ALPHA MICROSYSTEMS
RIGHT. FROM THE START.

SuperSort User's Guide

DSM-00195-01

REVISIONS INCORPORATED	
REVISION	DATE
00	April 1994
01	August 1996

SuperSort User's Guide
To re-order this document, request part number DSO-00195-00.

This document applies to SuperSort versions 1.0 and later.

ALPHA MICROSYSTEMS
2722 S. Fairview St.
P.O. Box 25059
Santa Ana, CA 92799

Table of Contents

Chapter 1 - Introducing SuperSort

SuperSort is a high-performance replacement for the standard sorting software that comes with AMOS. This new implementation uses modern sorting techniques to provide sorting designed for today's computer configurations.

Although SuperSort can be used to sort a sequential file from AMOS command level, it is primarily designed to be used by your AlphaBASIC and AlphaBASIC Plus application programs to substantially increase the speed with which they sort data in sequential and random files.

This document contains information on using the SuperSort program interface from within your AlphaBASIC, AlphaBASIC Plus, assembly language, and AlphaC programs. SuperSort is directly compatible with the older AMOS sort module, and your application software can make use of SuperSort with no programming changes whatsoever—all you have to do is install SuperSort onto your computer system and load it into memory to see an automatic increase in the speed of the applications that sort data.

Of course, if you want to change your application programs' sort interface, you will be able to take advantage of SuperSort's advanced sorting features:

- **Sorting Based on the Requirements of National Languages.** Different cultures and languages have different expectations of how characters should be sorted. For example, a Spanish customer requesting data between the range of "luz" and "maca" expects to see included in that range the words "llama," "lleno," and "lluvia," but the simple language-specific collating sequence used by the older AMOS sorting software will not provide these results. (See "Collating Sequences and Culturally Expected Results," later in this chapter for more information on this feature.)
- **Unlimited AlphaBASIC and AlphaBASIC Plus Keys.** In the case of AlphaBASIC and AlphaBASIC Plus, SuperSort expands the previous limit of three sort keys to an unlimited number by allowing your program to point to an array specifying a table of keys. The new AlphaC and assembly language interfaces also allow an unlimited number of keys.
- **New Key Types Supported.** In addition to the traditional key types of string, AMOS 6-byte floating point, and binary, you may now also use IEEE 4-byte floating point, IEEE 8-byte floating point, and integer keys.
- **Variable Length Records Supported.** When using SuperSort to sort sequential files, you may specify variable length records. This increases efficiency in speed and memory use, because records do not have to be "padded" to a fixed length.
- **Additional Features for SORT.LIT.** The SuperSort replacement for the AMOS SORT command provides several new features:

  Although each sort key must begin in the same character position in every record, the records may vary in length if you request the variable record length option.

  You may request sorting be done using a culturally correct collating sequence for a specific language.

- Program Interfaces for Assembly Language and AlphaC Programs. The older AMOS sort program provided a sorting program to be used at AMOS command level and an interface for AlphaBASIC and AlphaBASIC Plus programs. SuperSort provides all of these, plus programming interfaces for assembly language and AlphaC.

COMPATIBILITY

SuperSort is compatible with AMOS 1.4C and AMOS 2.2C and later. Programming interfaces are provided for AlphaBASIC, AlphaBASIC Plus, AlphaC, and assembly language.

SUPERSORT: TODAY'S SORT PROGRAM

SuperSort has been optimized for modern computer configurations and applications. The older AMOS sort program, AMSORT.SYS, was developed during the days when memory was at a premium, and was therefore optimized for a small memory partition that could not contain an entire data file. In fact, when used in small memory partitions (80KB or less), AMSORT.SYS is still very efficient. But, increasing the memory partition does not improve AMSORT.SYS's performance. On the other hand, SuperSort has been written to take advantage of today's larger memory partitions, and shows a substantial speed increase over AMSORT when used in memory partitions larger than 80KB.

In the following chapters, you will learn how to install SuperSort, and how to use it from within your application programs.

SORTING DEFINITIONS

This section introduces you to some of the terms you need to know when sorting data. If you are already familiar with these terms, you may want to move to Chapter 2, "Installing SuperSort.":

- Key
- Record and Field
- Sequential file
- Random file

Key

Whenever you sort data, you need to identify which piece of data you want to base the sort on. The item on which you base the sort is called the "key." For example, if you have a list of customer names and phone numbers, to find the phone number of a particular customer, it would probably be most convenient to have the list sorted alphabetically by customer name. In this case, the customer name is the key. If your data is more complicated, you might have more than one sort key. For example, if you have a list of customers with addresses, in order to send a mailing by bulk mail, you need to sort your mailing labels by the address zip code. Once the data is in order by zip code, you will probably want to sort on the customer name. In this case, your first key is the zip code, and the second is the customer name.

Record and Field

A record is a collection of related data. For example, a customer name, address, and phone number might make up a single record. Each of these separate items in a record is called a "field," and each field can also be used as a sort key. Each data file usually consists of many records. Using our example of a record consisting of customer name, address, and phone number, a set of three records might look something like this:

```
Avian Exotics         670 San Pablo, San Clemente, CA 92672       714 496 8999
Parrot Paradise       167 Via Estrella, San Clemente, CA 92672    714 496 5678
Tropical Bird Farm    2701 N. Elm St., Santa Ana, CA 92704        714 875 1256
```

Note that each field begins at the same character position in each record.. You will need to know the character position of each key you want to sort on when you use the AMOS SORT command or the SuperSort program interfaces.

Random and Sequential Files

A random file is one in which the records are physically grouped together in one area of the disk, and where any point within that file can thus be found immediately by calculating an offset from the file's beginning. A sequential file's records are not necessarily contiguous on the disk, but are linked in sequence by pointers in each segment indicating where on the disk the next segment can be found. Because it is quick to locate a record in a random file, random files are commonly used by application programs to store data.

You can only use the AMOS SORT command to sort sequential files. The AlphaBASIC, AlphaBASIC Plus, assembly language, and AlphaC program interfaces to SuperSort allow you to sort either random or sequential files, but you set up the SuperSort interface slightly differently for each type of file.

COLLATING SEQUENCE AND CULTURALLY EXPECTED RESULTS

We are all used to sorting information. From the time we are children, we learn to find a name in a phone book by scanning the names alphabetically. The set of rules we use to sort data is called a "collating sequence." (One such rule is, "a" comes before "b.")

In order for computers to sort, numeric values are assigned to characters, and the computer sorts the characters by comparing those values. Traditionally, the collating sequence used by computers has been the ASCII standard, which assigns numeric values to printable characters "a" through "z," "0" through "9," and punctuation, as well as non-printing characters such as Control-C.

However, if the language we are using contains special characters, computers using the ASCII standard sort data in ways we don't expect. For example, a Spanish speaker expects that "ñ" comes after "nz" and before "o," but using the ASCII collating sequence does not provide these results.

To sort data according to the requirements of a national language (that is, to achieve "culturally expected results"), a new collating sequence must be used. Before SuperSort, the Alpha Micro language definition files were used to specify the correct collating sequence. Although the language definition file comes closer to reflecting the culturally expected results than the ASCII standard, it cannot handle special

multiple characters like the Spanish “ch” or the ligature “æ.” However, by using the SuperSort collating option, you can achieve true, culturally correct results.

If you request the optional SuperSort collating feature, SuperSort selects the appropriate collating sequence based on the language used by the job performing the sort. For example, if your job has used the AMOS SET LANGUAGE command to select German as the language you are going to use, SuperSort uses the proper collating sequence for German. For details on the collating rules used by SuperSort’s collating sequences, see Chapter 7, “Language-specific Rules for Collating.”

SuperSort supports the following languages:

ENGLISH
FRENCH
GERMAN
SPANISH
DANISH

If you are using a language other than these, you will need to define your own collating sequence table. See Chapter 7 for instructions on doing so. To see what languages are supported on your computer, type:

SET LANGUAGE RETURN

You now see something like this:

```
Current language is ENGLISH(AMERICAN)
The following alternate languages are available:
     FRENCH (FRANCAIS)
     ENGLISH (AMERICAN)
```

Chapter 2 - Installation

SuperSort will overwrite some of the existing sort programs on your computer. Before installing SuperSort, decide whether you want to preserve the older software. If you do, enter the following commands:

```
LOG OPR: RETURN
COPY *.SAV = SYS:SORT.LIT,BAS:BASORT.SBR,BP:BASORT.XBR RETURN
```

To install SuperSort, download the SuperSort files from the software media. In the case of a streamer tape, your command will look like this:

```
LOG OPR: RETURN
MTUSAV = ALL:[] RETURN
```

If you have questions about the correct transfer command to use, contact your dealer for help. The command above installs the SuperSort files into the proper accounts. To verify that all files copied correctly, use the AMOS VERIFY command:

```
LOG OPR: RETURN
VERIFY SSORT.DIR RETURN
```

ENTERING THE PRODUCT INSTALLATION CODE

You must enter the Product Installation Code (PIC) to activate SuperSort. Contact your dealer for the PIC for your computer system. The first time you attempt to sort a file using SuperSort (either within your own program or when using the SORT command), you will be asked to enter the PIC. The computer will lock out other users for a few seconds while the PIC is being processed. Now SuperSort is ready to use.

WHAT'S INCLUDED?

The SuperSort product consists of the following files:

SYS:SSORT.SYS, SYS:SST000.OVR	SuperSort modules. Used by the AMOS SORT command and AlphaBASIC, AlphaBASIC Plus, assembly language, and AlphaC program interfaces.
SYS:SORT.LIT	Replacement for the standard AMOS SORT command; standalone sort program that can be used from AMOS command level.
BAS:BASORT.SBR	AlphaBASIC external assembly language subroutine; interfaces to SuperSort. Replaces older subroutine.
BP:BASORT.XBR	AlphaBASIC Plus external assembly language subroutine; interfaces to SuperSort. Replaces older subroutine.
BAS:SRTSYM.BSI, BP:SRTSYM.BPI, MAC:SRTSYM.M68 , MAC:SRTSYM.H	Include files giving sample key table array definitions for the different program interfaces supported by SuperSort.
LCS:COLLAT.SYS	Module used when you use SuperSort's culturally correct sorting feature.
LCS:LDFTBL.H	Source to COLGEN program, which can be changed to define new collating sequence.
LCS:COLGEN.C, LCS:COLGEN.LIT, LCS:COLGEN.RTI	Collating sequence table generation program. Uses LDFTBL.H to define the correct collating sequence for the language your job is using, CULCMP.lng
OPR:SSORT.DIR	File used by AMOS VERIFY command to verify correct installation.

Chapter 3 - The AMOS SORT Command

The AMOS SORT command sorts a sequential file from AMOS command level. The SuperSort SORT command acts very much like the older one—except, of course, that it's faster!

However, additional options are included:

1. You may request sorting be done using a culturally correct collating sequence for a specific language.
2. Although each key must begin in the same character position in every record, the records may vary in length if you request the variable record length option.

For your convenience, we have included on the next few pages a command reference sheet for the SuperSort SORT command; please insert it in your alphabetically ordered command reference sheets in your *AMOS System Commands Reference Manual*.

sort

FUNCTION:

Sorts data records in a sequential text file..

CHARACTERISTICS:

SORT.LIT can be loaded into system memory. It sorts logical records in ascending or descending order, and sorts only sequential files, not random files. Before SORT sorts your file, it loads SSORT.SYS into user memory if that file is not already in system memory. SORT replaces the file you specify with a sorted file of the same name.

If you do not use the /C switch, SORT performs a simple sort based on the ASCII collating sequence. If you use /C without specifying a language file extension, SORT performs a culturally correct sort using the collating sequence for the language your job is set to. If you use /C and specify a language file extension, SORT sorts using the collating sequence for that language.

You may sort a text file too large to fit into memory all at one time. SORT does not understand wildcard symbols.

FORMAT:

```
SORT {/C{:lng}}{/V} filespec
```

where:

`filespec`	Selects file you want to sort. SORT assumes an extension of .DAT and the account and device you're logged into.
`/C{:lng}`	Selects sorting based on culturally correct collating sequence. If you do not specify ***lng***, SORT assumes the language your job is set to. (***lng*** is the three-character extension of the language definition file on your computer specifying the language you want to use. For example, /C:USA indicates you want sorting to be done based on the collating sequence for U.S. English.)
`/V`	Specifies variable length records will be used.

OPERATION:

Enter SORT and the specification of the file you want to sort. For example:

`SORT LABELS.DAT` RETURN

SORT now asks you for the following information. After you have entered the information for all of the keys you want to use, just press RETURN the next time it asks for key information to end the questions.

RECORD SIZE	If you don't use the /V option, SORT recognizes a RETURN as the end of each data record, but needs to know the size of the largest data record it is going to be dealing with. Enter the maximum size (in bytes) of the logical records in your file. Every character is one byte of data, including spaces and punctuation. Exclude carriage return and line-feed bytes.
KEY SIZE	The key is the field in the record on which you want to sort (for example, customer name). SORT asks this question for each key you define. Enter the size (in bytes) of the key.
KEY POSITION	SORT asks this question for each of the keys you define. Enter the column number in the record where the first byte of the sort key occurs. The first byte of a record is position #1.
KEY ORDER	SORT asks this question for each of the keys you define. Enter an A for ascending or D for descending order.

If you are using the /V option to select variable length records, SORT asks:

```
Enter record ending (0=CR, 1=LF, 2=CR&LF)
```

SORT now sorts the file. After the sort is done, SORT displays statistics.

MESSAGES:

`?Cannot delete [filespec] - write protected`
Write-enable your disk and try again.

`?Cannot open [filespec] - file type mismatch`
You tried to sort a random file—you may only sort sequential files.

`?Enter A or D`
Answer A for ascending order, or D for descending order.

`?Illegal key size...`
Either the record size is too small or the key size is too big. Re-enter the numbers, adjusting one or the other.

`?Insufficient memory`
Delete any unnecessary memory modules from your user memory, or see your System Operator about increasing your memory.

```
?Key size must be > 0
?Key size must be less than record size
?Record size must be > 0
```
Enter a correct number.

```
?Sort error - Device (VDK0:) has overflowed!
```

The virtual disk (VDK0:) was used for temporary file storage, and that temporary file became too large, aborting the COPY. Either increase the VDK size, or remove the temporary (*.SRT) name from your VDK.INI file.

```
%Warning - A null byte was encountered in the input file.
Nulls are discarded and not rewritten to the sorted file,
resulting in lost data.  You may press ^C at this time to
terminate the sort and leave your data intact, or press
RETURN to continue.
```

There is at least one non-ASCII character in the file you are sorting. Make sure you specified the correct data file; trying to sort a program file would cause this kind of error.

```
?Main sort module not found.
```

If SSORT.SYS is not found in memory, you see this message

Chapter 4 - AlphaBASIC and AlphaBASIC Plus Program Interface

This chapter discusses:

- What is the interface between your AlphaBASIC or AlphaBASIC Plus program and SuperSort?
- How do I use the SuperSort interface?
- How do I sort a random or sequential file?

WHAT IS BASORT?

BASORT is an external assembly language subroutine called by your AlphaBASIC or AlphaBASIC Plus program as the interface to SuperSort. (If your program is AlphaBASIC, you use BASORT.SBR; if it is AlphaBASIC Plus, use BASORT.XBR.)

The two main types of AlphaBASIC data files are random files and sequential files. Using BASORT, you can use SuperSort to sort either type of file. See Chapter 1 for a discussion of sequential and random files.

You can use BASORT to sort the contents of a file into numeric order, a list of names or words into alphabetic order, and so on. BASORT permits the use of an unlimited number of keys if you define a table of keys in an array, and point BASORT to that array. See the details in the sections below for more information.

BEFORE YOU CAN USE BASORT

SSORT.SYS must be loaded into either system or user memory prior to running an AlphaBASIC program that uses BASORT. To load SSORT.SYS into user memory, enter:

```
LOAD DSK0:SSORT.SYS[1,4]RETURN
```

To load SSORT.SYS into system memory, you must have a line in your system initialization command file like this:

```
SYSTEM DSK0:SSORT.SYS[1,4]
```

For more information on loading subroutines into system memory when your computer boots, see your *System Operator's Guide*.

When an AlphaBASIC or AlphaBASIC Plus program calls a subroutine by using an XCALL statement, AlphaBASIC attempts to locate the subroutine in user or system memory. If it cannot, it attempts to load

the subroutine from the disk. For details on where on the disk AlphaBASIC looks for the subroutine, see the *AlphaBASIC XCALL Subroutine User's Manual.*

If an AlphaBASIC or AlphaBASIC Plus program fetches a subroutine from the disk, it loads the subroutine into memory only for the duration of its execution. Once the subroutine has completed its execution, it is removed from memory. Therefore, if a subroutine is to be called a large number of times, it is wise to load it into memory (using the AMOS LOAD command) to avoid the overhead of fetching the subroutine from disk.

Subroutines loaded into memory by use of the AMOS LOAD command remain in memory until you reboot or until you use the AMOS command DEL to delete them.

DEFINING A KEY ARRAY

The rest of this chapter discusses how to set up the BASORT calling statement to sort random or sequential files. If you use the format of the BASORT statement that allows you to define the sort keys in an array table, use MAP statements to set up the array. For each key, you will need to define:

- **Size of Key**
- **Position of Key** (the Key's character position in the record)
- **Type of Key**:
  0 = String
  1 = AMOS 6-byte floating point
  2 = Binary
  3 = IEEE 4-byte floating point
  4 = IEEE 8-byte floating point
  5 = Integer
- **Key Sort Order** (0 for ascending, 1 for descending).

Define one more key in the array than you need; then set the last key to null (0)—this tells BASORT it has reached the end of your key table. For example, to define an array of 12 keys:

```
MAP1 TABLE(13)
     MAP2 KEY'SIZE,B,2
     MAP2 KEY'POS,B,2
     MAP2 KEY'TYPE,B,1
     MAP2 KEY'ORDER,B,1

KEY'SIZE(13) = 0           ! Define end of table.
          .
          .
          .
! Set up keys.
KEY'SIZE(1) = 16      ! First key is 16 characters.
KEY'POS(1) = 1        ! First key begins in first character
                      !   position of the record.
KEY'TYPE(1) = 0       ! First key is a string
KEY'ORDER(1) = 1      ! First key is to be sorted in
                      ! descending order.
          .
          .
          .
```

Remember that KEY'TYPE is always String for sequential files. For more information on defining keys and setting up the BASORT statement, see the sections below.

SORTING A RANDOM FILE

Your program can call BASORT in one of two formats. The first format includes information on each of up to three sort keys. Because this format is compatible with that used with the older AMOS sorting software, you can use this format with existing AlphaBASIC programs without making any changes to your programs. The second format omits much of the key information, but instead points to a table you define containing key information for an unlimited number of keys. Only the second format allows an optional argument at the end of the statement to request a language-specific, culturally correct collating sequence be used.

When you use BASORT to sort random files, BASORT sorts the file onto itself (that is, it replaces the original, unsorted file with a file containing the sorted data). Therefore, if you wish to keep a backup copy of the unsorted file, you must create a separate copy to be sorted.

XCALLs in AlphaBASIC Plus assume an integer value unless a number is specified with a decimal point, so be sure to include a decimal point in numbers on the BASORT statement line if you mean a floating point number instead of an integer.

FORMAT #1:

```
XCALL BASORT, Channel, Record'Count, Record'Size,&
  Key1'Size, Key1'Position, Key1'Order, Key2'Size,&
  Key2'Position, Key2'Order, & Key3'Size, Key3'Position,&
  Key3'Order, & Key1'Type, Key2'Type, Key3'Type
```

CHANNEL	File channel on which file to be sorted is open for random processing.
RECORD'COUNT	Number of records in the random file you are sorting.
RECORD'SIZE	Size of the longest record in the file you are sorting. The size of a record is the number of bytes in that record (including characters, spaces, etc.).
KEY1'SIZE	The size in bytes of the first key. Give the size of the largest instance of the first key (i.e., if the first key is the customer's name, find the longest name in any record, or perhaps allow for a very long one).
KEY1'POSITION	The first character position occupied by the first key. If KEY1'POSITION is 50, for example, BASORT will fit the characters beginning at the fiftieth byte in the record into the sequence it is creating.
KEY1'ORDER	Sort order of the first key. Enter 0 to indicate you want the first key of each record to be sorted in ascending sequence, or enter 1 to indicate descending sequence.
KEY2'SIZE	The size in bytes of the second key.
KEY2'POSITION	The first character position of the second key.
KEY2'ORDER	Sort order of the second key. Enter a 0 or a 1. (See KEY1'ORDER, above.)
KEY3'SIZE	The size in bytes of the third key.
KEY3'POSITION	The first character position of the third key.
KEY3'ORDER	Sort order of the third key. Enter a 0 or a 1. (See KEY1'ORDER, above.)
KEY1'TYPE	The data type of the first key. Key types are: 0 = String 1 = AMOS 6-byte floating point 2 = Binary 3 = IEEE 4-byte floating point 4 = IEEE 8-byte floating point 5 = Integer
KEY2'TYPE	The data type of the second key. See KEY1'TYPE.
KEY3'TYPE	The data type of the third key. See KEY1'TYPE.

If you want to use less than three keys, all entries in the XCALL command line for the unused keys must be zero.

If you omit the key types, BASORT assumes string data type. All arguments in the XCALL command line are numeric, but may be passed as either floating point or string values. For example, "99" is a valid entry. Arguments must *not* be in binary format. The first character in a record is considered position 1.

FORMAT #2:

The second BASORT calling format is:

```
XCALL BASORT, Channel, Record'Count, Record'Size, &
  Key1'Size, Key'Pointer{,Collat}
```

CHANNEL	File channel on which file to be sorted is open for random processing.
RECORD'COUNT	Number of records in the random file you are sorting.
RECORD'SIZE	Size of the longest record in the file you are sorting. The size of a record is the number of bytes in that record (including characters, spaces, etc.).
KEY1'SIZE	Set this argument to -1 to indicate key information is defined in a separate array table.
KEY'POINTER	Pointer to first element of an array in which you have used MAP statements to define as many keys as you want (e.g., KEYS(1)). See "Defining a Key Array," above for a sample definition.
COLLAT	If set to 1, perform a simple sort based on language set; if 0, use culturally correct collating sequence for that language

All arguments in the XCALL command line are numeric, but may be passed as either floating point or string values. For example, "99" is a valid entry. Arguments must *not* be in binary format.

Sample Random File Sort

We'll use the following unsorted file as an example. The file PO.DAT contains customer names, dates, and purchase order numbers. (The dates are in the format YEAR/MONTH/DAY so they will sort by year, then by month, then by day.)

Leucadia Begonia Farms	94/01/30	49130
Durango Nurseries	93/03/07	1207
Springtime Growers	94/02/28	K79876
Capistrano Gardens	93/06/24	7S729
Daisy's Daisies	94/04/21	A00326
Durango Nurseries	93/11/01	4103
Springtime Growers	93/10/13	K65843
Leucadia Begonia Farms	94/01/30	57045
Durango Nurseries	93/07/03	1209
Durango Nurseries	92/12/31	0301
Leucadia Begonia Farms	92/07/16	24150
Capistrano Gardens	93/06/24	7S730

Here's the program we'll use to sort the file:

```
! Sample program to sort a random data file:
!
MAP1 CUSTOMER'INFO                ! Definition of the Record:
     MAP2 NAME,S,35               !   35 bytes maximum
     MAP2 PURCHASE'DATE,S,8       !    8 bytes maximum
     MAP2 PURCHASE'ORDER,S,7      !    7 bytes maximum
MAP1 RECORD'SIZE,F,6,50           !   50 bytes maximum
MAP1 RECORD'NUMBER,F,6,0          ! First record is #0
MAP1 CHANNEL,F,6,100              ! Open channel is #100
MAP1 RECORD'TOTAL,F,6,12          ! Total of 12 records
MAP1 ASCENDING,F,6,0              ! Sort in ascending order
MAP1 STRING,F,6,0                 ! All keys are strings

START:
     OPEN #100,"PO.DAT",RANDOM,RECORD'SIZE,RECORD'NUMBER
     PRINT "Now sorting..."
     XCALL      BASORT,CHANNEL,RECORD'TOTAL,RECORD'SIZE,35,1, &
     ASCENDING,8,36,ASCENDING,7,44,ASCENDING,STRING,STRING,STRI
NG
     PRINT "We will sort on name, purchase date,"
     PRINT "and purchase order number:" : PRINT
     FOR RECORD'NUMBER = 0 TO RECORD'TOTAL
          READ #100,CUSTOMER'INFO
          PRINT NAME,
          PRINT PURCHASE'DATE,
          PRINT PURCHASE'ORDER
     NEXT
     CLOSE #100
END
```

Note the line right after START: that opens the file, PO.DAT. The XCALL statement then calls the BASORT subroutine, where the variables (defined in the MAP statements) define the BASORT parameters. BASORT writes the new data back into the original file, overwriting the old data. The program also displays the results on your screen, and then closes the file. The resulting display looks like this:

```
Now sorting...
We will sort on name, purchase date,
and purchase order number:

Capistrano Gardens         93/06/24     7S729
Capistrano Gardens         93/06/24     7S730
Daisy's Daisies            94/04/21     A00326
Durango Nurseries          92/12/31     0301
Durango Nurseries          93/03/07     1207
Durango Nurseries          93/07/03     1209
Durango Nurseries          93/11/01     4103
Leucadia Begonia Farms     92/07/16     24150
Leucadia Begonia Farms     94/01/30     49130
Leucadia Begonia Farms     94/01/30     57045
Springtime Growers         93/10/13     K65843
Springtime Growers         94/02/28     K79876
```

SORTING SEQUENTIAL FILES

When you sort a sequential file, you must specify both an input and an output file. Before BASORT is called, your program must open the file to be sorted for input. BASORT leaves the file open for output. If you wish to sort a file back onto itself, you may specify the same file for both input and output.

Your program can call BASORT in one of two formats. The first format includes information on each of up to three sort keys. Because this format is compatible with that used with the older AMOS sorting software, you can use this format with existing AlphaBASIC programs without making any changes to your programs. The second format omits much of the key information, but instead points to a table you define containing key information for an unlimited number of keys. Only the second format allows optional arguments at the end of the statement to: 1) request a language-specific, culturally correct collating sequence be used or 2) Define the end-of-record character (allowing the use of variable length records).

Sequential files contain only ASCII data. For that reason, when you sort sequential files you do not have to specify the data type of the sort keys; BASORT knows all keys in a sequential file are strings.

XCALLs in AlphaBASIC Plus assume an integer value unless a number is specified with a decimal point, so be sure to include a decimal point in numbers on the BASORT statement line if you mean a floating point number instead of an integer.

FORMAT #1:

The first format for calling BASORT for sequential files is:

```
XCALL BASORT, Input'Channel, Output'Channel, Record'Size, &
     Key1'Size, Key1'Position, Key1'Order, &
     Key2'Size, Key2'Position, Key2'Order, &
     Key3'Size, Key3'Position, Key3'Order
```

INPUT'CHANNEL	The file channel on which the input file is open.
OUTPUT'CHANNEL	The file channel on which the output file is open.
RECORD'SIZE	The size, in bytes, of the largest record in the file, including the terminating carriage return/linefeed characters. **Too small a value results in truncation of data records.**
KEY1'SIZE	The size, in bytes, of the first key. Give the size of the largest instance of this key (i.e., if the first key is the customer's name, find the longest name in any record, or perhaps allow for a very long one).
KEY1'POSITION	The first character position of the first key. If KEY1'POSITION is 50, for example, BASORT will fit the characters beginning at the fiftieth byte in the record into the sequence it is creating.
KEY1'ORDER	Sort order of the first key. Enter 0 to indicate you want the first key of each record to be sorted in ascending sequence, or enter 1 to indicate descending sequence.
KEY2'SIZE	The size in bytes of the second key.
KEY2'POSITION	The first character position of the second key.
KEY2'ORDER	Sort order of the second key. Enter a 0 or a 1. (See KEY1'ORDER, above.)
KEY3'SIZE	The size in bytes of the third key.
KEY3'POSITION	The first character position of the third key.
KEY3'ORDER	Sort order of the third key. Enter a 0 or a 1. (See KEY1'ORDER, above.)

If you want to use less than three keys, all entries in the XCALL command line for the unused keys must be zero. Key types are always string for sequential files. All arguments in the XCALL command line are numeric, but may be passed as either floating point or string values. For example, "99" is a valid entry. Arguments must *not* be in binary format. The first character in a record is considered position 1.

FORMAT #2:

The second BASORT calling format is:

```
XCALL BASORT, Input'Channel, Output'Channel, Record'Size, &
     Key1'Size, Key'Pointer{,Collat}{,EOR}
```

INPUT'CHANNEL	The file channel on which the input file is open.
OUTPUT'CHANNEL	The file channel on which the output file is open.
RECORD'SIZE	The size, in bytes, of the largest record in the file, including the terminating carriage return/linefeed characters. **Too small a value results in truncation of data records.** If set to -1, BASORT assumes variable length records, and uses EOR (below) to determine the end of record.
KEY1'SIZE	Set this argument to -1 to indicate key information is defined in a separate array table.
KEY'POINTER	Pointer to first element of an array (e.g., TABLE(1)) in which you have used MAP statements to define as many keys as you want. See "Defining a Key Array," above for a sample definition.
COLLAT	If set to 1, perform a simple sort based on language set; if 0, use culturally correct collating sequence for that language. If omitted, 0 is assumed.
EOR	Defines end of record character: 0 = CR only (0D hex) 1 = LF only (0A hex) 2 = Both CR and LF (0D & 0A) RECORD'SIZE must be set to -1 indicating variable length records if you are using EOR. If omitted, 0 is assumed**.**

All arguments in the XCALL command line are numeric, but may be passed as either floating point or string values. For example, "99" is a valid entry. Arguments must *not* be in binary format.

Sample Sequential File Sort

The following is an unsorted sequential file containing a list of street names and the cities they are located in. The file is called STREET.DAT.

We want to record the sorted, alphabetic list of all the streets in a file called STREET.LST. The street names sorted according to the city they are located in we'll place in a file called CITY.LST.

Here is the unsorted file:

Redeye Circle	Laguna Niguel
Rancho Laguna Road	Laguna Beach
Redfield Road	Mission Viejo
Random Drive	Lake Forest
Reposado Drive	Laguna Hills
Ramona Drive	Mission Viejo
Revere Road	Laguna Beach
Ravenscroft Road	Mission Viejo
Redbird Street	Irvine
Red Bluff Drive	Laguna Hills
Random Drive	Mission Viejo
Raspberry Lane	San Juan Capistrano
Ranchero	San Clemente
Ramona Drive	San Juan Capistrano
Revere Road	Lake Forest
Raintree Drive	Irvine
Rhodolite Court	Lake Forest
Rancho Laguna Road	Laguna Hills
Ramona Drive	San Clemente
Revere Road	Mission Viejo

Now we create the AlphaBASIC program. The first thing we have to remember to do is open the file channel for the file we want to sort, and two more file channels and files where we want to put the sorted data (or we could use just one other file if we wanted to write over the original, unsorted data). Our program might look like this:

```
! Sample program to sort a sequential data file:
START:
     OPEN #1,"STREET.DAT",INPUT
     OPEN #2,"STREET.LST",OUTPUT
     OPEN #3,"CITY.LST",OUTPUT
     PRINT "Now sorting all streets alphabetically."
     XCALL BASORT,1,2,50,30,1,0,20,31,0,0,0,0
     CLOSE #1
     PRINT "Now sorting according to city."
     OPEN #1,"STREET.DAT",INPUT
     XCALL BASORT,1,3,50,20,31,0,30,1,0,0,0,0
     PRINT "All done." : PRINT
     PRINT "See STREET.LST and CITY.LST for sorted files."
     CLOSE #1 : CLOSE #2 : CLOSE #3
END
```

The file opened for input is our unsorted source file. The files opened for output are what will contain our sorted data.

The first BASORT statement sorts STREET.DAT using the street name as the first key and the city name as the second. Refer back to the discussions on BASORT statement format if you need a refresher on what the numeric arguments on the BASORT statement line mean. Note that since we are only using two keys, the third key data is 0, 0, 0.

Next, the file STREET.LST is created and the data in STREET.DAT is rewritten in alphabetical order. The next lines in the program close and then re-open file channel #1 and the file STREET.DAT. If those two lines are omitted, the new file CITY.LST, though created, would be empty because no further data would be found in the file STREET.DAT. These two lines cause the BASORT subroutine to look at the beginning of the file, rather than the end.

We again call the BASORT program. This line is different than the first because we are now specifying the city as the first key and the street name as the second key.

STREET.LST, the sorted version of all the streets contained in the file STREET.DAT, looks like this:

Raintree Drive	Irvine
Ramona Drive	Mission Viejo
Ramona Drive	San Clemente
Ramona Drive	San Juan Capistrano
Ranchero	San Clemente
Rancho Laguna Road	Laguna Beach
Rancho Laguna Road	Laguna Hills
Random Drive	Lake Forest
Random Drive	Mission Viejo
Raspberry Lane	San Juan Capistrano
Ravenscroft Road	Mission Viejo
Red Bluff Drive	Laguna Hills
Redbird Street	Irvine
Redeye Circle	Laguna Niguel
Redfield Road	Mission Viejo
Reposado Drive	Laguna Hills
Revere Road	Laguna Beach
Revere Road	Lake Forest
Revere Road	Mission Viejo
Rhodolite Court	Lake Forest

The file CITY.LST, which is the streets first sorted according to their city location, looks like this:

Raintree Drive	Irvine
Redbird Street	Irvine
Rancho Laguna Road	Laguna Beach
Revere Road	Laguna Beach
Rancho Laguna Road	Laguna Hills
Red Bluff Drive	Laguna Hills
Reposado Drive	Laguna Hills
Redeye Circle	Laguna Niguel
Random Drive	Lake Forest
Revere Road	Lake Forest
Rhodolite Court	Lake Forest
Ramona Drive	Mission Viejo
Random Drive	Mission Viejo
Ravenscroft Road	Mission Viejo
Redfield Road	Mission Viejo
Revere Road	Mission Viejo
Ramona Drive	San Clemente
Ranchero	San Clemente
Ramona Drive	San Juan Capistrano
Raspberry Lane	San Juan Capistrano

Chapter 5 - Assembly Language Program Interface

SuperSort is fully accessible from the AMOS assembly language programming environment. To call SuperSort from assembly language, follow these basic steps:

1. Locate SSORT.SYS in system or user memory.
2. Allocate memory within the job partition for sorting.
3. Define the keys on which to sort.
4. Define other sort parameters.
5. Call SSORT.SYS to perform the actual sort.

Definitions used by SuperSort programs are contained in the file DSK0:SRTSYM.M68[7,7].

LOCATING SSORT.SYS IN MEMORY

The sort module, SSORT.SYS, must be located in user or system memory prior to sorting. From within assembly language programs, you can use the SRCH monitor call to locate the memory module in memory, as shown in the sample program later in this chapter

ALLOCATING MEMORY FOR THE SORT PROCESS

SuperSort performs its best when given the maximum amount of memory. For this reason, you will want to allocate as much memory as possible for the sort process. Once allocated, you pass a pointer to this memory to SuperSort. After the sort is complete, you are free to dispose of this memory.

DEFINING KEYS

SuperSort supports a virtually unlimited number of sort keys. You must, however, define the characteristics of each key prior to calling SuperSort. Keys are defined using a key table, as defined in SRTSYM.M68.

Each element in the key table consists of five fields, which describe the key to use for sorting. You can specify as many keys as you like. You are only limited by the amount of memory you have to sort in.
The structure of the key table is as follows:

KT.CMP	4 bytes	Optional comparison routine
KT.SIZ	2 bytes	Size of key
KT.OFF	2 bytes	Offset within the record of key
KT.TYP	1 byte	Type of key
KT.ORD	1 byte	Sort order

S..KT = Size of one element in the key table. Multiply this with the maximum number of keys you might use in a sort to find the size of this area when allocating memory for it.

Description of each key table field:

KT.CMP Pointer to an optional comparison routine. If you have a unique key type you wish to sort on you would use this option. If used, KT.TYP should specify a value greater than the last defined type (e.g., >5).

Register use: Input: D0 => Pointer to first string.
D1 => Pointer to second string.
Output: D0 := Result code:
1 = string1 > string2
-1 = string1 < string2
0 = string1 = string2

KT.SIZ Load this field with the size of the key. It must be more than zero and no larger that the size of the record specified in the main interface structure passed to SSORT.

KT.OFF Load this field with the offset, in bytes, of the key within the data record. It must not exceed the size of the record. Also, key offsets start from a base of 1.

KT.TYP This field specifies the type of key:

0 = string
1 = 6-byte (AMOS) floating point
2 = binary (AlphaBASIC compatible format)
3 = 4-byte (IEEE) floating point
4 = 8-byte (IEEE) floating point
5 = integer
6 = use user-supplied comparison routine

KT.ORD Sort order. This field tells SSORT which order to sort this particular key. Set as follows: 0 = Ascending; 1 = Descending

DEFINING SORT PARAMETERS

After the keys have been defined, you must set up the remaining parameters which control the way the sort is to be performed. We have provided a SRTSYM.M68 symbol file that defines the layout of this structure to help you get your program sorting with a minimum of difficulty. This is the layout of the interface structure:

Symbol	Size	Description
IS.MEM	4 bytes	Address of free memory area
IS.MSZ	4 bytes	Size of the free memory area
IS.NRC	4 bytes	Number of records for a random file
IS.INP	4 bytes	Address of optional input routine
IS.OUT	4 bytes	Address of optional output routine
IS.ERR	4 bytes	Address of optional error handling routine
IS.CLT	4 bytes	Address of optional collate table
IS.KTB	4 bytes	Address of the key table
IS.NOR	4 bytes	Number of records sorted (returned by SSORT)
IS.CMP	4 bytes	Number of comparisons made
IS.SWP	4 bytes	Number of swaps made
IS.IDB	110 bytes	Input DDB
IS.ODB	110 bytes	Output DDB
IS.RSZ	2 bytes	Record size
IS.KYS	2 bytes	Total number of keys
IS.ERC	2 bytes	Error code
IS.BGF	2 bytes	“Big Flag” used for random files
IS.COL	1 byte	Sort flag, simple or collate
IS.EOR	1 byte	End of record indicator

S..IS = Size of this structure

Description of each element:

IS.MEM This is an address to an area of memory that SSORT will use to do its work. To get the best results, this area should be as large as possible. Use the GETMEM call to allocate this area and leave at least 4096 bytes for SSORT to allocate disk buffers, etc. It is important to note that you should allocate this memory in accordance with the way that AMOS allocates memory modules, because SSORT will be using standard AMOS calls to allocate buffers and load disk drivers.

IS.MSZ This is the size of the free memory area.

IS.NRC This is the number of records to be sorted. It is only used with random files. This can be left uninitialized for sequential file sorting.

IS.KTB Pointer to the key table. The key table contains information on all the keys you wish to sort on, starting with the primary, then secondary and so on, with an unlimited number. The key table structure is also defined in SRTSYM.M68, supplied with the SuperSort software. It is described in the next section.

IS.IDB This is the DDB for the input file. If you are supplying your own I/O routines this can be left uninitialized; otherwise, it should be set up for both random and sequential file sorts.

IS.ODB This is the DDB for the output file. In the case of random files, this should be all cleared. In the case of sequential files, it can be null if you want the input file to be overwritten with the result file; otherwise, it should be set up so that a separate output file is created by the sort.

IS.RSZ This is for the record size. It should be set for either type of file to be sorted but if you are specifying variable record sizes for a sequential file, it should be set to -1. If you are using your own I/O routines this should contain the maximum record size the input routine might pass.

IS.KYS Total number of keys to sort on. Set this according to the number of keys being used to sort with, not the number of entries you have allocated for the key table.

IS.BGF This is the "big flag." This is used with random files and signifies that the data records span physical block boundaries, i.e., exceed 512 bytes. Set it to a non-zero value if this is true, else set it to zero.

IS.COL Collate flag. Set this to 0 for culturally correct sorting or set to 1 for a simple ASCII sort.

IS.EOR End of record indicator. Used for sequential files. Use this if you are specifying variable record sizes. Set to: CR = 0, LF = 1, CRLF = 2.

Optional routines:

IS.INP Pointer to an optional input routine. SSORT will do its own I/O when you use standard files, such as those created by AlphaBASIC. But if you are sorting data that comes from a file with a special format or from memory, use this so your program can present the data to SSORT so it can be sorted properly.

Register use:	Input:	D0 => Pointer to memory area to put record data.
	Output:	D0 := Size of record. Set to -1 to signal no more records.

IS.OUT Pointer to an optional output routine. If you supply an input routine you must supply an output routine.

Register use:	Input:	D0 => Pointer to record data
		D1 := Size of record.
	Output:	nothing.

IS.ERR Pointer to an optional error handler. This option is independent of the other routines, i.e., you can use this option without the optional I/O routines. Use this option when you want to control error processing in your program.

Register use:	Input:	D0 := Error code.
	Output:	nothing.

IS.CLT Pointer to an optional collate table. If you are doing a simple compare, this field should be left 0. If you want to sort using a special collating sequence, load this field with the address of a CULCMP.lng file, created with COLGEN.LIT. For more information on COLGEN.LIT, see Chapter 7.

Statistics returned by SSORT:

IS.NOR When the sort is complete SSORT loads this with the number of records that were actually sorted.

IS.CMP This field is loaded by SSORT upon completion with the number of comparisons made to achieve the correct sort.

IS.SWP This field is set with the number of swaps SSORT performed during the sorting operation.

IS.ERC Upon completion of the sort, SSORT loads this field with the resulting error code.

Below is a list of symbols that define the errors that SSORT might return:

SE$NOMEM	= 1	Insufficient memory.
SE$FILERR	= 5	File improperly opened in SSORT.
SE$REDERR	= 6	Read file error.
SE$WRTERR	= 7	Write file error.
SE$SIZERR	= 8.	Wrong record size given.
SE$NOIO	= 9.	input/output routines not found.
SE$CTRLC	= 12.	CTRLC detected.

CALLING SSORT.SYS

Calling SuperSort is simply a matter of using the location of SSORT.SYS, found in the first step, and calling that address, passing in the parameter definitions you have set up.

PUTTING IT ALL TOGETHER

Now let's look at a sample program which sorts a file.

Note that the basic structure to pass to SSORT.SYS is defined in the SRTSYM.M68 file.

```
          SEARCH  SYS
          SEARCH  SYSSYM
          SEARCH  SRTSYM
;
.OFINI                              ;
.OFDEF  KEYTBL, <S..KT*4>           ; Key table.
.OFDEF  INTER, S..IS  ; Interface structure.
.OFSIZ  S..IMP                      ;
        GETIMP  S..IMP, A5
        LEA     A1, KEYTBL(A5)      ; Index the key table.
        LEA     A0, INTER(A5)       ; Index the interface structure.
;
; Set up the input DDB.
;
        LEA     A2, DATFIL          ; Index the data filename.
        LEA     A4, IS.IDB(A0)      ; Index the input DDB.
        FSPEC   @A4                 ; Get the filename.
        INIT    @A4                 ; Init it.
;
; Set up the key table.
;
        MOV     #0, KT.CMP(A1)      ; No comparison routine.
        MOVW    #6, KT.SIZ(A1)      ; Set size of key.
        MOVW    #1, KT.OFF(A1)      ; Offset of the key.
        MOVB    #1, KT.TYP(A1)      ; Set the type.
        MOVB    #0, KT.ORD(A1)      ; Set the sorting order.
;
; Now the interface structure.
;
        USRFRE  D1                  ; Get the start of available memory.
        USREND  D0                  ; Get the end of memory.
        SUB     D1, D0              ; Find the difference.
        SUB     #4096., D0          ; Leave some extra room.
        PUSH    D0                  ; Move the desired size onto the stack.
        PUSH                        ; Make some room for the address.
        GETMEM  @SP                 ; Get some memory.
        POP     A3                  ; Get the address of the module.
        POP                         ; Clean up the stack.
        MOV     A3, IS.MEM(A0)      ; Save the address in the structure.
        MOV     D0, IS.MSZ(A0)      ; Save the size as well.
                                    ;
        MOV     #250., IS.NRC(A0) ; Set the number of records.
        MOV     #0, IS.INP(A0)      ; No input routine.
        MOV     #0, IS.OUT(A0)      ; No output routine.
        MOV     #0, IS.ERR(A0)      ; No error handler.
        MOV     #0, IS.CLT(A0)      ; No collate table.
        MOV     A1, IS.KTB(A0)      ; Save the key table address.
        MOVW    #86., IS.RSZ(A0)  ; Set the record size.
        MOVW    #1, IS.KYS(A0)      ; Set the number of keys.
        MOVW    #0, IS.BGF(A0)      ; Set the "big" flag.
        MOVB    #1, IS.COL(A0)      ; Do a simple sort.
; Use OPENIO for any files that span blocks, or for any file created
;   using OPENIO.
```

```
        ;OPENIO  @A4, #F.WAT, IS.RSZ(A0) ; Open the file.
         OPENR   @A4
;
; Index SSORT.SYS
;
         SRCH    SSORT, A6         ; Index SSORT.SYS.
         BEQ     10$               ; Got it.
         TYPECR  <%SSORT.SYS not found in memory>
         EXIT                      ;
10$:
         MOV     A0, D0            ; Index the interface structure.
         CALL    10.(A6)           ; Call SSORT.
         TYPECR  <Done>
         EXIT
DATFIL:  ASCIZ   /SST001.DAT/      ; Data filename.
         EVEN
SSORT:   RAD50   /SSORT SYS/       ; SSORT.SYS filename.

         END
```

CULTURALLY CORRECT SORTING AND STRING COMPARISON

Another option is the use of the file COLLAT.SYS. This program is used for comparing or translating string information using the current CULCMP.lng file found in the LCS: account. To generate the CULCMP.lng file, use the COLGEN.LIT program found in the LCS: account. As shipped, LDFTBL.H it will generate a CULCMP.USA file that will be compatible for English and most other languages. If, on the other hand, the language you wish to use for sorting purposes does not match this table, you need to modify the LDFTBL.H file in the LCS: account, and recompile COLGEN.C. When finished, rename the resulting file so that it has the proper extension for the language to be used.

Note: When recompiling COLGEN, you must use AlphaC 2.0 or later (GNU C). Alpha C 1.x will not work.

When using COLLAT.SYS to translate strings, pass it a normal ASCII string of characters and COLLAT.SYS will translate it according to the CULCMP.lng file found in the LCS: account. This result string will not consist of any printable characters; rather, it will contain the codes found in the CULCMP file that correspond with the ASCII character in the original string. These are the codes that SSORT would use in a culturally correct sort.

Calling sequence for translating a string:

```
    PEA   dest'buff       ; Address of destination buffer.
    POP   D0              ; Save in D0.
    PEA   source'string   ; Address of source string.
    POP   D1              ; Save in D1.
    PUSH  #1              ; Function code (translate).
    PUSH  collat'table    ; Address of CULCMP file.
```

SSORT returns: D0 := Length of the translated string.

When using COLLAT.SYS to compare two strings, you pass it the addresses of the two ASCII strings you want compared along with the address of the CULCMP file and SSORT returns a code indicating the result of the compare.

Calling sequence for comparing two strings:

```
PEA   string1          ; Address of 1st string.
POP   D0               ; Save in D0.
PEA   string2          ; Address of 2nd string.
POP   D1               ; Save in D1.
PUSH  #0               ; Function code (compare).
PUSH  collat'table     ; Address of CULCMP file.
```

SSORT returns the result code in D0:

1 = string1 > string2
0 = string1 = string2
-1 = string1 < string2

Chapter 6 - AlphaC Program Interface

SuperSort is fully accessible from the AlphaC 2.0 programming environment. While the main sorting module, SSORT.SYS, uses a non-standard calling method, glue logic is provided which makes calling from C quite straightforward.

As when calling SuperSort from assembly language, you follow the same simple steps:

1. Locate SSORT.SYS in system or user memory.
2. Allocate memory within the job partition for sorting.
3. Define the keys on which to sort.
4. Define other sort parameters.
5. Call SSORT.SYS to perform the actual sort.

Definitions used by SuperSort programs are contained in the file DSK0:SRTSYM.H[7,7].

NOTE: SuperSort is only compatible with AlphaC 2.0 or later (GNU C). It will not work with AlphaC 1.x.

LOCATING SSORT.SYS IN MEMORY

The sort module, SSORT.SYS, must be located in user or system memory prior to sorting. From within C, you can use the afetch() function to locate the memory module in memory.

ALLOCATING MEMORY FOR THE SORT PROCESS

SuperSort performs at its best when given the maximum amount of memory. For this reason, you will want to allocate as much memory as possible for the sort process. Once allocated, you pass a pointer to this memory to SuperSort. After the sort is complete, you are free to dispose of this memory.

DEFINING KEYS

SuperSort supports a virtually unlimited number of sort keys. You must, however, define the characteristics of each key prior to calling SuperSort. Keys are defined using the KEYTBL structure, as defined in SRTSYM.H. Because multiple keys are supported, SuperSort expects multiple copies of the KEYTBL structure, one following the other without intervening data.

The entry for a key is defined by the following fields:

Symbol	Size	Description
*fCompare	Pointer to function returning int.	Function pointer defining the function to be called to compare this key, or NULL if SuperSort's built-in key comparison routines are to be used. This field allows you to expand the types of keys supported by SuperSort by providing your own comparison routine.
wKeySize	Unsigned short	The size of the key in bytes.
wKeyOffset	Unsigned short	The offset from the start of the record to the start of the key.
cKeyType	char	The type of the key, defined as: 0 = string 1 = 6-byte (AMOS) floating point 2 = binary (AlphaBASIC compatible) 3 = 4-byte (IEEE) floating point 4 = 8-byte (IEEE) floating point 5 = Integer 6 = use user-supplied comparison routine.
cKeyDirec	char	0 for ascending sort on this key, 1 for descending.

DEFINING SORT PARAMETERS

After the keys have been defined, you must set up the remaining parameters which control the way the sort is to be performed. You specify these parameters in a structure call INTERFACE, which is defined in SRTSYM.H.

The entries in this structure are as follows:

Symbol	Size	Description
*dwAddressOfMem	unsigned long	Pointer to the memory you allocated for the sort process.
dwSizeOfMem	unsigned long	The size of the memory area you allocated.
dwRndNoOfRec	unsigned long	If sorting a random file, this field must contain the number of records to be sorted.
*fAddrOfInput	pointer to function returning short	Pointer to the input function used to read data from the input file, or NULL if you want SuperSort to use its built-in routines.
*fAddrOfOutput	pointer to function returning void	Pointer to the output function used to write data to the output file, or NULL if you want SuperSort to use its built-in routines.

Symbol	Size	Description
*fErrorHandler	pointer to function returning void	Pointer to the error handling function called in case of an error during the sort process, or NULL if you want SuperSort to use its built-in routines.
*pwCollatTbl	pointer to short	Pointer to the collating table or NULL if no collating table is to be used.
pstKeyTable	pointer to KEYTBL	Pointer to the key definition table, as defined in the previous step.
dwNoOfRec	unsigned long	This field is set by SuperSort upon completion of the sort. It will contain the total number of records sorted.
dwNoOfCmps	unsigned long	This field is set by SuperSort upon completion of the sort. It will contain the total number of key comparisons done.
dwNoOfSwps	unsigned long	This field is set by SuperSort upon completion of the sort. It will contain the total number of record swaps performed.
stInpFile	ddb	This entry is the DDB used by the input file. You must fill in this DDB with all necessary items to access this file.
stOutFile	ddb	This entry is the DDB used by the output file. You must fill in this DDB with all necessary items to access this file.
nRecSize	short	The size of the records to be sorted.
nTotalKeys	short	The total number of sort keys provided in the key definition table.
wErrorNo	unsigned short	This field is set by SuperSort upon completion of the sort. It will contain an error code if the sort did not complete successfully. Error codes are defined in SRTSYM.H.
wBigFlag	unsigned short	Set to 1 if random file being sorted is structured as "big" records (records span block boundaries), set to 0 otherwise.
bCollat	boolean	Set to FALSE for simple key comparison, set to TRUE for culturally correct collating sequence.
cEorInd	char	The character to be treated as a end-of-record indicator when sorting sequential files.

CALLING SSORT.SYS

Calling SuperSort is simply a matter of using the location of SSORT.SYS, found in the first step, and calling that address, passing in the parameter definitions you have set up.

PUTTING IT ALL TOGETHER

Now let's look at a sample program which sorts a sequential file, DATA.IN, on a single, ascending, string key, starting at the first byte of each record. The results of the sort are placed in DATA.OUT.

```
/* ---------------------------------------------------------------- *
   Sample program which uses SSORT.SYS
 * ---------------------------------------------------------------- */

#define BOOL char

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include <moncal.h>
#include <amos.h>
#include <rad50.h>
#include <srtsym.h>

version (1, 0, , 100, , -1, 0, PH$REE!PH$REU);

#define      ELBOWSPACE   4048         /* Mem. to be left from alloc. block */

INTERFACE    stInterface;        /* SuperSort interface structure */
ddb          stUnSortFile;             /* ddb for input file */
unsigned char * pcFreeMemArea;         /* pointer to currently free memory */
unsigned char * pcStartMemArea;        /* pointer to the start of mem */
unsigned long       dwMemAreaFree;            /* total amount of mem remaining */
mcb          stMemArea;          /* mcb for getmem */

/* ------------------------------------------------------ *
 *     function prototypes
 * ------------------------------------------------------ */
void   InitKey (void);           /* initialize global key table */
void   OpenInpFile (char **);    /* opens input file */
void   ErrorHandler(unsigned short);    /* displays errors and sets joberr */

/* ---------------------------------------------------------------- *
 * Main program
 * ---------------------------------------------------------------- */
void main(int argc, char **argv)
{
const char   * InpFileName = "SYS:SSORT.SYS";
int          (* fnSsort)();
ddb          stInpddb;
int          iRetVal;

       stInterface.fErrorHandler = NULL;

       /* fetch SSORT.SYS which gives the address of sort routine */
       iRetVal = fspec (&InpFileName, &stInpddb, 0);
       if (!afetch (&stInpddb, &fnSsort, 0)) {
              ErrorHandler(NOAMSORT);
              exit(1);
       }
```

```
        /* jump to offset 10, as initially ver. info. stored  */
        fnSsort = fnSsort + 10;

        /* Allocate memory for sort process */
        /* check if memory available is enough or not */
        stMemArea.size = usrend() - usrfre() - ELBOWSPACE;

        if (stMemArea.size <= 10240) {  /* 10k min. mem. requirement */
              ErrorHandler(NOMEMORY);
              exit(1);
        }

        if ( !getmem(&stMemArea)) {      /* allocate memory */
              ErrorHandler(CANTALLOCATE);
              exit(1);
        } else {
              /*
               * Store size and ptr. to free mem.
               */
              dwMemAreaFree = stMemArea.size;
              pcStartMemArea = stMemArea.adr;
              pcFreeMemArea = pcStartMemArea;
        }

        /* Open the sequential file we are going to sort */
        OpenInpFile(&argv[1]);

        /* Initialize the file ddbs in interface structure */
        stInterface.stInpFile = stUnSortFile;
        clear (stInterface.stOutFile);

        /* Define the keys we are going to sort on */
        InitKey();

        /* Set up the remaining arguments to sort */
        stInterface.dwAddressOfMem = (unsigned long)pcFreeMemArea;
        stInterface.dwSizeOfMem = dwMemAreaFree;
        stInterface.fAddrOfInput = NULL;
        stInterface.fAddrOfOutput = NULL;
        stInterface.bCollat = SIMPLECOMPARE;
        stInterface.dwNoOfCmps = 0;
        stInterface.dwNoOfSwps = 0;

        /* Now call SSORT.SYS itself, passing a ptr to interface structure */
        if (!fnSsort (&stInterface))
              /* error returned from ssort.sys */
              if (stInterface.fErrorHandler == NULL) {
                    ErrorHandler(stInterface.wErrorNo);
                    exit(1);
              }

        printf ("\nRecords sorted: %ld", stInterface.dwNoOfRec);
        printf ("\nComparisons: %ld", stInterface.dwNoOfCmps);
        printf ("\nSwaps: %d\n", stInterface.dwNoOfSwps);

} /* end main */

/* ------------------------------------------------------------------ *
 *  InitKey
 *    Input  : none
 *    Output : none
 *    This function initializes the global key table
 * ------------------------------------------------------------------ */
void InitKey()
{
PKEYTBL            pstKeyTable;
```

```
       /* allocate space for key table */
       pstKeyTable = (PKEYTBL) pcStartMemArea;
       stInterface.pstKeyTable = (PKEYTBL) pcStartMemArea;
       stInterface.nTotalKeys = 0;

       /* set up record size */
       stInterface.nRecSize = 80;

       /* set up a single key */
       pstKeyTable->fCompare = NULL;
       pstKeyTable->wKeySize = 8;
       pstKeyTable->wKeyOffset = 1;
       pstKeyTable->cKeyType = 0;
       pstKeyTable->cKeyDirec = 0;
       pstKeyTable++;
       stInterface.nTotalKeys++;
       pcFreeMemArea += sizeof(KEYTBL);
       dwMemAreaFree -= sizeof(KEYTBL);

       /* could define more keys here if we wanted to */

} /* end init_key */


/* ------------------------------------------------------------- *
 *  ErrorHandler
 *     Input  : Error number.
 *     Output : none
 *     This function prints the error
 * ------------------------------------------------------------- */
void ErrorHandler(unsigned short wError)
{
       switch(wError) {
              case NOMEMORY:
                     typecr("?Insufficient memory");
                     break;
              case CANTALLOCATE:
                     typecr("?Can't allocate user specified memory");
                     break;
              case NOINPOUT:
                     typecr ("?Input Output routines not found");
                     break;
              case NOAMSORT:
                     typecr ("?Main sort module not found");
                     break;
              case NOFILE:
                     typecr ("?Input file not found");
                     break;
              case ERRORARG:
                     typecr ("?Argument error");
                     break;
              case ERRORFILE:
                     typecr ("?File not open in correct mode");
                     break;
              case READERROR:
                     typecr ("?Read error");
                     break;
              case WRITEERROR:
                     typecr ("?Write error");
                     break;
       }
}

/* ------------------------------------------------------------- *
 *  OpenInpFile
 *     Input  : Pointer to Input file name
 *     Output : none
```

```
 *     This function opens the input file.
 * ------------------------------------------------------------- */
void OpenInpFile (char ** pcInpFile)
{
int    iRetVal;

       clear (stUnSortFile);

       iRetVal = fspec ((const char **)pcInpFile, &stUnSortFile,
irad503('D','A','T'));

       iRetVal = init (&stUnSortFile);

       if (!lookup (&stUnSortFile)) {
              ErrorHandler(NOFILE);
              exit(1);
       }

       iRetVal = openi (&stUnSortFile, 0);
       stInterface.dwRndNoOfRec = 0;
}
```

Chapter 7 - Language-Specific Rules for Sorting

Chapter 1 contains a discussion of collating sequences and culturally expected sort results. This chapter gives a more technical discussion of the collating rules used for different languages. SuperSort adheres to these rules. If you want a more detailed explanation, refer to *Keys to Sort and Search for Culturally Expected Results* from the International Technical Support Center of IBM, document number GG24-3516.

The following terms are used in this chapter:

- **Diacritics:** These are special marks such as accent marks that serve to mark phonetic differences in words or to distinguish two words otherwise graphically identical (for example, “résumé” and “resume”).
- **Ligatures:** A single character combining two or more characters (for example, “æ”).
- **Specials:** Characters that are not letters or numbers (for example, “@”).
- **Quasi-homographs:** Words spelled identically when case, diacritics, ligatures, and special characters are ignored. For example, the following word pairs are quasi-homographs: “résumé” and “resume”; “co-op” and “coop”; and “Caesar” and “cæsar.”

ENGLISH

Words are sorted according to the alphabetic rank of the letters (“a” comes before “b”).

A longer string that begins with the same characters contained in a shorter string will come after the shorter string regardless of case. For example, “conveyance” comes after “Convey.”

Except in the case of quasi-homographs, diacritical marks are ignored. Embedded special characters are also ignored. For example, “co-op” comes before “cooperate” and after “coo.” Ligatures are also ignored, being treated as two separate characters.

In the case of quasi-homographs: diacritics and ligatures are considered first. The quasi-homograph with diacritics follows the one without. The word with the ligature comes after the one without. Next, case is considered: lowercase comes before uppercase. Finally, specials are considered; the word with special characters follows the one without.

Strings containing only specials (e.g., “@#$%” are sorted first).

Numbers follow specials and precede letters.

FRENCH

Words are sorted according to the alphabetic rank of the letters (“a” comes before “b”). The alphabet contains 26 characters, “a” through “z.”

A longer string that begins with the same characters contained in a shorter string will come after the shorter string regardless of case.

Except in the case of quasi-homographs, diacritical marks are ignored. Embedded special characters are ignored. Ligatures are also ignored, being treated as two separate characters.

In the case of quasi-homographs: diacritics and ligatures are considered first. The word with diacritics follows the one without. Diacritic distinction proceeds from right to left. The relative weight of diacritics is: acute, grave, circumflex, trema, and cedilla. For example, the following words are sorted correctly: “cote, côte, coté, côté.”

A quasi-homograph with a ligature comes after the one without. Next, case is considered: lowercase comes before uppercase. Finally, specials are considered; the word with special characters follows the one without.

Strings containing only specials (e.g., “@#$%” are sorted first).

Numbers follow specials and precede letters.

GERMAN

Words are sorted according to the alphabetic rank of the letters (“a” comes before “b”). The alphabet contains 26 characters, “a” through “z.” “ß” (ess-zed) is sorted as “ss” but precedes “ss.” Umlauted vowels “ä,” “ö,” “ü” are sorted after the vowels “a,” “o,” and “u.”

A longer string that begins with the same characters contained in a shorter string will come after the shorter string regardless of case.

Except in the case of quasi-homographs, diacritics are ignored. Embedded specials are ignored. Ligatures are ignored, being treated as two separate characters.

In the case of quasi-homographs: diacritics and ligatures are considered first. The word with diacritics follows the one without. A quasi-homograph with a ligature comes after one without. Next, case is considered: lowercase comes before uppercase. Finally, specials are considered; the word with special characters follows the one without.

Strings containing only specials (e.g., “@#$%” are sorted first).

Numbers follow specials and precede letters.

SPANISH

Words are sorted according to the alphabetic rank of the letters (“a” comes before “b”). However, the Spanish alphabet is different than English. There are 28 letters in the Spanish alphabet: “ch” follows “cz” and precedes “d”; “ll” follows “lz” and precedes “m”; “ñ” follows “nz” and precedes “o.” These characters are treated as if they were each a single character. A final special character, “rr,” is sorted as two characters, and is only treated as a single character for hyphenation. The letters “k” and “w” are generally omitted, but may be used in foreign names.

A longer string that begins with the same characters contained in a shorter string will come after the shorter string regardless of case.

Except in the case of quasi-homographs, diacritical marks are ignored. Embedded special characters are also ignored. Ligatures are also ignored, being treated as two separate characters.

In the case of quasi-homographs: diacritics and ligatures are considered first. The word with diacritics follows the one without. A quasi-homograph with a ligature comes after one without. Next, case is considered: lowercase comes before uppercase. Finally, specials are considered; the word with special characters follows the one without.

Strings containing only specials (e.g., “@#$%” are sorted first).

Numbers follow specials and precede letters.

DEFINING YOUR OWN COLLATING SEQUENCE

If you are using a language other than the ones listed in Chapter 1, you will need to define your own collating sequence appropriate to the language you are using. To generate a SuperSort collating sequence table, at AMOS command level type:

`COLGEN` RETURN

COLGEN will use the LDFTBL.H file to generate a CULCMP.USA file. As shipped, LDFTBL.H causes a CULCMP file to be created that is compatible with English and most other languages. If, on the other hand, the language you wish to use for sorting purposes does not match this table, you need to modify the LDFTBL.H file in the LCS: account, and recompile COLGEN.C. When finished, rename the resulting file so that it has the proper extension for the language to be used.

Note: When recompiling COLGEN, you must use version 2.0 or later of AlphaC (GNU C); AlphaC 1.X will not work.

Details on the content of LDFTBL.H and how these tables are constructed, may be found in the IBM document *Keys to Sort and Search for Culturally Expected Results* referred to at the beginning of this chapter. You will need to fully understand this material before proceeding with modifying LDFTBL.H.

Document History

Date	Revision	Description
April 1994	00	Initial Release
August 1996	01	Correct code sample and table reference in Chapter 4.

Index

V