Use with a 2-byte field for MBASIC or FORTRAN INTEGER data.

CAUTION

To users of prior releases of SuperSort: the meaning of INTEGER has been changed; in releases 1.02 and older it meant what TWOS-COMPLEMENT now means.

FLOATING-POINT	indicates data is in MicroSoft floating point binary format. Use with a 4-byte field for MBASIC SINGLE PRECISION data or FORTRAN REAL data, or with an 8-byte field for MBASIC or FORTRAN DOUBLE PRECISION data.
PACKED-BCD COMPUTATIONAL-3	Either of these indicates packed binary coded decimal data, with or without sign. Use for COBOL COMPUTATIONAL-3 data.

CAUTION

COMPUTATIONAL-3 should be abbreviated only as COMP-3 or C3. Other abbreviations will be misinterpreted as COMPUTATIONAL without the "3".

NOTE

For UNSIGNED binary fixed point data (where values with the high order bit set represent large positive numbers, not negative numbers) stored low order first, use LOHI. For such data stored high order first, no attribute is needed.

Additional examples of KEY commands:

```
KEY = 1,1 TWOS-COMPLEMENT, 3,4 TWOS-COMPLEMENT
KEY = 1,2 INTEGER, DESC, 3,6 FLOATING-POINT, 11,14 FLOAT
KEY + 11,12 TWOS-COMPLEMENT, 5,10 PACKED-BCD
KEY + 11,12 COMPUTATIONAL, 5,10 COMP-3   (same as preceding)
```

10. STARTING EXECUTION

When all of the desired commands have been entered, start SuperSort execution by entering the command

```
          GO
```

The GO command (or simply G) will generate an error message if you have not specified the input attributes, at least one input file, the output file, and at least one key. Errors will also occur in cases of inconsistent specifications, such as a key position larger than the input record length. After such an error, the previously entered commands are still in effect. Enter the necessary commands to correct the error condition, then type GO again.

We have now described enough commands to permit using SuperSort's basic functions of sorting, merging, and extracting records by record number. Here are some examples of complete sets of SuperSort commands:

```
     INPUT-ATTRIBUTES = 62, CR-DELIMITED
     SORT-FILE = SAMPLE.DAT
     OUTPUT-FILE = BYACCOUN.DAT
     KEY = #9, 4 ASCENDING RIGHT-JUSTIFY
     GO

     INPUT = 80, FIXED
     SORT MAILLIST.NEW(1,2500)
     MERGE MAILLIST.MAS
     KEY = 72,80 ASC, 50,71 ASC, 1,49 ASC UPPER
     OUTPUT = MAILLIST.MAS/C
     GO

     INPUT = 1200, CR-DEL
     MERGE PART.001, PART.002, PART.003
     KEY = # 7, 16, NUMERIC-ASCII, DESC,  #5, 5, UPPER RIGHT ASCE
     OUTPUT = PART123.DAT, 1024, FIXED
     GO

     I=62,CR-D; S=SAMPLE.DAT; O=OUTPUT.DAT; K=#4,6,DE; G
```

----------------*----------------

At this point we recommend you now do a few sorts and merges, either on SAMPLE.DAT or your own data. After thus completing your familiarization with SuperSort's basic functions, continue reading to obtain familiarity with record selection and other additional features.

----------------*----------------

11. USING COMMAND FILES

Some or all of the commands needed to do a particular sort/merge may be stored in a command file, making it unnecessary to enter them each time the particular procedure is to be repeated.

Commands in a file have exactly the same form as those entered from the console. The command file may be created with your text editor. This is easiest if your text editor is MicroPro's WordMaster. Enter exactly what you would type to SuperSort. Include a GO command in the file if execution is to commence without further console input.

Invoke use of the command file with the command:

```
               CFILE = filename
```

```
For example:   CFILE = SORTSAMP.COM
               CF B:CMDFIL.TXT
```

A command file need not contain the entire set of commands. For example, it might only contain a commonly-used but long and tedious-to-enter KEY command.

Other commands may be entered before the CFILE command, and after it completes if no GO was present in the file.

To make the commands from the file print out on the console, include the command

```
               LIST
```

in the file, or enter it from the console before invoking the file. LIST causes command file lines (starting with the one AFTER the one containing LIST) to echo on the console.

12. SPECIFYING THE WORK FILE DRIVE

For all sorts except those involving very little data, SuperSort makes use of a WORK FILE on disk. By default, this file is put on the current drive; the user may optionally specify the drive with the command:

```
               WORK-DRIVE = drive
```

where drive is a letter, A thru Z, corresponding to a drive present in the system, optionally followed by a colon.

```
For Example:   WORK-DRIVE = B:
               WORK C
```

When a large amount of data is being sorted, the user should plan sufficient space for the work file on one of the disk drives, and specify that drive in a work-drive command. See "Utilization Hints" for further discussion.

When the output diskette change option is invoked the work file must be on a different drive than the output file.

NOTE

> The work file is called SORT.$$$ and is deleted at the beginning of execution and upon successful completion.

13. USING TAGSORT

The command

```
TAGSORT
```

causes SORT to use a different internal technique to produce a sorted output file. The effect is to reduce the work file disk space usage without effecting the output of the run.

TAGSORT is allowed only where there is one sort input file and no merge input files.

See "Concepts and Facilities" and "Utilization Hints" for further discussion.

14. SPECIFYING DEGREE OF MESSAGE PRINTOUT

As described in the "Concepts and Facilities" section, SuperSort has six message levels, called PRINT LEVELs Ø through 5, with the default being 2. The operator may select a print level with the command

```
PRINT-LEVEL = n
```

where n is a number from Ø to 5.

Examples:

```
PRINT-LEVEL = 3
PR Ø
```

See the "Execution Messages" section below for examples of the printout at each level, and detailed specifications of the printout.

15. MISCELLANEOUS COMMANDS

Following are descriptions of those of the remaining commands that admit of brief description:

CANCEL deletes all previously entered commands, allowing you to start over. Exception: does not cancel COLLATING-SEQUENCE commands (described later).

BYE exit SORT; equivalent to Control-C

NO-ERROR-MESSAGES

if an error is detected during SORT execution, print only the error number, not the full message. The error number may be looked up in the "Warning and Error Messages" section, or the ERROR-MESSAGE command may be used. The RAM space occupied by message texts is made available as working storage, speeding execution and reducing work file space usage, particularly in small systems. May be abbreviated to NO-E.

ERROR-MESSAGE number

prints the full message for the specified error number. May be used to inquire about an error after the NO-ERROR-MESSAGES command was used.

RETURN-TO-CONSOLE

after execution, accept commands for another sort/merge operation instead of exiting to the operating system. This command reduces the amount of RAM available for working storage; it is thus not recommended for large sorts in small systems, or when the fastest execution is desired.

CHANGE

allows MP/M users to change disks immediately after SORT is invoked. Has no affect on CP/M users.

16. INTRODUCTION TO RECORD SELECTION

SELECT and EXCLUDE define record extraction criteria on the basis of data tests. Records meeting SELECT test criteria are included in the sort/merge process. Conversely, EXCLUDE does the opposite: only records which fail its test criteria are included; all others are bypassed. SELECT and EXCLUDE operate on both sort and merge-only inputs.

The SELECT and EXCLUDE commands have a comprehensive syntax allowing many options and variations; the most commonly used form is:

```
(SELECT)  (=) ( FIELD start, end ) operator "text" { NUMERIC-ASCII   }
(EXCLUDE) (+) ([FIELD] # field no)                 { UPPER-CASE      }
                                                   { RIGHT-JUSTIFY   }
                                                   { LOHI            }
                                                   { MASK-PARITY-BIT }
                                                   { EBCDIC          }
                                                   { ALTSEQ          }
```

=	means replace previously entered SELECT/ EXCLUDE commands
+	add this command to previously entered SELECT/ EXCLUDE commands
FIELD	start, end specifies a positional field, by start column and end column, in a manner similar to the KEY command.
FIELD # field no	specifies a comma-delimited variable length field to be tested. Note that, unlike in the KEY command, the maximum length is not specified: the entire field is always used.
operator	is one of the following comparison operators, to specify the test to be performed:

<	or LT	less than
<=	or LE	less or equal
=	or EQ	equal
<>	or NE	not equal
>=	or GE	greater or equal
>	or GT	greater than

"text"	is the text to test the field against, enclosed in quotes. DO NOT OMIT THE QUOTES when testing a field against a text. (The quotes are omitted for numeric constants, a less used form described below.) The quoted text may be as long as desired.
NUMERIC-ASCII UPPER-CASE RIGHT-JUSTIFY LOHI MASK-PARITY-BIT EBCDIC ALTSEQ	These have meanings analogous to those described above for the KEY command. They are applied to BOTH the field of the record and the "text", before the comparison test is performed. (Additional test attributes are also allowed; their application to record selection is discussed in a later section).

The following examples of SELECT relate to the file SAMPLE.DAT (supplied with SuperSort), using the field definitions given for this file in the "Getting Started" section.

```
SELECT = FIELD # 4 >= "780701"
```

choose only those records with dates on or after July 1, 1978. (Recall that field 4 is the date in the form YYMMDD). Note the quotes; they are necessary since the data being tested consists of characters.

```
SELECT = FIELD # 1 = "ABC"
```

choose only the records for firm ABC.

```
SE #1="ABC"
```

the same, abbreviated. Note that when # is present, the word FIELD may be omitted.

```
SELECT = FIELD # 8 > "3000" NUMERIC-ASCII
```

select records for transactions over $3000.00 (positive only). NUMERIC-ASCII indicates testing the numeric value of text.

```
SELECT = FIELD #4 >= "780701"
SELECT + FIELD #4 < "780715"
SELECT + FIELD #1 = "ABC"
SELECT + FIELD #9 >= "1500" RIGHT-JUSTIFY
SELECT + FIELD #8 > "3000" NUMERIC-ASCII
```

Together, the above five commands cause the output to contain only the records for transactions in the first 15 days of July for firm ABC with general ledger accounts of 1500 or higher and amounts over $3000.00.

The RIGHT-JUSTIFY in the fourth line is to make fields containing fewer digits test low, despite the lack of leading zeroes in the file. Specifying NUMERIC-ASCII would achieve the same result.

When multiple SELECT and/or EXCLUDE commands are given, the records included are those that PASS ALL of the SELECT conditions AND FAIL ALL of the EXCLUDE conditions.

The following examples of SELECT relate to a hypothetical file containing names and addresses in positional fields.

```
SELECT = FIELD 75, 79 < "10000"
```

If the ZIP code is stored right-adjusted in columns 75 through 79 inclusive, this chooses addresses in New England and New Jersey.

```
SELECT = FIELD 1,1 >= "M"
```

If the name begins in column 1, this chooses names beginning with M through Z.

```
SE FI 1,1="M"
```

Minimum abbreviation of preceding.

17. RANGE TESTING IN RECORD SELECTION

Frequently, it is desired to test for a field value within, or outside of, a given range. SELECT/EXCLUDE range testing can be specified with "between" and "not-between" operators in the following form:

```
(SELECT ) (=) field as above (BT) "text1", "text2", attributes
(EXCLUDE) (+)                (NB)                     as above
```

BT	means BeTween "text1" and "text2" inclusive
NB	means Not Between "text1" and "text2"
"text1"	is the lower limit of the range
"text2"	is the upper limit of the range

Examples of the between/not between form of SELECT:

```
SELECT #1 NB "-3ØØØ", "3ØØØ" NUMERIC-ASCII
```

choose large postive or negative transactions from SAMPLE.DAT: over $3ØØØ, or more negative than -$3ØØØ.ØØ.

```
SELECT = FI 75,79 BT "1ØØØØ", "19999"
```

choose ZIP codes in New York and Pennsylvania.

18. MORE RECORD SELECTION FEATURES

The previous section described the commonly-used tip of the SELECT/ EXCLUDE iceberg; now we will introduce more features. Those reading this manual for the first time may wish to skip ahead two sections.

Interchangabilty of Fields and Constant Values

As described previously, the field specification was always shown on the left of the comparison operator, and the constant to test against on the right. However, fields and constants can be swapped and substituted as desired. You may put the constant before and the field after the operator, and you may test a field against another field or fields in the same record. Examples:

```
SELECT = "ABC" <= FIELD #1
SELECT = FIELD #1 > FIELD #2
SELECT = "1234" BT FIELD #1, FIELD #2
```

The latter tests for field 1 less than or equal to "1234" and field 2 greater than or equal to "1234".

Logical Operators

Complex conditions can be formed by combining tests with AND, OR, and XOR (exclusive OR):

```
SELECT = #9 > "3000" NUM OR #1 = "ABC"
```

All transactions from SAMPLE.DAT over +$3000.00, and also every transaction for firm "ABC".

Note that all attributes (such as NUM for NUMERIC-ASCII in the above) must follow the comparison test they apply to, and be repeated for all tests they apply to.

An individual test may be negated by preceding it with NOT:

```
SELECT = #9 GT "3000" NUM AND NOT #1 = "ABC"
```

transactions over +$3000.00 for firms other than ABC

AND, OR, XOR, and NOT may not be abbreviated.

Parentheses

Parentheses may be used to group logical operations. In the absence of parentheses, the order of operations is: all comparisons first, then NOT, then AND, then XOR and OR. Example:

```
SELECT = #2="R" AND ( #9 >="1500" RIGHT OR #1="ABC" )
```

19. RECORD SELECTION WITH NON-ASCII DATA

This section describes the field test attributes and the numeric constant features which make it possible to do record selection on BCD and various types of binary data.

Additional Test Attributes

The following additional test attributes may be used in record selection tests. Their meanings are as described above in the "Key Specification" and "Concepts and Facilities" sections.

```
TWOS-COMPLEMENT   or   COMPUTATIONAL
INTEGER
FLOATING-POINT
PACKED-BCD       or    COMPUTATIONAL-3
```

Numeric constants, to compare binary and BCD data to, will be described shortly; first, we will give some examples using additional field test attributes to compare fields to each other.

```
SELECT = FIELD 1,2 > FIELD 3,4 INTEGER
```

Select records in which bytes 1 and 2 are greater than bytes 3 and 4 when interpreted as FORTRAN or MBASIC binary INTEGERs (twos complement fixed point binary, stored low order first).

```
SELECT = FIELD 1,2 > FIELD 3,4 LOHI
```

As above, except data is unsigned.

```
SELECT = FIELD 1,8 BT FIELD 9,16, FIELD 17,24 FLOATING
```

Select records in which the first field in the record has a value between (inclusive) the second and third fields. The fields are treated as MicroSoft FORTRAN or MBASIC binary DOUBLE PRECISION data. Note the use of the FLOATING-POINT attribute and the field lengths of 8 bytes for DOUBLE PRECISION data.

```
SELECT = FIELD 1,1Ø < 21,3Ø  COMPUTATIONAL-3
```

Accept records in which positions 1-1Ø are less than positions 21-3Ø when compared as packed BCD data, as might be written by MicroSoft COBOL.

Numeric Constants

Values to test against (compare to) may be specified as numeric constants as well as quoted texts. The numeric constant represents the binary values stored inside the computer, the value being expressed in octal, decimal, hexadecimal, or BCD at the user's convenience; for quantities expressed in decimal, the user may specify high order first or low order first storage.

Numeric constants are generally appropriate when binary or BCD data is being tested; a quoted text is generally more convenient for comparison to ASCII data. (Numeric constants can be used with ASCII data, if the numeric values of the ASCII codes are entered.)

When using numeric constants, the user generally must be aware of how many bytes the constant occupies, to ensure its length matching the comparison field. This is because the blank fill generated by SuperSort when comparing values of unequal length will not normally provide meaningful results with binary values.

Thus, the syntax used to specify constants allows specification of the constant's length in bytes. If the significant digits require fewer bytes, high order bytes are added by SuperSort which are filled with all one bits for negative numbers (except for PACKED-BCD constants), and zero bits in all other cases. If no size is specified, the minimum that will hold the significant digits is used.

A numeric constant consists of an optional sign, any number of digits you can type on a line, an optional base indicator letter, and, if the base indicator letter is present, an optional decimal number representing the size of the constant in bytes. No spaces are accepted before or after the base indicator letter.

The base indicators available are:

T	decimal, converted to two's complement binary
I	decimal, converted to two's complement binary, stored low order first
P	decimal, converted to packed BCD
H	hexadecimal
Q	octal

If no base indicator is given, decimal (T base indicator) is assumed, except if one of the digits A-F is present, hexadecimal is assumed.

All numeric constants are stored with the high order byte first (at the lowest memory address) except those in which the I base indicator is used. I-base decimal constants must be used in comparisons where the field being tested is stored low order first, i.e. wherever the LOHI or INTEGER test attribute is pertinent. If a hexadecimal constant is used in such a test, rearrange the digits to put the low order byte first.

The I base indicator implies the LOHI attribute, since it is pertinent only to tests of data stored low order first. If the data is signed (and it usually is), TWOS-COMPLEMENT or INTEGER must be explicitly specified.

The P base indicator implies the PACKED-BCD (COMPUTATIONAL-3) test attribute.

CAUTION

Users of prior releases of SuperSort: the meaning of the I base indicator has been changed; in release 1.02 and preceding releases it meant what T means now.

Illustrative examples of numeric constants:

12	one byte containing 12 decimal (ØC hex)
12T	same
12T2	two bytes containing 12 decimal, stored high order first (ØØØC hex)
12I2	two bytes containing 12 decimal, stored low order first (ØCØØ hex)
12H	one byte containing 12 hex (18 decimal)
12P	one byte containing 12 in unsigned packed decimal (12 hex)
1234	decimal 1234 in two bytes (the minimum it will fit), stored high order first.
1234T4	decimal 1234 extended to 4 bytes with sign (Ø for positive) bytes, stored high order first.
-2	decimal -2, stored in 1 byte (FE hex)
-2I4	decimal -2 extended to 4 bytes with sign bit bytes (all 1 bits for negative), stored low order first (FEFFFFFF hex)
1234P	1234 with no sign stored as packed decimal. two bytes, 1234 hex.
+1234P	1234 with positive sign in packed decimal. three bytes, Ø1234C hex.
-1234P4	1234 with negative sign, packed decimal, extended to 4 bytes with leading zero byte: ØØ01234D hex.
1Ø29AH	three bytes containing, in order, Ø1 hex, Ø2 hex, and 9A hex.
1Ø29A	same: default hex since A present
ØØØ1H	1, stored in ONE byte. Leading zeroes do NOT reserve bytes.
1H2	correct way to get two bytes containing ØØØ1 hex, high order first
1ØØH2	two bytes containing 1, stored low order first: when using hex constants, rearrange the digits if low order first storage is desired.

Examples of SELECT commands using numeric constants:

```
SELECT = FIELD 23,23 = FFH
```

test for FF hex in position 23.

```
SELECT = FIELD 1,10 < 1234P10  COMPUTATIONAL-3
```

Accept records in which positions 1-10, interpreted as packed decimal, contain a value less than 1234. Note explicit sizing of constant to 10 bytes to match length of field.

```
SELECT = FIELD 1,10 < +1234P10
```

Same as preceding, since the presence of the P in the constant implies testing as packed decimal; the presence of the + sign is does not effect the outcome of the comparison.

```
SELECT = FIELD 1,2 > 123I2
```

test for 2-byte low order first value greater than 123 decimal in bytes 1 and 2 of record. Note that LOHI is implied by use of the I base indicator. Comparison is UNSIGNED, that is, data in record with high order byte on is considered large positive data, rather than negative.

```
SELECT = FIELD 1,2 > 123I2 INTEGER
```

similar, but data is assumed to be signed.

```
SELECT = FIELD 1,2 > 123T2 COMPUTATIONAL
```

here the data is assumed to be high order first and signed. Note use of the T base indicator.

```
SELECT = 11,14 <= 00002084H4 FLOATING-POINT
```

select records in which the FORTRAN REAL or MBASIC SINGLE PRECISION value in positions 11 through 14 has a value less than 10.0. Note the explicit 4-byte size specification; the leading zeroes in the constant were entered only for readability. Since SuperSort has no provision for entering floating point constants in decimal, the hexadecimal value was obtained from a FORTRAN compilation listing; it could also be obtained by DUMPing a FORTRAN unformatted or MBASIC random file.

Constant Lists

In addition to fields and single constants, another form of value is accepted: the constant list. A CONSTANT LIST is a list of acceptable constants, separated by commas or blanks, and enclosed in BRACKETS. The constants in the list are evaluated individually, then concatenated together. They need not be of the same type.

For instance, you might want to express part of the value as quoted text (for printable ASCII characters), and part as a numeric constant (for control characters or special codes):

```
SELECT = FIELD 1,6 = ["ABCD",0D0AH]
```

The preceding example tests for ABCD, followed by the control characters carriage return and line feed. (This would only work in a fixed length or VARIABLE record, as the latter two characters are delimiters for CR-DELIMITED records.)

Or, you might want to express several byte values in octal, with three digits for each byte:

```
SELECT = FIELD 1,3 = [ 101Q, 102Q, 103Q ]
```

If the digits were all run together, 101102103Q, SuperSort would pack each digit into three bits, bridging byte boundaries.

Or, you might want to enter a constant too big to conveniently type on one line:

```
SELECT = FIELD #1 = [ "THIS IS A VERY LONG ", &
"TEST TEXT THAT RAMBLES ON AND ON, AND KEEPS", &
" GOING ON AND ON, AND ON, TILL THE END" ]
```

Recall that the & is the SuperSort convention for continuing the same command on the next line. The &'s and carriage returns could be enclosed in the quotes, but this would store them in the string constant. Since the user desired to test for the text shown without any imbedded carriage returns, he used the constant list construction to break it into three constants.

The SELECT/EXCLUDE features that way have described can be used in an almost unlimited number of combinations to specify complex record selection conditions. For example, if you can think of a use for this, SuperSort will do it:

```
SELECT = #4 > "1"  AND  #7 BT "LLLL","ZZZZ" &
            OR  #5 BT "1234","9999"&
                 AND  (NOT FIELD 1,4 = 0P4 PACKED &
                      XOR  ( "ABC" BT #11, #12 &
                           OR  FIELD 5,8 BT [ 15Q, 12Q], &
                                 ["STOP HERE", DH ] ) )
```

20. SELECT/EXCLUDE SUMMARY

The SELECT and EXCLUDE commands allow records to be chosen from the input file on the basis of field values and value relationships between fields. The arguments to these commands are comparisons of field values and constants, optionally preceded by NOT and/or combined with the AND, OR, and XOR operators, and parentheses, if desired. If multiple commands are given (with + included in all but the first!), only records which are accepted by ALL of the SELECTs and rejected by NONE of the EXCLUDEs will be included in the sort/merge.

In addition to the usual comparison operators - less than, equal to, etc - operators for "between" (inclusive) and "not-between" are included. All comparisons operate on both ASCII and binary data of any length up to SuperSort's maximum record length of 4096 characters.

Two types of constants, quoted text string and numeric, are available. The numeric constants may be octal, hexadecimal, decimal (fully packed into binary), or BCD (decimal packed two digits to a byte, with optional trailing sign nibble), and may be any number of digits long. Further, you can specify the length of a numeric constant in bytes, and you can concatenate several constants (of the same or different types) into one value.

The "test attributes" available with the KEY command are also available with the SELECT and EXCLUDE commands. These permit right-justification, right-to-left comparison, use of alternate collating sequences, and masking of the high order bit. Additional attributes permit the data to be treated as BCD, fixed point binary (signed or unsigned, stored high or low order first), or floating point binary.

All of the SELECT/EXCLUDE features have been described individually in the preceding three sections; A formal syntax specification of the SELECT and EXCLUDE commands, and text expanding a few details, is given in Appendix 2.

21. SPECIFYING ALTERNATE COLLATING SEQUENCES

The COLLATING-SEQUENCE command is used to specify or modify the alternate collating sequence that is invoked with the ALTSEQ field test attribute in the KEY, SELECT, and EXCLUDE commands.

The base or default collating sequence which the COLLATING-SEQUENCE command modifies is normally the ASCII code value sequence, or the 8-bit binary byte value. Any characters for which no position is specified with the COLLATING-SEQUENCE command retain their base position. The base sequence can be altered by linking in a customized COLTAB.REL, as described in the "Programmer's Guide"; this custom table may then be operated on with the COLLATING-SEQUENCE command, if desired.

The general form of the COLLATING-SEQUENCE command is

```
          COLLATING-SEQUENCE (=) argument, argument,...
                             (+)
```

The = and + have the same effect.

COLLATING-SEQUENCE arguments can be of two types, which we will call implied position and explicit position. With implied position arguments, characters can be simply listed in the order in which they are to collate; with explicit position arguments, characters can be assigned to specific positions in the sequence.

Implied Position Arguments

An IMPLIED POSITION argument specifies a character or a range of characters, whose position in the collating sequence implicitly is immediately after that of the preceding argument, or position Ø (out of 256 possible positions) if it is the first argument in the command. The characters are commonly specified enclosed in quotes, but may also be specified as decimal, octal, or hexadecimal codes.

For example,

```
COL = " ", "A"-"Z", "Ø"-"9", 7FH
```

specifies blank as the first character of the collating sequence, then the upper case letters, then the digits, then the character whose hexadecimal code is 7F. The unspecified characters retain their original positions, thus in this example some of the control codes and punctuation characters have positions equal to the position of a letter, digit, blank, or 7FH.

In an implied position argument, if a range of characters is given (two character values separated by -), the characters in the range always occupy sequentially increasing positions in the sequence, starting from the first character. This is true even if the second character in the range is less than (has a smaller ASCII code than) the first character.

For example, the following would sort in ascending order on the upper case letters and in descending order on the lower case letters:

```
COL = "A"-"Z", "z"-"a"
```

Explicit Position Arguments

In an EXPLICIT POSITION argument, the position of the character or range of characters in the collating sequence is explicitly specified and may be all the same, sequentially increasing, or sequentially decreasing for the characters in the range.

To form an explicit position argument, follow the character or range with an =, then a numeric constant or quoted character to specify the first postion, followed by a + for increasing positions, a - for decreasing positions, or neither for all characters in the range to occupy the same position.

For example, the following makes the letters occupy positions 25 up, with upper and lower case being equal, and makes the digits occupy postions 5Ø down:

```
COL = "a"-"z"=25+, "A"-"Z"=25+, "Ø"-"9"=5Ø-
```

To clarify the effect of -, we will point out that the following are equivalent:

```
COL = "Ø"-"9"=5Ø-
```

and

```
COL = "9"-"Ø"=4Ø+
```

Which form you use is a matter of convenience only. The following, however, is quite different:

```
COL = "Ø"-"9"=5Ø
```

it makes all the digits have the same position, 5Ø.

Interaction of Arguments

If an implied position argument follows an explicit position argument, its starting position is the position occupied by the last character of the explicit argument, plus one, regardless of whether said preceding argument occupied increasing, decreasing, or constant positions.

Any argument will override any arguments to its left. Thus, all the unspecified characters can be gotten out of the way at the high end of the sequence by, for example,

```
COL Ø-255=255, " "=Ø, "A"-"Z", ".", "Ø"-"9"
```

which sets up a sequence of A thru Z, then period, then Ø thru 9, with everything else high. The first argument in the above example EXPLICITLY sets the position of all characters (Ø-255) to the last position (=255). The second argument explicitly sets space to the first position (Ø). The remaining three arguments IMPLICITLY set the positions of the letters, period, and digits, in that order, to sequential positions after space, since no ='s and position values follows them. That is, A's position is set to 1, B's, to 2, ... Z's to 26, period's to 27, etc.

Actually, since the position wraps around to Ø after 255, the following is equivalent the preceding:

```
COL Ø-255=255, " ", "A"-"Z", ".", "Ø"-"9"
```

Multiple COLLATING-SEQUENCE commands

Any number of COLLATING-SEQUENCE commands may be entered; each one modifies whatever the last one left; the implied position starts over at Ø at the beginning of each command.

The selection of = or + after the keyword COLLATING-SEQUENCE has no effect other than to clarify the meaning of the command to possible future readers of command files.

Reinitializing the collating sequence table

SuperSort NEVER automatically sets the collating sequence table back to its original values. The CANCEL command does NOT cancel COL commands. If the RETURN-TO-CONSOLE command is used, the COL commands remain in effect upon completion of the sort.

However, the user can restore the collating sequence table to its standard original state with the command:

```
COL Ø-255
```

Note on COLLATING-SEQUENCE arguments

The values for characters and positions can actually be ANY quoted string or numeric constant accepted by the SELECT and EXCLUDE commands that has one-byte value. Thus, leading + and - signs, the base indicators H, I, P, and Q, and a size in bytes of 1 byte, are accepted, as long as the resulting value occupies exactly one byte. This feature could could lead to confusing results if used indiscriminately.

22. INVOKING USER-EXIT ROUTINES

A user-exit routine is a subroutine installed by the user in SuperSort (SuperSort I only). The function of such routines is discussed in "Concepts and Facilities"; their coding and installation is described in the "Programmer's Guide".

User-exit routines installed in the SuperSort program are dormant unless invoked with the command:

```
USE-USER-EXIT [1] [2]
```

where 1 means to use the "XIT1" routine, and 2 means to use the "XIT2" routine.

Examples:

```
USE-USER-EXIT 1,2

USE 2

USE 1

US 1, 2
```

These commands have no effect unless user-exit routines have been installed (linked into) your copy of SORT.COM.

D. UTILIZATION HINTS

This section includes notes on applying SuperSort to data created by programs written in several specific languages, descriptions of a number of non-sorting utility uses of SuperSort, and considerations relating to sorting large files.

1. CREATING SORT PROCEDURE COMMAND LIBRARIES

Two tools are available to eliminate the need, and the possibility of error, of entering SuperSort commands every time a regularly used sort/merge procedure is to be executed.

SUBMIT

The operating system's batch-processing facility, SUBMIT (see your CP/M system documentation), may be used to invoke a sort if all of the necessary SuperSort commands can be fit onto one system command line -- which is often possible by using the shortest abbreviations and separating commands with semicolons. For example, a line in the submit file might be:

```
SORT I=80 CR;S=NEW.DAT;M=OLD.DAT;O=NX.DAT;K=#1,10; GO
```

Command Files

SuperSort's command file facility and its usage is described in the "Operating the SuperSort Program" section.

Both Together

Using a SuperSort command file, a sort/merge requiring any number of commands may be invoked from a submit file. For example, if the file SORTRANS.COM contains the desired SuperSort commands, the line

```
SORT CFILE=SORTRANS.COM
```

could be used in a submit file. The submit file could also specify additional sorts or other operations, all of which would take place without operator intervention when the submit file was invoked.

2. THE ONE-INPUT MERGE -- THE NON-SORT

Whenever you wish to apply some SuperSort facility to a data file without reordering the records, use a ONE-INPUT MERGE: specify the desired file as a merge-only input file, and specify no other input files. You can thus select records, apply user exists, perform file type conversions, utilize output options, etcetera, etcetera, while keeping the records in the original order.

When doing a one-input merge, it is necessary to specify a key in order to prevent the "no key specified" error message, but the key has no effect (unless K-OUTPUT is used).

NOTE

> If the input is a COBOL RELATIVE file containing deleted or unwritten records, the "holes" will be lost. The output records will be in the input order, but the record numbers will not be the same.

3. NON-SORTING USES

In addition to sorting and merging, SuperSort's features permit it to perform many utility functions. This section suggests a number of such uses. All can be applied without reordering the data, via a one-input merge.

Converting a File to a Different Type

Specify the present type -- FIXED-LENGTH, CR-DELIMITED, VARIABLE, or RELATIVE -- in the INPUT-ATTRIBUTES command, and the desired type in the OUTPUT-FILES command.

For example, if you wish to print a FIXED, VARIABLE, or RELATIVE file containing ASCII data, convert it to CR-DELIMITED, then copy the result to the printer.

Type conversions are also useful to make files written by one language readable in another language.

Changing Record Length (FIXED and RELATIVE files)

Specify the present record length in the INPUT-ATTRIBUTES command, and the desired record length in the OUTPUT-FILE command. SuperSort will truncate or blank fill the records as required.

Truncating Long Records (CR-DELIMITED and VARIABLE files)

Specify the present maximum record length (or longer) in the INPUT-ATTRIBUTES command; specify the desired maximum in the OUTPUT-FILE command. Records longer than the specified output record length will be truncated to that length, with the data in columns beyond the specified output length being discarded.

Selecting and Rearranging Positional Fields within the Record

Specify the desired positional fields, in the desired order, as keys. In the OUTPUT-FILE command, specify K-OUTPUT, and specify FIXED, CR-DELIMITED, VARIABLE, or RELATIVE as desired. The result will be a file containing only the specified fields, in the specified order. Use no test attributes -- sort in a separate run if necessary -- or read about K-OUTPUT in the "Programmer's Guide", then use KEY test attributes with caution.

Converting Comma-Delimited Fields to Positional Fields

Specify the desired comma-delimited fields (#-fields), in the desired order, with the desired lengths, as keys. Specify K-OUTPUT. The output file will contain the specified fields in the specified order, stripped of quotes, and blank-filled to the specified lengths. RIGHT-JUSTIFY may be used to obtain leading instead of trailing blank fill on individual fields.

Such a conversion might be used to make a file generated by a BASIC program readable by a FORTRAN or COBOL program, or to format an ASCII file for printout.

Converting Data to a Different Character Set

To convert data from one set of codes to another, set up the desired conversions with the COLLATING-SEQUENCE command, specify the entire record as a key with the ALTSEQ attribute, and specify K-OUTPUT. The output file will receive records in which the specified byte substitutions have been performed. Specific conversions can also be performed more simply by using the EBCDIC, UPPER-CASE, and/or MASK-PARITY-BIT attributes, without using COLLATING-SEQUENCE. A side effect of this conversion technique is that CR-DELIMITED and VARIABLE records will be extended to the specified key length with blanks.

Inserting Blank Spaces Between Fields

To format an ASCII file attractively for printout, you may wish to insert blanks between the fields. This may be done using SuperSort with a two-step process.

First, sort or one-input-merge the file, specifying a FIXED-LENGTH output file with records several bytes longer than the input records. The additional record length will be filled with blanks.

Second, select and arrange the fields, using K-OUTPUT as just described. Wherever blanks are desired, specify a key of the desired length from the blank fill at the end of the extended records. Specify this second output file as CR-DELIMITED, so that it will be ready to copy to the printer.

For example, suppose you wish to print a report based on file SAMPLE.DAT showing only the firm number, right-adjusted in columns 1-4, and the transaction date, in columns 9-14. You do not want to print the other fields, nor the commas and quotes, and you only want to show transactions over +$3000.00. Proceed as follows:

First, select the desired records and convert to overlong blank-filled FIXED records. You could also sort, if desired, in this run:

```
A>SORT
INPUT-ATTRIBUTES = 62 CR-DELIMITED
MERGE-FILE = SAMPLE.DAT
OUTPUT-FILE = TEMP.DAT, 100 FIXED-LENGTH
KEY = 1,1
SELECT = #8 > "3000" NUMERIC-ASCII
GO
```

Second, create the print file:

```
A>SORT
INPUT-ATTRIBUTES = 100 FIXED-LENGTH
MERGE-FILE = TEMP.DAT
OUTPUT-FILE = TEMP.PRN, CR-DELIMITED, K-OUTPUT
KEY = #1,4,RIGHT-JUSTIFY,   91,94,   #4,6
GO
```

The 91,94 key in the above gets 4 blanks from the end of the record, to fill columns 5 through 8.

Then copy file TEMP.PRN to the printer.

Garbage Collecting a COBOL RELATIVE file

Sort or merge the file. All deleted records, and all record numbers for which records were never written, will be removed; the output records will be written contiguously from record number 1.

Obtaining a list of Existing Record Numbers in a RELATIVE file

Specify CR-DELIMITED R-OUTPUT. The output file will be a printable list of record numbers. Using KR-OUTPUT, you could show selected fields, or the entire records, along with the record numbers.

4. NOTES ON USE WITH MicroSoft MBASIC

MBASIC Sequential Files

MBASIC Sequential files are CR-DELIMITED and contain ASCII data.

Commas between fields and quotes around string data are not automatically written. We suggest writing data variable length, and explicitly comma-delimiting the fields. The following example outputs the variables I and J with a comma between them:

```
150 PRINT #1, I; ","; J
```

String data which can contain commas, or significant leading or trailing blanks, should be explicitly quoted. The following example outputs A$ and B$, enclosed in quotes (produced by CHR$(34)), with a comma between them:

```
200 PRINT #1, CHR$(34); A$; CHR$(34); ","; CHR$(34); B$; CHR$(34
```

Alternately, you could produce positional fields with PRINT USING, with the TAB function, or (with caution) by using commas to separate the variable names in the PRINT statement.

All numeric data in MBASIC sequential files will be correctly interpreted by the NUMERIC-ASCII attribute, subject to the 14-digit accuracy limitation for double precision values.

MBASIC Random Files

MBASIC random files (the kind processed with PUT and GET) contain no delimiters unless explicitly output by the program. Normally, they should be processed in SuperSort as FIXED-LENGTH records with positional fields.

Numeric data in random files is normally binary integer or floating point. Since such data can contain bytes with the value 1A hex, you should read the section "End of File Indication Considerations" in the "Programmer's Guide" before using MBASIC random files with SuperSort. Usually, using a logical record length of 128 or more and using the NO-ZZZ input attribute will solve the problem.

MBASIC Data Types - Random Files

MBASIC Data Type	Field Length	Use SuperSort Test Attribute
INTEGER	2	INTEGER
SINGLE PRECISION	4	FLOATING-POINT
DOUBLE PRECISION	8	FLOATING-POINT
STRING	as written	none required

5. NOTES ON USE WITH BASIC-E AND CBASIC

File Types

BASIC-E and Software System's CBASIC "sequential" files are CR-DELIMITED; the "random" files of these BASICs have records of constant length, terminated with carriage returns, and may be processed as either CR-DELIMITED or FIXED-LENGTH.

Data in Files

BASIC-E and CBASIC data in files is written in ASCII in comma-delimited fields, with strings enclosed in quotes, when PRINT USING is not used. The NUMERIC-ASCII attribute will correctly interpret numeric data.

6. NOTES ON USE WITH MicroSoft FORTRAN

FORTRAN File Types

FORMATTED files are CR-DELIMITED.

UNFORMATTED files are usually FIXED-LENGTH, with the record length 128 or a multiple thereof.

FORTRAN Data Types - FORMATTED Files

NUMERIC-ASCII can be used for all numeric formats. For DOUBLE PRECISION data, note that it disregards significant digits in excess of 14.

Positive numbers written in positional fields with I and F formats will be correctly interpreted with no attributes, with no limitation on the number of signicant digits.

FORTRAN Data Types - UNFORMATTED Files

FORTRAN Data Type	Field Length	Use SuperSort Test Attribute
LOGICAL	1	INTEGER
INTEGER	2	INTEGER
REAL	4	FLOATING-POINT
DOUBLE PRECISION	8	FLOATING-POINT

7. NOTES ON USE WITH MicroSoft COBOL

COBOL File Types

COBOL File Type	SuperSort File Type
Line Sequential	CR-DELIMITED
Fixed Length Sequential	FIXED-LENGTH
Variable Length Sequential	VARIABLE
Relative	RELATIVE
Indexed	---

COBOL Data Types

COBOL Data Type (USAGE)	Field Length	Use SuperSort Test Attribute
DISPLAY	# characters	none required
COMPUTATIONAL	2	COMPUTATIONAL
COMPUTATIONAL-3	one-half # digits, plus 1	COMPUTATIONAL-3 or PACKED-BCD

When entering commands, remember that COMPUTATIONAL-3 should be abbreviated only as "COMP-3" or "C3"; other keywords may be abbreviated with the first two or more characters.

8. CONSIDERATIONS RELATING TO LARGE SORTS

The amount of data that can be sorted and/or merged is limited by diskette space: The output file must fit entirely on one diskette. The work file must also fit entirely on one diskette; however, in most cases it is no bigger than the output file, or can be manipulated (e.g. by using tagsort) to be no bigger than the output file.

Merging generally uses no work file space; when sorting large amounts of data, the user will want to plan adequate diskette space for the work file.

In a system with three or more drives, reserving one drive for the output file and another for the work file generally allows sorting the maximum amount of data.

In a two-drive system, use one of the following approaches to sorting large amounts of data:

a. for sorting one input file only:
   put input file on one drive
   put output file on other drive
   put work file on either drive
   use tagsort

b. for any number of sort input files:
   reserve one drive for work file
   put output file on same drive as input,
      with diskette change option
   tagsort cannot be used

Method (b) usually can sort the largest amount of data; but if it yields a "disk full" error for the work file (<output filename>.$$$), try method (a). Also, read the next section.

Additional hint: it is not necessary to have the file SORT.COM on-line while sorting. Change diskettes after invoking SORT (but before the GO command!) in order to make the diskette space SORT.COM occupies available for data.

9. MINIMIZING WORK FILE DISK SPACE USAGE

a. Use tagsort for single-file sorts

b. Use maximum available RAM working storage:

   with the SuperSort program:

      Relocate your operating system to use all available RAM
         ("CPM" command; see system documentation)
      Do not use the RETURN-TO-CONSOLE command
      Use the NO-ERROR-MESSAGES command

   with the SuperSort subroutine:

      Supply largest possible working storage area

   with either:

      Use NOCOL, NOREPORT, NOSEL, and/or NOERR load options as
         described in the "Programmer's Guide".

c. Maximize effectiveness of working storage use:

   Do not greatly over-specify the maximum length of CR-DELIMITED records, VARIABLE records, and comma-delimited fields

Additional measures to try if work file overflows disk although total data is less than a diskette's capacity and the preceding measures are not adequate:

d. if you are using record selection or user-exits, try not using them. Instead, do a separate merge-only run to apply these features.

e. if you have a long key or several keys: sort first on first few characters of highest priority key only (short keys make better use of RAM working storage), then sort the result on all keys (partially ordered data requires less work file space).

f. if you have little RAM and long records: divide the data into two or more files, sort each in a separate run, then merge the results together. Possible ways to divide the data include using record selection in SuperSort, and using a text editor.

Another hint: If your data is already mostly in order, with out of order records not far after their correct position, and you have no merge only files, put the work file on the output disk if other considerations permit. In some cases, if the input file and output file attributes are the same, SuperSort will be able to rename the work file to be the output file, skipping the merge phase and using no work file space concurrently with output file space.

10. MAXIMIZING SORTING SPEED

Do everything specified in the previous section, except do not use tagsort unless necessary to prevent work file disk overflow.

E. EXECUTION MESSAGES

This section details SuperSort's normal execution messages (as opposed to error and warning messages). As explained in the "Concepts and Facilities" section, all execution messages are controlled by the PRINT-LEVEL setting. The print-level is set by command in the SuperSort program, and by parameter in the subroutine. In the program, it defaults to 2 if not specified. Many of the printable items also print only if non-Ø, or only if different from another quantity printed.

A brief list of the printable quantities was given in "Concepts and Facilities"; here we will show a number of examples, then tabulate detail about each item, keyed to the examples.

For our first example, we will do a simple sort, that does not produce any of the conditionally printing items, at print-levels Ø through 5. The parenthesised numbers in the left margin are references to a table below. The operator's typing is on lines beginning with * and A>; the rest is computer output. Note the use of command abbreviation, multiple commands on one line, the PRINT-LEVEL command, and the RETURN-TO-CONSOLE command.

```
          A>SORT

          MicroPro SuperSort ready for command input
          * PRI Ø; I 4Ø CR-D; SORT I; OUT O; KEY 1,4; RE; GO

          MicroPro SuperSort release 1.6Ø
          * PRI 1; I 4Ø CR-D; SORT I ; OUT O; KEY 1,4; RE; GO
(9)       639 RECORDS SORTED
(21)      WORK FILE DISK SPACE USAGE 8K
(22)      *** SORT/MERGE COMPLETE ***

          MicroPro SuperSort release 1.6Ø
          * PRI 2; I 4Ø CR-D; SORT I; OUT O; KEY 1,4; RE; GO

          639 RECORDS SORTED
(18)      OUTPUT FILE SIZE 8K
          WORK FILE DISK SPACE USAGE 8K
          *** SORT/MERGE COMPLETE ***

          MicroPro SuperSort release 1.6Ø
          * PRI 3; I 4Ø CR-D; SORT I; OUT O; KEY 1,4; RE; GO

          639 RECORDS SORTED
          OUTPUT FILE SIZE 8K
(19-21)   2 SORT RUNS 1 MERGE RUN WORK FILE DISK SPACE USAGE 8K
          *** SORT/MERGE COMPLETE ***
```

First example, continued...

```
           MicroPro SuperSort release 1.60
           * PRI 4; I 40 CR-D; SORT I; OUT O; KEY 1,4; RE; GO
(2)        SORTING...
(2)        MERGING...
           639 RECORDS SORTED
           OUTPUT FILE SIZE 8K
           2 SORT RUNS    1 MERGE RUN    WORK FILE DISK SPACE USAGE 8K
           *** SORT/MERGE COMPLETE ***


           MicroPro SuperSort release 1.60
           * PRI 5; I 40 CR-D; SORT I; OUT O; KEY 1,4; GO
(1)        24248 BYTES WORKING STORAGE
                SORTING...
(3)                  468+170 1+0
                MERGING...
(4)                  2
           639 RECORDS SORTED
           OUTPUT FILE SIZE 8K
           2 SORT RUNS    1 MERGE RUN    WORK FILE DISK SPACE USAGE 8K
           *** SORT/MERGE COMPLETE ***

           A>
```

For our second example, we show runs with user-installed user-exit routines invoked. At level 1, the number of records output prints because it is different from the number of records sorted; at level 2, the numbers of insertions and deletions that account for the difference are also printed.

```
           A>SORT
           MicroPro SuperSort release 1.60
           * PRI 1; USE 1,2; I 40 CR-D; SORT I; OUT O; KEY 1,4; RE; GO
           488 RECORDS SORTED
(17)       328 RECORDS OUTPUT
           WORK FILE DISK SPACE USAGE 8K
           *** SORT/MERGE COMPLETE ***


           MicroPro SuperSort release 1.60
           * PRI 2; USE 1,2; I 40 CR-D; SORT I; OUT O; KEY 1,4; GO

           639 RECORDS INPUT FOR SORT
(7,9)           213 SORT INPUT DELETIONS      62 SORT INPUT INSERTIONS
           488 RECORDS SORTED
(15,16)         162 OUTPUT DELETIONS     2 OUTPUT INSERTIONS
(17,18)    328 RECORDS OUTPUT OUTPUT FILE SIZE 6K
           WORK FILE DISK SPACE USAGE 8K
           *** SORT/MERGE COMPLETE ***

           A>
```

This one is a good example of the level 4 and 5 printouts (items 1-4 by the reference numbers at the left):

```
          A>SORT

          MicroPro SuperSort release 1.60
          * PRI 6; I 2048 CR-D; SORT SORT.SYM; OUT SY; KEY 1,4; GO
(1)       38060 BYTES WORKING STORAGE
(2)            SORTING...
(3)                 10+9       10+40      10+86      10++19
                    9+27       9+26       9+14       9+18
                    9+5        9+65       9+45       9+43
                    9+54       9+26       9+23
(2)            MERGING...
(4)                 4
(2)            FINAL MERGE...
(4)                 12
(9)       639 RECORDS SORTED
(18)      OUTPUT FILE SIZE 8K
(19-21)   15 SORT RUNS   2 MERGE RUNS   WORK FILE DISK SPACE USAGE 12K
(22)      *** SORT/MERGE COMPLETE ***

          A>
```

Here's the whole works: both sort and merge-only inputs, and SELECT in use as well as XIT1 and 2. The reference numbers in the left margin are used in the table on the next page.

```
          A>SORT
          MicroPro SuperSort release 1.60
          * PRI=5; IN=70 CR-D; SORT=A; MERGE=O; OUT=OO 80; KEY=#1 30
          * USE 1,2; SEL FI 1,1 <> "A"; GO
(1)       38029 BYTES WORKING STORAGE
(2)            SORTING...
(3)                 11+0
(2)            MERGING...
(4)                 2
(5)       18 RECORDS INPUT FOR SORT
(6)            6 SORT RECORDS EXCLUDED OR NOT SELECTED
(7,8)          4 SORT INPUT DELETIONS    3 SORT INPUT INSERTIONS
(9)       11 RECORDS SORTED
(10)      328 RECORDS INPUT FOR MERGE ONLY
(11)           4 MERGE-ONLY RECORDS EXCLUDED OR NOT SELECTED
(12)           64 MERGE-ONLY INPUT DELETIONS
(14)      260 RECORDS MERGED ONLY
(15,16)        91 OUTPUT DELETIONS       8 OUTPUT INSERTIONS
(17,18)   188 RECORDS OUTPUT OUTPUT FILE SIZE 4K
(19-21)   1 SORT RUN      1 MERGE RUN    WORK FILE DISK SPACE USAGE 1K
(22)      *** SORT/MERGE COMPLETE ***

          A>
```

The following table shows exactly what's printed at what levels and under what conditions. The "ref" column is the number shown in the left margin in the preceding examples.

ref	description	printed only at level and above	printed only if different from
1	working storage size	5	
2	SORTING...,	4	
2	MERGING..., (indicates beginning of merging)	4	
2	FINAL MERGE..., (indicates beginning of last merge, to output file, prints only if not also the first merge)	4	
2	ADDITIONAL MERGE... (indicates beginning of second complete pass through data: tagsort suggested tagsort if this prints)	4	
3	number of records input to each sort run	5	
4	number of runs input to each merge run	5	

The following quantities are printed only if one or more sort input files are specified:

5	records read from sort input file(s) (see note on next page)	2	records sorted
6	records rejected by SELECT and/or EXCLUDE during sort	2	zero
7	records deleted by XIT1 during sort	2	zero
8	records inserted by XIT1 during sort	2	zero

NOTE

a record REPLACED by an XIT routine is counted as both a deletion and an insertion.

9	records sorted - number remaining after SELECT/EXCLUDE and XIT1	1	

Table, continued...

ref	description	printed only at level and above	printed only if different from
	The following quantities are printed only if one or more merge-only input files are specified:		
1Ø	records read from merge-only input files (see NOTE below)	2	records merged
11	records rejected by SELECT and/or EXCLUDE during merge-only input	2	zero
12	records deleted by XIT1 during merge-only input	2	zero
14	records merged only	1	
15	records deleted by XIT2	2	zero
16	records inserted by XIT2	2	zero
17	records output (always printed if both sort and merge-only inputs)	1	records sorted or merged
18	output file size, kilobytes	2	
19	number of sort runs	3	zero
2Ø	number of merge runs	3	zero
21	work file size	1	zero
22	SORT/MERGE COMPLETE	1	

NOTE

For RELATIVE files, the numbers of records input (refs 5 and 1Ø) count all deleted records and never-written record numbers less than the highest record number in the file.

The NOREPORT Load Option

As described in the "Programmer's Guide", the NOREPORT load option reduces SuperSort's memory requirements by eliminating much text. Following is an example run of a SuperSort main program loaded with the NOREPORT option:

```
        A>SORT
        MicroPro SuperSort release 1.6Ø
        * PRI=5; IN=7Ø CR-D; SORT=A; MERGE=O; OUT=OO 8Ø; KEY=#1 3Ø
        * USE 1,2; SEL FI 1,1 <> "A"; GO
(1)     38Ø29 BYTES WORKING STORAGE
(3)               11+Ø
(N1)    188 RECORDS
        A>
```

In the above, item (N1) is the number of OUTPUT records, which is printed at level 1 or higher. Items (1) and (3) are as described previously. Items (2), (4)-(16) and (18)-(22) of the previous description are never printed with the NOREPORT load option.

F. WARNING AND ERROR MESSAGES

This section describes the various warning and error messages SuperSort can emit. The full text of all messages is shown, supplemented by additional explanation. The errors are arranged in numerical order, for reference in cases where only the error number is known: after error which occured when the NO-ERROR-MESSAGES command was in use, and for error codes returned by calls to the SuperSort subroutine.

1. WARNING MESSAGES

There are several conditions which are sufficiently serious that the user should be informed, but not sufficiently serious that execution should be terminated. When one of these conditions arise in either the SuperSort program or subroutine, one of the following messages is unconditionally printed on the console. Execution then continues.

WARNING SW1: MARGINAL SORT WORKING STORAGE

The amount of memory the SuperSort main program is being run in, or the size of the RAM working storage area supplied by the caller to the SuperSort subroutine, is barely big enough to permit SORT to function with the given record lengths and options. Slow execution and overflow of the work file disk are likely after this message appears. Increase working storage size or shorten record lengths if feasible; with the sort main program, use the NO-ERROR-MESSAGES command.

WARNING SW5: INSUFFICIENT FIELDS OR COLUMNS FOR KEY OR SELECT IN ONE OR MORE RECORDS, BLANKS ASSUMED

A positional (column-specified) field extended beyond the end of a CR-DELIMITED record, or the number of comma-delimited fields in a record was smaller than a #-field number specified. The missing field or portion of field is assumed to contain blanks and execution continues. Message prints once, at end of execution, regardless of the number of occurences.

WARNING SW6: ONE OR MORE CR-DELIMITED INPUT RECORDS WERE LONGER THAN SPECIFIED MAXIMUM LENGTH, AND WERE DIVIDED INTO MULTIPLE RECORDS BEFORE SORT/MERGE

Occurs at end of execution if one or more CR-DELIMITED input records were too long. The excess length becomes another record; usually you will wish to correct the input file or re-run SORT with a longer input record length specified.

NOTE

> If there are NO delimiting characters in the entire file, this message appears and the treatment of the input is identical to that for FIXED-LENGTH records of the specified length.

2. ERROR MESSAGES

The various errors can be classified as follows according to when they occur and what happens when they occur:

Command Entry

When an error occurs while commands are being entered to the main program, SuperSort displays a "^" (caret, or up-arrow on some terminals) under the APPROXIMATE position of the error, prints the error messages, and prompts for another command. The corrected command may then be reentered. Any previously accepted commands remain in effect; no part of the erroneous command takes effect.

When a command entry error occurs on an input line containing multiple commands separated by semicolons, the command that caused the error, and any commands to its right, have not taken effect, but any commands to its left have been accepted and remain in effect.

When a command entry error occurs on a command read from a command file with "LIST" off, the command is printed on the console before the "^" and error message are printed.

When a command entry occurs in a command "line" that was entered on multiple lines (by use of the continuation character "&"), the "^" will be spaced down as well as over to indicate the position of the error.

At GO

These errors occur when the GO command is given to the SuperSort main program. The program returns to command entry and previously accepted commands remain in effect; the operator may correct the error or omission and enter GO again.

At Subroutine Initialization

These errors occur when the SuperSort subroutine is first invoked by a calling program. The error number is returned to the subroutine caller. At this time no files have been opened; the input is unaltered even if the output was to the same file.

During Execution

These errors apply to both the program and the subroutine. Execution is aborted; files in use are not closed; the work file <OUTPUT FILENAME>.$$$ may not have been erased. If the output was the same file as one of the sort inputs (generally not recommended), it may or may not have been altered.

The subroutine returns the error number to the caller.

The SuperSort program normally returns to the operating system after such an error, however, if the RETURN-TO-CONSOLE command is in effect, it will return to command input but all previously entered commands will have been cleared.

Internal Errors

These are errors which only occur when some one of SORT's internal mechanisms malfunctions. If you can find a set of conditions which reproduce one of these errors (other than invalid record select parameters in a SuperSort subroutine call), you have found a bug in SuperSort.

Many errors fit two or more of the above classes, particularly those that apply to both the main program and the subroutine. In the following error descriptions, the classification(s) of each error are indicated.

Following are the full texts of each message; note that the error number is included in the text. The full text is printed by the SuperSort program, except that errors during execution under the NO-ERROR-MESSAGES command print only "ERROR S" and the error number. Items such as file names, shown in lower case in the message, are replaced with actual values when the message is printed.

ERROR S28: TOO LITTLE MEMORY

Occurs when the SuperSort program is first started. The solution is to run the program in more RAM memory, with CP/M relocated appropriately. (The operating system message LOAD ERROR or TOO BIG may also occur, if the CP/M in use is relocated for too little RAM to accomodate SORT.COM.)

ERROR S29: NOT ENOUGH WORKING STORAGE

Occurs at GO in the sort main program or at initialization in the subroutine. The amount of RAM memory (sort main program) or the caller-supplied working storage area (sort subroutine) is is not large enough for the particular combination of record lengths and options specified.

ERROR S30: COMMAND INPUT WORKSPACE FULL,
TRY MORE MEMORY OR SIMPLER COMMANDS

Occurs during main program command entry. A command was entered that was presumably valid but too complex to handle in the amount of RAM in use.

ERROR S35: BADLY STRUCTURED SORSUB PARAMETER BLOCK

Occurs at subroutine initialization, indicating invalid parameters in the call. Probably one of the DW -1's required after the sort file names, merge-only file names, and key specifications is missing or misplaced, or the number of sort files, merge-only files, or keys actually given is different from the number that precedes them.

ERROR S37: INVALID FILE NAME

During main program command entry, means whatever was entered where a file name is required contains no letters or digits (before the . if one is given), or contains * or ?.

At subroutine initialization, indicates first byte of file name in the parameter block is zero or blank.

ERROR S38: INVALID DRIVE NAME

During main program command entry, drive name given not between A and Z (WORK-UNIT command).

At subroutine initialization, first byte of any of the file names in the parameter block not 0-26 (the normal CP/M values) nor ASCII 'A'-'Z' (accepted for convenience and translated to 1-26).

ERROR S39: UNRECOGNIZED COMMAND

During command entry, first word of command not a SuperSort command keyword nor an acceptable abbreviation thereof.

ERROR S40: INVALID ARGUMENT

During command entry, an argument to a command (something after the initial keyword of the command) not understood by SuperSort. "^" is printed under approximate position of error, for this and all command entry errors.

ERROR S41: MISSING ARGUMENT
Command entry: something mandatory appears to have been omitted, about where the ^ points; or, possibly a preceding error caused SuperSort to expect another argument when you intended to end a variable length list.

ERROR S42: INVALID LINE TERMINATION

During command entry, indicates command acceptable up to approximate position of "^", after which the end of the command (carriage return or ; or |) was expected, but something else was found.

ERROR S43: NUMBER REQUIRED

During command entry, at least one digit is required where the "^" points, but none was found.

ERROR S44: NUMBER TOO LARGE

During command entry, number larger greater than 65535. (This limit does not apply to numeric constants in SELECT and EXCLUDE arguments, which are limited only by the number of digits you are willing to enter.)

ERROR S45: RECORD LENGTH MISSING

Command entry: number to specify record length is mandatory in INPUT-ATTRIBUTES command.

ERROR S46: THAT COMMAND MUST BE LAST ON LINE

Command entry: "GO" and "CFILE" may not be followed by ; and another command.

ERROR S50: NO INPUT FILE SPECIFIED

At GO command to the main program, means no SORT-FILES nor MERGE-FILES command has been given. Specify your input, then GO again. At subroutine initialization, means subroutine parameter block specified no file(s) to sort and no file(s) to merge.

ERROR S51: MORE THAN 32 SORT INPUT FILES

Occurs during main program command entry, or at subroutine initialization.

ERROR S52: NO KEYS SPECIFIED

At GO command to main program, means no KEY command has been entered and accepted. At subroutine initialization, means zero key count specified in parameter block.

ERROR S53: MORE THAN 32 KEYS

Occurs during main program command entry, or at subroutine initialization.

ERROR S54: MORE THAN 32 SELECT CRITERIA

During main program command entry, means number of SELECT commands, plus the number of EXCLUDE commands, greater than 32. At subroutine initialization, means the "number of select criteria" field in the sort subroutine parameter block contains a number greater than 32.

ERROR S55: NO OUTPUT FILE SPECIFIED

At GO command to main program, means no OUTPUT-FILE command has been entered. At subroutine initialization, output file name field in sort subroutine parameter block begins with a zero or blank.

ERROR S56: PRINT LEVEL NOT Ø TO 5

Occurs during main program command entry, or at subroutine initialization.

ERROR S57: RECORD LENGTH NOT 1 TO 4Ø96

Occurs during main program command entry, or at subroutine initialization. Indicates invalid input or output record length specification. For VARIABLE files, the minimum record length that the user may specify to SuperSort is 3, corresponding to records containing 1 data byte.

ERROR S58: FIELD END COLUMN < START COLUMN

Command entry: applies to positional field specifications (no #) in KEY, SELECT, and EXCLUDE statements.

ERROR S59: #-FIELD NUMBER NOT 1 TO 255

Occurs during main program ommand entry, or at subroutine initialization. Applies to KEY, SELECT, and EXCLUDE comma-delimited fields.

ERROR S61: FIELD END COLUMN > RECORD LENGTH

Occurs in the main program during command entry or at GO; in the subroutine, at initialization. Applies to KEY, SELECT, and EXCLUDE positional (column-specified, no # in command) fields.

ERROR S62: FIELD START COLUMN < 1 OR > RECORD LENGTH

Occurs in the main program during command entry or at GO; in the subroutine, at initialization. Applies to KEY, SELECT, and EXCLUDE positional fields.

ERROR S63: FIELD LENGTH NOT 1 TO 4Ø96

Occurs at entry of a KEY, SELECT, or EXCLUDE command, or during subroutine initialization.

ERROR S64: BAD SELECT OR EXCLUDE ARGUMENT

During execution, indicates invalid operation code in record select parameter string supplied by subroutine caller.

ERROR S65: NO 'GO' AFTER ERROR IN COMMAND FROM COMMAND FILE -- CORRECT ERROR, ENTER 'GO' AGAIN

Occurs at GO command from a command file after an error was found in a preceding command in the command file.

If the sort main program detects an error in a command input from a command file (invoked with the CFILE command), it prints the line containing the error, and the error message, on the console. If a GO command is subsequently encountered in the command file, it is not executed, instead the above message is printed and console input is accepted.

This gives the operator the opportunity to enter the necessary command(s) to correct the error. After the error is corrected, another GO may be entered to start sort execution.

ERROR S67: TAGSORT WITH INPUT FILE SAME AS OUTPUT

Occurs at GO command in the main program, or at initialization in the subroutine, when tagsort has been requested and the same file has been specified for both input and output.

ERROR S68: MORE THAN 32 MERGE-ONLY INPUT FILES

Occurs during main program command entry, or at subroutine initialization.

ERROR S69: NO INPUT RECORD LENGTH SPECIFIED

At GO in the main program, indicates no INPUT-ATTRIBUTES command has been entered. At subroutine initialization, indicates a zero in the parameter block input record length field.

ERROR S7Ø:) MISSING

At entry of a SELECT or EXCLUDE command: matching parentheses not found where expected, or not recognized due to some other syntax error.

ERROR S71: COMPARISON OPERATOR REQUIRED

Occurs at entry of a SELECT or EXCLUDE command. After a value (field specification, quoted text constant, or numeric constant), one of the following is required: GT, GE, EQ, NE, LE, LT, BT, NB, >, >=, =, <>, <=, or <.

ERROR S72: FIELD OR CONSTANT REQUIRED

Occurs at entry of a SELECT or EXCLUDE command.

Before and after one of the comparison operators listed in the description of the preceding error, a FIELD specification or a numeric or quoted text constant is required (2 values are required after BT and NB).

The message also occurs if the beginning of the argument list is unrecognizable, where a (or "NOT" are acceptable in addition to a field or constant.

ERROR S73: SECOND " MISSING

Command entry: closing " missing in SELECT or EXCLUDE text constant.

ERROR S74: ILLEGAL DIGIT FOR NUMBER BASE

At entry of a SELECT or EXCLUDE command: 8-9 or A-F encountered in an octal constant, or A-F encountered in a decimal constant.

Might also be confusion resulting from lack of a space or comma to separate a constant from the following item.

ERROR S75: SIZE IN BYTES IS TOO SMALL TO HOLD THE VALUE GIVEN

At entry of a SELECT or EXCLUDE command: The significant digits of a constant require more bytes than the size in bytes specified after the base indicator. Example: 12345H2: 12345 hexadecimal will not fit in two bytes.

Or, confusion resulting from the lack of a space or comma between the constant and the next item.

ERROR S76:] MISSING"

At entry of a SELECT or EXCLUDE command: in [ ]-enclosed constant list, neither another valid constant nor the closing] found at position of "^".

ERROR S78: 1-BYTE CONSTANT REQUIRED

At entry of a COLLATING-SEQUENCE command: a character or position value was given as a pair of quotes with Ø characters or two or more characters between them, or as a number whose value was greater than 255 decimal.

ERROR S79: TOO MANY NESTED ()'S OR INTERNAL ERROR

At entry of a SELECT or EXCLUDE command: can be caused by more than 5 levels of nested parentheses.

ERROR S81: WORKING STORAGE FULL:
TRY USING MORE WORKING STORAGE, OR TRY TAGSORT.

Occurs during execution. An unusually large amount of input being sorted or merged in a relatively small amount of RAM memory (sort main program) or a small caller-supplied working storage area (sort subroutine) has caused the working storage area to overflow during sorting or merging, despite the fact that the available working storage appeared adequate when checked at initialization.

ERROR S82: ILLEGAL XIT1 INSERTION

Occurs during execution, when the operator has invoked a user-installed "XIT1" user-exit subroutine. The XIT1 subroutine attempted to insert or replace a record in a merge-only input file, or when tagsort, R-OUTPUT, P-OUTPUT, KR-OUTPUT, or KP-OUTPUT was in use.

XIT1 functions are limited to acceptance or rejection of the record except for sort input records when full record sort or K-OUTPUT is in use.

ERROR S85: MERGE-ONLY INPUT FILE NAME SAME AS OUTPUT FILE

Occurs at GO command in main program, or at subroutine initialization. The output file cannot be the same as one of the merge input files.

ERROR S86: TAGSORT, AND INPUT FILE DRIVE SAME AS OUTPUT/C DRIVE

Occurs at GO command in the main program, or at subroutine initialization. When tagsort and output diskette change are both requested, the output file cannot be on the same drive as the input file.

ERROR S87: WORK FILE DRIVE SAME AS OUTPUT/C DRIVE

Occurs at GO command in the main program, or at subroutine initialization. When output diskette change is specified, the work file must be on a different drive. (The work file is on the current drive if not otherwise specified with the WORK-DRIVE command.)

ERROR S88: INVALID COMBINATION OF FIELD TEST ATTRIBUTES

An invalid combination of KEY or SELECT/EXCLUDE field test attributes has been specified, such as NUMERIC-ASCII and INTEGER together, or UPPER-CASE and FLOATING-POINT. Also occurs if LOHI is given for a comma-delimited field. Occurs at command entry, or at subroutine initialization or execution.

ERROR S89: MORE THAN ONE OF CR-DELIMITED, FIXED, VARIABLE, AND RELATIVE

Two file types specified for input, or for output. Note that it is legal for the output to be different from the input. Occurs at command entry, or at subroutine initialization.

ERROR S91: MORE THAN ONE INPUT FILE WITH OUTPUT OPTION OR TAGSORT

R-OUTPUT, P-OUTPUT, KR-OUTPUT, KP-OUTPUT must be used with a single (sort or merge) input file; TAGSORT must be used with a single sort input file only. (K-OUTPUT can be used with multiple input files.) Occurs at GO or at subroutine initialization.

ERROR S92: MORE THAN ONE OUTPUT OPTION, OR AN OUTPUT OPTION AND TAGSORT

More than one of TAGSORT, K- , R- , P- , KP- , and KR-OUTPUT specified. Occurs at GO or at subroutine initialization.

ERROR S94: MERGE-ONLY INPUT FILE WITH TAGSORT

Tagsort must be used with a single sort input file only. Occurs at GO or at subroutine initialization.

ERROR S1ØØ: SELECT/EXCLUDE NOT PRESENT IN THIS VERSION

Occurs at main program command entry, or during subroutine execution, if record selection is invoked in a version of SuperSort which the user has created with the NOSEL load option, as described in the "Programmer's Guide".

ERROR S1Ø1: COLATING-SEQUENCE, ALTSEQ, AND EBCDIC
NOT IN THIS VERSION

Occurs at command entry when the indicated features are invoked in a version of the main program created by the user with the NOCOL load option invoked.

CAUTION

If the subroutine is loaded with NOCOL, and EBCDIC is invoked, no error occurs, but incorrect sorting will occur.

Use of ALTSEQ is unaffected by NOCOL in the subroutine since in the subroutine the collating sequence table is supplied by the user as an argument, rather than loaded from SORLIB.

ERROR S128: FILE name.typ NOT FOUND

At entry of SORT-FILES or MERGE-FILES command, named file not found on specified disk. If wrong diskette was mounted, it is permissable to change diskettes and re-enter command.

Occurs during execution if one of the specified inputs to the subroutine is not found.

ERROR S129: DISK d: FULL WHILE WRITING FILE name.typ

Occurs during execution when there is insufficient space for the indicated file. If the file is the work file (whose name.typ appears as SORT.$$$), you may wish to put this file on another drive via the WORK-DRIVE command, or to use tagsort to reduce the size of the work file.

ERROR S130: DIRECTORY OF DISK d: FULL WHILE WRITING FILE name.typ

Occurs during sort execution. A CP/M diskette's directory fills up before its file space is full if there are a number of small files on the diskette. See previous error with regard to work file.

ERROR S131: CLOSE FAILURE FILE name.typ

Internal error, or diskette was changed while program was running, other than before the GO command or as prompted by the output diskette change option.

ERROR S133: OVERLONG RECORD (LENGTH nnn) IN FILE name.typ

Record encountered during execution in a VARIABLE input file with length longer than the maximum specified by the user.

ERROR S134: NOT A RELATIVE FILE: FILE name.typ

The RELATIVE input attribute was specified, but the indicated file did not have a proper COBOL RELATIVE file header with a zero in the first byte. Occurs during execution.

ERROR S135: INCORRECT RECORD LENGTH GIVEN FOR RELATIVE FILE name.typ

User-specified input record length did not match that in RELATIVE file header. Occurs during execution.

ERROR S136: INVALID LENGTH nnn IN VARIABLE LENGTH OUTPUT RECORD

VARIABLE output file in use, and record returned by user-installed XIT2 subroutine did not contain a record length between 1 and 4094, fixed point binary, high order first, in its first two bytes; or VARIABLE input in use, and record with invalid length returned by user-installed XIT1 routine.

NOTE

If the record length is in the range 1 to 4094, but longer than the user-specified record length, SuperSort will truncate the record, adjust the length, and give no error message.

ERROR S140: COMMAND DECODING WORKSPACE FULL,
TRY MORE MEMORY OR SIMPLER COMMANDS

During command entry, either a very complicated command (probably a SELECT or EXCLUDE) was input while operating in a minimal amount of memory, or an internal error.

ERROR S150: INTERNAL ERROR
ERROR S151: INTERNAL ERROR
ERROR S152: INTERNAL ERROR
ERROR S153: INTERNAL ERROR
ERROR S154: INTERNAL ERROR

The preceding five errors can be caused during execution by an incorrectly formed record select parameter string in a call to the SuperSort subroutine. If you are using record select in your subroutine call, check the pertinent parameters carefully. If you are using the subroutine without intending to invoke record selection, make sure the parameter block ends correctly with a 16-bit zero followed by a 16-bit -1 to indicate null selection parameters.

ERROR S160: INTERNAL ERROR
ERROR S253: INTERNAL ERROR

Internal errors.

ERROR Snn: NO SUCH ERROR

ERROR-MESSAGE command entered with undefined error number, otherwise internal error.

3. "NOERR" LOAD OPTION MESSAGES

The NOERR load option reduces the amount of memory SuperSort occupies, in order to increase the working storage available, by shortening the error message texts. Invocation of this option is described in the "Programmer's Guide".

Many of the shortened error message texts are null -- only the word ERROR and the error number is printed. Most of these are command entry errors, where the "^" printed under the approximate positition of the error usually makes the problem immediately apparent to an experienced operator.

Similarly, the several errors which say TOO MANY are distinct in use (in the main program case) by what command causes them.

Following are the shorter error message texts provided by the NOERR load option. For equivalent long texts and descriptions, refer above by number.

```
ERROR S28: TOO LITTLE MEMORY
ERROR S29: NOT ENOUGH WORKING STORAGE
ERROR S30: TOO LITTLE MEMORY
ERROR S33:
ERROR S35: BAD PARM BLK
ERROR S36:
ERROR S37: BAD FILE NAME
ERROR S38: BAD DRIVE
ERROR S39: WHAT?
ERROR S40:
ERROR S41: ARG MISSING
ERROR S42: HUH?
ERROR S43: # REQUIRED
ERROR S44: # > 65545
ERROR S45: # REQUIRED
ERROR S46: END LINE REQUIRED
ERROR S50: NO INP FILE
ERROR S51: TOO MANY
ERROR S52: NO KEY
ERROR S53: TOO MANY
ERROR S54: TOO MANY
ERROR S55: NO OUT FILE
ERROR S56: BAD PRN LVL
ERROR S57: BAD REC LEN
ERROR S58: FIELD END < START
ERROR S59: #-FIELD # BAD
ERROR S61: FIELD END > REC LEN
ERROR S62: FIELD START BAD
ERROR S63: BAD FIELD LEN
ERROR S64: BAD SEL/EXCL ARG
ERROR S65: CORRECT PREVIOUS ERR THEN GO
ERROR S67: TAGSORT MISUSED
ERROR S68: TOO MANY
ERROR S69: NO INP REC LEN
ERROR S70: ) MISSING
ERROR S71:
ERROR S72:
ERROR S73: " MISSING
ERROR S74: BAD DIGIT
ERROR S75: SIZE TOO SMALL
ERROR S76: ] MISSING
ERROR S78:
ERROR S79: TOO MANY ((((()))))'S
ERROR S81: FULL WORKING STORAGE
ERROR S82: ILLEG XIT1 INSERT
ERROR S85: MERGE FILE = OUTPUT FILE
ERROR S86: INPUT DRIVE = OUTPUT/C DRIVE
ERROR S87: WORK DRIVE = OUTPUT/C DRIVE
ERROR S88: ILLEG ATTRIBUTE COMBINATION
ERROR S89:
ERROR S91: OUTPUT OPTION OR TAGSORT MISUSED
ERROR S92: OUTPUT OPTION OR TAGSORT MISUSED
ERROR S94: TAGSORT MISUSED
ERROR S100: FEATURE NOT PRESENT
ERROR S101: FEATURE NOT PRESENT
ERROR S128: NOT FOUND: FILE d:name.typ
ERROR S129: DISK FULL FILE d:name.typ
ERROR S130: DIRECTORY FULL FILE d:name.typ
ERROR S131: CLS ERR FILE d:name.typ
ERROR S133: REC TOO LONG FILE d:name.typ
ERROR S134: NOT RELATIVE: FILE d:name.typ
ERROR S135: REC LENGTH BAD: FILE d:name.typ
ERROR S136: BAD OUTPUT REC LEN: nnn
ERROR S140: TOO LITTLE MEMORY
ERROR S150: INTERNAL ERR
ERROR S151: INTERNAL ERR
ERROR S152: INTERNAL ERR
ERROR S153: INTERNAL ERR
ERROR S154: INTERNAL ERR
ERROR S160: INTERNAL ERR
ERROR S253: INTERNAL ERR
ERROR Snn:
```

4. "NOREPORT" LOAD OPTION MESSAGES

The NOREPORT Load Option, another option that makes more working storage available by shortening message texts, alters two of the warning messages as follows. Refer to the "Warning Messages" section above for full text and discussion:

```
WARNING SW5: RECORD(S) WITH INSUFFICIENT FIELDS OR COLUMNS
WARNING SW6: OVERLONG CR-DELIMITED INPUT RECORD(S)
```

IV. PROGRAMMER'S GUIDE

The "Programmer's Guide" is intended primarily for programmers and systems designers. Additional detail on file and data formats is presented. Calling sequences for the SuperSort subroutine, SORSUB, and various supporting routines are given. Loading procedures are described for user-supplied main programs that use the SuperSort subroutine, and for reloading the SuperSort main program in order to invoke load options supplied with SuperSort or to install user-supplied custom modules such as user-exit routines.

The "Programmer's Guide" assumes familiarity with the "Operator's Handbook", particularly the "Concepts and Facilities" section. Refer back as necessary for basic descriptions of and motivations behind the features which are described here in technical detail only.

A. FILE AND RECORD FORMATS

This section is intended mainly for those who wish to use non-ASCII data with SuperSort, and for programmers who wish to write assembly language programs to process files of one or another of SuperSort's record types.

1. CR-DELIMITED RECORD FILES

CP/M Text File Conventions

Under CP/M, a file containing lines of text normally has both a carriage return (ØD hex) and a line feed (ØA hex) after each line. If the last record in the file does not end exactly at the end of a physical sector, a Control-Z (1A hex) character follows the last record. If the last record ends exactly on a sector boundary, there may or may not be an additional sector containing a Control-Z, depending on what language or utility wrote the file.

The above format is produced by and expected by the editor and the various programming languages. SuperSort produces the above format for CR-DELIMITED output files; on input, it will correctly read the above format and some variations, as follows.

SuperSort handling of CR-DELIMITED Files

When reading a CR-DELIMITED input file, SuperSort accepts a single carriage return or line feed as a record delimiter. It then ignores any additional carriage returns or line feeds before the beginning of the next record. Thus, a file intended to be read only by SuperSort needs to contain only one character between records; empty lines, containing not even a blank, are ignored in the input. A Control-Z is also taken as a record delimiter (as well as an EOF indicator), so that the last record of the file will be correctly read even if the carriage return after it is omitted.

SuperSort considers the input end-of-file to have been reached upon reading the physical end-of-file, or upon reading a single Control-Z (1A hex) character. The NO-SINGLE-Z and NO-ZZZ input attributes are inoperative with CR-DELIMITED input.

When writing a CR-DELIMITED output file, SuperSort puts one carriage return and one line feed after each record, regardless of how the corresponding input record was delimited. The file is terminated by filling any unused space in the last sector with Control-Z's (1A hex), or two hex FF's followed by Control-Z's if the FFZZZ output attribute is specified. If the data happens to end on a sector boundary, no end-of-file characters are written.

Data Considerations for CR-DELIMITED Files

Normally, only blanks, printing characters (ASCII codes 21 hex through 7E hex), and possibly tabs (Ø9 hex) and form feeds (ØC hex) are put in text files. When putting other values into a CR-DELIMITED file, the user be sure to output no bytes with value 1A hex except as an end-of-file indicator, and no bytes with value ØD hex (carraige return) or ØA hex (line feed), except as record delimiters.

2. FIXED-LENGTH RECORD FILES

In SuperSort, FIXED-LENGTH records have a constant length and no character is ever interpreted as a record delimiter. Carriage returns, line feeds, or any special character(s) other than Control-Z (1A hex) may be present without restriction, provided they are allowed for in the record length specification.

The record length need not bear any relationship to the physical sector length; SuperSort handles FIXED-LENGTH records 1 to 4Ø96 bytes long, spanning sector boundaries to efficiently use disk space.

On input, SuperSort considers the end-of-file to have been reached when there is less than a record length of data remaining before the physical end of file, or, if neither NO-SINGLE-Z nor NO-ZZZ is specified, upon detecting a Control-Z (1A hex) in position 1 of the record. Users of binary data which could possibly take on the value 1A hex (26 decimal) in the first byte of the record should carefully read the section below on End-of-File Indication Considerations.

On output, SuperSort fills any unused space in the last sector of the file with Control-Z's (1A hex), or with two FF hex bytes followed by Control-Z's if the FFZZZ output attribute is in effect. If the last record ends at the end of a sector, no end-of-file characters are written. Note that in the SuperSort main program, FFZZZ defaults on for output if it is specified for input.

3. VARIABLE RECORD FILES

VARIABLE records contain the length of the data in the record in the first two bytes, stored as a binary fixed point number with the high order byte first. This record data length must be between 1 and 4Ø94. In SuperSort (but not in the file), the length bytes counted in the length; thus, the record length specification must be in the range 3 to 4Ø96 and positional data fields begin at column three.

When reading a VARIABLE input file, SuperSort considers the end of the file to have been reached when less than 2 bytes remain before the the physical end-of-file, or upon reading two bytes in the record length position containing control-z's.

When writing a VARIABLE output file, SuperSort ends the file by writing two zero bytes (a zero-length record), then filling any remaining space in the last sector with (two hex FF's if FFZZZ specified, then) Control-Z's (1A hex).

Since the length of each record, and the end-of-file, are indicated externally to the data, VARIABLE files may contain any data type with any value in any column position after the length bytes.

4. RELATIVE FILES

RELATIVE files are a special COBOL-compatible type of fixed length record file. In COBOL programs, records in a RELATIVE file can be randomly read, written, and deleted by record number. The records present in the file need not have contiguous record numbers. Deleted records are represented by a record length of binary zeroes; never-written records are all zero or have no disk space allocated to them. The record length and highest active record number are stored in a 6-byte header at the beginning of the file.

Since the records are not delimited, and the end-of-file is unambiguously indicated by the highest active record number in the header, RELATIVE file records can contain any data type with any value, provided the entire record is not binary zeroes.

The SuperSort main program automatically sets the NO-SINGLE-Z and NO-ZZZ input attributes when RELATIVE input is specified; users of the subroutine version should specify these attributes in the parameter block when using RELATIVE input.

The format of the relative file header in the first sector is:

Byte	Use
Ø	must be Ø to indicate RELATIVE file
1,2	record length, binary fixed point, low order first
3	unused
4,5	highest record number that has been written

Record data begins in byte 6 of the first sector.

5. END-OF-FILE INDICATION CONSIDERATIONS FOR FIXED-LENGTH FILES

The CP/M operating system keeps track of the number of 128-byte sectors in a file, but it has no provision for recording, externally to the data, the number of used and unused bytes in the last sector of the file. Thus, to allow for cases where the number of data bytes in the file is not an exact multiple of 128, the end of file must be indicated in the data. By convention, Control-Z (1A hex) characters are for this purpose. This convention works without ambiguity for ASCII and BCD data, but some types of non-ASCII data can take on the value 1A hex, creating the possibility of a spurious end-of-file indication and the resultant loss of following input data.

CR-DELIMITED files should contain only ASCII data, which cannot take on the value 1A hex. For VARIABLE and RELATIVE files, the end-of-file indication problem is handled by the file and record structure and EOF attribute defaulting as described above. For FIXED-LENGTH record files, the user of binary data must consider the end-of-file indication problem.

SuperSort defaults to terminating a FIXED-LENGTH record input file upon reading a Control-Z (1A hex) in position 1 of the record. This works so long as the data in this position cannot take on this value, e. g. if it is ASCII text or if the user's program writes a zero fill byte in this position. For users who wish less restriction on their binary data, other options are available; before discussing them, we will review the available EOF criteria.

SuperSort has the following four possible criteria for defining the end of a FIXED-LENGTH input file. If more than one is enabled, SuperSort terminates input on the first to occur.

Physical end of file

> If less than a record length of data remains in the file, SuperSort always considers the end of the file to have been reached. This is an unambiguous logical end of file indicator if the logical record length is 129 or greater; for files (e.g. FORTRAN unformatted output) known not to have an appended Control-Z, it is also an unambiguous end-of-file indicator for records of length 128.

Single Control-Z at beginning of record

> This is the default; it may be suppressed with NO-SINGLE-Z input attribute and is also suppressed by the NO-ZZZ input attribute in the SuperSort main program.

Full record length of Control-Z's (1A hex)

> This also defaults on; it may be disabled with NO-ZZZ.

Two FF hex bytes, rest of record Control-Z's

> defaults off; enable with FFZZZ. Note that this provides an unambiguous EOF indication if the data record begins with a non-negative integer quantity.

Some approaches to the binary FIXED-LENGTH record end-of-file definition problem are:

a. Make sure there cannot be 1A hex in the first byte of the record. For example, write a zero instead of data in this byte. While this can waste a byte in each record, it is simple and requires no special input attributes to be specified in SuperSort.

b. Make sure the entire record cannot consist of 1A's, and use NO-SINGLE-Z in SuperSort. If there is a datum anywhere in your record which cannot take on the value 1A hex in all bytes, and you know that the program which writes the file fills the entire unused portion of the last sector with Control-Z's (many language processors onlyØ write a single Control-Z), the full-record-length-of-Control-Z's end-of-file criterion will work.

c. Use the FFZZZ convention in your files, and specify FFZZZ and NO-ZZZ in SuperSort. This convention is effective if the first datum in the record cannot take on the value FFFF hex -- for instance, if it is a positive FORTRAN integer. Code the program that writes the file to fill any unused space in the last sector with FF hex for the first two bytes, followed by 1A hex to the end of the sector. Or, if more convenient, code the program to always write two FF hex bytes followed by enough 1A hex bytes to make a record length.

d. Make the logical record length 128 or greater. Specify NO-ZZZ in SuperSort. The physical end of file will be an unambiguous logical end of file indication and no restrictions will apply to the data in the file. Exception: if the program writing the file unconditionally appends a terminating Control-Z, then the record length must be 129 or more.

B. FIELD FORMATS

1. POSITIONAL FIELDS

A positional field is located in the record via a user-specified start position and end position (or length, in calls to the the SuperSort subroutine). The positions (columns) start with 1 for the first byte of the record, or 3 for the first data byte of a VARIABLE record. No commas, quotes, or other characters get special treatment; if present, they are treated as part of the data.

If a positional field's ending position is beyond the end of a CR-DELIMITED record, it is filled out with blanks. If its start position is beyond the end of the record, all blanks are used and a warning message occurs. An attempt to specify a starting or ending position beyond the specified record length causes an error message.

2. COMMA-DELIMITED FIELDS

A comma-delimited field is specified by field number. Field n is extracted from the record as follows: first, the n-1 th comma in the record is located, not counting commas enclosed in quotes. If n is 1, the beginning of the record (column 3 for VARIABLE records) is used. This is the beginning of the field. Then, the next comma not enclosed in quotes is located; if the end of the record is encountered, this position is used. This is the end of the field. Next, any blanks at the beginning or end of the field are removed. Finally, all quotes (") in the field are removed. For a key field, the result is then right-truncated if longer than the user-specified maximum length.

Though quotes are normally used around the entire field, if at all, there can be any even number of quotes in the field, and they can be in any position as long as they enclose the desired commas or blanks. The presence or absence of quotes has no effect except when they enclose commas, leading blanks, or trailing blanks; blanks not at the beginning or end of the field are never deleted.

If there are not n-1 commas not enclosed in quotes in the record, a warning message occurs and a blank field is assumed.

Commas with nothing between them are taken as blank fields and produce no error message.

A comma-delimited field can be up to 4096 characters long.

The field number must be 1 to 255.

C. DATA FORMATS

SuperSort's basic data format is a stream of unsigned bytes. When comparing two keys during sorting or merging, or when doing a record selection test, the two byte streams are compared left to right until a position is found where the bytes are not equal. This correctly sorts ASCII data, and unsigned binary fixed point data stored high order first; there is no distinction between these types, and no user specification is needed to distinguish them.

This section details the data formats expected by some of the test attributes the user may specify to modify the above basic method of handling data; for an introduction to all of the test attributes, see the "Concepts and Facilities" section.

1. NUMERIC-ASCII

A NUMERIC-ASCII datum is a number represented as a string of ASCII characters, as written by the PRINT statement of BASIC, by the formatted write of FORTRAN, by putting USAGE IS DISPLAY data into a record in COBOL, or as entered with a text editor or other program. Any sequence of characters, within the specified field, matching the following, in order, will be correctly interpreted:

```
any number of blanks
optional + or -, followed by any number of blanks
any number of leading zeroes
any number of digits
optional decimal point
additional digits if decimal point was present
any number of blanks
E, e, D, or d, to indicate exponent following
If E, e, D, or d present:
    any number of blanks
    optional + or -, optionally followed by blanks
    up to three digits, with value between -255 and +255.
```

Significant digits after 14 are ignored, but the scan continues to locate the point correctly and to check for an exponent.

There are no error or warning messages for NUMERIC-ASCII data interpretation. If the field is not numeric, zero is assumed; if only part of it can be interpreted, the value thus obtained will be used; an exponent out of the range -255 to +255, or more than three exponent digits, will produce an unspecified value without an error message.

2. PACKED-BCD (COMPUTATIONAL-3)

The PACKED-BCD (Binary Coded Decimal) test attribute signifies COBOL format packed decimal data. Such data is stored with two decimal digits in each byte, with the more significant digit in the high order bits and the more significant bytes first (at lower addresses). Each digit occupies four bits and has a value between Ø and 9. The optional sign is at the end, in the low order four bits of the last byte. Values in these four bits are have the following meanings:

Ø-9:	digit: number is unsigned
A, C, E or F hex:	+ sign
B or D hex:	- sign

SuperSort will correctly compare signed and unsigned PACKED-BCD numbers, and signed numbers with different sign representations.

D. OPTIONAL OUTPUT FILE FORMATS

1. OUTPUT OPTIONS — GENERAL

The general nature of the output options, and the motivations behind them, were given in the "Concepts and Facilities" section.

The data output by each output option is described in the following sections. In all cases, if VARIABLE output is specified, two bytes of record length begin the record; if CR-DELIMITED output is specified, a carraige return and line feed follow the record.

When an output option is specified; any output record length specified by the user is disregarded. The output record lengths for the various options are as follows:

file type	K-OUTPUT	R-OUTPUT	P-OUTPUT	KR-OUTPUT	KP-OUTPUT
CR-DELIMITED	keys	6	13	keys+6	keys+13
FIXED-LENGTH	keys	3	4	keys+3	keys+4
RELATIVE	keys	3	4	keys+3	keys+4
VARIABLE	keys+2	5	6	keys+5	keys+6

where "keys" means the lengths of the keys as described below for K-OUTPUT. Note that the record lengths for CR-DELIMITED files do not include the carriage return and line feed.

2. K-OUTPUT (keys only)

Each output record in a K-OUTPUT file contains only the concatenated, internalized keys. The record length is the total length of the modified keys. Note that PACKED-BCD fields have a byte added to them, and all NUMERIC-ASCII keys become 9 bytes.

The keys are concatenated together, highest priority key first, in a partially internalized form suitable for comparison as an unsigned byte string. Comma-delimited fields have been extracted, deblanked, and de-quoted as described above, and extended to their maximum length with blanks. Attributes specified have been applied, as follows:

NUMERIC-ASCII	data converted to 9-byte format detailed below
UPPER-CASE	lower case letters converted to upper case
RIGHT-JUSTIFY	leading, rather than trailing, blanks added to comma-delimited fields shorter than maximum length
LOHI	reverse order of bytes

MASK-PARITY-BIT	set bit 7 of all bytes to Ø
EBCDIC	each byte replaced from EBCDIC table
ALTSEQ	each byte replaced from alternate collating sequence table
TWOS-COMPLEMENT	complement bit 7 of first byte
INTEGER	reverse order of bytes, complement bit 7 first byte
FLOATING-POINT	special conversion detailed below
PACKED-BCD	add FF hex byte in front of key; if signed, shift right 4 bits to delete sign; if sign was negative, ones complement entire key, including the added byte
DESCENDING	ones complement entire key, after all other attributes applied

A K-OUTPUT file created using no attributes will contain the original field data; some attributes, such as NUMERIC-ASCII and DESCENDING, thoroughly disguise the original data.

When using attributes in creating a CR-DELIMITED or FIXED-LENGTH K-OUTPUT file, use caution not to inadvertently create an end-of-file indication; see the "File and Record Formats" section above.

The 9-byte internal format produced by the NUMERIC-ASCII attribute is a sortable BCD floating point number, as follows:

bytes/bits	definition
byte Ø bit 7	one
byte Ø bits 6 - Ø, and byte 1	excess 513 exponent (base ten), computed as: value of digits after E, if any, plus number of digits before decimal point, plus 512, but zero if significant digits zero.
bytes 2-6	significant digits, 4 bits each, trailing Ø's.

All one's complemented if number was negative.

The FLOATING-POINT attribute transforms the key as follows:

1. reverse order of bytes.
2. if byte Ø (after reversal) is Ø, set all bytes to Ø.
3. set bit 7 of byte 1 to bit Ø of byte Ø.
4. shift byte Ø right one bit position.
5. set bit 7 of byte Ø to 1.
6. if bit 7 of byte 1 was a 1 before step 3, ones-complement all bytes.

3. R-OUTPUT (record numbers)

An R-OUTPUT file contains only the input record numbers. The record numbers start at 1; the maximum record number handled is 65535.

If the output file is specified as CR-DELIMITED, the numbers are output in ASCII, with leading zeroes to produce six digits.

If the output file is specified as FIXED-LENGTH, VARIABLE, or RELATIVE, the numbers are output in binary, three bytes each. The first byte is always zero, the second byte is the low order byte of the record number, and the third byte is the high order byte: the bytes are "low order first", forming a FORTRAN integer.

When reading a FIXED-LENGTH R-OUTPUT file with another program, the end of the file is indicated by physical end of file or by 1A hex (26 decimal) in the first byte, whichever occurs first. (or FF hex in the first byte if the FFZZZ output attribute was specified).

4. P-OUTPUT (pointers)

A P-OUTPUT file contains "pointers" to the beginning of each input record. Each pointer consists of two numbers: a SECTOR NUMBER, running up from 1, indicating which 128-byte sector of the file contained the beginning of the record, and a BYTE OFFSET, a number 1 to 128 indicating at which byte in the sector the record begins.

If the output file is CR-DELIMITED, the numbers are output in ASCII, six digits each, with a comma between them. If the output file is another type, the numbers are output in binary, two bytes each, low order byte first.

When reading a FIXED-LENGTH P-OUTPUT file with another program, the end of file is indicated by physical end-of-file, or by a record containing 1A1A hex for both numbers, whichever occurs first. (If the file is written with the FFZZZ option, then -1 in the first number indicates end of file).

5. KR-OUTPUT (keys and record numbers)

A KR-OUTPUT file contains the keys, followed by the input record numbers, both as described above.

6. KP-OUTPUT (keys and pointers)

A KP-OUTPUT file contains the keys, followed by the pointers to the input records, both as described above.

E. SUBROUTINES AND CALLING SEQUENCES

This section describes the subroutines supplied on the SuperSort distribution diskette and the user-exit routines which may be coded by the user. Calling sequences for use from FORTRAN, COBOL, and assembler programs are given in this section. The procedure to load a program using these routines is given later in the "Loading Procedures" section. This section applies to SuperSort I only.

1. INTRODUCTION

The subroutines described in this section are:

SORSUB	the SuperSort sort/merge subroutine
SORMSG	print error message if error occurred on preceding SORSUB call
SORCNT	return counters in array provided by caller
XIT1, XIT2	user-provided routines called by SuperSort to inspect and/or alter records during input and output respectively.

To make use of the SuperSort subroutines, the user must have one or more of the following language processors, in which to write the calling program:

MicroSoft FORTRAN (F80)
MicroSoft Relocateable Assembler (M80)
MicroSoft COBOL

plus the Microsoft loader, L80, to load the compiled code and SuperSort routines.

2. THE SORT/MERGE SUBROUTINE (SORSUB)

The SORSUB subroutine provides the functions of the SuperSort main program, in subroutine form for invocation from a main program supplied by the user.

It has the following detail difference from the main program: there is no provision for changing diskettes at the start of the routine. If the operator has been allowed to change a diskette after invocation of the caller's main program and before that program calls SORSUB, the caller's program should have issued an "initialize disk system" system call.

Arguments to SORSUB

Five arguments must be transmitted in a call to SORSUB. One of them is a variable-length parameter block in which most of the parameter information is actually transmitted. The arguments are named as follows in this manual:

PARBLK	parameter block
WORK	working storage area location
WLEN	working storage area length
STAT	variable for status return
COLTAB	alternate collating sequence table

In more detail, the arguments are as follows:

PARBLK parameter block containing file names, key specifications, etc. as described in detail in the next section, "The SORSUB Parameter Block"

WORK work area location
WLEN work area length

Together the above should define the largest area of RAM that it is practical to make available. For moderate amounts of data with a short record length, a large FORTRAN array or assembler DS will suffice. In this case, WORK should be the address of the array, and WLEN should contain its length in bytes.

For large amounts of data and long record lengths, one very effective technique is to use the area between the top of the program and the bottom of the operating system as the working storage area. The end of the program may be defined in a module that is loaded last, after all library searches, or via the loader's $MEMRY feature. If the program uses FORTRAN I/O, be sure increase this address to allow for the buffers FORTRAN allocates at this address. The address thus obtained should be transmitted as the WORK argument.

A zero may be transmitted in the WLEN argument if the working storage area is to extend up to the base of the operating system. SORSUB will then determine the base of the operating system, reduce the address appropriately if the stack is in this area, and compute the actual working storage area length.

STAT	16-bit variable for completion status return. Zero is returned upon successful completion; otherwise a SuperSort error number is returned. For example, if 28 decimal is returned in this variable, then the error condition is that described for ERROR S28 in the "Warning and Error Messages" section. Warning conditions are not returned in the status; the message is unconditionally typed on the console device and SORSUB execution continues. The completion status is also returned in A register, with the flags set, and also in BX.
COLTAB	user's optional collating sequence table. A 256-byte table (128 acceptable if MASK-PARITY-BIT attribute used), indexed by character (byte) value, with each byte containing position (0-255) that character is to occupy in collating sequence. The COLTAB argument is applied to key fields and record selection tests with the ALTSEQ test attribute specified; It may be omitted if there are no such fields specified in the parameter block.

After describing the overall calling sequences in various languages, we will describe the PARBLK argument in detail.

SORSUB Calling Sequence in FORTRAN

SORSUB may be called as a SUBROUTINE subprogram in the form:

```
CALL SORSUB(PARBLK,WORK,IWLEN,ISTAT,COLTAB)
```

Where PARBLK is an array that has been initialized to the desired parameters as described in the next section, WORK is an array for use as the working storage area, IWLEN is an integer variable or constant containing the length of WORK in bytes, ISTAT is an integer variable for completion status return (values described above), and COLTAB is a 256-element LOGICAL (INTEGER*1) array specifying the alternate collating sequence to use for key fields and selection tests with the ALTSEQ attribute specified in the PARBLK argument.

SORSUB may also be declared and called as an INTEGER FUNCTION, or as a LOGICAL (INTEGER*1) function. In both cases the completion status will be the function value. The ISTAT argument, however, must be be given and will always be set.

SORSUB Calling Sequence in COBOL

SORSUB may be called from a COBOL program with a sentence such as the following in the procedure division:

```
CALL 'SORSUB' USING PARBLK, WORK, WLEN, STAT, COLTAB.
```

When calling SORSUB from COBOL, the user must produce 16-bit binary values with the low order byte stored first for arguments such as WLEN. This can be done using the COMPUTATIONAL data type, which generates a 16-bit binary datum with the high order byte stored first, after adjusting the desired value as follows:

1. Divide by 256 decimal to get a quotient and remainder.

2. Multiply the remainder by 256 decimal.

3. Add the quotient from step 1 to the product from step 2. Use this value.

Normally, the value is known when the program is being written, and the adjustment may be performed by hand and the resulting value entered in a VALUE IS clause in the data division.

The following example shows portions of a COBOL program that calls SORSUB. A complete COBOL program example is given at the end of the "SORSUB Parameter Block" section.

```
  ...
 DATA DIVISION.
  WORKING-STORAGE SECTION.
* WORK AREA, 4096 BYTES LONG IN THIS EXAMPLE
   77 WORK-AREA DISPLAY PIC X(4096).
* WORK AREA LENGTH
   77 WORK-LEN COMPUTATIONAL VALUE 16 PIC 9999.
*      ABOVE VALUE IS 4096, ADJUSTED FOR LOW ORDER FIRST
* STATUS RETURN VARIABLE: SORSUB RETURNS STATUS HERE.
   77 STAT-RETURN COMPUTATIONAL PIC 9999.
* DUMMY COLTAB - FULL TABLE NOT NEEDED IF NOT USED
   77 COLTAB PIC X.
* SORT SUBROUTINE PARAMETER BLOCK
   01 PARAM-BLOCK.
        (an example with a parameter block in COBOL
        is given at the end of the next section)
   ...
 PROCEDURE DIVISION.
   ...
* INVOKE SORT SUBROUTINE
      CALL 'SORSUB' USING PARAM-BLOCK, WORK-AREA,
          WORK-LEN, STAT-RETURN, COLTAB.
* TEST STATUS RETURNED TO SEE IF AN ERROR OCCURED
      IF STAT-RETURN = 0 GO TO SUCCESS.
* PROCESS SORSUB ERROR HERE.  TEST STAT-RETURN FOR 256
* TIMES SORT ERROR CODE TO IDENTIFY PARTICULAR ERRORS.
   ...
```

SORSUB Calling Sequence in Assembler

When SORSUB is called from an assembly language program, the first two arguments are transmitted in the BX and DX register pairs, and the location of a block of pointers to the remaining arguments is transmitted in CX. To wit:

```
BX:    location of parameter block
DX:    location of working storage area
CX:    location of pointer block containing:
                pointer to location containing working
                storage area length
                pointer to status return variable
                pointer to collating sequence table, if used
```

approximately 12Ø bytes of stack space must be available.

CALL SORSUB

upon return the completion status (Ø if successful, else error code) is in the A register, with the flags set, and also in the HL register pair and in the status return variable.

The following examples shows portions of an assembly language program that calls SORSUB:

```
            EXT SORSUB       ;DECLARE SORSUB EXTERNAL
            ...

            MOV BX,PARBLK    ;LOCATION OF PARAMETER BLOCK
            MOV DX,WORK      ;LOC WORKING STORAGE ARRAY
            MOV CX,SORARS    ;LOC OF POINTERS TO THE REST
            CALL SORSUB      ;CALL SUPERSORT SUBROUTINE
            JNZ ERROR        ;PSW NON-ZERO IF ERROR
            ...

;BLOCK OF POINTERS TO SORSUB ARGUMENTS 3,4,5
SORARS: DW WLEN             ;LOC WORK AREA LGTH
        DW SSAT             ;LOC STATUS RETURN VBL
        DW COLTAB           ;LOC ALTSEQ TABLE
;
WLEN:   DW 5ØØØ             ;SORSUB WORKING STORE AREA LENGTH
SSTAT:  DS 2                ;STATUS RETURNED HERE
COLTAB: DB Ø,1,2,3,173,25,17, ...  ;COLLATING SEQUENCE TBL
PARBLK: ...      (SORSUB parameter block here)
        ...      (see next section)
WORK:   DS 5ØØØ             ;SORSUB WORKING STORAGE AREA
```

Another example, complete with parameter block, is included as a file on the distribution diskette: file SUBRDEMO.MAC is an assembly language program that calls SORSUB.

3. THE SORSUB PARAMETER BLOCK

The SORSUB parameter block gives the complete specifications for a sort/merge operation, except for the collating sequence table, which is transmitted as a separate argument. The parameter block can specify up to 32 files to be sorted, up to 32 files to be merged only, up to 32 keys, an indefinitely complex record selection criterion, and all other SuperSort features. Yet for simple cases it is compact, due to the use of variable length data structures.

Refer to the previous section for the usage of the parameter block in the SORSUB calling sequence.

In summary, the SORSUB parameter block consists of:

a. Fixed length information

   This includes the output file name, input and output record lengths and other file attributes, the print level, the work file drive, and flags to invoke features such as output options, tagsort, and user-installed XIT routines.

b. Variable length information

   sort input file specifications
   merge-only input file specifications
   key field specifications
   record select specifications

   The input file specifications and the key field specifications each consist of a fixed length block which is repeated for each file or key field to be specified. The number of items, as a 16-bit quantity, precedes each group, and a 16-bit -1 (FFFF hex) must follows each group.

   The record selection specifications are of indefinite length in a variable format to be detailed in the following section.

In detail, the parameter block is laid out as shown in the table beginning on the next page; several parameters are detailed in notes following the table.

All parameters are binary unless otherwise specified; all 2-byte items are stored low order byte first.

Refer to the "Concepts and Facilities" section and to preceding sections of the "Programmer's Guide" for descriptions of the features and options mentioned in the parameter block description.

SORSUB PARAMETER BLOCK

name	byte offset	length, bytes	description
IRECL	0	2	(Maximum) record length for input files
ORECL	2	2	Record length for output file, or zero for same as input files.
	4	4	(reserved, should be zero)
ICRDF	8	1	Input file type (Note 1)
IEOF	9	1	Input files end-of-file options (Note 2)
OFNAME	10	12	Output file name, in FCB format (Note 5)
OFOP	22	2	Bit 15 on for OUTPUT DISKETTE CHANGE
OCRDF	24	1	Output file type (Note 1)
OEOF	25	1	Output file end-of-file option (Note 3)
WDRV	26	1	Work file disk drive (Note 4)
RNOUTF	27	1	non-0 for R-OUTPUT, zero normal
KANOUT	28	1	non-0 for KR-OUTPUT, zero normal
TAGSF	29	1	non-0 for TAGSORT, zero normal
	30	1	(reserved, should be zero)
KONLYF	31	1	non-0 for K-OUTPUT, zero normal
KPOUTF	32	1	non-0 for KP-OUTPUT, zero normal
POUTF	33	1	non-0 for P-OUTPUT, zero normal
XIT1F	34	1	non-0 to invoke use of user-installed XIT1 routine, zero to not call XIT1.
XIT2F	35	1	non-0 to invoke use of user-installed XIT2 routine, zero to not call XIT2.
PRNLVL	36	1	Print level: 0 to 5
	37	1	(reserved; should be zero)
	38	-	end of fixed length part of parameter block

continued on next page...

SORSUB parameter block continued...

name	byte offset	length, bytes	description
NSORFL	38	2	Number of sort input files, 1 to 32. May be Ø if merge-only files are given; must always be present.
Repeat next 4 items for each sort input file; omit if none:			
		12	Input file name, FCB format (Note 5)
		2	(reserved; should be zero)
		2	Start record (Ø = beginning of file)
		2	End record (Ø or FFFF = entire file)

		2	Must contain FFFF hex to mark end of input sort files. Must always be present
NMOFL	after sort input files	2	Number of merge-only input files, 1 to 32. May be Ø if sort input files are given; must always be present
Repeat next 2 items for each merge-only file; omit if none:			
		12	Merge-only file name, FCB format (Note 5)
		6	(reserved; should be Ø)

		2	Must contain FFFF hex to mark end of merge-only input files. Must always be present.

concluded on next page...

SORSUB parameter block concluded...

name	byte offset	length, bytes	description
NKEY	after merge files	2	Number of key fields: 1 to 32.
Repeat the next 3 items for each key:			
		2	Start column (positional field), or field number (comma-delimited field)
		2	Field length in bytes; maximum length for comma-delimited field. Note that length, not end postion, is given.
		2	Field type and attributes - see Note 6

		2	Must contain FFFF hex to mark end of keys. Must always be present
NSEL	after keys	2	Ø to not use record selection, any value 1-32 to use record selection
Omit the next item if select is not to be used:			
		variable length	properly formatted select specification string, as described in next section.
		1	FF hex to terminate select string (FFFF hex in 2 bytes also acceptable).

Notes to SORSUB PARAMETER BLOCK Table

NOTE 1: The file record type is specified by a 1-byte value:

Type	Contents of ICRDF or OCRDF (decimal)
CR-DELIMITED	1
FIXED-LENGTH	2
VARIABLE	64
RELATIVE	128

A zero value is also taken to mean FIXED-LENGTH.
For VARIABLE and RELATIVE input files, always specify NO-SINGLE-Z and NO-ZZZ, by putting 3 in IEOF (next note).

NOTE 2: Input end-of-file indication options for FIXED-LENGTH record files may be specified by putting the sum of one or more of the following in parameter IEOF. For default end-of-file detection, use Ø. For VARIABLE or RELATIVE input, use 3.

Option	IEOF Value
NO-SINGLE-Z	1
NO-ZZZ	2
FFZZZ	4

NOTE 3: parameter OEOF should be Ø for normal output end-of-file last sector fill (1A hex), or 2 for FFZZZ fill.

NOTE 4: The disk drive number for the work file or in a file name is a one-byte quantity in one of the following formats:

BINARY	Ø = current logged drive, 1 = A:, 2 = B:,...
ASCII	space = current, 'A' = A:, 'B' = B:, ...

NOTE 5: The following 12-byte format is used for all file names in the SORSUB parameter block:

Bytes	Contents	Description
Ø	disk drive	see NOTE 4
1-8	file name	ASCII, left-adjusted, blank-filled
9-11	file type	ASCII, left-adjusted, blank-filled

Examples for NOTE 5:

```
    file B:DETAIL.DAT, in assembler:
        DB 2,'DETAIL  DAT'      ;NB 2 blanks between
                                ; DETAIL and DAT
    in FORTRAN:
        LOGICAL NAME(12)
         ...
        EQUIVALENCE(NAME,desired place in PARBLK)
         ...
        DATA NAME/2,'D','E','T','A','I','L',' ',' ','D','A','T'/
```

in COBOL: in the WORKING-STORAGE SECTION of the DATA DIVISION. Note ASCII drive specification; begin the VALUE with a space if current drive is desired.

```
        Ø1 OFNAME USAGE IS DISPLAY PICTURE IS X(12)
                VALUE IS 'BDETAIL  DAT'.
```

NOTE 6: The field type and attributes for each key are specified by a 16-bit quantity in which each bit has a function. For each desired attribute, obtain the value from the following table, then use the sum of these values. For convenience, both hex and decimal values are shown; the bit number column is for information only.

Attribute	Decimal Value	Hex Value	Bit Number
comma-delimited (omit for positional)	1	1	Ø
NUMERIC-ASCII	2	2	1
UPPER-CASE	8192	2ØØØ	13
RIGHT-JUSTIFY	64	4Ø	6
LOHI	4	4	2
MASK-PARITY-BIT	16384	4ØØØ	14
EBCDIC	2Ø48Ø	5ØØØ	12+14
ALTSEQ	1024	4ØØ	1Ø
TWOS-COMPLEMENT (COMPUTATIONAL)	32	2Ø	5
INTEGER	36	24	2+5
FLOATING-POINT	16	1Ø	4
PACKED-BCD (COMPUTATIONAL-3)	128	8Ø	7
DESCENDING (omit for ASCENDING)	256	1ØØ	8

Examples:
comma-delimited, NUMERIC-ASCII, DESCENDING:
use 1 + 2 + 256 or 259.
positional, ASCENDING, UPPER-CASE:
positional and ASCENDING require no specification;
use 8192 to specifiy UPPER-CASE.

All but bits Ø and 8 of the above are also used in coding test attributes in record selection specification strings, described in the next section.

The following is an example of a SORSUB parameter block coded in Assembler. It specifies a run in which two files are sorted, an additional file is are merged with the result of the sort, and the diskette is changed before the output file is written.

```
    ;
    ; SORT SUBROUTINE PARAMETER BLOCK
    ;
    ;     FIXED LENGTH PORTION
    PARBLK: DW 256          ;INPUT RECORD LENGTH: 256 BYTES
            DW Ø            ;OUTPUT RECORD LENGTH: SAME
            DB Ø,Ø,Ø,Ø
            DB 1,Ø          ;INPUT CR-DELIMITED, NO EOF OPTIONS
            DB 2,'RESULT  DAT'      ;OUTPUT TO B:RESULT.DAT
            DW X'8ØØØ'      ;REQUEST OUTPUT DISKETTE CHANGE
            DB 1,Ø          ;OUTPUT CR-DELIMITED, NO EOF OPTIONS
            DB 1            ;WORK DRIVE IS A:
            DB Ø,Ø          ;NO OUTPUT OPTIONS
            DB Ø            ;DO NOT USE TAGSORT
            DB Ø
            DB Ø,Ø,Ø        ;NO OUTPUT OPTIONS
            DB X'FF',Ø      ;INVOKE XIT1, DO NOT INVOKE XIT2
            DB 5            ;MAXIMUM PRINT LEVEL
            DB Ø
    ;   SORT INPUT FILES
            DW 2                    ;NUMBER OF SORT INPUT FILES
            DB Ø,'FILE1   DTA'      ;FIRST FILE IS FILE1.DTA
            DW Ø,1ØØ,455            ;TAKE RECORDS 1ØØ-455
            DB 1,'FILE2   DTA'      ;SECOND FILE IS A:FILE2.DTA
            DW Ø,Ø,Ø                ;ENTIRE FILE
            DW -1                   ;MARK END OF SORT FILES
    ;   MERGE INPUT FILES
            DW 1                    ;NUMBER OF FILES
            DB Ø,'MASTER  Ø7A'      ;MERGE FILE MASTER.Ø7A
            DW Ø,Ø,Ø
            DW -1                   ;MARK END MERGE FILES
    ;   KEY SPECIFICATIONS
            DW 2                    ;NUMBER OF KEYS
            ;  FIRST (HIEST PRIORITY) KEY IS COLUMNS 1Ø TO 14
            DW 1Ø           ;START COLUMN
            DW 5            ;LENGTH
            DW Ø            ;ALL ATTRIBUTES OFF; POSITIONAL
            ;  SECOND KEY IS FIELD #5 LENGTH 17 DESCENDING NUM-ASC
            DW 5            ;FIELD NUMBER 5
            DW 17           ;MAX LENGTH
            DW 1+2+X'1ØØ'   ;BITS: 1 MEANS COMMA-DELIM NOT POSNL,
                            ;      2 SPECIFIES "NUMERIC-ASCII"
                            ;    1ØØ HEX SPECIFIES DESCENDING
            DW -1           ; TERMINATES KEY SPECIFICATIONS
    ;
            DW Ø            ;NO RECORD SELECTION: MUST BE PRESENT
            DW -1           ;TERMINATE THE NULL SELECTION INFO
```

The following is a complete COBOL program that calls SORSUB; it consists largely of the SORSUB parameter block. The SORMSG subroutine, described subsequently, is also used. 16-bit binary values have been adjusted to interchange the bytes, as explained above under the subheading "SORSUB Calling Sequence in COBOL". 8-bit binary values are generated two at a time, using COMPUTATIONAL data items.

```
*   DEMONSTRATION COBOL PROGRAM TO CALL SORT SUBROUTINE.
*
*   SEE "DISPLAY" COMMANDS AT BEGINNING OF PROCEDURE DIVISION
*   (2 PAGES AHEAD) FOR EXPLANATION OF WHAT PROGRAM DOES.
*
*   TO LOAD THIS PROGRAM USE: L80 DEMO,SORLIB/S,COBLIB/S,DEMO/N/E
*
 IDENTIFICATION DIVISION.
  PROGRAM-ID. DEMO.

 DATA DIVISION.
  WORKING-STORAGE SECTION.

* SORSUB WORK AREA, 4096 BYTES LONG.
   77 WORK-AREA DISPLAY PIC X(4096).

* WORK AREA LENGTH
   77 WORK-LEN COMP VALUE 16 PIC 9999.
*          ABOVE VALUE IS 4096 WHEN INTERPRETED AS LOW-ORDER-FIRST.

* STATUS RETURN VARIABLE: SORSUB RETURNS STATUS HERE.
   77 STAT-RETURN COMP PIC 9999.

* DUMMY FOR OPTIONAL COLATING SEQUENCE TABLE
*     - TABLE NOT USED IN THIS SORT, SO FULL TABLE NOT NEEDED.
   77 COLTAB PIC X.

* SORT SUBROUTINE PARAMETER BLOCK
*          THE PARAMETER BLOCK ARGUMENT TO THE SORT SUBROUTINE
*          BEGINS ON THE NEXT PAGE; THE FOLLOWING COMMENTS APPLY:
*
* DATA NAMES CORRESPOND TO DESCRIPTION IN SUPER-SORT MANUAL.
* THE 02, 03, AND 04 LEVELS ARE USED TO IMPROVE READABILITY.
* THE 'COMPUTATIONAL' DATA TYPE IS USED TO GENERATE BINARY VALUES
*     WHERE REQUIRED; THE VALUES ARE ADJUSTED TO PRODUCE THE DES-
*     IRED LOW-HIGH VALUE (AS EXPECTED BY SUPER-SORT) FROM A HIGH-
*     LOW VALUE (AS STORED BY COBOL).
* ONE-BYTE BINARY VALUES ARE GENERATED IN PAIRS, USING A SINGLE
*     'COMPUTATIONAL' ITEM TO GENERATE TWO ADJACENT ONE-BYTE PARAM-
*     ETERS. THE VALUE FOR SUCH ITEMS IS DETERMINED BY MULTIPLYING
*     THE VALUE DESIRED IN THE FIRST PARAMETER BY 256 AND ADDING
*     THE VALUE DESIRED IN THE SECOND PARAMETER.
* DISK DRIVES IN DISK FILE NAMES ARE SPECIFIED IN ASCII WITH A
*     SINGLE LETTER A, B, C ..., OR BLANK FOR CURRENT.
```

```
   01 PARAM-BLOCK.
      02  FIXED-INFO.
          03 IRECL COMP VALUE 20480 PIC 99999.
*                 20480 IS 80 TIMES 256, FOR LOHI STORAGE
          03 ORECL COMP VALUE 0 PIC 9999.
          03 UNUSED-1 COMP VALUE 0 PIC 9999.
          03 UNUSED-2 COMP VALUE 0 PIC 9999.
          03 ICRDF-IEOF COMP VALUE 256 PIC 9999.
*                 ABOVE GENERATES BYTE CONTAINING 1 TO SAY INPUT FILES
*                 ... ARE CR-DELIMITED, FOLLOWED BY BYTE CONTAINING
*                 ... ZERO TO SAY NO EOF OPTIONS.
          03 OFNAME DISPLAY VALUE 'ARESULT  DAT' PIC X(12).
*                 ABOVE REPRESENTS FILE NAME FOR A:RESULT.DAT
          03 OFOP COMP VALUE 0 PIC 9999.
          03 OCRDF-OEOF COMP VALUE 256 PIC 9999.
*                 SEE COMMENT FOR ICRDF-IEOF
          03 WDRV-RNOUTF COMP VALUE 512 PIC 9999.
*                 ABOVE PUTS 2 IN FIRST BYTE, FOR WDRV (USE B:),
*                 0 IN SECOND BYTE FOR RNOUTF.
          03 KANOUT-TAGSF COMP VALUE 0 PIC 9999.
*                 ABOVE GENERATES 2 BINARY ZERO BYTES.
          03 UNUSED-KNOLYF COMP VALUE 0 PIC 9999.
          03 KPOUTF-POUTF COMP VALUE 0 PIC 9999.
          03 XIT1F-XIT2F COMP VALUE 0 PIC 9999.
          03 PRNLVL-UNUSED COMP VALUE 512 PIC 9999.
*                 ABOVE PUTS 2 IN PRNLVL, 0 IN UNUSED BYTE.
      02 SORT-INPUT-FILES.
          03 NSORFL COMP VALUE 256 PIC 9999.
*                 ABOVE IS A 1, FOR ONE FILE, IN HI ORDER BYTE.
          03 SORT-FILE-NAME VALUE 'AA       DAT' PIC X(12).
*                 ABOVE SPECIFIES FILE A:A.DAT
          03 UNUSED COMP VALUE 0 PIC 9999.
          03 START-RECORD COMP VALUE 0 PIC 9999.
          03 END-RECORD COMP VALUE 0 PIC 9999.
          03 END-MARKER COMP VALUE -1 PIC 9999.
      02 MERGE-INPUT-FILES.
          03 NMOFL COMP VALUE 512 PIC 9999.
*                 ABOVE IS A 2, FOR TWO FILES, IN HI ORDER BYTE.
          03 MERGE-FILE-1.
             04 MERGE-FILE-NAME-1 VALUE 'AB       DAT' PIC X(12).
*                 ABOVE SPECIFIES FILE A:B.DAT
             04 UNUSED COMP VALUE 0 PIC 9999.
             04 UNUSED-2 COMP VALUE 0 PIC 9999.
             04 UNUSED-3 COMP VALUE 0 PIC 9999.
          03 MERGE-FILE-2.
             04 MERGE-FILE-NAME-2 VALUE ' C       DAT' PIC X(12).
*                 FILE C.DAT: NOTE BLANK TO INDICATE CURRENT DRIVE
             04 UNUSED COMP VALUE 0 PIC 9999.
             04 UNUSED-2 COMP VALUE 0 PIC 9999.
             04 UNUSED-3 COMP VALUE 0 PIC 9999.
          03 END-MARKER COMP VALUE -1 PIC 9999.
          02 KEY-INFO.
              03 NKEY COMP VALUE 256 PIC 9999.
*                     ABOVE IS A 1, FOR ONE KEY, IN HI ORDER BYTE.
              03 FIELD-NR COMP VALUE 256 PIC 9999.
*                     ABOVE IS A 1, FOR FIELD 1, IN HI ORDER BYTE.
              03 FIELD-MAX-LEN COMP VALUE 5120 PIC 9999.
*                     ABOVE IS 20, TIMES 256 TO MOVE TO HI BYTE.
              03 FIELD-ATTRIBUTES COMP VALUE 16640 PIC 9999.
*                     ABOVE IS 64 FOR RIGHT JUSTIFY,
*                             PLUS 1 FOR COMMA-DELIMITED FIELD,
*                             TIMES 256 TO PUT IN OTHER BYTE.
*                             (64+1)*256 = 16640
              03 END-MARKER COMP VALUE -1 PIC 9999.
          02 SELECT-INFO.
              03 NSEL COMP VALUE 0 PIC 9999.
              03 END-MARKER COMP VALUE -1 PIC 9999.


     PROCEDURE DIVISION.
      REQUIRED-PARA-NAME.
*     EXPLAIN WHAT THIS PROGRAM IS GOING TO DO
         DISPLAY 'THIS PROGRAM SORTS FILE "A:A.DAT" AND MERGES THE'.
         DISPLAY 'RESULT WITH FILES "A:B.DAT" AND "C.DAT" (PRESUMED'.
         DISPLAY 'ALREADY SORTED) TO FORM FILE "A:RESULT.DAT".'.
         DISPLAY 'THE WORK FILE IS PLACED ON DRIVE B.'.
         DISPLAY 'THE SORT KEY IS COMMA-DELIMITED FIELD #1, RIGHT-'.
         DISPLAY 'JUSTIFIED, MAXIMUM LENGTH 20.'.
*     USE SORT SUBROUTINE TO DO IT
              CALL 'SORSUB' USING PARAM-BLOCK, WORK-AREA, WORK-LEN,
                      STAT-RETURN, COLTAB.
*     TEST STATUS RETURNED TO SEE IF AN ERROR OCCURED
              IF STAT-RETURN = 0 GO TO SUCCESS.
*      SORSUB RETURNED ERROR. PRINT MESSAGE.
              DISPLAY "SORT SUBROUTINE ERROR:".
*          NOTE: SPECIFIC ERRORS COULD BE DISTINGUISHED BY TESTING
*             STAT-RETURN FOR 256 TIMES THE SORT ERROR CODE.
*          SORMSG SUBROUTINE - SUPPLIED WITH SUPER-SORT -
*          PRINTS APPROPRIATE MESSAGE. NO PARAMETERS REQUIRED.
              CALL 'SORMSG'.
*     DISPLAY COMPLETION MESSAGE - WHETHER ERROR OR SUCCESSFUL
*               NOTE: IF SUCCESSFUL, SORT SUBROUTINE DISPLAYED
*               "SORT/MERGE COMPLETE", SINCE WE SPECIFIED A NON-0
*               PRINT LEVEL.
      SUCCESS. DISPLAY "DEMO-PROGRAM EXECUTION COMPLETE".
```

4. SORSUB RECORD SELECTION SPECIFICATIONS

All fields of the SORSUB parameter block, PARBLK, were described in the previous section except the optional record selection specification string.

To use SORSUB without record selection, put a Ø in the NSEL field in PARBLK (see table, previous section). You may defer reading this section until you wish to use record selection.

To use SORSUB with record selection, you must code a record selection specification string in the parameter block. Before attempting to understand the coding of SORSUB record selection strings, you should be thoroughly familiar with the SELECT and EXCLUDE commands of the SuperSort main program, as described in the Operator's Handbook. This section does not describe the facilities available; it only covers the method of invoking them in the subroutine version of SuperSort.

The selection specification string consists of a sequence of codes, each of which represents an operand (field or constant), a comparison operator, or a logical operator. Each code is one byte; many of the codes take following bytes to specify information such as field number or comparison test attributes. The individual codes and the following argument bytes each takes will be described presently, via a table.

The record selection specification string must be coded with OPERATORS FOLLOWING THEIR OPERANDS, in a manner analogous to the operation of a reverse-polish-notation calculator. For example, the string correspon-ding to the SuperSort main program command

```
SELECT FIELD #1 < "ABC"
```

is coded in the following order (detailed programming examples will be given later):

```
code for FIELD #1
code for constant "ABC"
code for "less than"
```

Each comparison code must be preceded by two field or constant codes. Each comparison produces a TRUE or FALSE value; SORSUB is capable of remembering several TRUE/FALSE values temporarily while applying the specification string to a record.

The logical operators AND, OR, and XOR use the two MOST RECENTLY GENERATED (but not yet used) TRUE/FALSE values, and produce a single TRUE/FALSE result. Thus, those operators must be coded after the comparisons (or after the preceding logical operators) whose results are to be AND'd, OR'd, or XOR'd. For example,

```
SELECT FIELD #1 > "ABC" AND FIELD #2 = "Ø2138"
```

must be coded in the order

```
codes for FIELD #1 > "ABC"
codes for FIELD #2 = "Ø2138"
code for AND
```

Note that the AND was coded at the end, not in the middle.

NOTE

Some of the difficulties experienced in the writing of select strings can be overcome by the insertion of the special AND operator (Ø9H) just prior to the string termination.

The record selection string, taken as a whole, must produce exactly ONE true/false result. Thus, it must contain a comparison operator, and the operands required by that operator. Any number of comparisons may be coded, provided the right number of AND's (or other logical operators as desired) are coded to combine the multiple test results into one. An error message will occur if the specification string, when finished, leaves more or less than one TRUE/FALSE result, or if it contains the wrong number of operands for the operators coded, or if the operands do not precede the operators.

If application of the specification string to a given input record produces a TRUE result, the record is included in the SORT/MERGE; a FALSE result causes the record to be excluded. To cause a record to be excluded when a condition is met (e.g. when a comparison produces a TRUE result), you may code the desired condition FOLLOWED BY a NOT code.

Operations are done in the order they are specified in the string. There is no heirarchy of operations, nor any provision for coding parentheses. Thus, in coding complex tests, the programmer must compose the string with the operators in the desired order of execution.

There is no explicit code for BETWEEN or NOT-BETWEEN. These operations must be composed with two comparisons and an AND or OR.

The composition of the record select specification string will be detailed via examples after the following table, which details the individual codes of which the string is formed. As in other parts of the SORSUB parameters, all values are binary and all two-byte items are stored low order first.

SORSUB RECORD SELECTION SPECIFICATION CODES

code (hex)	function	follow by	description
0	positional field	2 bytes: start position 2 bytes: length	specifies positional (column-specified) field to be tested by a subsequent comparison operator
1	comma-delimited field	2 bytes: field number	specifies comma-delimited field to be tested by a subsequent comparison operator
2	constant	2 bytes: length (n) n bytes: binary value	specifies a constant to be used by a subsequent comparison operator. Value should have high order byte first except when it is to be compared to a field with low order byte first using the LOHI test attribute.
83	less than)		
C3	less or equal)	2 bytes, test attribute bits,	
43	equal)	same as for key fields except	
A3	not equal)	bits 0 and 8 do not apply.	
63	greater or equal)	Refer to table in Note 6 to the SORSUB parameter block,	
23	greater than)	above, about 7 pages back.	
			the above six comparison operators compare the two most recently specified (but not yet used) fields or constants, producing a TRUE/FALSE result.
5	NOT	--	complement most recent TRUE/FALSE result
6	AND)		perform logical operation
7	OR)	--	on two most recently produced TRUE/FALSE results,
8	XOR)		producing a TRUE/FALSE.
FF	halt	--	terminates specification string. MUST BE PRESENT.

ADDITIONAL SORSUB RECORD SELECTION SPECIFICATION CODES

code (hex)	function	follow by	description
9	special AND	--	same as AND except record is immediately rejected if result is false. Use judiciously to speed execution

NOTE

If you experience unexplained internal errors, try including the code for special AND just prior to terminating the select string.

code (hex)	function	follow by	description
84	special less than	2 bytes, test attribute bits, as above.	these six special comparison operators compare the first operand of the MOST RECENT PREVIOUS COMPARISON to the recently specified (but not yet used) field or constant, producing a TRUE/FALSE result. Use to facilitate coding BETWEEN and NOT BETWEEN tests as shown in illustrative example below.
C4	special less or equal		
44	special equal		
A4	special not equal		
64	special greater or equal		
24	special greater than		

Each of the following examples illustrates one or more basic principles of the formation of record select strings. Each shows a record selection condition, expressed in the form of a command that could be used to produce it in the SuperSort main program, followed by the coding of a select specification string to produce the same selection condition in SORSUB.

Example 1: selection equivalent to the SuperSort program command

```
        SELECT FIELD #7 < "ABC"
```

could be coded as a SORSUB record selection specification string, in assembler, as follows:

```
        ;FIELD #7
        DB      1       ;CODE FOR COMMA-DELIM FIELD
        DW      7       ;FIELD NUMBER OF COMMA-DELIM FIELD
        ;"ABC"
        DB      2       ;CODE FOR CONSTANT
        DW      3       ;LENGTH OF CONSTANT: 3 BYTES
        DB      'ABC'   ;THE 3 BYTES: ASCII ABC
        ;< -- COMPARISON FOLLOWS THINGS TO COMPARE
        DB      X'83'   ;CODE FOR LESS THAN: 83 HEX
        DW      Ø       ;TEST ATTRIBUTES: NONE
        ;TERMINATE SELECT SPECIFICATION
        DB      X'FF'   ;FF MANDATORY AT END
```

Or, in FORTRAN, as follows

```
        INTEGER*1 SELSTR(13)
         ...
        EQUIVALENCE (SELSTR,proper place in PARBLK)
         ...
        DATA SELSTR /1,7,Ø,2,3,Ø,'A','B','C',X'83',Ø,Ø,X'FF'/
```

Note that where a 2-byte quantity (field number 7, etc.) was entered as two single bytes in the the data statement, the LOW ORDER byte was put first, per the 8Ø8Ø convention of storing all two-byte quantities backwards.

Note that the appropriate syntax of the language in use was used to convert the ASCII constant to binary. There is no distinction between constant data types when SORSUB is called; all are coded as code 2 (1 byte), followed by the length in two bytes (low order first), followed by the binary value in "length" bytes, high order byte first. (Exception: constants for comparison to low order first fields, such as FORTRAN INTEGERS, should be stored low order byte first. In this case the LOHI bit must also be set in the comparison operator's test attributes.)

Example 2:

```
        EXCLUDE FIELD 1Ø,14 GE 123P5 PACKED-BCD
```

In assembler:

```
        ;FIELD 1Ø,11
        DB      Ø                       ;CODE FOR POSITIONAL FIELD
        DW      1Ø                      ;START COLUMN
        DW      5                       ;INCLUSIVE LENGTH
        ;123P5 -- MUST ENTER AS HEXADECIMAL EQUIVALENT
        DB      2                       ;CODE FOR CONSTANT
        DW      5                       ;LENGTH: 5 BYTES
        DB      Ø,Ø,Ø,1,X'23'           ;VALUE
        ;GE PACKED-BCD
        DB      X'63'                   ;CODE FOR >=
        DW      X'8Ø'                   ;ATTRIBUTES: PACKED-BCD BIT ON
        ;NOT -- TO MAKE IT EXCLUDE, NOT SELECT
        DB      5                       ;MAKE TRUE FALSE, MAKE FALSE TRUE
        ;TERMINATE STRING
        DB      X'FF'
```

In FORTRAN, the following data statement might be used (SELSTR is a LOGICAL or INTEGER*1 array EQUIVALENCEd into the parameter block):

```
        DATA SELSTR /Ø, 1Ø,Ø, 5,Ø,  2, 5,Ø, Ø,Ø,Ø,1,X'23',
     +              X'63', X'8Ø',Ø, 5, X'FF'/
```

In COBOL, the following code might be used within a SORSUB parameter block being specified at level Ø1 in the working-storage section of the data division. Single-byte values are generated two at a time with the COMPUTATIONAL data type using a value equal to the first byte contents times 256 decimal plus the second byte value.

```
        Ø2 SELSTR.
         Ø3 SELSTR-1 COMPUTATIONAL   PIC 99999 VALUE 1Ø.
         Ø3 SELSTR-2 COMPUTATIONAL   PIC 99999 VALUE 5.
         Ø3 SELSTR-3 COMPUTATIONAL   PIC 99999 VALUE 2.
         Ø3 SELSTR-4 COMPUTATIONAL   PIC 99999 VALUE 128Ø.
         Ø3 SELSTR-5 COMPUTATIONAL-3 PIC 9(1Ø) VALUE 123.
         Ø3 SELSTR-6 COMPUTATIONAL   PIC 99999 VALUE 25472.
         Ø3 SELSTR-7 COMPUTATIONAL   PIC 99999 VALUE 5.
         Ø3 SELSTR-8 COMPUTATIONAL   PIC 99999 VALUE -1.
```

The FORTRAN form of this example shows the byte values required. The first three pairs of bytes happen to have zero in the first byte. In SELSTR-4, 128Ø is 5 times 256 plus Ø. In SELSTR-5, the appropriate COBOL data type was used to generate the constant value; this could be done since an even number of bytes preceded it. In SELSTR-6, 25472 is 63 hex (= 99 decimal) times 256, plus 8Ø hex (= 128 decimal). In SELSTR-8 a -1 is used to generate a byte containing FF hex; the second byte also generated does no harm at the end of the string.

Example 3:

```
        SEL #2 = "SPEC" OR #1 < "-100" NUM AND #2 = "PMT"
```

Note that the default order of operations causes the AND to be done before the OR, that is, as though parentheses were present like this:

```
        SEL #2 = "SPEC" OR ( #1 < "-100" NUM AND #2 = "PMT" )
```

In assembler:

```
        ;FIELD #2 = "SPEC"
        DB      1               ;# FIELD
        DW      2                       ;FIELD NUMBER
        DB      2               ;CONSTANT
        DW      4                       ;LENGTH
        DB      'SPEC'          ;VALUE
        DB      X'43'                   ;COMPARE FOR EQUAL
        DW      0                       ;NO ATTRIBUTES
                ;"OR" IS CODED LATER, AFTER THING TO OR WITH
        ;FIELD #1 < "-100" NUMERIC-ASCII
        DB      1
        DW      1
        DB      2
        DW      4
        DB      '-100'
        DB      X'83'
        DW      2                       ;NUMERIC-ASCII ATTRIBUTE BIT
                ;"AND" IS ALSO CODED LATER
        ;FIELD #2 = "PMT"
        DB      1
        DW      2
        DB      2
        DW      3
        DB      'PMT'
        DB      X'43'
        DW      0
        ; AT THIS POINT THERE ARE 3 SAVED TRUE/FALSE RESULTS:
        ;    RESULT OF   FIELD #2 = "SPEC"       (LEAST RECENT)
        ;    RESULT OF   FIELD #1 < "-100" NUMERIC-ASCII
        ;    RESULT OF   FIELD #2 = "PMT"        (MOST RECENT)
        ; DO "AND" TO COMBINE LAST TWO RESULTS:
        DB      6                       ;CODE FOR "AND"
        ; NOW THERE ARE TWO SAVED RESULTS:
        ;    RESULT OF   FIELD #2 = "SPEC"
        ;    RESULT OF THE AND OPERATION
        ; "OR" THE TWO RESULTS TOGETHER:
        DB      7                       ;CODE FOR "OR"
        ;TERMINATE STRING
        DB      X'FF'
```

Note that the logical operators were coded after their operands, especially that the OR was placed after the AND, since the result of the AND was one of the operands of the OR.

Example 4:

    SELECT FIELD #7 BT "07000", "07999"

Since there is no code for BETWEEN, this must be coded as:

    SELECT FIELD #7 >= "07000" AND FIELD #7 <= "07999"

We will show coding using the "Special less than or equal" operation, which avoids the need to code FIELD #7 twice. In assembler:

```
;FIELD #7 >= "07000"
 DB      1
 DW      7
 DB      2
 DW      5
 DB      '07000'
 DB      X'63'
 DW      0
 ;SAME THING <= "07999"
 DB      2            ;SECOND OPERAND: CONSTANT,
 DW      5            ;OF LENGTH 5,
 DB      '07999'      ;OF VALUE "07999" ASCII
 DB      X'C4'        ;SPECIAL LESS OR EQUAL: COMPARE
           ;...SAME THING AS LAST COMPARED (IE FIELD 7,
           ;...FIRST OPERAND OF X'63' CODE ABOVE)
           ;...TO OPERAND JUST CODED ("07999").
 DW      0            ;ATTRIBUTES: NONE
 ;AND RESULT OF THE >= WITH RESULT OF <=
 DB      6            ;CODE FOR AND
 ;TERMINATE STRING
 DB      X'FF'
```

Another Approach to Record Selection Specifications

The following model is intended to elucidate the coding of record selection specification strings. Since it is a close analogy to the implementation of record selection in SORSUB, it will clarify the coding of complex tests and give you an increased understanding of the wide range of possiblities available. However, if it confuses you, skip it.

SORSUB record select specification strings are coded as though one is writing a program for a hypothetical machine with two stacks, the "Operand Stack" and the "Result Stack". Each record selection specification code may be thought of as an instruction in the machine language of the hypothetical computer. The stacks operate in normal pushdown fashion: the last value stored (pushed) is the first value retrieved (popped). Both stacks are initially empty, and at completion of the program there must be exactly one value in the Result Stack and no values in the Operand Stack.

Each field or constant code, together with its argument bytes, pushes one value onto the Operand Stack of the hypothetical machine.

Each non-special comparison operator code pops two values off the Operand Stack (generating an error if two values are not present), and pushes one value (TRUE or FALSE according to the result of the comparison of the values popped from the Operand Stack) onto the Result Stack.

A NOT code pops one value off the Result Stack (generating an error if no value is present), then pushes the opposite value.

AND, OR, and XOR pop two values off the Result Stack (generating an error if two values are not present), perform the indicated logical operation on those values, and push the result onto the Result Stack. Since the Result Stack operates in stack fashion, and is independent of the Operand stack, AND, OR, and XOR always combine the two most recently generated but not yet used TRUE/FALSE values, regardless of whether these values were generated by comparison operators or preceding logical operators.

When the "halt" code is encountered, the hypothetical machine pops a value off the Result Stack, verifies that both stacks are empty, then uses the TRUE/FALSE value popped from the result stack to determine whether the record just tested should be included in the sort.

5. THE SORMSG SUBROUTINE

The SORMSG subroutine types the appropriate SuperSort error message, if a preceding call to SORSUB ended with an error condition. If the error related to a file, the file name is included in the message. If there was no error, nothing is typed. SORMSG does not type warning messages - these are unconditionally typed by SORSUB during execution.

SORMSG has no parameters and preserves all registers.

SORMSG Calling Sequence in FORTRAN

```
        CALL SORMSG
```

SORMSG Calling Sequence in COBOL

```
        CALL 'SORMSG'
```

SORMSG Calling Sequence in Assembler

```
        EXT SORMSG
        ...
        CALL SORMSG
```

NOTE

Since a call to SORMSG causes all of SuperSort's error message texts to load (these are not normally needed by SORSUB), its memory requirement is substantial. The memory requirement can be reduced, and the texts shortened, with the NOERR load option.

6. THE SORCNT SUBROUTINE

The SORCNT subroutine allows a calling program to have access to most of the data that SORSUB can print on the console, including the number of records input, output, sorted, and merged; the number of records deleted and inserted by SELECT, XIT1, and XIT2; and the size of the output and work files.

SORCNT stores information into an array provided by the caller, as shown in the following table. Each item is two bytes, binary, low order first.

byte offset decimal	FORTRAN or COBOL subscript	description
0	1	number of sort input records read
2	2	number of sort record select rejections
4	3	number of sort XIT1 deletions
6	4	number of sort XIT1 insertions
8	5	number of records sorted
10	6	number of merge-only records read
12	7	number of merge record select rejections
14	8	number of merge XIT1 deletions
16	9	reserved for future use
18	10	number of records merged only
20	11	number of XIT2 deletions
22	12	number of XIT2 insertions
24	13	number of output records
26	14	number of 128-byte sectors in input file
28	15	number of sectors in output file
30	16	number of sectors in work file
32	17	number of sort runs (sort blocks)
34	18	number of merge runs
36	19	number of runs input to merges
38	20	non-0 if record(s) with insuffient fields or characters read
40	21	reserved
42	22	reserved

SORCNT Calling Sequence in FORTRAN

```
             INTEGER IARRAY(22)
             ...
             (use SORSUB)
             ...
             CALL SORCNT(IARRAY)
```

SORCNT Calling Sequence in COBOL

```
             ...
             01 COUNT-ARRAY.
              02 COUNT-ITEM OCCURS 22 COMPUTATIONAL PIC 99999.
             ...
             (use SORSUB)
             ...
             CALL 'SORCNT' USING COUNT-ARRAY.
             ...    (swap high and low order bytes
             ...    of COUNT-ITEMs to be used)
             ...
```

SORCNT Calling Sequence in Assembler

```
             EXT SORCNT
             ...
             (use SORSUB)
             ...
             LXI H,ARRAY
             CALL SORCNT
             ...
      ARRAY: DS 44
```

7. USER-EXIT ROUTINES (XIT1, XIT2)

The user-exit routines are routines which are provided by the user and which the SuperSort main program or subroutine calls for each input record (XIT1) or each output record (XIT2). The user-exit routines can inspect, alter, insert, replace, and delete records. There are many uses for user-exit routines; a few possibilities were listed in the "Concepts and Facilities" section.

Procedures for installing XIT routines are given in the "Loading Procedures" section; calling sequences are given in this section. After a user-exit routine has been installed, it is used only if the appropriate command or parameter block flag was given.

Each XIT routine is called for each each record, and an additional time at the end of the file. Record selection is done on the input records before XIT1 is called.

Each XIT routine is called with three arguments: an input or output record, an empty buffer in which the routine may return a record, and a flag variable used for communication in both directions:

RECORD	A record of the input or output file record length. If the file is CR-DELIMITED, a carriage return, line feed, or control-Z will be present if the record is shorter than the maximum length. If the file is VARIABLE, the first 2 bytes contain the length of the data in the record.
BUFFER	A space of the input or output file record length. A record returned in BUFFER for a FIXED or RELATIVE file should be of the appropriate length (blank padded if necessary); a record for a CR-DELIMITED file may be shorter than the maximum length if it is terminated with a carriage return, line feed, or control-Z; a record for a VARIABLE file should have the data length in the first two bytes, high order first. Records of the entire output file record length will be properly transmitted from XIT2 to the output file even if the output file record length length is longer than the input file record length. The XIT routine should not store into BUFFER beyond the input or output record lengths, plus two bytes for CR-DELIMITED files.
FLAG	At entry to XIT routine: bit Ø on if record is transmitted (only off at EOF) bit 2 on if record is a merge-only input record bit 3 on if this is end-of-file call XIT routine may return (modulo 4 - bits 2-7 ignored): Ø delete RECORD 1 keep RECORD 2 use record in BUFFER instead (such replacement is counted as a deletion and an insertion in in SuperSort's counters) 3 Insert record in BUFFER into file ahead of RECORD. Transmit same RECORD and FLAG to XIT routine again on next call. Beware infinite loops!

There are a few limitations on what FLAG values an XIT routine can return, mostly with regard to when XIT1 can insert or replace a record, as detailed in the following table.

XIT	condition / comments	flag value to XIT	legal flag values for XIT to return
XIT1	each sort input record records may be deleted or kept in all cases. * records may be replaced and inserted only when full record sort or K-OUTPUT is in use, not when another output option or tagsort is in use.	1	Ø, 1, 2*, 3*
	end of last sort input file zero, one, or several records may be added to end of input.	8	Ø, 2, 3
	each merge-only record records may be kept or deleted only.	5	Ø, 1
XIT2	each output record records may be deleted, kept, replaced, or inserted.	1	Ø, 1, 2, 3
	end of output file records may be added to end, or not. 2 means add BUFFER only, 3 means add BUFFER and call XIT2 again.	8	Ø, 2, 3

Form of an XIT1/2 Routine in FORTRAN

```
        SUBROUTINE XITn (RECORD, BUFFER, FLAG)
        INTEGER FLAG
        LOGICAL RECORD (nn), BUFFER (nn)
                             or whatever data type
                             works best for your data
        ...
        store new record into BUFFER if desired
        ...
        FLAG = desired value
                         or may leave unchanged to
                         keep record, not add at EOF.
        ...
        RETURN
        END
```

Form of an XIT1/2 Routine in Assembler

```
        ENTRY XITn
XITn:   ;AT ENTRY: BX POINTS TO RECORD,
        ;          DX POINTS TO BUFFER,
        ;          CX POINTS TO FLAG.
        ...
        MOV AX,[CX]     ;PICK UP FLAG TRANSMITTED BY SORSUB
```

```
...
copy new record into buffer if desired
...
MOV CX,[AX]      ;STORE NEW FLAG VALUE FROM A REGISTER
                     or leave unchanged to keep
                     record, not add record at EOF
...
register contents need not be restored
RET
END
```

8. COLLATING SEQUENCE TABLES (COLTAB, EBCTAB)

The alternate collating sequence table COLTAB may be replaced in the SuperSort main program with a table specifying a sequence of the user's choice. The new table's sequence will then be used for ALTSEQ keys and select comparisons, subject, of course, to additional modification with the COLLATING-SEQUENCE command.

COLTAB is (normally) 256 bytes long, with byte Ø containing the sequence position (Ø-255) for data bytes of value Ø, byte 1 containing the position for data bytes of value 1, etc. The external label COLTAB must be defined at the beginning of the table, and the label ENDCLT at the end. The assembly language source file for the standard table, COLTAB.MAC, is furnished with SuperSort.

The procedure to link a modified COLTAB into the SuperSort program is given in the "Loading Options and Procedures" section. With the subroutine, COLTAB is not used since the caller passes his own table as an argument.

The collating table EBCTAB, used for keys and record select comparisons with the EBCDIC attribute specified, may similarly be replaced. It is 128 bytes long, has label EBCTAB at the beginning, and label ENDEBC at the end. EBCTAB is used by the subroutine as well as the main program.

9. MEMORY REQUIREMENTS

Program or Subroutine	Approximate memory usage in bytes (decimal)
SUPERSORT I, II Main Program	188ØØ
SORSUB	114ØØ
SORMSG - in addition to memory required by SORSUB	2ØØØ
SORCNT - in addition to memory required by SORSUB	25

The reader is also referred to the "Load Options" section for information on reduction of memory requirements obtainable with various load options.

F. LOAD OPTIONS AND LOADING PROCEDURES

This section tells you how to load the SuperSort subroutine (SORSUB) with your own main program, how to reload the SORT main program to install exit routines or collating sequence tables, how to reduce memory requirements by deleting unwanted features, and how to modify SORT and SORSUB for variants of the CP/M with different load addresses or system call entry points. This section applies to SuperSort I only.

The preceding section gave the calling sequences for SORSUB, the anciliary routines SORMSG and SORCNT, and the user-suppliable modules XIT1, XIT2, COLTAB, and EBCTAB.

1. LOAD OPTIONS

There are several options supplied with SuperSort that may be invoked when loading the SuperSort program or when loading the SuperSort subroutine with your own program. The options are described in this section; the means of invoking them is described in the next section, "Loading Procedures".

Eliminating Features to Reduce Memory Requirements

Several modules are supplied which may be loaded with the SORT main program or subroutine and which have the effect of preventing code for certain features from loading, in order to save memory. These are summarized in the table on the next page.

Generally, if a feature that was not loaded is invoked, an error message occurs. The exceptions are: with NOREPORT, the printout is just shortened. With NOCOL and the SUBROUTINE version only, if EBCDIC is requested, the results are unspecified.

MEMORY SAVING LOAD OPTIONS

Module Name	Effect	approximate size reduction, bytes: subroutine	main program
NOERR	much shortened set of error messages tests, as detailed in "Warning and Error Messages" section. * saves memory with subroutine only if SORMSG used.	Ø (1100 *)	1100
NOREPORT	shortens information printed at end of execution. see "Execution Messages" section.	1000	1000
NOCOL	main program: eliminates COLATING-SEQUENCE command and ALTSEQ and EBCDIC keywords in KEY, SELECT, and EXCLUDE commands. subroutine: prevents EBCDIC table from loading.	100	500
NOSEL	eliminates record selection	900	1100
NOCOL and NOSEL together	main program size reduction is greater than sum of each separately	1000	2200

Modifying for a Different Variant of CP/M

The file SYSEQA.MAC contains EQU's which define a number of version-dependent quantities, such as the system call entry point and the program exit jump address. The values distributed are correct for Digital Research CP/M.

If you have a different system, refer to the comments in the file and to the programmer's manual or interface manual for your operating system to determine any changes needed, edit the file as required, assemble, and specify in the L80 command before SORLIB/S whenever loading the SORT main program or subroutine.

The load address (normally 100 hex) can be changed by reloading. No other changes are required.

There is no provision for changing the system call function numbers, the system call specifications, or the disk sector size.

User-Suppliable Modules

The user may install his own version of one or more of the following tables and subroutines whenever he loads he loads the SuperSort main program or subroutine:

name	function	effect if omitted
XIT1	subroutine to inspect each input record and accept, delete, replace, or insert an additional record. Operative only if installed and invoked with appropriate command or parameter flag.	all records accepted
XIT2	similar to XIT1, but operates during output.	same
EBCTAB	collating sequence table for EBCDIC option; may be altered to any desired 7-bit sequence.	standard table loads
COLTAB	collating sequence for ALTSEQ option, before modification by COLATING-SEQUENCE command. Applicable to main program only; with subroutine user table is passed as an argument.	standard table loads (byte value sequence)

The interface specifications for XIT1 and XIT2, and the form of COLTAB and EBCTAB, were described previously in the "Subroutines and Calling Sequences" section.

2. LOADING PROCEDURES

This section gives the procedure for loading the SuperSort program or subroutine, for any of the following purposes: use of subroutine in user's main program, installation of user-exit routines or custom collating-sequence tables, invocation of load options supplied with SuperSort, and/or changing the system symbols defined in SYSEQA.MAC.

By "loading" we mean the creation of an absolute load module (COM file) from the relocatable object (REL files), as opposed to the loading code into RAM for execution.

To use the procedures described here, it is necessary to have a copy of the Microsoft loader (L80). Refer to the documentation supplied with the Microsoft software for information on L80 usage.

Due to the large number of external symbols, loading SuperSort requires 36K of RAM, or more if there is a lot of user-supplied code.

Prepare a working diskette containing L80.COM, the REL files supplied on the distribution diskette, and the REL files for any of your own modules that you wish to install. Note that there is list of the files on the SuperSort distribution diskette in the "Installation" section.

Reloading the SuperSort Main Program

The SuperSort main program is supplied already loaded (i.e. as a .COM file) on the distribution diskette. To reload it to install options, use the following L80 command:

```
L80 SORT,options,SORLIB/S,filename/N/E@
```

Where

filename	is the desired file name for the newly loaded sort. (With versions of L80 older than 3.0, the /N switch is not available; use SAVE after exiting from L80.)
options	is the name(s) of the REL files for any desired load options or user-supplied modules.

L80 should print no error messages and no undefined symbols.

When the SuperSort main program is loaded, nothing must be loaded after SORLIB is searched. If a user-supplied optional module requires code from the FORTRAN library (all programs written in FORTRAN do), insert FORLIB/S before SORLIB/S.

Examples showing reloading of the SuperSort main program:

```
L80 SORT,SORLIB/S,NSORT/N/E
```

reproduces SORT as distributed, on file NSORT.COM.

```
L80 SORT,XIT1,NOSEL,NOCOL,SORLIB/S,XSORT/N/E
```

installs user-supplied module XIT1 and MicroPro-supplied load option modules NOSEL and NOCOL in a sort called XSORT.

Loading User Main Program with SuperSort Subroutine

To load a program of your own that calls SORSUB, and optionally also SORMSG or SORCNT, use the Microsoft loader as follows:

```
L80 program,options,SORLIB/S,filename/N/E@
```

Where

program	is the user's main program module(s)

options	load option REL file name(s) as above
filename	name of COM file to receive loaded code

When the subroutine is used, loading additional modules after SORLIB/S is permissable; therefore, L80's automatic FORLIB search at /E may be allowed to occur.

Note that the subroutines SORSUB, SORMSG, and SORCNT are not named explicitly in the loading command. They are contained in the library SORLIB and those that are called will be loaded during the search invoked by the /S switch.

Examples of SuperSort subroutine loading:

```
L80 SUBRDEMO,SORLIB/S,TEST/N/E
```

loads the sample program (after assembly with M80) supplied on the distribution diskette, onto file TEST.COM.

```
L80 SORTER,USRSUBR,XIT2,NOCOL,SORLIB/S,SORTER/N/E
```

loads the user's program SORTER, the user's subroutine USRSUBR (presumably used in SORTER), the user-supplied SORT optional module XIT2, the MicroPro-supplied optional module NOCOL, and the SuperSort subroutine.

APPENDIX A: EBCDIC COLLATING SEQUENCE AND ASCII CODE TABLE

EBCDIC SEQUENCE POSITION	HEX VALUE	ASCII GRAPHIC	ASCII NAME(S)	
00	00		NUL	
01	01	^A	SOH	
02	02	^B	STX	
03	03	^C	ETX	
37	04	^D	EOT	
2D	05	^E	ENQ	
2E	06	^F	ACK	
2F	07	^G	BEL	
16	08	^H	BS	
05	09	^I	HT	
25	0A	^J	LF	
0B	0B	^K	VT	
0C	0C	^L	FF	
0D	0D	^M	CR	
0E	0E	^N	SO	
0F	0F	^O	SI	
10	10	^P	DLE	
11	11	^Q	DC1	X-ON
12	12	^R	DC2	TAPE
13	13	^S	DC3	X-OFF
3C	14	^T	DC4	/TAPE
3D	15	^U	NAK	
32	16	^V	SYN	
26	17	^W	ETB	
18	18	^X	CAN	
19	19	^Y	EM	
3F	1A	^Z	SUB	
27	1B	^[	ESC	
22	1C	^\	FS	
1D	1D	^]	GS	
35	1E	^^	RS	
1F	1F	^_	US	
40	20		SPACE	
5A	21	!		
7F	22	"		
7B	23	#		
5B	24	$		
6C	25	%		
50	26	&		
7D	27	'		
4D	28	(		
5D	29	)		
5C	2A	*		
4E	2B	+		

EBCDIC SEQUENCE POSITION	HEX VALUE	ASCII GRAPHIC	ASCII NAME(S)
6B	2C	,	
60	2D	-	
4B	2E	.	
61	2F	/	
F0	30	0	
F1	31	1	
F2	32	2	
F3	33	3	
F4	34	4	
F5	35	5	
F6	36	6	
F7	37	7	
F8	38	8	
F9	39	9	
7A	3A	:	
5E	3B	;	
4C	3C	<	
7E	3D	=	
6E	3E	>	
6F	3F	?	
7C	40	@	
C1	41	A	
C2	42	B	
C3	43	C	
C4	44	D	
C5	45	E	
C6	46	F	
C7	47	G	
C8	48	H	
C9	49	I	
D1	4A	J	
D2	4B	K	
D3	4C	L	
D4	4D	M	
D5	4E	N	
D6	4F	O	
D7	50	P	
D8	51	Q	
D9	52	R	
E2	53	S	
E3	54	T	
E4	55	U	
E5	56	V	
E6	57	W	
E7	58	X	
E8	59	Y	
E9	5A	Z	

EBCDIC SEQUENCE POSITION	HEX VALUE	ASCII GRAPHIC	ASCII NAME(S)	NOTE
FF	5B	[		(1)
E1	5C	\		
FF	5D	]		(1)
5F	5E	^		
6D	5F	_		
79	60	`		
81	61	a		
82	62	b		
83	63	c		
84	64	d		
85	65	e		
86	66	f		
87	67	g		
88	68	h		
89	69	i		
91	6A	j		
92	6B	k		
93	6C	l		
94	6D	m		
95	6E	n		
96	6F	o		
97	70	p		
98	71	q		
99	72	r		
A2	73	s		
A3	74	t		
A4	75	u		
A5	76	v		
A6	77	w		
A7	78	x		
A8	79	y		
A9	7A	z		
C0	7B	{		
6A	7C	|		
D0	7D	}	ALT MODE	
A1	7E	~		
07	7F		DEL RUBOUT	

NOTE 1: Haven't identified EBCDIC equivalent, collate at end of sequence.

APPENDIX B: SELECT/EXCLUDE SYNTAX

The many features available for record selection may be combined to form the argument to a SELECT or EXCLUDE command in a manner similar to the formation of expressions in a programming language. This appendix uses a formal notation to show exactly how they may be combined, for the benefit of readers who are facile with such notation and who wish to use complicated record selection criteria.

To express the SELECT and EXCLUDE command syntax, we will use RULES of the form:

```
        <name> := definition
```

Each rule should be read as "<name> is defined as ... ". The following notational conventions will be used in the definition portion of each rule (to the right of the :=) include:

<name>	substitute anything matching rule <name>
[ ]	contents optional
|	means "or": separates alternatives.
( )	enter one of the enclosed alternatives; used to limit range of | 's
{ }	enter zero or more times
" "	enclose (,), [, or] where they are to be entered in command

The SELECT/EXCLUDE syntax specification follows, interspersed with occasional elucidation of the notation and amplification of the meaning. Refer to the Operator's Handbook for introductory descriptions of all forms shown.

```
<select-command> :=   SELECT   (=|+)  <logical-expression>

<exclude-command> :=  EXCLUDE  (=|+)  <logical-expression>

<logical-expression> := <and-expression> {(OR|XOR) <and-expression>}

<and-expression> :=  <sub-expression> { AND <sub-expression> }

<sub-expression> :=  <comprison> | NOT <sub-expression>
                    | "(" <logical-expression> ")"
```

The preceding rules say that the argument to SELECT or EXCLUDE is a logical-expression, and that a logical-expression consists of comparisons, optionally preceded by NOT, connected with AND, OR, and XOR, optionally grouped with parenthesis.

Careful reading of the rules reveals that after NOT, any sub-expression is accepted, including another NOT or a parenthesized logical-expression, and that inside parentheses, any logical-expression is accepted, including more parentheses.

In the absence of parentheses, the order of operations is: NOT, AND, OR and XOR. Operations at the same level are done left to right.

```
<comparison> :=  <value> <compop2> <value> { <attribute> }
               | <value> ( BT|NB > <value>, <value> {<attribute>}

<compop2> :=  LT | LE | EQ | NE | GE | GT | < | <= | = | <> | >= | >

<attribute> :=  UPPERCASE | RIGHT-JUSTIFY | LOHI | MASK-PARITY-BIT
              | EBCDIC | ALTSEQ | TWOS-COMPLEMENT | COMPUTATIONAL
              | INTEGER | FLOATING-POINT | PACKED-BCD
              | COMPUTATIONAL-3
```

A comparison consists of a value, a comparison operator, and another value, and a third value if the operator was BT or NB, and an optional list of test attributes. The test attributes are described in the Operator's Handbook.

```
<value> :=  FIELD start-posn, end-posn | [FIELD] # field-number
          | <constant-list>
```

A value consists of a field specification (as described in Operator's Handbook) or a constant-list.

```
<constant-list> := <constant> | "[" <constant> { <constant> } "]"

<constant>  :=    "<text>" | <numeric-constant>
```

A constant-list consists of one constant, or a list of constants enclosed in [ ]'s. Each constant may be text enclosed in quotes, or a numeric constant. The constants in a list are evaluated individually, then concatenated.

```
<numeric-constant> :=  [+|-] <digits> [ H | T | I | P | Q [<size>] ]

<digits>    one or more digits Ø-9, also A-F if hexadecimal

<size>      decimal number indicating desired length in bytes
```

A numeric constant consists of an optional sign, one or more digits, and an optional base indicator: H (hexadecimal), T (decimal, stored high order first), I (decimal, stored low order first), P (packed BCD decimal), or Q (octal). If no base indicator is given, the default is decimal (T), except hexadecimal if one of the digits A-F is present. If a base indicator is given, it may be followed by a decimal number specifying the number of bytes the constant is to occupy.

No imbedded blanks are allowed between the digits, base indicator, or length. The user should be careful to always put a blank or comma after each numeric constant, especially if the next character would otherwise be a letter or digit.

Any number of digits that fit on one input line may be given. The digits are converted in the specified radix to a byte string of the necessary length (ignoring leading zeroes). The entire string is twos-complemented if a - sign was present.

If a <size> is specified, the constant is extended with leading fill bytes if necessary. The fill is zero, except all 1 bits if a - was present in other than P constants.

Note that the conversion algorithm does not force the first bit to be a sign; it only goes to another byte if the significance overflows. Thus, 65535, -1T2, and FFFF all have the same 2-byte value.

The I base indicator implies the LOHI attributes.

P constants are slightly different: A + or - sign causes an appropiate trailing nibble to be added; no trailing sign is included if neither + nor - is given; and any fill required by a specified length is done with Ø's. Also, the PACKED-BCD attribute is implied for a comparison if a P constant appears anywhere in it.

Octal constants greater than 377Q are packed three bits to a digit, bridging byte boundaries. Use a constant list if you wish to put three digits in each byte.

Note that there is no implicit interaction between the field length and the constant length. Since SuperSort uses blank, not zero or sign bit, fill when comparison operands are of unuequal length, the user of binary data must explicitly specify constants of the same length as the field being tested.

This page intentionally left blank

APPENDIX C: SUPERSORT DIFFERENCES FROM PRIOR RELEASES

SuperSort 1.60 now handles files to the CP/M file size limit. No initial disk reset is effected when operating under MP/M. If initial disk change is desired under MP/M the new command CHANGE must be issued.

SORT's temporary file is now named <output filename>.$$$ instead of SORT.$$$ to eliminate conflicts in Multiuser Systems. Note that under certain conditions if two users invoke a sort of the same file on the same drive conflict may still occur.

SuperSort contains additions that permit it to input and output COBOL-style "relative" and "variable length sequential" files, to sort and select records on the basis of binary integer and floating point data, and to produce three new types of optional output files. The maximum record length has been increased to 4096. User-exit routines can now add multiple records at end of file. A summary of the changes follows; refer to "Concepts and Facilities" and other manual sections for detailed information.

General

The minimum memory requirement for the SORT main program is now 26K.

SuperSort is sold in two versions: SuperSort I (complete), and SuperSort II (without relocatable code, permitting customization and use as a subroutine).

Command Additions in the SORT Main Program

NEW FILE AND RECORD TYPES: VARIABLE or RELATIVE may now be specified in the INPUT-ATTRIBUTES and OUTPUT-FILE commands.

OUTPUT OPTIONS: Five forms of optional output can be invoked, by specifying one of the following in the OUTPUT-FILE command:

K-OUTPUT R-OUTPUT P-OUTPUT KR-OUTPUT KP-OUTPUT

FIXED POINT AND FLOATING POINT BINARY DATA: The following test attributes may now be used in KEY, SELECT, and EXCLUDE commands:

LOHI	TWOS-COMPLEMENT	COMPUTATIONAL
INTEGER	FLOATING-POINT	COMPUTATIONAL-3

The meaning of INTEGER has been changed to imply low-high storage; TWOS-COMPLEMENT was always present, but not documented.

The "I" base indicator for numeric constants for use in SELECT and EXCLUDE commands has been changed to imply low order first storage; the "T" base indicator has been added, with the former meaning of "I".

COMPATIBILITY: All command changes are additions, and all old commands are still present, with the following exceptions: The INTEGER test attribute has been changed to imply low order first storage; the "I" base indicator for SELECT/EXCLUDE numeric constants now implies low order first storage; the obsolete OUTPUT-FILE keywords RECORD-NUMBER-OUTPUT and KEYS-AND-NUMBERS are still accepted, with their old meanings, but with longer minimum abbreviations.

Changes Applicable to Main Program and Subroutine

RECORD LENGTH: The maximum record length is now 4096.

USER-EXIT ROUTINES may now add multiple records to the end of the output or the end of the sort input.

PACKED DECIMAL in keys and record selection tests: Packed decimal values now compare correctly even if the operands differ in sign convention or presence or absence of sign.

Sort Subroutine Changes

Disk drives in the parameter block may now be specified with an upper-case ASCII letter (or space for currently logged drive), as well as in binary as before.

Parameters formerly documented as "reserved - should be zero", and unused bits in certain existing parameters, are now used to invoke new features. All calls set up for prior releases will work as before, provided all "reserved - should be zero" parameters, and all undefined bits, are zero. In particular, there is no change in the effect of previously defined attribute bits, despite the non-compatible change in the meaning of INTEGER in the SuperSort main program. Refer to the "Programmer's Guide" for details.

APPENDIX D: INFOSTAR FILE SORTING

InfoStar File Sorting

InfoStar's file format makes it very easy to select or exclude and sort the records in a data file (.DTA). InfoStar's .DTA files are always comma-delimited fields within carriage-return-delimited records. To sort such records always remember to specify CR-DEL in the input command.

For example:

```
*INPUT = 100, CR-DEL
```

would specify a carriage-return-delimited record which has a maximum length of 100. Note that when the length is specified it must be greater than the length of the longest record in the file.

Normally no special attributes need be given to the input and output filenames.

When specifying the key to be used, it's field # can be specified along with that field's maximum length. Usually no attribute need be given because the data is simply ASCII text. Specifying no attribute, however, will not correctly sort a field of unpadded numbers (i.e. 2, 12, 2083, 84). To get the varying lengths of the numbers in the field properly ordered would require the NUMERIC-ASCII attribute.

Subsets of InfoStar files can also be created using the select/exclude capabilities of SuperSort. Remember that each "sub" datafile needs its own (.NDX) index file. If you have just erected a sub datafile with SuperSort and now wish to once again enter data into it with InfoStar read the section in this Appendix on Creating an Index File.

Creating an Index File

Assuming that you have a datafile in a format acceptable to InfoStar, an Index file (.NDX) can be created for it by SuperSort. The following example will illustrate the procedure.

```
INPUT = 100, CR
SORT = DATA .DTA
OUTPUT = DATA .NDX , fixed, KP
KEY = #1, 12, #2, 24, #1, 2
GO
```

Notice the [fixed, KP] which has been appended to the output filename. This specifies that the output will consist of the concatenated keys in fixed position format in addition to pointers. These keys and pointers are then used by InfoStar as in index to the datafile. The key(s) which are specified, and their associated lengths, must correspond exactly with the key field(s) which were defined on the InfoStar form for which they are targeted. It is very important that both the length and the order of the key fields match the InfoStar form.

The duplicate entry for field #1 in the above example is used to insure that an additional two bytes is reserved for InfoStar's later use. Note that what is actually in these positions is not important at this time but it is very important to specify two additional bytes over and above the space needed for the key(s). Using the fixed, KP output option and the correct key specification, the index file will be very quickly generated.

File Maintenance

The following two sorts are necessary to do a complete file maintenance on a InfoStar file.

1. Sort the datafile into the order dictated by the Key excluding deleted records as we go.

2. Using fixed keys and pointers output, extract an index file (type .NDX) from the now sorted data (.DTA) file.

An example of sorting a datafile into key order is shown below:

```
*INPUT = <max record length>, CR-DEL
*SORT = <input filename>.DTA
*OUTPUT = <output filename>.DTA
*KEY = #1, 12, #2, 24
*EXC = Field 1,1 = ØFFH
*GO
```

In this example the key fields we were sorting to, were field #1 and field #2 which had lengths of 12 and 24, respectively. In most cases the key fields on your InfoStar form and the keys you specify to SuperSort (along with their lengths) are the same.

Notice the EXClude line. This line eliminates any deleted records (i.e. those which begin with a hex FF).

Extraction of the index file once the datafile is sorted and purged of deleted records is described in the previous section on Creating an Index File for InfoStar.

** SUPER-SORT 1.5 ** COMMAND SUMMARY **

COMMAND KEYWORD	DESCRIPTION	POSITIONAL PARAMETER(S)	DESCRIPTION	ATTRIBUTE KEYWORD	DESCRIPTION
INPUT-ATTRIBUTES=	Specify format of input records	number 1 to 4096	(Maximum) record size	(FIXED-LENGTH)	Record length identical for all records
				(CR-DELIMTED)	Varying records separated by CR-LF
				(VARIABLE)	COBOL-type variable length records
				(RELATIVE)	COBOL "relative" file
SORT-FILES(=) (+)	Specify input file(s) to sort	[d:]filename[.typ],..	sort filenames		
MERGE-FILES(=) (+)	Specify input file(s) to merge	[d:]filename[.typ],..	merge filenames		
KEY(=) (+)	Specify one or more sort and/or merge keys	(start-col,end-col)	Specify sort/ merge key field location	(ASCENDING)	Specify ascending sequence for this field
		(#fld-num,max-size)		(DESCENDING)	Specify descending sequence for this field
				NUMERIC-ASCII	Specify signed/unsigned numeric text string with unaligned decimal point position and optional exponential notation
				RIGHT-JUSTIFY	Specify right justification
				UPPER-CASE	Treat lower case as upper case
				LOHI	Compare field right to left
				TWOS-COMPLEMENT	Signed binary stored hi-low (COMPUTATIONAL)
				INTEGER	Signed fixed point binary stored low-high
				FLOATING-POINT	MicroSoft single or double precision
				PACKED-BCD	Packed decimal (COMPUTATIONAL-3)
				MASK-PARITY-BIT	Ignore high order bit in each field byte
				ALTSEQ	Use alternate collating sequence, this field
				EBCDIC	Collate this field as though in EBCDIC
OUTPUT-FILE=	Specify output file	[d:]filename[.typ]	output filename	no-entry	Output file to consist of full records
				(R-OUTPUT)	Output file of record numbers only
				(P-OUTPUT)	Output file of sector/offset pointers only
				(K-OUTPUT)	Output concatenated sort/merge keys only
				(KR-OUTPUT)	Output concatenated keys, record numbers only
				(KP-OUTPUT)	Output concatenated keys, pointers only
WORK-DRIVE=	Specify drive for work file	d:	drive letter		(current drive is used if this command ommitted)
(SELECT) (=)	Specify record selection or exclusion	[NOT]	Specify negation	NUMERIC-ASCII	Refer to the previous definitions
(EXCLUDE) (+)		(start-col,end-col)	Specify field to examine	RIGHT-JUSTIFY	of these attribute keywords
		(#field-number)		UPPER-CASE	
		<,<=,=,>=,>,<>, BT,NB,LT,LE, EQ,NE,GE,GT	comparison operator	TWOS-COMPLEMENT	COMPUTATIONAL
				INTEGER	FLOATING-POINT
				PACKED-BCD	COMPUTATIONAL-3
		(literal-constant)	Specify comparison value	MASK-PARITY-BIT	LOHI
		(start-col,end-col)		ALTSEQ	EBCDIC
		(#field-number)			
		[AND, OR, XOR ...]	logical operator		(follow logical operator with another comparison definition)
CFILE=	Invoke command file	[d:]filename[.typ]	command file name		

Notes:
1. Commands may be entered in any order. Keywords may be abbreviated with the first two letters.
2. An = sign indicates replacement of any previous entry for this command keyword. A blank in this position means the same.
3. The + sign indicates additional values for this command keyword.
4. Positional parameters must follow the associated command keyword in the order indicated.
5. Attribute keywords may be entered in any order following the field definition or comparison definition referred to.
6. Use as many attribute keywords as is necesary to describe a KEY or SELECT/EXCLUDE field.
7. SELECT/EXCLUDE syntax is:

```
(SELECT) <field or literal> operator <field or literal> [attribute-keyword(s)], [AND],...
(EXCLUDE)                                                                        [OR],...
                                                                                 [XOR],...
```

8. Brackets [] enclose optional items.
9. Vertically stacked parenthetical items in a column indicate mutually exclusive choices.
10. Use this table as a reference guide only. Refer to the appropriate manual sections for more information.