digital

VAX–11 SORT User's Guide

Order No. AA-D113A-TE

VAX11

January 1979

This manual describes how to use the VAX-11 native mode SORT utility. The manual is intended for all users.

VAX-11 SORT User's Guide

Order No. AA-D113A-TE

SUPERSESSION/UPDATE INFORMATION:	This is a new document for this release.
OPERATING SYSTEM AND VERSION:	VAX/VMX V01.5
SOFTWARE VERSION:	VAX/VMS V01.5

digital equipment corporation · maynard, massachusetts

First Printing, January 1979

The postage-paid READER'S COMMENTS form on the last page of this document requests your critical evaluation to assist us in preparing future documentation.

DEC	FOCAL
DECnet	IAS
DECsystem-10	MASSBUS
DECSYSTEM-20	PDP
DECtape	RSX
DECUS	UNIBUS
DIBOL	VAX
DIGITAL	VMS

Contents

Tables

Commercial Engineering Publications typeset this manual using DIGITAL's TMS-11 System.

784all

Preface

Intended Audience

This manual is written for the full range of VAX/VMS system users, from beginners to the most advanced level: system operator, applications programmer, system manager, or software developer. Emphasis is on the *how to use* information, and detailed descriptions of SORT internals are kept to a minimum.

You can use SORT as an interactive utility (Chapter 2), or as a set of subroutines, callable from VAX-11 programming languages (Chapter 3).

New users or those with simple sort requirements, can learn how to use SORT by reading Chapters 1 and 2. To use SORT efficiently or for more than simple sorts, read also Chapters 3 and 5.

Structure of this Document

Chapter 1 introduces the VAX-11 SORT program and describes its environment, features, and requirements, and explains user requisites.

Chapter 2 explains how the SORT command is used to run VAX-11 SORT interactively or in batch mode.

Chapter 3 explains how to call SORT routines from user programs, and describes how to use subroutine parameters.

Chapter 4 provides complete lists of SORT error messages and recovery procedures.

Chapter 5 provides information and programming techniques for improving VAX-11 SORT efficiency.

The *Glossary* defines terms used in this manual.

Appendixes A, B and C consist of helpful programming aids such as: code conversion charts, character sets, and data types used by VAX-11 SORT.

Appendix D summarizes basic concepts.

Finally, page references to key terms appear in the index.

Associated Documents

The following documents are relevant to VAX-11 SORT users:

- PDP-11 SORT Reference Manual
- VAX/VMS Primer
- VAX/VMS Summary Description
- VAX/VMS Command Language User's Guide
- VAX/VMS System Messages and Recovery Procedures Manual
- Introduction to VAX-11 Record Management Service
- VAX-11 Record Management Services User's Guide
- VAX-11 Record Management Services Reference Manual
- VAX-11 Software Installation Guide
- VAX/VMS System Services Reference Manual
- VAX-11 Common Run-Time Procedure Library Reference Manual
- VAX-11/780 Architecture Handbook
- VAX-11/780 Processor Handbook
- VAX-11/780 Software Handbook
- VAX-11/780 Technical Summary

Symbology

You will encounter the following symbols, colors, and special graphics in this manual.

dollar sign $	The system prompt; indicates that the VAX/VMS command interpreter is ready for command input. The next $ prompt indicates successful completion of command processing, and the system's readiness to accept another command. In addition, the $ must appear in the first character position of a command to be executed in an indirect command file.
Return RET	Indicates RETURN or ESC key entry required. Pressing this key after entering a full command line ends the command input and begins processing. When using the prompted command format, RET or ESC is required after each command segment.
Square Brackets []	Used in manual text to indicate qualifiers; not entered. Used in command syntax to indicate enclosed portion is optional.
Braces {}	Used in manual text to indicate input options where one in the vertical list must be selected; not entered.
n	Used in this manual text to indicate variable data input (typically some number value); not entered.
Underscore __	Indicates an entered underscore character.
Hyphen -	Indicates line continuation.
Comma ,	Commas are entered to separate listed subqualifiers.
Circumflex ^	Represents the CTRL key on many terminals. Normally entered simultaneously with the alphabetic character that immediately follows. For example, ^C is the same as CTRL/C.
Uppercase Letters ABC	Indicates command inputs that must be entered as shown.
Lowercase Letters abc	Used in text to describe the command syntax; not entered.
Red print	Indicates characters you type at the terminal. All system printouts appear in black print.
Shading	Used to highlight that portion of an example that is being described in text.

Chapter 1 Introduction

VAX-11 SORT rearranges and reformats records in any VAX-11 record management service (VAX-11 RMS) file organization. SORT consists of two functional parts: a control program called the utility, and a callable subroutine package. The utility can be used in an interactive terminal session or in batch mode using the VAX/VMS DIGITAL command language (DCL) SORT command. The callable subroutines are invoked by the SORT utility. Users can write control programs in most VAX-11 languages using these callable subroutines.

You can invoke SORT interactively by entering a SORT command with qualifiers and input/output parameters. The command specifies one of four sort types and the sorting keys. During program execution, SORT indicates all errors. At the completion of each session, SORT prints a statistical summary.

1.1 Sort Types

The four sort types (or sorting processes) are:

- Record Sort
- Tag Sort
- Address Sort
- Index Sort

Record Sort produces a reordered data file sorted by specified key fields (that is, entire records are reordered). This sort uses any VAX/VMS input device and can process any valid VAX-11 RMS format. Record, a relatively slow sort, is the default process.

Tag Sort produces the same kind of output file as record sort by sorting only the record keys. Tag sort then randomly reaccesses the input file to create a resequenced output file according to those record keys. This method conserves temporary storage, but can only accept input files residing on disk.

Address Sort produces an address file. That is, a reordered address file, on disk only, of record's file addresses (RFAs). The address file, sorted by record keys, can be used by programs as an index to read the original file in the desired sequence. This is the fastest of the four sorting processes.

Index Sort produces an address file containing the key field of each data record and a pointer (RFA) to its location in the input file. The address file can be used by programs to randomly access data from the original file in the desired sequence. Like address sort, this is a high-speed process.

For more information on sort types, see Chapter 2.

1.2 Input and Output

As input, SORT accepts sequential, relative or indexed-sequential data files containing records of fixed, variable, or variable with fixed-length control (VFC) format. Character, binary, or decimal data types, and files from disk, magnetic tape, card reader or terminals are accepted.

As output, SORT produces sequential, relative or indexed-sequential data files. These files can be of fixed, variable or VFC format and output to disk, magnetic tape, printer or terminal. In addition, SORT outputs address files (on disk only) for sequential access by programs.

1.3 Statistics

SORT prints statistics at the end of each session. These statistics include:

- Elapsed execution time
- Number of records read, sorted, and output
- The longest record length

For more information on statistics and how they can be useful, see Section 2.2.3 and Chapter 5.

1.4 Functions Supported by VAX–11 SORT

1. Sort types: record, tag, address, index.
2. File organizations as input and output: sequential, relative, indexed-sequential. All VAX–11 RMS file types are supported.
3. Record format for input and output: fixed, variable, and VFC. All VAX–11 RMS record formats are supported.
4. All VAX/VMS devices are supported for input and output.
5. Multivolume support as provided by VAX–11 RMS.

6. Callable subroutine package. VAX-11 programming languages producing native mode code are supported. Included are:

   VAX-11 COBOL-74

   VAX-11 FORTRAN IV-PLUS

   VAX-11 MACRO

   VAX-11 BLISS

7. Controlled by command string or specification file.
8. Free field and fixed position specification file formats.
9. Data Types:
   - Character data is ASCII representation
   - Binary data is VAX representation
   - Packed decimal data is VAX representation
   - Zoned data is VAX representation
   - Decimal data supports:
     - leading separate sign
     - leading overpunched sign
     - trailing separate sign
     - trailing overpunched sign
10. Ascending/descending output based on each key field.
11. Output file blocking and allocation size.
12. Sort statistics provided at completion.
13. ASCII collating sequence for character keys.
14. RSX SORT-11 utility option.

Chapter 2
Running SORT in Interactive and Batch Mode

This Chapter explains how to use the SORT command to sort files interactively or in batch mode.

2.1 The SORT Command

The SORT command consists of three parts: the command name, the input file specification parameter, and the output file specification parameter, in that order. Each part must be separated by one or more spaces or tabs, and is invoked by terminating with RET when the command is entered as a continuous command string.

This section describes how sorts are performed using the SORT command without the specification file qualifier. The specification file is a more sophisticated method of controlling SORT, and therefore is described later in Section 2.5. The specification file is a command qualifier and should not be confused with the file specifications for the input and output files.

Format:

```
           ❶                              ❷
$ SORT[qualifiers] input-file-specification[qualifiers]
   output-file-specification[qualifiers] RET
                        ❸
```

❶ Command Name (SORT)

SORT is the command name that invokes the VAX-11 SORT utility. Command name qualifiers specify the sort process, describe the sorting key(s), specify the number of work files, indicate the specification file if a sort other than a standard sort is to be performed, and finally indicate whether the VAX-11 SORT utility or the RSX SORT-11 utility is to be invoked.

❷ Input File Specification Parameter

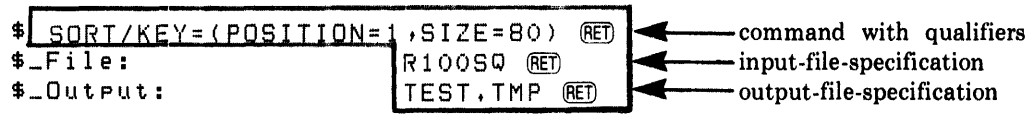
This parameter specifies the physical location of the input file (see Appendix D for additional file specification information). Input file qualifiers define the input file attributes such as record format and file size.

❸ Output File Specification Parameter

This parameter specifies the physical location of the sorted output file (see Appendix D for additional file specification information).

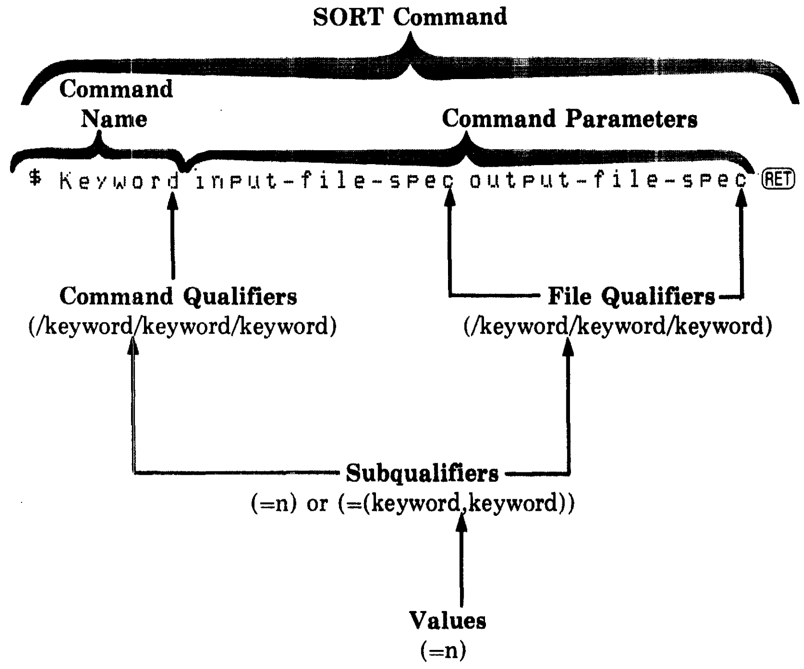
Output file qualifiers define the output file attributes such as record format, record size, block size, file organization, allocation quantity, contiguous allocation, overlay existing file, and bucket size.

The VAX/VMS command interpreter will prompt you for input and output file specifications if they are not entered in the first command string. The following is an example of prompted format:



```
$ SORT/KEY=(POSITION=1,SIZE=80) RET   <---- command with qualifiers
$_File:                R100SQ RET     <---- input-file-specification
$_Output:              TEST.TMP RET   <---- output-file-specification
```

The following example shows how the SORT command is structured:


SORT Command
Command Name
Command Parameters
$ Keyword input-file-spec output-file-spec RET
Command Qualifiers
(/keyword/keyword/keyword)
File Qualifiers
(/keyword/keyword/keyword)
Subqualifiers
(=n) or (=(keyword,keyword))
Values
(=n)

Notes: 1. keywords may be truncated and are unique.
2. n indicates variable data input (typically some number value).

Table 2–1 summarizes all the SORT command qualifiers, subqualifiers, and input values. The complete details on qualifiers and input values are discussed in Section 2.4.

Table 2-1: SORT Command Summary

Notation Used:

- Underlined upper-case characters indicates the minimum entry required.
- Brackets [ ] indicate enclosed portion is optional.
- If several enclosed words are listed vertically, only one may be used.
- Qualifiers, subqualifiers and values that must be specified are shown without brackets.
- Braces { } indicate a selection must be made from the vertical list.
- Defaults are shown in bold type.

Command Qualifiers	Subqualifiers and Values	Notes
$ SORT		
[/PROCESS=	[**RECORD** TAG ADDRESS INDEX]]	
/KEY=	([NUMBER=[1-10]]	/KEY: is not required if specified in a specification file.
	,POSITION=[1-16383]	
	,SIZE={1-255 for CHARACTER data type 1, 2, or 4 for BINARY data type 1-31 for DECIMAL data type}	
	[**,CHARACTER** ,BINARY ,ZONED ,DECIMAL ,PACKED_DECIMAL]	
	[,LEADING_SIGN **,TRAILING_SIGN**]	TRAILING_SIGN is default if data type is DECIMAL.
	[,SEPARATE_SIGN **,OVERPUNCHED_SIGN**]	OVERPUNCHED_SIGN is default if data type is DECIMAL.
	[**,ASCENDING)** ,DESCENDING)]	
[/WORK_FILES=[0,**2**-10]]		
[/SPECIFICATION[=file-specification]]		SYS$INPUT is default file name.
[/RSX11]		VAX-11 SORT is default. /RSX11 requires PDP-11 SORT command switches. Refer to the *PDP-11 SORT Reference Manual.*

(continued on next page)

Table 2-1: SORT Command Summary (continued)

Input File Qualifiers	Subqualifiers and Values	Notes
input-file-specification		See Appendix D for file-specifications. DAT is default file-type.
[/FORMAT=	(RECORD_SIZE=[1-16383]]	RECORD_SIZE is not normally specified.
	[,FILE_SIZE=[1-4294967295])]	FILE_SIZE is not normally specified.

Output File Qualifiers	Subqualifiers and Values	Notes
output-file-specification		See Appendix D for file specifications. The default output file type is the same as input file type.
[/FORMAT] =	[(**FIXED**=[1-16383] / (VARIABLE=[1-16383] / (CONTROLLED=[1-16383]]	FIXED record format is default if sort process is index or address.
	[,SIZE=[1-255]]	Used for VFC records only. Default value is 2 if CONTROLLED is specified and SIZE is not.
	[,BLOCK_SIZE=[18-32767])]	For magnetic tape files only.
[**/SEQUENTIAL** / /RELATIVE / /INDEXED_SEQUENTIAL]		Default is the input file organization if sort process is record or tag, otherwise /SEQUENTIAL is default. If /INDEXED_SEQUENTIAL is specified, /OVERLAY must be specified.
[/ALLOCATION=[1-4294967295]]		Required if /CONTIGUOUS is specified. The default value is determined by the number of records sorted.
[/CONTIGUOUS]		/NOCONTIGUOUS is default. /CONTIGUOUS is invalid if /ALLOCATION is not specified.
[/OVERLAY]		/NOOVERLAY is default. /OVERLAY is required if /INDEXED SEQUENTIAL output file organization is specified.
[/BUCKET_SIZE=[1-32]]		Default value is the same as the input file value if the input and output file organizations are the same, otherwise default is 1.

2.2 Interactive Sessions

To invoke SORT in interactive mode simply enter the SORT command. Any errors in the command are immediately reported at your terminal (see Chapter 4, Error Conditions). At the end of a successful run, SORT prints the statistics message (see Section 2.2.3).

SORT accepts two kinds of command formats: a keyboard-oriented command string containing all the command qualifiers (excluding /SPECIFICATION), or a keyboard-oriented command string containing the /SPECIFICATION qualifier pointing to a specification file containing the command qualifiers.

For example:

```
$ SORT/KEY=(POSITION=1,SIZE=10) input-file-specification
   output-file-specification RET
```

or:

```
$ SORT/SPECIFICATION=file-specification
   input-file-specification output-file-specification RET
```

The use of the specification file is the more involved method and therefore explained in Section 2.5.

In order to specify a sorting sequence, you must select key fields within the data itself. Remember, SORT reorders the entire file. The information provided in Section 2.6 can help you to set up the key fields (keys).

You can extract key information from a file and store it in a reordered format for future use in accessing data in your original file in the order of your reordered file. In addition, the contents of your sorted file can be entire records, key fields with record pointers, or record indices relative to the position of each record within the file. Your intentions for the sorted output file usage, together with input and output file organizations, determine what sort process to use. The information provided in Section 2.2.2 can help you to choose the correct sorting process.

Because SORT is designed to process all RMS file organizations, you also must consider how to direct the sorting process you have chosen, so that your output file organization will be usable on your peripheral device. The information provided in Section 2.2.2 and Table 2–2 compares file organizations and sorting processes.

If your sorting task requires more than two work files, Section 2.7 can help you to set up additional work files. Most sorts will normally use the default number of work files.

Finally, you must specify input and output file specifications. Appendix D reviews the standard VAX/VMS file-specification information, and file specification qualifiers are summarized in Table 2–1, and described in detail in Section 2.4.

2.2.1 A Sample Sort

Users can invoke the SORT command by simply providing the required key position and size for a single key and the file name of a single input file located on the user's default disk.

The format of the command is:

```
$ SORT/KEY=(POSITION=[1-16383],SIZE=[1-255])
   input-file-specification output-file-specification RET
```

This means:

- A record sorting process is performed on the specified input file
- The input file key data type must be character
- The input file must reside on the user's default disk
- All the records in the input file are reordered in the output file in ascending alphabetic order
- Input file type DAT is assigned, and output file type DAT is assigned
- SORT assigns two work files for temporary storage
- Output file organization is the same as the input file organization
- Output file record format is the same as input records format
- Output file bucket size is the same as input file bucket size
- SORT statistics are printed at the terminal that executed the sort

An abbreviated representation of the preceding command example is:

```
$ SORT/K=(PO=1,SI=80) INPUT OUTPUT RET
```

Description:

If you specify the key position and size, and character data type by default; this sort reads the single input file specified (on the user's default disk), sets up two work files on the user's default disk, and performs a record sort.

This process creates an output file named OUTPUT.DAT having the same file organization as the input file. All the records in INPUT.DAT are reordered in ascending alphabetic order in the output file. The alphabetic order is determined by the contents of the 80-character key field (SI=80) starting in position one (PO=1) of each record.

NOTE:

A quick test can be run at your terminal by using SYS$OUTPUT as the output-file-specification. This technique displays the sorted output file before the sort statistics are printed.

Finally, upon completion of the run the sort statistics are printed at the terminal that executed the sort.

2.2.2 Selecting the Sort Type

SORT offers a choice of four processes: record, tag, address, and index. You specify the sort process by using the proper qualifier in the command or in the specification file code. Each process has its particular input requirements, processing methods, device requirements, and resultant output files.

SORT provides four sorting techniques:

- **RECORD** (/PROCESS=RECORD, or SORTR if specification file)
  Record sort produces a reordered data file sorted by specified key fields (that is, entire records are reordered). This sort can be used on any acceptable input device, and can process any valid VAX-11 RMS format. Record, a relatively slow sort, is the default process.

- **TAG** (/PROCESS=TAG, or SORTT if specification file)
  Tag sort produces the same kind of reordered data file as record sort by sorting only the record keys. This method conserves temporary storage, but can only accept input files residing on disk. Tag sort is faster than record sort, if the key size is much smaller than the record size and the file size is small so that the reaccessing process is short.

- **ADDRESS** (/PROCESS=ADDRESS, or SORTA if specification file)
  Address sort produces an address file without reordering the input file. That is, a reordered address file (on disk only) of record's file addresses (RFAs).

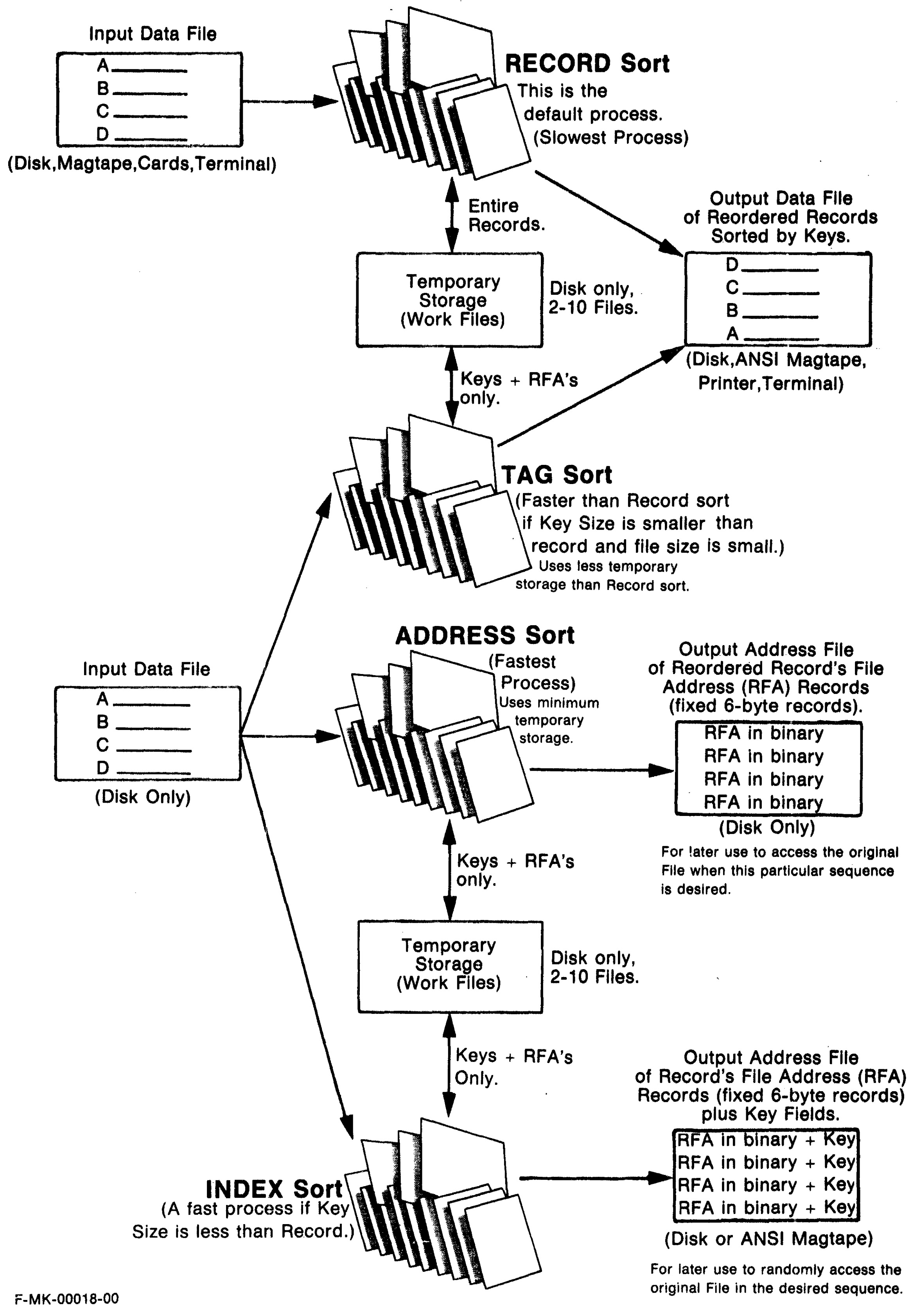
  The address file, sorted by record keys, can later be used as an index* to read the original file in the desired sequence. Any number of address files may be created for the same data base. A customer master file, for instance, may be referenced by either customer-number index or sales-territory index for different reports. This is the fastest of the four sorting processes.

- **INDEX** (/PROCESS=INDEX, or SORTI if specification file)
  Index sort produces an address file containing the key field of each data record and a pointer (RFA) to its location in the input file. The address file can be used by programs to randomly access data from the original file in the desired sequence. Like Address sort, this is a high speed process.

Figure 2-1 summarizes these options to help you determine which process is best for your sorting application. Chapter 5 provides additional information regarding sorting processes where performance considerations are important.

* Not indexed by VAX-11 RMS.

Figure 2-1: SORT's Four Sorting Processes


Input Data File
A
B
C
D
(Disk,Magtape,Cards,Terminal)
RECORD Sort
This is the default process. (Slowest Process)
Entire Records.
Temporary Storage (Work Files)
Disk only, 2-10 Files.
Output Data File of Reordered Records Sorted by Keys.
D
C
B
A
(Disk,ANSI Magtape, Printer,Terminal)
Keys + RFA's only.
TAG Sort
(Faster than Record sort if Key Size is smaller than record and file size is small.)
Uses less temporary storage than Record sort.
Input Data File
A
B
C
D
(Disk Only)
ADDRESS Sort
(Fastest Process)
Uses minimum temporary storage.
Output Address File of Reordered Record's File Address (RFA) Records (fixed 6-byte records).
RFA in binary
RFA in binary
RFA in binary
RFA in binary
(Disk Only)
For later use to access the original File when this particular sequence is desired.
Keys + RFA's only.
Temporary Storage (Work Files)
Disk only, 2-10 Files.
Keys + RFA's Only.
Output Address File of Record's File Address (RFA) Records (fixed 6-byte records) plus Key Fields.
RFA in binary + Key
RFA in binary + Key
RFA in binary + Key
RFA in binary + Key
(Disk or ANSI Magtape)
For later use to randomly access the original File in the desired sequence.
INDEX Sort
(A fast process if Key Size is less than Record.)
F-MK-00018-00

File I/O Considerations

Input and output file organizations are another important factor in determining which sort type to use. Figure 2-2 shows how the I/O flows through SORT, and Table 2-2 list all possible I/O combinations and shows the default output file organizations.

Inputs to VAX-11 SORT can be files of sequential, relative, or indexed organization containing records of fixed, variable, or VFC format from disk, magnetic tape, card reader, or terminals.

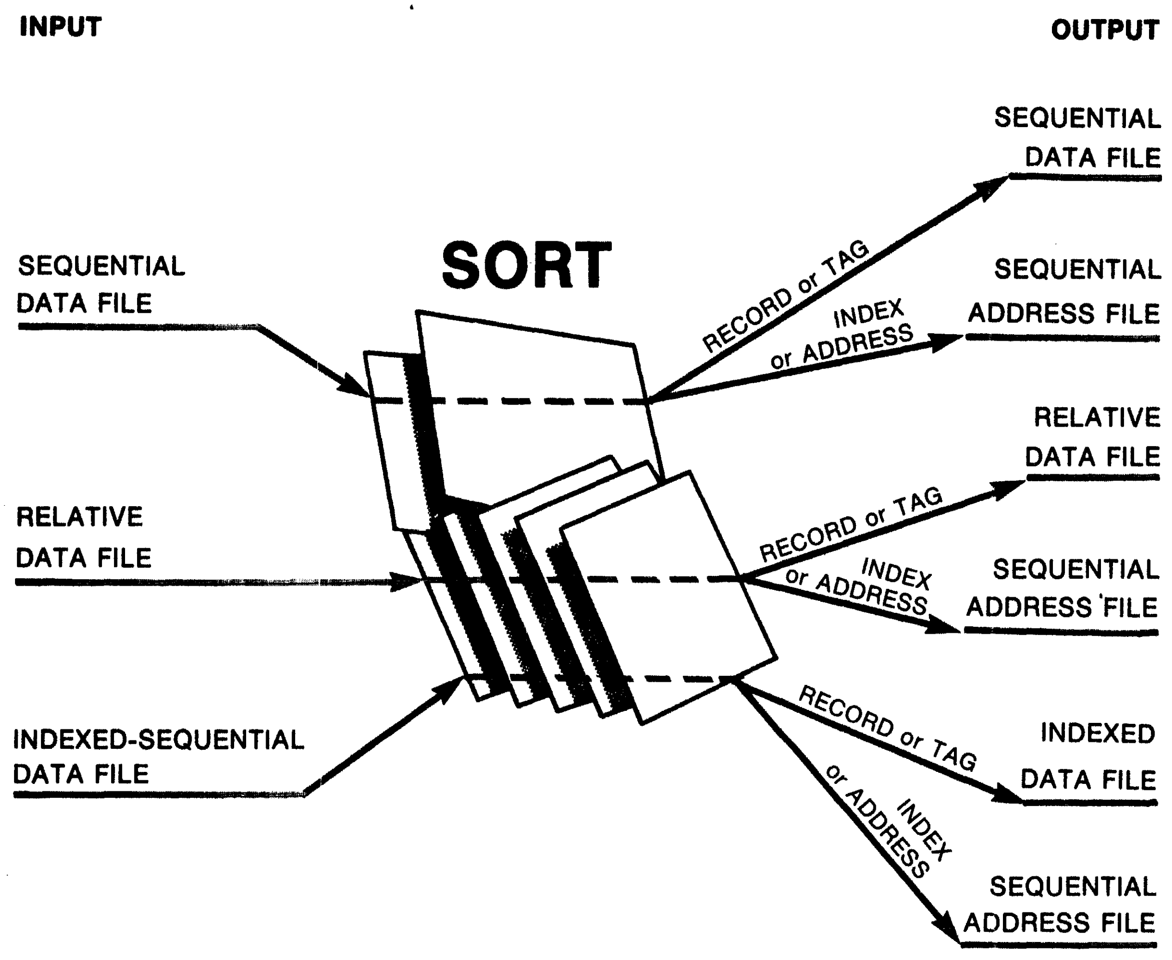
Input parameters to the sort program are either provided by RMS after processing the input file header records, or specified in the command in the form of input-file-specification qualifiers (that is, /FORMAT ...).

Outputs from VAX-11 SORT are files of records reordered by key fields and are created in sequential, relative, or indexed organization. These files may contain record types of fixed, variable, or VFC format. Output files can be written to disk, magnetic tape, printer, or terminals.

Sorted output address files of 6-byte RFAs in binary coded records are output to disk only for sequential access by programs. *These output address files are intended for software use as indices into input files, and cannot be output to printers or terminals without further processing.*

Output parameters to the sort program are specified in the command in the form of output-file-specification qualifiers (that is, /FORMAT ...).

Figure 2-2: File Organization I/O Flow



NOTES:

1. **RECORD** & **TAG** produce reordered data files of the same organization as input by default.
2. **INDEX** produces reordered address files of RFA's plus keys in sequential file organization.
3. **ADDRESS** produces reordered address files of RFA's only in sequential file organization.

F-MK-00019-00

Table 2-2: File I/O Considerations

Type of Input File	Sort Process	Output File Organization Specified	Results
Sequential Data File	Record	*Sequential*	Reordered sequential data file.
		Relative	Reordered file of data records.
		Indexed-Seq	Populates (overlays) an already existing Indexed-Sequential output file with reordered data records.
	Tag	*Sequential* Relative Indexed-Seq	Same as for record.
	Address	*Sequential* Relative Indexed-Seq	Sequential address file of RFAs.
	Index	*Sequential* Relative Indexed-Seq	Sequential address file of RFAs with keys.
Relative Data File	Record	Sequential *Relative* Indexed-Seq	Same as above for each process.
	Tag	Sequential *Relative* Indexed-Seq	
	Address	*Sequential* Relative Indexed-Seq	
	Index	*Sequential* Relative Indexed-Seq	
Indexed-Sequential Data File	Record	Sequential Relative *Indexed-Seq*	Same as above for each process.
	Tag	Sequential Relative *Indexed-Seq*	
	Address	*Sequential* Relative Indexed-Seq	
	Index	*Sequential* Relative Indexed-Seq	

Note: The default output file organization is shown in *italic* type.

2.2.3 SORT Statistics

Statistics are automatically printed at the completion of each sort session. These consist of: elapsed execution time, the number of records read, sorted, and output; the longest record length; the multiblock count used and the multibuffer count used for input and output; the merge order; the number of merge passes; the working set size used; the number of initial runs; and the virtual memory used for the sort tree.

In addition, SORT statistics include statistics kept by VAX/VMS for the number of buffered and direct I/O operations, CPU time, and the number of page faults. Figure 2-3 illustrates a typical SORT statistics printout of a single sequential input file (filename R100SQ.DAT) that is 10,000 records in length, and each record is 80 characters long. The sorting is done on an 80 character key starting at position 1 of each record. The output filename is TEST.TMP and is output in the same format as the input file by default.

The command string that caused the sample printout in Figure 2-3 was:

```
$ SORT/KEY=(POS=1,SIZE=80) RET
$_File: R100SQ RET
$_Output:TEST.TMP RET
```

The statistics can be used to help tune the parameters you specify for a specific sort, such as the best working set quota size to use (see Chapter 5, Section 5.2.2.5).

Figure 2-3: Sample Sort Statistics Printout

```
                              SORT STATISTICS:

RECORDS READ:    10000                LONGEST RECORD LENGTH:  80
RECORDS SORTED: 10000                 INPUT MULTI BLOCK COUNT: 11
RECORDS OUTPUT: 10000                 OUTPUT MULTI BLOCK COUNT: 20
MAXIMUM WORKING SET USED: 128 ❶       INPUT MULTI BUFFER COUNT: 2
VIRTUAL MEMORY ADDED: 236032 ❷        OUTPUT MULTI BUFFER COUNT: 2
DIRECT IO COUNT: 227                  NUMBER OF INITIAL RUNS: 38
BUFFERED IO COUNT: 23                 ORDER OF THE MERGE:     7
PAGE FAULTS:  15596                   NUMBER OF MERGE PASSES: 2
ELAPSED TIME: 00:02:26.97 ❸           CPU TIME: 5055 ❹
$
```

Notes: ❶ Maximum working set used is in blocks.
❷ Virtual memory added is in bytes.
❸ Elapsed time is the total sort run time from start to end in hrs: min: sec. 1/100secs.
❹ CPU time is the data processing time less I/O time in 1/100secs. (that is, 5055 is 50 seconds and 55/100th's seconds.)

2.2.4 Samples

Figure 2-4 shows a step-by-step session for an interactive sort on a single key. Figure 2-5 shows how an interactive sort would appear when sorting on two keys.

Figure 2-4: Interactive Session Sample #1

Step 1: Observe the input file you want to sort to determine where the key fields are located and their size.

To sort this input file named BOATS.LST in alphabetic order by manufacturer you must specify a single key field starting at character position 2 and having a key field size of 10 characters.

```
MANUFACTURER  MODEL      RIG   LENGTH  WEIGHT BEAM   PRICE

 NORTHERN    37         KETCH   37     14,000  11  $50,000
 CHALLENGER  41         KETCH   41     26,700  13  $51,228
 OLYMPIC     ADVENTURE  KETCH   42     24,250  13  $80,500
 EASTWARD    HO         M/S     24      7,000  09  $15,900
 AMERICAN    26-MS      M/S     26      5,500  08  $18,895
 LINDSEY     39         M/S     39     14,500  12  $35,900
 WINDPOWER   IMPULSE    SLOOP   16        650  07   $3,500
 CAPE DORY   TYPHOON    SLOOP   19      1,900  06   $4,295
 VENTURE     222        SLOOP   22      2,000  07   $3,564
 SALT        19         SLOOP   25      2,600  07   $6,590
 AMERICAN    26         SLOOP   26      4,000  08   $9,895
 HUNTER      27         SLOOP   27      6,500  09  $14,999
 TANZER      28         SLOOP   28      6,800  10  $17,500
 ALBIN       BALLAD     SLOOP   30      7,276  10  $27,500
 GRAMPIAN    2-34       SLOOP   34     11,800  10  $29,675
 CHRIS-CRAF  CARIBBEAN  SLOOP   35     18,000  11  $37,850
 ISLANDER    36         SLOOP   36     13,450  11  $31,730
 COLUMBIA    41         SLOOP   41     20,700  11  $48,490
```

/KEY=(POSITION=2,SIZE=10)

Step 2: Enter the following SORT command to sort the input file named BOATS. LST and create an output file named BOATS.ALB:

```
$ SORT/KEY=(POS=2,SIZE=10) BOATS.LST BOATS.ALB RET
```

Step 3: Observe this printout when SORT has completed.

```
                              SORT STATISTICS:

RECORDS READ:    18                 LONGEST RECORD LENGTH:  57
RECORDS SORTED: 18                  INPUT MULTI BLOCK COUNT: 20
RECORDS OUTPUT: 18                  OUTPUT MULTI BLOCK COUNT: 32
MAXIMUM WORKING SET USED: 200       INPUT MULTI BUFFER COUNT: 2
VIRTUAL MEMORY ADDED: 404992        OUTPUT MULTI BUFFER COUNT: 2
DIRECT IO COUNT: 2                  NUMBER OF INITIAL RUNS: 0
BUFFERED IO COUNT: 17               ORDER OF THE MERGE:     0
PAGE FAULTS:  146                   NUMBER OF MERGE PASSES: 0
ELAPSED TIME:  00:00:01.97          CPU TIME: 54
$
```

Step 4: Examine your newly sorted output file named BOATS.ALB. Notice that the records are now in alphabetical order.

```
MANUFACTURER  MODEL      RIG   LENGTH  WEIGHT BEAM   PRICE

 ALBIN       BALLAD     SLOOP   30      7,276  10   $27,500
 AMERICAN    26         SLOOP   26      4,000  08    $9,895
 AMERICAN    26-MS      M/S     26      5,500  08   $18,895
 CAPE DORY   TYPHOON    SLOOP   19      1,900  06    $4,295
 CHALLENGER  41         KETCH   41     26,700  13   $51,228
 CHRIS-CRAF  CARIBBEAN  SLOOP   35     18,000  11   $37,850
 COLUMBIA    41         SLOOP   41     20,700  11   $48,490
 EASTWARD    HO         M/S     24      7,000  09   $15,900
 GRAMPIAN    2-34       SLOOP   34     11,800  10   $29,675
 HUNTER      27         SLOOP   27      6,500  09   $14,999
 ISLANDER    36         SLOOP   36     13,450  11   $31,730
 LINDSEY     39         M/S     39     14,500  12   $35,900
 NORTHERN    37         KETCH   37     14,000  11   $50,000
 OLYMPIC     ADVENTURE  KETCH   42     24,250  13   $80,500
 SALT        19         SLOOP   25      2,600  07    $6,590
 TANZER      28         SLOOP   28      6,800  10   $17,500
 VENTURE     222        SLOOP   22      2,000  07    $3,564
 WINDPOWER   IMPULSE    SLOOP   16        650  07    $3,500
```

Figure 2-5: Interactive Session Sample #2

Step 1: Observe the input file you want to sort to determine where the key fields are located and their size.

To sort this input file named BOATS.LST in ASCII alphanumeric order first by beam, and then by price, you must specify two keys. The first key (or primary key) field starts at character position 47 and has a size of 2. The second key starts at character position 51 and has a size of 7.

```
MANUFACTURER  MODEL       RIG   LENGTH  WEIGHT BEAM  PRICE

 NORTHERN     37          KETCH   37    14,000  11  $50,000
 CHALLENGER   41          KETCH   41    26,700  13  $51,228
 OLYMPIC      ADVENTURE   KETCH   42    24,250  13  $80,500
 EASTWARD     HO          M/S     24     7,000  09  $15,900
 AMERICAN     26-MS       M/S     26     5,500  08  $18,895
 LINDSEY      39          M/S     39    14,500  12  $35,900
 WINDPOWER    IMPULSE     SLOOP   16       650  07   $3,500
 CAPE DORY    TYPHOON     SLOOP   19     1,900  06   $4,295
 VENTURE      222         SLOOP   22     2,000  07   $3,564
 SALT         19          SLOOP   25     2,600  07   $6,590
 AMERICAN     26          SLOOP   26     4,000  08   $9,895
 HUNTER       27          SLOOP   27     6,500  09  $14,999
 TANZER       28          SLOOP   28     6,800  10  $17,500
 ALBIN        BALLAD      SLOOP   30     7,276  10  $27,500
 GRAMPIAN     2-34        SLOOP   34    11,800  10  $29,675
 CHRIS-CRAF   CARIBBEAN   SLOOP   35    18,000  11  $37,850
 ISLANDER     36          SLOOP   36    13,450  11  $31,730
 COLUMBIA     41          SLOOP   41    20,700  11  $48,490
```

/KEY=(POS=47,SIZE=2)

/KEY=(POS=51,SIZE=7)

Step 2: Enter the following SORT command to sort the input file named BOATS.LST and create an output file named BOATS.BEM:

```
$ SORT/KEY=(POS=47,SIZE=2)/KEY=(POS=51,SIZE=7)
    BOATS.LST BOATS.BEM RET
```

Step 3: Observe this printout when SORT has completed.

```
                              SORT STATISTICS:

RECORDS READ:    18                 LONGEST RECORD LENGTH:  57
RECORDS SORTED: 18                  INPUT MULTI BLOCK COUNT: 20
RECORDS OUTPUT: 18                  OUTPUT MULTI BLOCK COUNT: 17
MAXIMUM WORKING SET USED: 200       INPUT MULTI BUFFER COUNT: 2
VIRTUAL MEMORY ADDED:  202240       OUTPUT MULTI BUFFER COUNT: 2
DIRECT IO COUNT: 2                  NUMBER OF INITIAL RUNS: 0
BUFFERED IO COUNT: 17               ORDER OF THE MERGE:     0
PAGE FAULTS:  142                   NUMBER OF MERGE PASSES: 0
ELAPSED TIME:  00:00:01.85          CPU TIME: 45
$
```

Step 4: Examine your newly sorted output file named BOATS.BEM. Notice that the records are now in order first by beam width, and second by price.

```
MANUFACTURER  MODEL       RIG  LENGTH   WEIGHT BEAM   PRICE

 CAPE DORY    TYPHOON    SLOOP   19      1,900  06    $4,295
 WINDPOWER    IMPULSE    SLOOP   16       650   07    $3,500
 VENTURE      222        SLOOP   22      2,000  07    $3,564
 SALT         19         SLOOP   25      2,600  07    $6,590
 AMERICAN     26         SLOOP   26      4,000  08    $9,895
 AMERICAN     26-MS       M/S    26      5,500  08   $18,895
 HUNTER        27        SLOOP   27      6,500  09   $14,999
 EASTWARD     HO         M/S     24      7,000  09   $15,900
 TANZER        28        SLOOP   28      6,800  10   $17,500
 ALBIN        BALLAD     SLOOP   30      7,276  10   $27,500
 GRAMPIAN     2-34       SLOOP   34     11,800  10   $29,675
 ISLANDER     36         SLOOP   36     13,450  11   $31,730
 CHRIS-CRAF   CARIBBEAN  SLOOP   35     18,000  11   $37,850
 COLUMBIA     41         SLOOP   41     20,700  11   $48,490
 NORTHERN     37         KETCH   37     14,000  11   $50,000
 LINDSEY      39         M/S     39     14,500  12   $35,900
 CHALLENGER   41         KETCH   41     26,700  13   $51,228
 OLYMPIC      ADVENTURE  KETCH   42     24,250  13   $80,500
```

2.3 Batch Sessions

To run the same sort as shown in Figure 2-4 using batch mode, perform the following steps:

Step 1: Create a command file named BOATS1.COM as follows:

```
$ PRINT BOATS.LST
$ SORT/KEY=(POS=2,SIZE=10) BOATS.LST BOATS.ALB
$ PRINT BOATS.ALB
```

Step 2: Enter this command:

```
$ SUBMIT BOATS1.COM RET
```

Observe this response:

```
Job n entered on queue SYS$BATCH
```

Step 3: Observe that the input file, output file, sort statistics, and batch statistics are all printed on the system printer.

2.4 The SORT Command Description

NOTE:

Review the SYMBOLOGY in the front of this manual before continuing.

Format:

```
$ SORT[qualifiers] input-file-specification[qualifiers]
   output-file-specification[qualifiers] RET
```

2.4.1 Command Name Qualifiers

Abbreviated Example:

```
$ SORT/PROCESS=n/KEY=(n)/WORK_FILES=n/SPECIFICATION=n
   input-file-specification[qualifiers]
   output-file-specification[qualifiers] RET
```

$$\left[\ \text{/PROCESS=}\quad \left[\begin{array}{l}\textbf{RECORD}\\ \text{TAG}\\ \text{ADDRESS}\\ \text{INDEX}\end{array}\right]\ \right]$$

Indicates the type of sort to be performed. /PROCESS=RECORD is the default.

/KEY=

This qualifier must be specified unless defined in a specification file. It defines a sorting key, and may appear several times in a single command string in order to specify several sort keys (up to 10).

NOTE:

The /KEY subqualifiers group must be enclosed in parentheses.

([NUMBER=n]

n specifies the precedence of the sort key being defined, where 1 is the primary sort key, 2 is the secondary sort key, and so on. If this option is not specified on the first /KEY qualifier, NUMBER=1 is assumed. If this option is not specified on subsequent /KEY qualifiers, the default NUMBER value is the NUMBER value of the previous key plus 1. Legal values are 1 - 10.

,POSITION=n

n specifies the position of the key within each record, where the first character of the record is 1. This subqualifier input must be specified.

,SIZE=n

n specifies the length of the sort key in either characters, bytes, or digits, depending on the key field data type. This subqualifier input must be specified. If the sort key data type is CHARACTER, key size must be less than or equal to 255 characters. If the data type is binary, key size must be 1, 2, or 4 bytes. If the data type is any of the decimal types, key size must be less than or equal to 31 digits. The total of all key field sizes must be less than or equal to 255 bytes. See Section 2.6 for additional key size information.

[**,CHARACTER**
,BINARY
,ZONED
,DECIMAL
,PACKED_DECIMAL]

This subqualifier indicates the type of data appearing in the sort key field. See Section 2.6 for data type descriptions. CHARACTER is the default.

[,LEADING_SIGN]
[,**TRAILING_SIGN**]

This subqualifier indicates whether the sign of a decimal data type key appears at the beginning or end of the key. If the key data type is DECIMAL and this option is not specifed, TRAILING_SIGN is the default. See Section 2.6 for key descriptions.

[,**OVERPUNCHED_SIGN**]
[,SEPARATE_SIGN]

This subqualifier indicates whether the sign of a decimal data type key is superimposed on the decimal value or is separate from the decimal value. If the key data type is DECIMAL and this option is not specified, OVERPUNCHED_SIGN is the default. See Section 2.6 for key descriptions.

[,**ASCENDING**])
[,DESCENDING]

Indicates whether the key is to be sorted into ascending or descending order. ASCENDING is the default value.

[/WORK_FILES=n]

n specifies the number of temporary work files to be used during the sort. Values of 0, or from 2 to 10 may be used. Default value is 2. 0 specifies no work files because data will fit in real memory. See Section 2.7 for additional information.

[/SPECIFICATION[=file-specification]]

Specifies the name of a file which contains SORT specification statements. If this qualifier is not specified, a standard sort is performed. See Section 2.5 and Appendix D for additional information. SYS$INPUT is the default value.

[/RSX11]

Indicates that SORT-11 (/RSX11) is to be invoked. The SORT-11 command format and switches are not described in this manual. Refer to the *PDP-11 SORT Reference Manual* when using the /RSX11 qualifier. VAX-11 SORT is the default value.

NOTE:

Only the minimal unique abbreviated form of qualifier and parameter inputs are required, but all four character abbreviations are accepted (for example, enter SPE= for SPECIFICATION=SYS$INPUT).

An abbreviated example is:

```
$ SORT/KEY=(NUM=1,POS=12,SIZE=2,DECI)/SPE=
   input-file-specification[qualifiers]
   output-file-specification[qualifiers] RET
```

The actual example including defaults is:

```
$SORT/PROCESS=RECORD/KEY=-
 (NUMBER=1,POSITION=12,SIZE=2,DECIMAL,TRAILING_SIGN,-
 OVERPUNCHED_SIGN,ASCENDING)/WORK_FILES=2-
 /SPECIFICATION=SYS$INPUT -
   input-file-specification[qualifiers]
   output-file-specification[qualifiers] RET
```

2.4.2 Input-File-Specification Qualifiers

Defines input file attributes.

Format:

```
$ SORT[qualifiers] input-file-specification-
   /FORMAT=(RECORD_SIZE=n,FILE_SIZE=n)
   output-file-specification[qualifiers] RET
```

NOTE:

If the input file name does not contain a file type, the default file type becomes DAT.

If only one FORMAT subqualifier is specified, the parentheses () can be omitted.

[/FORMAT=(RECORD_SIZE=n]

This input should be used only to override the record size input normally retrieved from RMS. Omitting RECORD_SIZE indicates that the file record format is to be obtained from the file header or label. n specifies the longest record length (LRL) in bytes. The LRL input is optional, but should be specified if the input file is not on disk or is inaccurate. The longest record length allowed is 16,383 bytes (not including control bytes). For additional information on determining the LRL, refer to the $FAB MRS parameter in the *VAX-11 Record Management Services Reference Manual.* Note, stream format is not supported because VAX-11 RMS does not support it.

[,FILE_SIZE=n)]

This input should only be used to supply the file size normally provided by RMS when the input file is not on disk. This input is used to determine the size of the work files based on input file size. n specifies the input file size in blocks. Default is 1000 if file size cannot be obtained from RMS and is not specified by the user. Maximum file size is 4,294,967,295 blocks.

2.4.3 Output-File-Specification Qualifiers

Defines output file attributes.

Format:

```
$ SORT[qualifiers] input-file-specification[qualifiers]
  output-file-specification-
  /FORMAT=(CONTROLLED=n,SIZE=n,BLOCK_SIZE=n)
  /INDEXED_SEQUENTIAL/ALLOCATION=n/CONTIGUOUS/OVERLAY
  /BUCKET_SIZE=n RET
```

NOTE:

If the output file name does not contain a file type, the output file type becomes the same as the input file type.

If only one FORMAT option is specified, parentheses () may be omitted.

[/FORMAT=]

(FIXED=n
(VARIABLE=n
(CONTROLLED=n

Indicates the output file record format. n specifies the longest record length (LRL) of the output records in bytes, and is optional. The longest record length allowed is 16,383 bytes (less any control bytes). Default is input file record format if record or tag sort, and FIXED if index or address sort. For additional information on determining the LRL, refer to the $FAB MRS parameter in the *VAX-11 Record Management Services Reference Manual.*

[,SIZE=n]

This input applies to CONTROLLED records only. That is, variable with fixed-length controlled (VFC) records. n specifies the size in bytes of the fixed portion of controlled records. Maximum fixed control area size is 255 bytes. If CONTROLLED is specified, and SIZE is not, default is two bytes.

[,BLOCK_SIZE=n)]

This input applies to magnetic tape files only. n specifies the block length in bytes of the output file. Default value is the block size of the input tape file, or that which was established at tape mounting time. Block length must be in the range of 18 to 65,535 bytes.

NOTE:

To ensure for correct data interchange with other DIGITAL systems, you should specify a block size less than or equal to 512 bytes. To ensure compatibilty with most non-DIGITAL systems, the block size should be less than or equal to 2048 bytes.

```
[ /SEQUENTIAL          ]
[ /RELATIVE            ]
[ /INDEXED_SEQUENTIAL  ]
```

Indicates the organization of the output file. If /INDEXED_SEQUENTIAL is specified, the output file must already exist and must be empty; therefore, /OVERLAY must be specified. Default is the input file organization if a record or tag sort (/PROCESS= RECORD or /PROCESS=TAG) is performed. Otherwise, /SEQUENTIAL is default.

[/ALLOCATION=n]

n specifies the number of 512-byte blocks of disk space to be allocated for the output file. The default value is whatever the output requires based on the number of records sorted. Blocks allocated must be in the range of 1 to 4,294,967, 295.

[/CONTIGUOUS]

Indicates contiguous allocation of blocks for output file. Default is /NOCONTIGUOUS. This qualifier is invalid if /ALLOCATION is not specified, or if /ALLOCATION value is insufficient for total output and the file must be extended.

[/OVERLAY]

/OVERLAY indicates that an existing file which has the same name as the output file should be overwritten with the SORT output. /OVERLAY requires that the existing file must be empty. Default is /NOOVERLAY.

[/BUCKET_SIZE=n]

n specifies the RMS bucket size (that is, the number of 512-byte blocks per bucket) for the output file. If the output file has the same organization as the input file, the default value is the same as input file bucket size. The maximum number of blocks per bucket is 32. If the output file organization is different from the input file organization, the default value is 1.

2.5 Specification File

Use of the /SPECIFICATION qualifier in the SORT command allows SORT to be controlled by SORT specification statements. These statements are contained in the header record and field records of a specification file, and provide a means for expanding the range of sorting features.

Having sort processes controlled by specification files enables dynamic program control of specification file statements, and therefore dynamic control of subsequent sort processes using the same specification file modified. Also, specification file libraries can be maintained for often-used sorts.

The command string for a typical standard sort using a specification file would look like this:

```
$ SORT/SPECIFICATION[=specification file]
   input-file-specification output-file-specification (RET)
```

There are several methods of entering the /SPECIFICATION qualifier. If you allow the predetermined specification file statements to control the sort, SORT will run automatically with no further operator prompts.

Example:

```
$ SORT/SPECIFICATION[=file-specification
  of the predetermined specification file]
   input-file-specification
   output-file-specification (RET)
```

However, if you use the default specification file (that is, =SYS$INPUT, and providing your terminal is set to be the input device), SORT will prompt you for the specification file values.

Example:

```
$ SORT/SPECIFICATION
   input-file-specification
   output-file-specification (RET)
PLEASE ENTER SPECIFICATION FILE RECORDS.
_enter the specification file header record values (RET)
_enter the specification file field record values for
   the 1st key field (RET)
                                   .
                                   .
                                   .
                                   .
_enter the specification file field record values for
   the last key field (CTRL/Z)
```

If you allow SORT to prompt you for the input and output files as well as the specification file values, then the SORT command will be input in the following sequence:

```
$ SORT/SPECIFICATION RET
_file: input-file-specification RET
_OUTPUT: output-file-specification RET
PLEASE ENTER SPECIFICATION FILE RECORDS.
-enter the specification file header record values RET
-enter the specification file field record values for
   the 1st key field RET
                                    .
                                    .
                                    .
                                    .
-enter the specification file field record values for
   the last key field CTRL/Z
```

2.5.1 Specification File Records

The specification file records can have either of two formats; fixed position field format (SORT–11), and logical position field format (VAX–11 SORT).

NOTE:

Since omit/include and alternate collating sequences are not supported, ALTSEQ records and record type records are invalid and cause errors. Only header and field specification records are processed.

Fixed Position Field Format (SORT–11)

In order to allow ease of conversion from SORT–11 V2 use to VAX–11 SORT, the existing fixed-position-fields format of SORT–11 is accepted.

Free Field Format (VAX–11 SORT)

To allow some flexibility for new users, fields may be separated by commas, and records may be variable length up to 132 characters. Blanks are ignored unless they are embedded within a field, such as 1 00, in which case an error is generated. Continuation lines are supported as in DCL. The individual fields, their length, meaning and order are identical to SORT–11.

Comments in this format are placed at the end of the line by placing an exclamation point (!) immediately before the comment. The format for a VAX–11 SORT header record would look like:

```
Page number,line number,H,sort type,total key field size,
sorting order,collating sequence,output key,record length
!comment
```

A specific example would be:

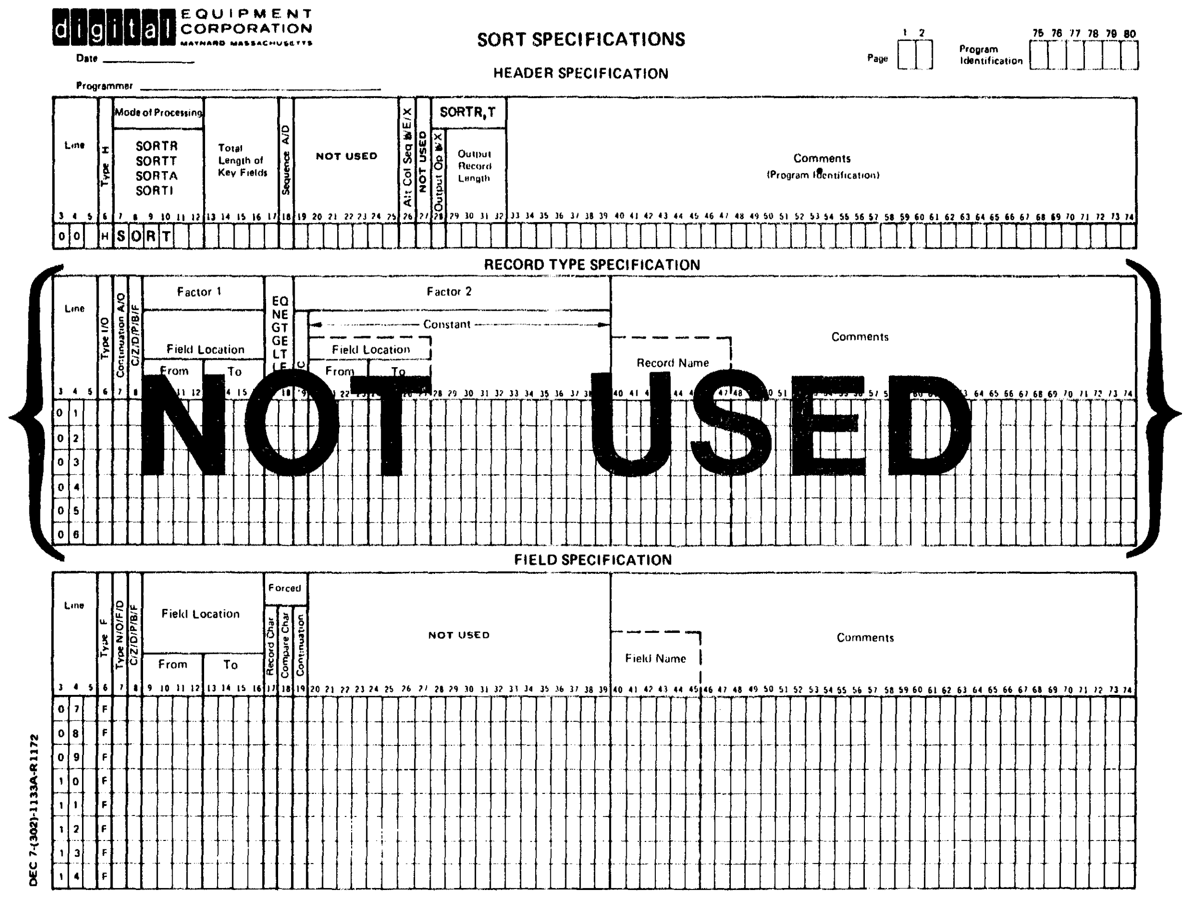
```
1,1,H,,10,A,,X,132     !header for record sort ascending -
order, Key field size 10.
```

2.5.2 Specification File Record Formats

Two record types and two record formats exist for specification files. The two record types are header records and field specification records. Each record type may contain either fixed position fields to support SORT-11 compatible files, or free fields for VAX-11 SORT.

2.5.2.1 For SORT-11 Type Files (fixed position fields) — A DIGITAL SORT specification form is available for use when setting up fixed position fields (see Figure 2-6).

Figure 2-6: SORT Specification Form



digital EQUIPMENT CORPORATION MAYNARD MASSACHUSETTS

Date ____________

Programmer ______________________________

SORT SPECIFICATIONS

Page 1 2 [ ][ ]   Program Identification 75 76 77 78 79 80 [ ][ ][ ][ ][ ][ ]

HEADER SPECIFICATION

Line (3 4 5)	Type H (6)	Mode of Processing: SORTR SORTT SORTA SORTI (7–12)	Total Length of Key Fields (13–17)	Sequence A/D (18)	NOT USED (19–25)	Alt Col Seq b/E/X (26)	NOT USED (27)	SORTR,T: Output Op b/X (28)	SORTR,T: Output Record Length (29–32)	Comments (Program Identification) (33–74)
0 0	H	S O R T								

RECORD TYPE SPECIFICATION

NOT USED

Line (3 4 5)	Type I/O (6)	Continuation A/O (7)	C/Z/D/P/B/F (8)	Factor 1: Field Location From (9–12)	Factor 1: Field Location To (13–16)	EQ NE GT GE LT LE (17–18)	C (19)	Factor 2: Constant / Field Location From (20–23)	Factor 2: Constant / Field Location To (24–27)	Factor 2: Constant (28–39)	Record Name (40–47)	Comments (48–74)
0 1												
0 2												
0 3												
0 4												
0 5												
0 6												

FIELD SPECIFICATION

Line (3 4 5)	Type F (6)	Type N/O/F/D (7)	C/Z/D/P/B/F (8)	Field Location From (9–12)	Field Location To (13–16)	Forced: Record Char (17)	Forced: Compare Char (18)	Continuation (19)	NOT USED (20–39)	Field Name (40–45)	Comments (46–74)
0 7	F										
0 8	F										
0 9	F										
1 0	F										
1 1	F										
1 2	F										
1 3	F										
1 4	F										

DEC 7-(302)-1133A-R1172

The format of each type of SORT-11 specification record is fixed. The SORT specification form is based on card columns, as shown in Figure 2-6. The following entries are common to both types of specification file lines.

Column	Entry	Notes
1-2	Page number	Required only when different types of records are to be described. A separate page, numbered in ascending sequence, should be used for each record type and its corresponding Field Specifications. Only the first page has a header specification.
3-5	Line number	Specifies line sequence. If column 5 is blank, 0 is assumed. Thus a digit entry in this column can be used to identify later line insertions.
6	Specification Type	H for Header, or F for Field

In the following material, unless otherwise stated, these criteria apply:

- Numeric data is decimal.
- Either leading zeroes or leading blanks are acceptable in right-justified entries.
- All field position definition records begin at column 1.

Table 2-3 summarizes all fixed position SORT specification entries.

Header Record:

The first record in a specification file must be the header. The header tells the SORT program what kind of sorting process to use, key field size, sorting order, and output record size.

Format:

Field Position	Function	Legal Values
1-2	Page number	Any number or blanks
3-5	Line number	Any number or blanks
6	Header record ID	H
7-12	Sorting process	SORT R,I,A,T or blanks
13-17	Total key field size	Any number or blanks
18	Sorting order	A, D, or blank
19-25	(not used)	Blanks (anything-ignored)
26	(not used)	Blank (anything-ignored)
27	(not used)	Blank (anything-ignored)
28	(not used)	Anything-ignored
29-32	Output record length	Any number or blanks
33-132	(not used by SORT, may be used for comments)	Anything-ignored

Notes and Comments on Header Specification Entries:

Columns	Explanations and Legal Entries
7–12	Type of SORT (must be left-justified) Legal values: SORTR or blanks – Record sort SORTT – Tag sort SORTA – Address sort SORTI – Index sort
13–17	Total of all key field sizes Legal values: 1–255 Must be equal to the total size in bytes of the largest record key on the file and right-justified.
18	Normal sort order sequence Legal values: A or blank – ascending D – descending This field may be qualified by N or O entered in column 7 of the field specification.
29–32	Output Record Length (for SORTR and SORTT only) Legal values: A decimal number (right-justified) equal to the number of bytes in the largest output record. To determine this number, add the sizes of the key fields in the field specifications for the largest record in the file. If neither SORTT nor SORTR are to be run at this time, an entry in this field is not needed.

Field Specification Records:

The field specification records follow the header record and specify key fields (up to ten).

NOTE:

Data fields are not supported since each entire record in a file is written to the output file for SORTR and SORTT. For SORTA or SORTI, output files contain only pointers and possibly some restricted-format key data.

Format:

Field Position	Function	Legal Values
1–5	Page/Line number	See Header Specification
6	Field record ID	F
7	Key field order	N or O
8	Key field type	B,C,D,I,J,K,P,Z
9–12	First byte of field	Any number or blanks
13–16	Last byte of field	Any number or blanks
17–19	(not used)	Anything-ignored
20–80	(not used – available for comments)	Anything-ignored

Notes and Comments on Field Specification Entries:

Columns	Explanations and Legal Entries
7	Key Field Order – specifies keys and their sort sequence (this entry can satisfy the column 18 entry for the Header Specification). Legal values: N – normal sort sequence O – opposite sort sequence
8	Key field data type codes: B – Binary (two's complement binary) C – Character (8-bit ASCII coded alphanumeric characters). This is the *default* data type. D – Decimal data with sign trailing and overpunched. I – Same as D, but with the sign leading and separate, so that the first byte of the field is a + or –. J – Same as I, but with the sign trailing and separate. K – Same as D, but with the sign leading and overpunched P – Packed-decimal format. Z – Zoned ASCII format.
9–12	Field location (location of the first byte of a multi-byte key field). Legal values: A decimal number (right-justified) specifying the first byte of a key field. Blanks can be used to specify a one-byte key field.
13–16	Field location (location of the last, or only, byte of a key field). Legal values: A decimal number (right-justified) specifying the last byte, or the only byte, in a key field.

Table 2-3: Fixed Position SORT Specification Summary

Header Specifications		
Column	**Entry**	**Explanation**
6	H	Header specification
7–12	SORTR SORTT SORTA SORTI	Record sort Tag sort Address sort Index sort
13–17	1–255	Decimal number specifying the total length of all key fields listed in the Field Specifications (must be the maximum for SORTR).
18	A or blank, D	Sort processing sequence: ascending or descending.
29–32	Decimal number (SORTR or SORTT only) 1–16,383.	This entry specifies the number of bytes for the largest record.

(continued on next page)

Table 2-3: Fixed Position SORT Specification Summary (continued)

Field Specifications		
Column	**Entry**	**Explanation**
6	F	Field specification
7	N	Normal - Key field sequenced as indicated in column 18 of Header Specification.
	O	Opposite - Key field sequenced opposite to column 18 of Header Specification.
8	C	Character type data (8-bit ASCII alphanumeric data in key field).
	Z	Zoned ASCII.
	D	Digit - use digit value or convert to binary for FORTRAN IV numbers.
	I	Same as D, but with sign leading and separate (that is, the first byte of the field is a + or -).
	J	Same as D, but with sign trailing and separate (that is, the last byte of the field is + or -).
	K	Same as D, but with sign leading overpunched (that is, the sign is superimposed on the first byte of the field).
	P	Packed-decimal data type.
	B	Binary data type - the key field is in two's complement binary notation.
9-12	Decimal number 1-16,383	Location of the first byte of the key field.
13-16	Decimal number 1-16,383	Location of the last (or only) byte in the key field.
17-19	(not used)	All values are ignored.
20-80	Anything	Comments.

Sample Fixed Position Specification File:

The following sample shows a header record and field record. Together they specify an index sort process on a character key of four bytes starting in position 10 of the record, and the output file is to be sorted in descending order.

```
column numbers | 1234567   11       33        46
Header Record  |      HSORTI         HEADER INDEX SORT ALL DEFAULT
Field Record   |      FOC00100013             FIELD SPEC ALPHA KEY
```

NOTE:

This sample specification file performs the same sorting process as the sample shown for free field position format.

2.5.2.2 For VAX-11 SORT (free fields, that is fields separated by commas): — Free fields are formatted in the same sequence as the fixed position fields described previously; however, instead of identifying fields with column numbers, commas are used. If you wish to enter blanks or use the default value, you must follow the entry with a comma.

Header records and field specification records are used in the same manner here as they are for the fixed position field records described previously, and the same explanations and legal entries also apply (see Table 2-3).

Header Records:

Field Position	Function	Legal Values
1	Page number	Any number, or blank, or comma,
2	Line number	Any number, or blank, or comma,
3	Header record ID	H,
4	Sorting process	SORT R,I,A,T, or blank, or comma
5	Total key field size	Any number, or blank, or comma,
6	Sorting order	A, D, or blank, or comma,
7	Collating sequence	Anything, or comma,
8	Output key	Anything, or comma,
9	Record length	Any number, or blank, or comma,
10	Comment	! anything, or blank

Field Specification Records:

Field Position	Function	Legal Values
1	Page number	See Header
2	Line number	See Header
3	Field record ID	F,
4	Key sorting order	N, O, or blank, or comma,
5	Key data type	B,C,D,I,J,K,P,Z, or blank, or comma,
6	start position of key	Any number, or blank, or comma,
7	ending position of key	Any number, or blank, or comma,
8	------	Anything-ignored, or comma,
9	comment	! Anything-ignored

Sample Free Field Position Specification File:

The following sample shows a header record and field record. Together they specify an index sort process on a character key of four bytes starting in position 10 of the record, and the output file is to be sorted in descending order.

Header Record	,,H,SORTI,,,,,!HEADER INDEX SORT ALL DEFAULT EXCEPT TYPE
Field Record	,,F,0,C,10,13,,!FIELD SPEC ALPHA KEY 4 BYTES POS 10 OPPOSITE ORDER

NOTE:

This sample specification file performs the same sorting process as the sample shown for fixed position format.

2.6 Setting Up the Keys

When entering the SORT command qualifier /KEY, you must specify /KEY=(subqualifiers and values) for each key field by which the records are to be sorted. The total size of all key fields must be less than or equal to 255 bytes, and the maximum number of key fields allowed is ten.

Before entering the key subqualifiers and values into the SORT command string, perform the following steps (1 through 6). See Section 2.4.1 for /KEY= specification information. Figure 2–7 provides a flowchart for quick reference when setting up keys.

Step 1

Each sort key is assigned a precedence number. You may choose to use either the default system or the key numbering system. First decide what your key fields will be and in what order you want them sorted. Make notes of their sequence for use when assigning the precedence number to each. Note, if you intend to enter the sort keys into the command string in the order of precedence you have chosen for your sorting operation, then the NUMBER= subqualifier is not necessary. Instead, the default feature will automatically assign the first key entered as key number 1 and each subsequent key the next higher number.

Example:

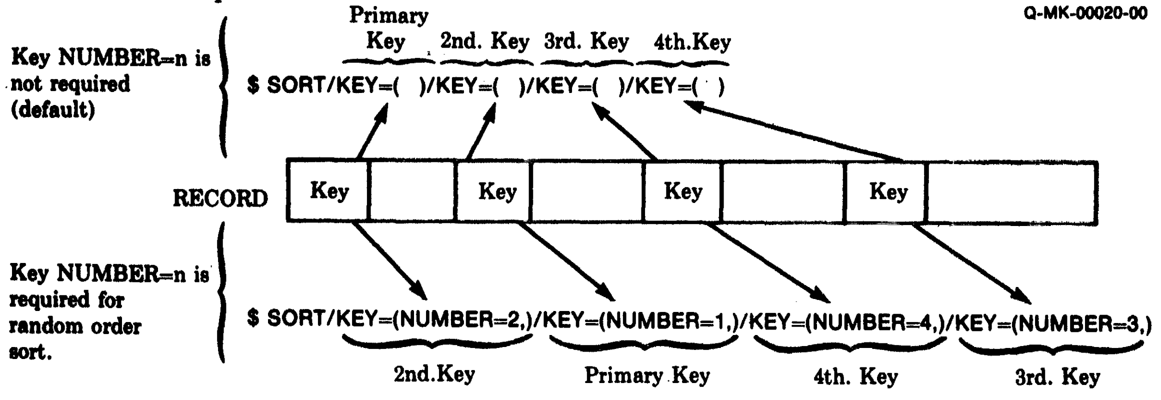
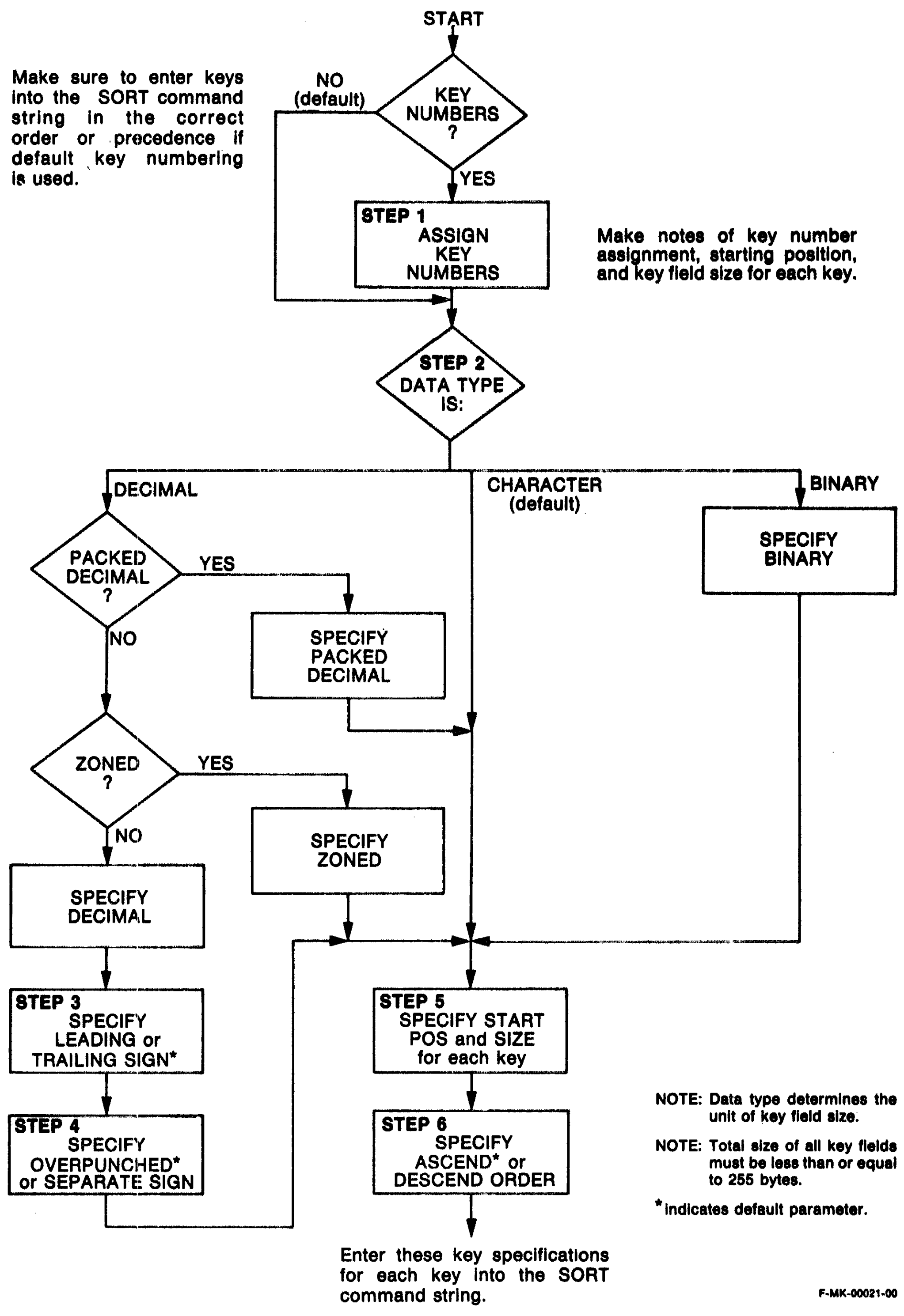

Q-MK-00020-00
Key NUMBER=n is not required (default)
Primary Key
2nd. Key
3rd. Key
4th.Key
$ SORT/KEY=( )/KEY=( )/KEY=( )/KEY=( )
RECORD
Key
Key
Key
Key
Key NUMBER=n is required for random order sort.
$ SORT/KEY=(NUMBER=2,)/KEY=(NUMBER=1,)/KEY=(NUMBER=4,)/KEY=(NUMBER=3,)
2nd.Key
Primary Key
4th. Key
3rd. Key

Figure 2-7: Setting Up the Keys

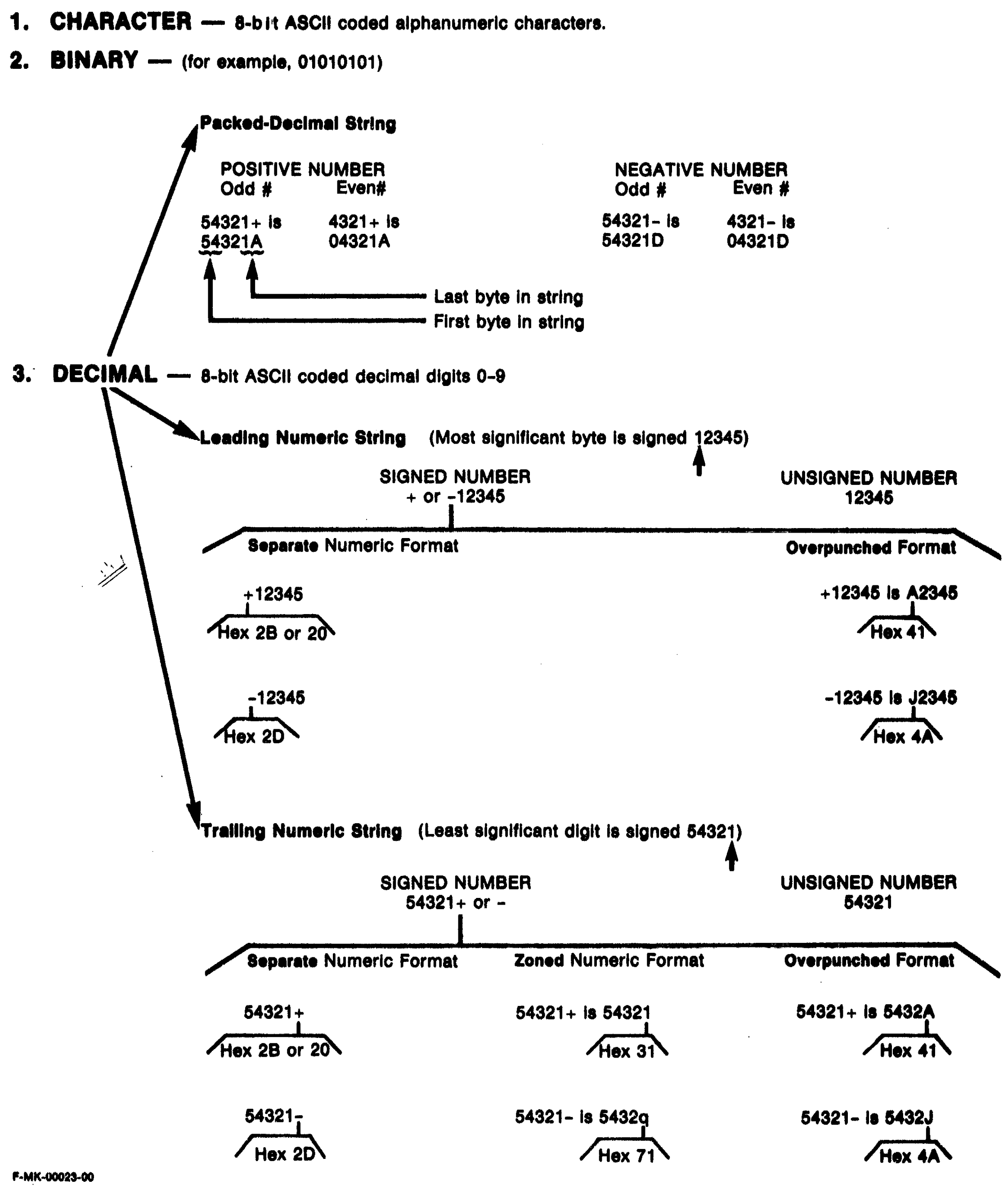

START
Make sure to enter keys into the SORT command string in the correct order or precedence if default key numbering is used.
NO (default)
KEY NUMBERS ?
YES
STEP 1 ASSIGN KEY NUMBERS
Make notes of key number assignment, starting position, and key field size for each key.
STEP 2 DATA TYPE IS:
DECIMAL
CHARACTER (default)
BINARY
SPECIFY BINARY
PACKED DECIMAL ?
YES
NO
SPECIFY PACKED DECIMAL
ZONED ?
YES
NO
SPECIFY ZONED
SPECIFY DECIMAL
STEP 3 SPECIFY LEADING or TRAILING SIGN*
STEP 4 SPECIFY OVERPUNCHED* or SEPARATE SIGN
STEP 5 SPECIFY START POS and SIZE for each key
STEP 6 SPECIFY ASCEND* or DESCEND ORDER
Enter these key specifications for each key into the SORT command string.
NOTE: Data type determines the unit of key field size.
NOTE: Total size of all key fields must be less than or equal to 255 bytes.
*Indicates default parameter.
F-MK-00021-00

Step 2

Before you can enter key size in Step 5, you must first determine what the data type of each key field is. Figure 2–8 shows a summary of the data types supported. See Appendix C for specific descriptions of the date types.

Example:
For decimal data type, specify DECIMAL in the command string. For example: $ SORT/KEY=(POS=1,SIZE=10,DECIMAL).

Figure 2–8: Recognizing Data Types and Signed Numbers



Step 3

If data type is decimal, specify the position of the sign. If data type is not decimal, proceed to Step 5.

Examples:

1. For +12345 or -12345, specify the optional keyword LEADING_SIGN in the command string. For example:

   ```
   $ SORT/KEY=(POS=1,SIZE=10,DECIMAL, LEAD)
   ```

2. For trailing sign numbers (that is, 12345+ or 12345-), the optional keyword TRAILING_SIGN is not required (TRAILING_SIGN is default).

Step 4

For decimal data types, specify if the sign is overpunched (superimposed) or separate from the decimal value.

Examples:

1. For separate sign (that is, +12345 or -12345), specify the optional keyword SEPARATE_SIGN in the command string. For example:

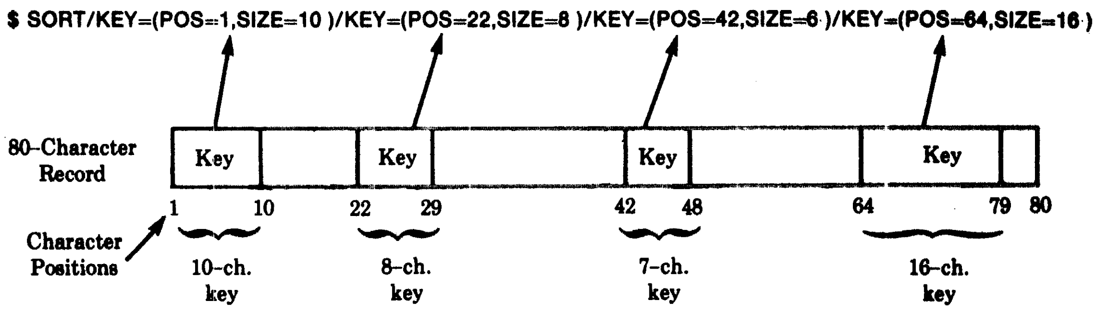
   ```
   $ SORT/KEY=(POS=1,SIZE=10,DECIMAL,LEAD,SEPA)
   ```

2. For overpunched sign (that is, 5432A or 5432J), the optional keyword OVERPUNCHED_SIGN is not required (OVERPUNCHED_SIGN is default).

Step 5

You must specify the starting position (first character in the key field) and the size for each key. The first character of the record is 1.

Example: (in this example, key NUMBER= is default, data type is CHARACTER).



```
$ SORT/KEY=(POS=1,SIZE=10 )/KEY=(POS=22,SIZE=8 )/KEY=(POS=42,SIZE=6 )/KEY=(POS=64,SIZE=16 )
```

Q-MK-00022-00

NOTE:

Key field size can represent either the number of bytes, or digits depending on the data type. The chart below describes which unit of key field size to use:

Data Type	Size Indicates	
Character	Number of characters (bytes) must be less than or equal to 255.	
Binary	Number of bytes must be either 1, 2, or 4.	
	1	byte for decimal values in the range of −128 to 127.
	2	bytes for decimal values in the range of −32,768 to 32,767.
	4	bytes for decimal values in the range of −2,147,483,648 to 2,147,483,647.
Decimal	Number of digits in the string must be less than or equal to 31.	

The number of bytes in the key field for character, binary, zoned numeric, and overpunched is identical to the number of characters or digits. The size of leading separate or trailing separate fields is equal to the number of digits plus one. The size of packed-decimal fields is equal to (number of digits/2)+1.

See Appendix C for additional information.

Step 6

Specify for each key, whether that key is to be sorted into ascending or descending order. Ascending order is default.

Example:

To sort the first key in ascending order and the second key in descending order, enter the key parameters into the command string as follows:

```
$ SORT/KEY=(POS=1,SIZE=10)/KEY=(POS=22,SIZE=8,DESCENDING)
```

Now that you have performed Steps 1 through 6 to assemble specifications for each key, you are ready to enter these key specifications into the SORT command string.

2.7 Setting Up the Work Files

SORT automatically assigns two work files to your SYS$DISK device if you choose to use the default. The size of these two work files (SORTWORK0 and SORTWORK1) is determined by SORT from the size of your input file. (Note; if no assignment is done, work files are created on SYS$DISK).

To assign your work files to a device other than the device your directory is on, type:

```
$ ASSIGN (device): SORTWORK0
$ ASSIGN (device): SORTWORK1
              .
              .
              .
$ ASSIGN (device): SORTWORK9
```

Example: `$ ASSIGN DB3: SORTWORK1`

Figure 2-9 illustrates how logical names are assigned to physical devices.

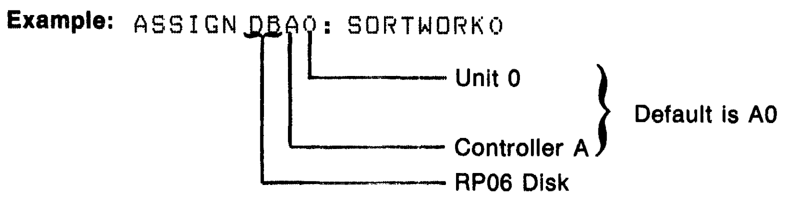
Figure 2-9: Specifying Work Files.

WORK FILE #	Logical Name
1	SORTWORK0
2	SORTWORK1
3	SORTWORK2
4	SORTWORK3
5	SORTWORK4
6	SORTWORK5
7	SORTWORK6
8	SORTWORK7
9	SORTWORK8
10	SORTWORK9

A specific physical device code is assigned to a specific logical name using the ASSIGN, or DEFINE commands.

Physical Device Codes

DB: RP04, RP05, RP06 Disk
DM: RK06 Disk



Chapter 3
Calling SORT from User Programs

You can use SORT as a set of callable subroutines from your programming language. There are two functional interfaces to choose from; the file I/O interface and the record I/O interface. Both I/O interfaces share the same set of six subroutines, and the same calls are used from all languages.

This SORT subroutine package consists of six external function calls. Each call causes a phase of the SORT program to be performed, and returns a status (32-bit) value indicating either success or the failure type of the phase. Calls and associated parameters conform to the VAX-11 standard calling interface. The calls are:

Subroutine Name	**Function**
1. SOR$INIT_SORT	Initialize scratch files, work area, sorting parameters
2. SOR$PASS_FILES	Pass a file name to SORT
3. SOR$RELEASE_REC	Pass a record to SORT
4. SOR$SORT_MERGE	Initiate sorting and intermediate merging of records
5. SOR$RETURN_REC	Initiate final merge pass and receive output record from SORT
6. SOR$END_SORT	Allow clean up of files and work area to complete the sort operation

3.1 File I/O Interface

The file I/O interface enables you to specify an input file and an output file to SORT. SORT then reads the data from the input file and sorts it into the output file.

For the file I/O interface, use the following four calls in the order listed:

Call	Function
1. SOR$PASS_FILES	Pass file specifications
2. SOR$INIT_SORT	Initialize work areas
3. SOR$SORT_MERGE	Sort records
4. SOR$END_SORT	Clean up work areas

3.2 Record I/O Interface

The record I/O interface enables you to pass individual data records to SORT. SORT orders them, then returns each record in correct order, individually.

For the record I/O interface, use the following five calls in the order listed:

Call	Function
1. SOR$INIT_SORT	Intialize work areas
2. SOR$RELEASE_REC	Pass an input record
3. SOR$SORT_MERGE	Sort records
4. SOR$RETURN_REC	Receive a sorted output record
5. SOR$END_SORT	Clean up work areas

NOTE:

Calls 2 and 4 are each repeated for as many times as there are records to be sorted.

3.3 Programming Considerations

Any program can use either SORT subroutine package interface, providing the language used produces VAX-11 native mode code and supports the following features.

- 32-bit integers
- Longword addresses
- Call by string descriptors
- Call by reference
- Either CALLS or CALLG (that is, VAX/VMS standard calling sequence)
- External function calls (each SORT subroutine returns a 32-bit status code)

Additional information regarding the VAX/VMS calling standards can be found in Appendix C of the *VAX-11 Common Run-Time Procedure Library Reference Manual.* SORT follows the Modular Procedure Standards and uses the common Run-Time Library routines to allocate memory and event flags. However, SORT is not re-entrant.

The SORT subroutines are a part of the standard VMS library, therefore to use the package a user only has to code the appropriate calls into his program, compile or assemble and link. During the linking process the appropriate SORT routines will automatically be linked with the user's program.

Figure 3-1 summarizes the callable subroutine set.

Figure 3-1: Subroutine Set Summary

NOTE:

Use the subroutine calls in the order shown.

Call	Function
1. SOR$PASS_FILES	Open the input file and create the output file.
2. SOR$INIT_SORT	Set up the key comparison buffer and validate key information.
	Get memory for sorting initial phase, input and output buffers, and set up to read input.
	Create work files and initialize the sort.
3. SOR$RELEASE_REC	Get record from user and build key.
4. SOR$SORT_MERGE	Insert record by key into sort tree.
	If sort tree is full, continue; if not, get another record.
	Output records to work files as a number of strings of sorted records.
	Output and input until no more records and all records are output.
	Read in strings from work file and merge them until there are ten or less left in work files.
5. SOR$RETURN_REC	Set up to output records to user.
	Do final merge pass to output records (not work files) to user.
6. SOR$END_SORT	Return memory, close output and input files, and delete work files.

3.3.1 Key Comparisons

Both Interfaces

When using either interface, you have the choice of allowing SORT to do key comparison to determine the correct order of any two records, or of writing a routine of your own that SORT can call to do the key comparisons.

The advantage of writing your own routine is that you may know a great deal more about the nature of the key data and therefore write a routine specifically tailored to that particular data. Because SORT does not know anything about the key data in advance of receiving it, SORT's key comparison routine must be general in order to handle all types of data. A routine tailored to a particular data type or set can therefore be much more efficient, both in space and performance.

If you want to use the SORT key comparison routine, you must provide the key definitions in the SOR$INIT_SORT call. Or, if you want to use your key comparison routine with SORT, you must pass the address of your routine's entry point (with parameters) to SORT in the SOR$INIT_SORT call. See Section 3.4.3, Definitions, for details. Users can write a program that uses any key data type.

For debugging purposes, it should be noted that the key comparison routine may not necessarily be called each time a call is made to SORT. This situation can occur with the following calls:

```
SOR$RELEASE_REC
SOR$SORT_MERGE
SOR$RETURN_REC
```

Record I/O Interface Only

For record I/O interface, you must set up the key data area before passing the record to SORT if SORT is to do the comparisons.

The key field must be set up with each key physically next to the one before it, in order of precedence from left to right.

For example:
If the key definitions looked like this:

Key 1 - Character, Ascending, Pos 1, Size 4.
Key 2 - Binary, Descending, Pos 15, Size 2.
Key 3 - Packed, Ascending, Pos 30, Size 4.

Then, the key area in the user's program should look like this:

$$\underbrace{\text{ABCD}}_{\text{key1}}\underbrace{\text{45}}_{\text{key2}}\underbrace{\text{634D}}_{\text{key3}}$$

SORT will handle ascending/descending considerations as long as SORT is doing the key comparisons. The user does not have to modify the key data in any way.

In addition the entire key area must physically preceed and be adjacent to the record. For example:

```
(KEYAREA)  ABCD45634D
(RECORD)   ABCDEFGHIJKLMN45........
```

When passing the record to SORT, the record descriptor must describe the entire string including the key area. Therefore, the length of the string is (total key length plus record length) and the address is the address of the first byte of the key area.

For record I/O the only valid key types are 1, 2, and 4; character, binary, and packed decimal. However, the instruction set provides a set of decimal instructions that allow conversion from all of the other decimal formats to packed. Therefore, when you build the key area from your record data you can convert the other decimal types to packed, and by doing so, sort on any of the nine valid key data types that the file I/O interface accepts.

When SORT returns the record it will strip off the key data. The length returned will be the length of the record alone and the first byte of the output buffer will contain the first byte of the record, not the key.

If you are passing the address of your own key comparison routine to SORT and you do not wish to set up the key field preceding the record, you may specify a 0 value as the total key size in the call to SOR$INIT_SORT. You then pass just the record to SORT. When SORT calls your key comparison routine the addresses of the two keys will be the addresses of the first byte of each record.

3.4 Subroutines (Parameters, Definitions, and Valid Returns)

Each call requires several user supplied parameters. Parameters, parameter definitions, and valid returns are provided in the following paragraphs for each call. Both symbolic and hexadecimal values are provided for the returned messages as an aid when debugging.

All user program value parameters must be passed to SORT using "call by reference" (that is, the address of the value in the user's data area is passed to the SORT routine, not the value itself).

All file specifications and records are passed to SORT using string descriptors. A descriptor is a 2–longword structure of format.

For example:

flags word	length of string	= 2 words = 1 longword
address of string		= 1 longword

The address of the descriptor is passed to SORT.

To omit an optional parameter, either leave it null or pass a 0 address in the argument list; do not pass the address of a data item with a 0 value. In general the meaning of a parameter and its legal values are identical to the equivalent parameter in the command line to the utility.

3.4.1 SOR$INIT_SORT

Function: Initialize scratch files, work area, and sorting parameters.

Parameters: Each of the following parameters is numbered to match its definition which follows.

1. Key buffer address ❶
2. Longest record length (LRL) ❷
3. File size ❸
4. Number of work files
5. Sort type ❹
6. Total key size ❺
7. Comparison routine address ❺

Notes: ❶ Mandatory for the file I/O interface and for the record I/O interface only if SORT is to do the key compares.

❷ Mandatory for the record I/O interface.

❸ Needed for the record I/O interface and input from unit record or magnetic tape devices in order for SORT to be efficient, but is not required.

❹ Valid only for the file I/O interface.

❺ Mandatory only if parameter 1 is not present and the user program is to do the key compares.

Definitions:

1. Set up the key buffer in your user data area. The key buffer describes the definition of the keys to be sorted on, and has the following format:

key type	one word = 1-9 for file I/O interface, and 1, 2, or 4 for record I/O interface
key order	one word = 0 or 1
start position	one word = 1 to (max record size)
length	one word = 1-255 (depends on key type)

Up to ten of these blocks can be specified in the order of key precedence.

The key buffer must be preceded by a word specifying the number of keys specified in the following blocks. For example:

	2	= number of keys
Key 1	1	= key type (character)
	0	= key order (ascending)
	10	= start position in record
	40	= length of key
Key 2	4	= key type (packed-decimal)
	0	= key order (ascending)
	60	= start position in record
	10	= length of key in number of digits

Key Types:	1 = Character	6 = Decimal leading overpunched
	2 = Binary	7 = Decimal leading separate
	3 = Zoned	8 = Decimal trailing overpunched
	4 = Packed-decimal	9 = Decimal trailing separate
	5 = not used	
Key Order:	0 = Ascending	1 = Descending

When passing the address of the key buffer, pass the address of the word with the number of keys.

2. Longest record length (LRL) is a decimal number (one word in length) indicating the longest record length in bytes not including key size.

3. File size (one longword in length) is the value for the input file size in blocks.

4. Number of work files (one byte in length) is the value of 2 - 10 or 0.

5. Sort type (one byte in length) is the value of 1 - 4 as listed:

1 = Record sort	3 = Index sort
2 = Tag sort	4 = Address sort

6. Total key size (one byte in length) is the value of 1 - 255.

7. Address of the user generated key comparison routine. You have the option of performing your own key comparisons, and not supplying a key definition to SORT. SORT calls your routine at the specified address, and with the following parameters:

   1) address of key 1
   2) address of key 2

   SORT expects the following return value:

   -1 if key 1 is less than key 2.
   0 if key 1 is equal to key 2.
   1 if key 1 is greater than key 2.

NOTE

Keys must not be modified in any way.

Valid Returns:

Symbolic	Hex Value	Meaning
SOR$_SORT_ON	1C802C	A sort is already in progress or this call is in the wrong sequence.
SOR$_MISS_KEY	1C8004	No key definition specified.
SOR$_BAD_TYPE	1C806C	An invalid sort process was specified.
SOR$_BAD_LRL	1C8084	An invalid LRL was specified.
SOR$_LRL_MISS	1C8074	No LRL was specified and is required.
SOR$_BAD_FILE	1C808C	An invalid file size.
SOR$_WORK_DEV	1C800C	Work file device not random access device or not local node.
SOR$_VM_FAIL	1C801C	SORT failed to get needed virtual memory.
SOR$_WS_FAIL	1C8024	SORT failed to get needed working set size.
SOR$_NUM_KEY	1C803C	Invalid number of keys specified (must be 1-10).
SOR$_KEY_LEN	1C80AC	Invalid key length specified.
SS$_NORMAL	1	Success
All RMS error codes		See Chapter 4.

3.4.2 SOR$PASS__FILES

Function: Pass a file specification to SORT.

Parameters: Each of the following parameters is numbered to match its definition which follows.

1. Input file descriptor ❶
2. Output file descriptor
3. Output file organizaton
4. Output file record format
5. Output file bucket size
6. Output file block size ❷
7. Output file maximum record size
8. Output file allocation
9. Output file file options

Notes: All output file parameters are specified as for VAX-11 RMS.

❶ These parameters are mandatory.

❷ These parameters are optional.

Definitions:

1. Input file descriptor is the string descriptor for the string in ASCII of the input file specification.
2. Output file descriptor is the string descriptor for the string in ASCII of the output file specification.
3. Value of output file organization (one byte in length):

   FAB$C__SEQ
   FAB$C__REL
   FAB$C__IDX

4. Value of record format for output (one byte in length):

   FAB$C__FIX
   FAB$C__VAR
   FAB$C__VFC

5. Value for bucket size (one byte in length) is 1 - 32.
6. Value for block size (one word in length) is 18 - 32,767.
7. Value for maximum record size (one word in length) is 1 - 16,383.
8. Value for output file allocation (one longword in length) is 1 to the maximum RMS file size.
9. Value for output file file options (one longword in length) is: see the $FAB FOP parameters in the *VAX-11 Record Management Services Reference Manual*.

Valid Returns:

Symbolic	Hex Value	Meaning
SS$_NORMAL	1	Success
SOR$_SORT_ON	1C802C	A sort is already in progress or this call is in the wrong sequence.
SOR$_VAR_FIX	1C8064	Cannot change variable records to fixed records.
SOR$_INCONSIS	1C805C	Inconsistent data for file.
SOR$_OPENIN	1C109C	Cannot open input file.
SOR$_OPENOUT	1C10A4	Cannot open output file.
All RMS error codes		See Chapter 4.

3.4.3 SOR$RELEASE_REC

Function: Pass a record to SORT.

Parameters: Each of the following parameters is numbered to match its definition which follows.

1. Record descriptor

Notes: Parameter 1 is mandatory.

Definitions:

1. Record descriptor is the address of the descriptor for the key and record being input to SORT. The length of the record must include the total key length plus the total record length. Also, the key field must physically immediately precede and adjoin the record, and the descriptor must point to the beginning of the key.

Valid Returns:

Symbolic	Hex Value	Meaning
SS$_NORMAL	1	Success
SOR$_SORT_ON	1C802C	A sort is already in progress or this call is in the wrong sequence.
SOR$_BAD_LRL	1C8084	Record length is longer than LRL specified.
SOR$_BAD_ADR	1C8094	Invalid descriptor address passed.
SOR$_KEY_LEN	1C80AC	Invalid key length specified.
SOR$_EXTEND	1C80A4	Failed to extend work file.
SOR$_MAP	1C809C	Internal sort map error.
SOR$_NO_WRK	1C8014	Cannot do sort in memory, need work files.

3.4.4 SOR$SORT_MERGE

Function: Initiate sorting and intermediate merging of records.

Parameters: None.

Valid Returns:

Symbolic	Hex Value	Meaning
SS$_NORMAL	1	Success
SOR$_SORT_ON	1C802C	A sort is already in progress or this call is in the wrong sequence.
SOR$_EXTEND	1C80A4	Failed to extend work file.
SOR$_NO_WRK	1C8014	Cannot do sort in memory, need work files.
SOR$_MAP	1C809C	Internal sort map error.
SOR$_READERR	1C10B4	Cannot read a specified input file record.
SOR$_WRITEERR	1C10D4	Cannot write a specified output file record.
SOR$_BADFIELD	1C101C	Bad data in key field.

3.4.5 SOR$RETURN_REC

Function: Initiate final merge pass and receive output record from SORT.

Parameters: Each of the following parameters is numbered to match its definition which follows.

1. Record descriptor ●
2. Record size

Notes: ● This parameter is mandatory.

Definitions:

1. Record descriptor for the output area that SORT is to place the output record into.
2. The location (one word in length) in which SORT is to place the actual size of the record returned.

Valid Returns:

Symbolic	Hex Value	Meaning
SOR$_MAP	1C809C	Internal sort map error.
SOR$_EXTEND	1C80A4	Failed to extend work file.
SS$_NORMAL	1	Success, a record has been returned.
SS$_ENDOFFILE	870	Success, no more records to return.

3.4.6 SOR$END_SORT

Function: Allow clean up of files and work area to complete the sort operation.

Parameters: None

Definitions: None

Valid Returns:

Symbolic	Hex Value	Meaning
SS$_NORMAL	1	Success
SOR$_CLEAN_UP	1C80B4	Failed to delete work files and reinitialize work areas and data areas.

3.5 Sample MACRO Program

```
    .TITLE TESTSUB
    .IDENT x01.01
;
; THIS IS A SAMPLE MACRO
; PROGRAM WHICH CALLS
; THE SORT SUBROUTINE
; PACKAGE. THERE IS AN
; EXAMPLE USING EACH
; INTERFACE.
;
;
; DATA AREA
;
FILENAMEIN:     .ASCII  /R010SQ.DAT/     ;INPUT FILENAME
FILENAMEOUT:    .ASCII  /TEST.TMP/       ;OUTPUT FILENAME
                .BLKB   2
IN_FAB:         .BLKB   80               ;RMS DATA BLOCKS
IN_RAB:         .BLKB   68
OUT_FAB:        .BLKB   80
OUT_RAB:        .BLKB   68
FILEIN:         .LONG   10               ;INPUT FILE NAME DESCRIPTOR
                .ADDRESS FILENAMEIN
FILEOUT:        .LONG   8                ;OUTPUT FILE NAME DESCRIPTOR
                .ADDRESS FILENAMEOUT
KEYBUF:         .WORD   1                ;KEY DEFINITION BUFFER
KEYTYPE:        .WORD   1
KEYORD:         .WORD   0
KEYPOS:         .WORD   1
KEYSIZ:         .WORD   10
INLRL:          .WORD   80               ;INPUT RECORD LONGEST LENGTH
WRKFILE:        .LONG   500              ;WORK FILE SIZE
NUMWRK:         .BYTE   4                ;NUMBER OF WORK FILES
TAGSRT:         .BYTE   2                ;TAG SORT
                .BLKB   2
KEYAREA:        .BLKB   10               ;KEY BUFFER
RECORDBUF:      .BLKB   80               ;RECORD BUFFER
RECDESC:        .LONG   90               ;RECORD DESCRIPTOR
    .ADDRESS KEYAREA

;
;
;
; FIRST THE FILE I/O INTERFACE. DO A TAG SORT ON THE FILE 'R010SQ.DAT'
; INTO THE FILE 'TEST.TMP' USING 4 WORK FILES. KEY IS CHARACTER, 10 BYTES
; LONG, STARTING POSITION 1.
;
;
    .EXTRN      SOR$PASS_FILES,SOR$INIT_SORT,SOR$SORT_MERGE,SOR$END_SORT,-
                SOR$RELEASE_REC,SOR$RETURN_REC
;
FILEIO::
    .ENTRY      ^M<R2,R3,R4,R5,R6,R7>   ;SAVE REGISTERS
                                        ;DEFAULT ALL OUTPUT OPTIONS
    PUSHAB      FILEOUT                 ;PUSH FILENAME DESCRIPTOR ADDRESS
    PUSHAB      FILEIN
    CALLS       #2,SOR$PASS_FILES       ;PASS FILENAMES TO SORT
    BLBC        R0,2$                   ;TEST FOR ERROR
    PUSHAB      TAGSRT                  ;PUSH SORT TYPE
    PUSHAB      NUMWRK                  ;PUSH NUMBER OF WORK FILES
    CLRQ       -(SP)                    ;DEFAULT LRL AND WORK FILE SIZE
    PUSHAB     KEYBUF                   ;PUSH KEY BUFFER ADDRESS
    CALLS      #5,SOR$INIT_SORT         ;INITIALIZE THE SORT
    BLBC       R0,2$                    ;TEST FOR ERROR
                                        ;LET SORT DO COMPARES
    CALLS      #0,SOR$SORT_MERGE        ;START SORTING
    BLBC       R0,2$                    ;TEST FOR ERROR
    CALLS      #0,SOR$END_SORT          ;DO CLEAN UP
    BLBC       R0,2$                    ;TEST FOR ERROR
;
;
; NOW TRY THE RECORD I/O INTERFACE. RECORDS ARE 80 BYTES LONG, KEY IS
; CHARACTER, 10 BYTES LONG, STARTING IN POSITION 1. WORK FILE SIZE IS
; 500 BLOCKS.
;
;
    CALLS      #0,OPEN_INPUT            ;OPEN USER INPUT AND OUTPUT FILE
    BLBC       R0,2$                    ;TEST FOR ERROR
                                        ;DEFAULT SORT TYPE AND WORK FILES
    PUSHAB     WRKFILE                  ;PUSH WORK FILE SIZE
    PUSHAB     INLRL                    ;PUSH LRL
    PUSHAB     KEYBUF                   ;PUSH KEY BUFFER ADDRESS
    CALLS      #3,SOR$INIT_SORT         ;INITIALIZE THE SORT
    BLBC       R0,2$                    ;TEST FOR ERROR
    MOVZWL     #1000,R6                 ;SET UP LOOP INDEX
1$: CALLS      #0,GET_RECORD            ;GET RECORD FROM MY FILE
    BLBC       R0,2$                    ;TEST FOR ERROR
    MOVC3      #10,RECORDBUF,KEYAREA    ;SET UP KEY IN KEY BUFFER
                                        ;SORT DOES COMPARES
    PUSHAB     RECDESC                  ;PUSH RECORD DESCRIPTOR
    CALLS      #1,SOR$RELEASE_REC       ;GIVE RECORD TO SORT
    BLBC       R0,2$                    ;TEST FOR ERROR
    SOBGTR     R6,1$
                                        ;SORT DOES COMPARES
    CALLS      #0,SOR$SORT_MERGE        ;NO MORE RECORDS TO GIVE
2$: BLBC       R0,6$
3$:
    PUSHAB     INLRL                    ;PUSH RECORD SIZE LOCATION
    PUSHAB     RECDESC                  ;PUSH RECORD DESCRIPTOR
    CALLS      #2,SOR$RETURN_REC        ;GET RECORD BACK
    CMPL       R0,SS$_ENDOFFILE         ;GOTTEN ALL RECORDS
    BEQL       4$                       ;YES
    BLBC       R0,6$                    ;ERROR
    CALLS      #0,PUT_RECORD            ;PUT RECORD INTO OUTPUT
    BRB        3$
4$: CALLS      #0,SOR$END_SORT          ;FINISH UP
    BLBC       R0,6$                    ;TEST FOR ERROR
    CALLS      #0,CLOSE_FILE  .         ;CLOSE UP FILES
    MOVL       #1,R0                    ;INDICATE SUCCESS
    RET
6$: CLRL       R0                       ;INDICATE FAILURE
    RET
;
    .END
```

3.6 Sample COBOL-74/VAX Program

```
IDENTIFICATION DIVISION.
PROGRAM-ID. TSTSORT.
*
* THIS IS A SAMPLE COBOL-74/VAX PROGRAM THAT CALLS THE NATIVE
* SORT SUBROUTINE PACKAGE USING THE RECORD I/O INTERFACE. IT
* REQUESTS A RECORD SORT USING A 5 BYTE CHARACTER KEY.
*
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT FILE-IN
        ASSIGN TO "SY".
    SELECT FILE-OUT
        ASSIGN TO "SY".
DATA DIVISION.
*
* ASSIGN FILE DEVICES
* AND NAMES AND DEFINE INPUT AND OUTPUT RECORD AREAS.
*
FILE SECTION.
FD  FILE-IN
    VALUE OF ID IS "SORTIN.DAT"
    LABEL RECORDS ARE STANDARD.
01  IN-REC.
    05  IN-1     PIC X(9).
    05  IN-2     PIC X(5).
    05  IN-3     PIC X(6).
FD  FILE-OUT
    VALUE OF ID IS "SORTOU.DAT"
    LABEL RECORDS ARE STANDARD.
01  OUT-REC             PIC X(20).
*
* SET UP DATA FOR SORT SUBROUTINE PARAMETERS.
*
WORKING-STORAGE SECTION.
77  END-OF-FILE-SW      PIC X     VALUE "0".
    88  END-OF-FILE               VALUE "1".
77  SHOW-STAT                     PIC 9(9).
*
* LONGEST RECORD LENGTH, WORK FILE SIZE
* AND RETURN STATUS VALUES.
*
77  LRL                 PIC 99    VALUE 20 COMP.
77  FILE-SIZ            PIC 9(8)  VALUE 1 COMP.
01  SORT-STATUS         PIC S9(8) COMP VALUE 0.
    88  SS-NORMAL              VALUE 1.
    88  SS-ENDOFFILE           VALUE 2160.
*
* KEY BUFFER INDICATING ONE 5 BYTE CHARACTER KEY STARTING IN
* POSITION 10 OF EACH RECORD, ASCENDING ORDER.
*
01  KEY-BUFFER.
    05  KEY-NUMBER      PIC 9(4)    VALUE 1 COMP.
    05  KEY-TYPE        PIC 9(4) COMP    VALUE 1.
    05  KEY-ORDER       PIC 9(4) COMP    VALUE 0.
    05  KEY-START       PIC 9(4) COMP    VALUE 10.
    05  KEY-LENGTH      PIC 9(4) COMP    VALUE 5.
*
* AREA FOR KEY AND RECORD.
*
```

```
01  WK-REC-ALL.
    05  WK-KEY1          PIC X(5).
    05  WK-REC.
    10  WK-1     PIC X(9).
    10  WK-2     PIC X(5).
    10  WK-3     PIC X(6).
PROCEDURE DIVISION.
MAIN-LOGIC.
*
* OPEN THE INPUT AND OUTPUT FILES. THEN INITIALIZE THE SORT
* SPECIFYING THE KEY DEFINITION, THE LRL AND WORK FILE SIZE.
*
    OPEN INPUT FILE-IN
        OUTPUT FILE-OUT.
    CALL "SOR$INIT_SORT" USING KEY-BUFFER LRL FILE-SIZ
        GIVING SORT-STATUS.
    IF NOT SS-NORMAL
        MOVE SORT-STATUS TO SHOW-STAT
        DISPLAY "FAILURE DURING SOR$INIT, STATUS WAS "SHOW-STAT
         PERFORM ABORT-JOB.

*
* READ RECORDS FROM FILE
* EXTRACT THE KEY AND THEN HAND EACH TO SORT.
*
    PERFORM RELEASE-RECS UNTIL END-OF-FILE.
*
* END OF FILE CALL SORT TO FINISH SORTING RECORDS.
*
    CALL "SOR$SORT_MERGE" GIVING SORT-STATUS.
    IF NOT SS-NORMAL
        MOVE SORT-STATUS TO SHOW-STAT
        DISPLAY "FAILURE DURING SOR$MERGE, STATUS WAS " SHOW-STAT
        PERFORM ABORT-JOB.
    MOVE "0" TO END-OF-FILE-SW.
*
* REQUEST RECORDS BACK FROM SORT UNTIL ALL RECEIVED.
*
    PERFORM RETURN-RECS UNTIL END-OF-FILE.
*
* CALL SORT TO CLEAN UP WORK AREAS.
*
    CALL "SOR$END_SORT" GIVING SORT-STATUS.
    IF NOT SS-NORMAL
        MOVE SORT-STATUS TO SHOW-STAT
        DISPLAY "FAILURE DURING SOR$END, STATUS WAS " SHOW-STAT
        PERFORM ABORT-JOB.
*
* CLOSE FILES.
*
    CLOSE FILE-IN
        FILE-OUT.
    STOP RUN.
*
* READ RECORDS AND BUILD KEY.
*
RELEASE-RECS.
    READ FILE-IN
        AT END
                MOVE "1" TO END-OF-FILE-SW.
    IF NOT END-OF-FILE
        MOVE IN-REC TO WK-REC
        MOVE IN-2 TO WK-KEY1
        CALL "SOR$RELEASE_REC" USING BY DESCRIPTOR WK-REC-ALL
        GIVING SORT-STATUS.
    IF NOT SS-NORMAL
        MOVE SORT-STATUS TO SHOW-STAT
        DISPLAY "FAILURE DURING SOR$RELEASE, STATUS WAS " SHOW-STAT
        PERFORM ABORT-JOB.
*
* RECEIVE RECORDS AND WRITE THEM OUT.
*
RETURN-RECS.
    CALL "SOR$RETURN_REC" USING BY DESCRIPTOR WK-REC
        BY REFERENCE LRL
        GIVING SORT-STATUS.
    IF SS-ENDOFFILE
        MOVE "1" TO END-OF-FILE-SW.
    IF NOT END-OF-FILE
        MOVE SPACES TO OUT-REC
    MOVE WK-REC TO OUT-REC
    WRITE OUT-REC.
ABORT-JOB.
    DISPLAY "ABNORMAL END OF JOB".
    CLOSE FILE-IN
        FILE-OUT.
    STOP RUN.
```

3.7 Sample FORTRAN IV PLUS Program

```
        PROGRAM CALLSORT
C
C
C       THIS IS A SAMPLE FORTRAN IV PLUS PROGRAM THAT CALLS THE
C       NATIVE SORT SUBROUTINE PACKAGE USING THE FILE I/O INTERFACE.
C       THIS PROGRAM REQUESTS AN INDEX SORT OF FILE 'R010SQ.DAT'
C       INTO THE FILE 'TEST.TMP'. THE KEY IS AN 80 BYTE CHARACTER
C       ASCENDING KEY STARTING IN POSITION ONE OF EACH RECORD.
C
C
C       DEFINE EXTERNAL FUNCTIONS AND DATA
C
        CHARACTER*10 INPUTNAME       !INPUT FILE NAME
        CHARACTER*8  OUTPUTNAME      !OUTPUT FILE NAME
        INTEGER*2 KEYBUF(5)          !KEY DEFINITION BUFFER
        INTEGER*2 NUMWRK             !NUMBER OF WORK FILES
        INTEGER*2 ISRTTYP            !SORT PROCESS
        INTEGER*4 SOR$PASS_FILES     !SORT FUNCTION NAMES
        INTEGER*4 SOR$INIT_SORT
        INTEGER*4 SOR$SORT_MERGE
        INTEGER*4 SOR$END_SORT
        INTEGER*4 ISTATUS            !STORAGE FOR SORT FUNCTION VALUE
C
C       INITIALIZE DATA - FIRST THE FILENAMES THEN THE KEY BUFFER FOR
C       ONE 80 BYTE CHARACTER KEY STARTING POSITION 1, 3 WORK FILES
C       AND AN INDEX SORT PROCESS
C
        DATA INPUTNAME,OUTPUTNAME/'R010SQ.DAT','TEST.TMP'/
        DATA KEYBUF,NUMWRK,ISRTTYP/1,1,0,1,80,3,3/
C
C       CALL THE SORT EACH CALL IS A FUNCTION
C
C
C       PASS SORT THE FILENAMES
C
        ISTATUS = SOR$PASS_FILES(INPUTNAME,OUTPUTNAME)
        IF (.NOT. ISTATUS) GOTO 10
C
C       INITIALIZE WORK AREAS AND KEYS
C
        ISTATUS = SOR$INIT_SORT(KEYBUF,,,NUMWRK,ISRTTYP)
        IF (.NOT. ISTATUS) GOTO 10
C
C       SORT THE RECORDS
C
        ISTATUS = SOR$SORT_MERGE( )
        IF (.NOT. ISTATUS) GOTO 10
C
C       CLEAN UP WORK AREAS AND FILES
C
        ISTATUS = SOR$END_SORT()
        IF (.NOT. ISTATUS) GOTO 10
        STOP 'SORT SUCCESSFUL'
10      STOP 'SORT UNSUCCESSFUL'
        END
```

Chapter 4
Error Conditions

You can encounter error conditions at three operating levels: first with the VAX/VMS DCL command interpreter, next with the SORT error messages, and last with VAX-11 RMS messages.

SORT handles two basic types of errors; fatal and warning. Fatal errors (severity level F) cause SORT to halt processing; warning errors (severity level W) cause a warning message to be output and allow sort processing to proceed. Errors in both these categories are grouped into three classes:

- Errors caused by I/O or other system failures.
- Errors caused by misinformation passed to SORT as a parameter of a subroutine call.
- Errors caused by invalid data in a key field.

For the SORT utility, errors of all types and classes are signaled to the system; this signal causes a message to be output. Execution is either stopped or continued based on the severity of the error. Execution can be resumed only if the severity level is W (that is, code = 0).

In summary, only invalid data errors and a few RMS errors cause warning error messages. System or I/O failures and bad subroutine parameters are fatal. For additional information regarding error condition handling, refer to the *VAX/VMS System Services Reference Manual.*

4.1 Command Interpreter Error Messages

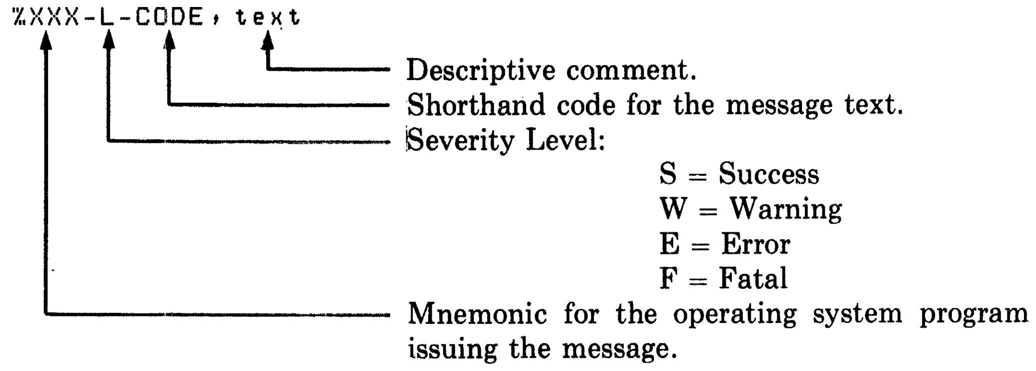
In interactive mode, when you enter a command line incorrectly, the command interpreter issues a descriptive error message telling you what was wrong. For example, if you specify more than one parameter for a command that accepts a single parameter, you receive the message:

```
%DCL-W-MAXPARAM, maximum parameter count exceeded
```

You must then retype the command line.

Other error messages may occur during execution of a command. These messages can indicate such errors as a nonexistent file or a conflict in qualifiers. Not all messages from the system indicate errors; other messages are informative, or merely warn you of a particular condition.

The VAX/VMS system messages have the general format:



```
%XXX-L-CODE, text
```

Descriptive comment.
Shorthand code for the message text.
Severity Level:
S = Success
W = Warning
E = Error
F = Fatal
Mnemonic for the operating system program issuing the message.

Example:

```
%SORT-W-CLOSEOUT, error closing output (output file-spec-
ification)
```

Because these messages are descriptive, you can usually understand what you need to do differently when you issue the command again. But, if you do not, the *VAX/VMS Messages and Recovery Procedures Manual* lists all the possible command interpreter error messages and describes what you can do to correct a command interpreter error.

4.2 SORT Error Messages

The following VAX-11 SORT error messages are listed in alphabetic order. All SORT error messages have the same format as command interpreter messages, that is:

```
%SORT-(severity level)-(code),(text).
```

The following descriptions of error messages observe the following conventions:

- Only the (code), (text) part of the message is shown in the following list.
- (filespec) indicates a file specification. For example: DB1:[153,10]TEST.TMP;3
- (number) indicates the user entered numeric value.
- LRL means the longest record length (specified in bytes).

```
BAD_ADR, invalid descriptor address specified.
```

You passed the subroutine package an address for a descriptor, and the descriptor was invalid format. Character string descriptors in VAX consist of two longwords. The first word of the first longword contains the character string length in bytes. The second longword contains the address of the string.

User Action: See Section 3.4 and check character string format.

```
BADFIELD, (filespec, or field text that is invalid) field in-
valid at (number).
```

Bad data in key field or command. In this message, (number) indicates the record number of the record containing bad data in hex.

User Action: Check key field data type and the starting positions and lengths (See Section 2.6, Setting Up the Keys).

```
BAD_FILE, file size invalid.
```

You specified a negative file size or a zero file size. File size must be greater than zero.

User Action: Specify a file size greater than zero.

```
BAD_KEY, invalid key specification.
```

Either the key field size, position, data type, or order is incorrect within the key definition. Positions start at one and cannot be greater than the maximum record size. Size must be less than or equal to 255 for character data, 1, 2, or 4 for binary data, and less than or equal to 31 for decimal.

User Action: See Section 2.6, Setting Up the Keys, and check the command string key specifications.

```
BAD_LEN, output record length less than 18 bytes for magtape.
```

Magnetic tape requires record lengths to be at least 18 bytes and no greater than 4096 bytes.

User Action: See Section 2.4.3, and check your output file block size parameters.

```
BAD_LRL, input file (filespec).
Record size greater than specified LRL.
```

In reading the input file, SORT encountered a record longer than the specified LRL. The record will be truncated to the LRL and sorted.

User Action: Re-execute SORT with a larger LRL.

```
BAD_SPEC, invalid specification file record.
FIELD:(record specification).
```

An incorrect field was specified in the specification file record. (record specification) indicates bad record contents.

User Action: See descriptions of specification file record formats (Section 2.5.2) and change field specifications.

```
BAD_TYPE, invalid sort process.
```

You passed the subroutine package a sort type code of less than 1 or greater than 4 if file I/O or not equal to 1 if record I/O, or an invalid key word in command /PROCESS. Legal values are 1-4 for file I/O, nothing for record I/O, and RECORD, TAG, INDEX, or ADDRESS for command /PROCESS parameter.

User Action: Specify a different sorting process.

```
CLEAN_UP, failed to reinitialize work area and files.
```

SORT was unable to deallocate the extra virtual memory, deassign work file channels, or readjust working set size. For the SORT utility, this is a warning of little importance. For the SORT subroutine packages, this could mean a failure to be able to recall SORT from the same program until it has exited. This is an internal error.

User Action: Exit from the user program before re-executing SORT.

```
CLOSEIN, error closing (filespec) as input.
```

An error occurred closing an input file. This message is usually accompanied by an RMS message indicating the reason for the failure.

User Action: Take corrective action based on the associated message.

```
CLOSEOUT, error closing (filespec) as output.
```

An error occurred closing an output file. This message is usually accompanied by an RMS message indicating the reason for the failure.

User Action: Take corrective action based on the accompanying message.

```
EXTEND, failed to extend work file.
```

SORT failed to extend a user's temporary work file. Either the device is full, or the user does not have extend privilege.

User Action: See Section 2.7 and reassign work files to a different device with more space, and make sure you have extend privilege on that directory.

```
INCONSIS, inconsistent data in file (filespec).
```

If you specified /OVERLAY plus other output file qualifiers, SORT will verify that the information in the existing file matches the information you provided. If it does not, this error message is reported. Unless you specifically want a verification, /OVERLAY should be used without other qualifiers.

User Action: Check the command string output file qualifiers (See Section 2.4.3).

```
IND_OVR, indexed sequential output requires overlay quali-
fier.
```

You specified indexed output file organization and did not specify /OVERLAY.

User Action: You must create the indexed file first with RMS DEFINE utility (or other). The primary key of the file should be the same as the sort key for efficiency but is not required to be. Then you must specify /OVERLAY in the SORT command string.

```
KEY_LEN, Key length invalid. Key number (number), size
(number).
```

The key size is incorrect for the data type, or the total key size is greater than 255.

User action: See Section 2.6, Setting Up the Keys, and specify correct key field size. Size must be less than or equal to 255 for character data, 1, 2, or 4 for binary data, and less than or equal to 31 for decimal. Also, only ascending or descending order is allowed.

```
LRL_MISS, LRL must be specified.
```

If record I/O interface subroutine package is selected, the longest record length (LRL) must be passed to SORT in the call.

User Action: See Section 3.4, and specify LRL.

```
MAP, failed to map work file.
```

This is an internal SORT failure.

User Action: Verify that the system parameter "maximum process sections" has been set up at 10. If it has, then report this failure to a specialist. Otherwise, set that system parameter to 10.

```
MISS_KEY, key specification missing.
```

SORT did not find any key definition in either the command line or specification file, or in the parameters to the subroutine package.

User Action: You must input at least one key definition in one of these three areas.

```
NO_WRK, need work files cannot do SORT in memory.
```

You specified /WORK-FILES=0 indicating the data would fit in memory, but the data was too large.

User Action: Either increase the working set quota, or allow SORT to use two or more work files.

```
NUM_KEY, too many keys specified.
```

Up to ten key definitions are allowed. Either too many have been specified, or the NUMBER value is wrong.

User Action: See Section 2.6, Setting Up the Keys, and check your command string key field specifications.

```
ONE_IN, only one input file allowed.
```

SORT will take only one input file at a time.

User Action: You can concatenate files of the same organization and record format using COPY, and then sort.

```
OPENIN, error opening (filespec) as input
```

An input file cannot be opened. This message is usually accompanied by an RMS message indicating the reason for the failure.

User Action: Take corrective action based on the associated message.

```
OPENOUT, error opening (filespec) as output
```

An output file cannot be opened. This message is usually accompanied by an RMS message indicating the reason for the failure.

User Action: Take corrective action based on the associated message.

```
READERR, error reading (filespec)
```

An input file record specified cannot be read. This message is usually accompanied by an RMS message indicating the reason for the failure.

User Action: Take corrective action based on the associated message.

```
SORT_ON, sort already in progress.
```

You tried to call the SORT subroutine package with calls in the wrong order, or to recall it before it finished running the previous sort.

User Action: Reorder the subroutine calls and then re-execute SORT.

```
VAR_FIX, cannot change variable length records into fixed
length.
```

You specified variable length input records and requested fixed length output.

User Action:Output records must be variable or controlled in this case.

```
VM_FAIL, failed to get required virtual memory (number).
```

SORT could not get the amount of virtual memory required for the sort. (number) indicates the number of bytes needed.

User Action: If the SORT utility is being run, decrease the working set quota; if either SORT subroutine package is being run, either decrease the quota or return some memory to the system inside the user's program before calling SORT.

```
WORK_DEV, work file (filespec)
device specified not random access or not local.
```

Work files must be specified for random access devices that are local to the CPU the sort is being performed on (that is, not on node in a network). Random access devices are disk devices.

User Action: See Section 2.7, Setting Up the Work Files, and specify the correct device.

```
WRITEERR, error writing (filespec)
```

An output file record cannot be written. This message is usually accompanied by an RMS message indicating the reason for the failure.

User Action: Take corrective action based on the associated message.

```
WS_FAIL, failed to get required working set space (number).
```

SORT could not set the required amount of real memory space. A minimum 75 page working set is needed. (number) indicates number of pages available.

User Action: Increase the working set quota.

4.3 VAX-11 RMS Error Codes

Listed below, in alphabetic order, are the VAX-11 RMS completion status codes. This list includes both symbolic and hexadecimal codes for error messages and success messages. These RMS codes are returned to your program by the operating system.

All VAX-11 RMS error messages have the same format as command interpreter messages, that is:

```
%RMS-(severity level)-(code),(text).
```

For additional information refer to the *VAX-11 Record Management Services Reference Manual.*

Valid Returns for Error Messages:

Symbolic	Hex Value	Meaning
RMS$_ACC	0001C002	File access error.
RMS$_ACT	0001825A	File activity precludes operation.
RMS$_AID	000183F4	Bad area identification number field in allocation XAB.
RMS$_ALN	000183FC	Invalid alignment boundary type in allocation XAB.
RMS$_ALQ	00018404	Incorrect allocation quantity in allocation XAB; the value either exceeds the maximum allowed, or is equal to zero for the extend service.
RMS$_ANI	0001840C	Records in a magnetic tape file are not ANSI D format.
RMS$_AOP	00018414	Invalid allocation option in allocation XAB.
RMS$_ATR	0001C0CC	Read error on file header.
RMS$_ATW	0001C0D4	Write error on file header.
RMS$_BKS	0001841C	Invalid bucket size in FAB.

Symbolic	Hex Value	Meaning
RMS$_BKZ	00018424	Invalid bucket size in the allocation XAB for relative file.
RMS$_BLN	0001842C	Invalid value in block length field.
RMS$_BOF	00018198	File is already at beginning of the file (backspace operation).
RMS$_BUG_DDI	0001843C	Invalid default directory. Internal VAX-11 RMS error; no recovery possible - contact a software specialist.
RMS$_CCR	00018494	Cannot connect RAB (only one record stream permitted for sequential files).
RMS$_CDA	0001C0E4	Cannot deliver AST.
RMS$_CHN	0001C0EC	Channel assignment failure.
RMS$_COD	000184AC	Invalid type code in XAB.
RMS$_CRE	0001C00A	File create error.
RMS$_CUR	000184B4	No current record; operation not immediately preceded by a successful get or find service.
RMS$_DAC	0001C012	File deaccess error during a close service.
RMS$_DEL	00018262	Record accessed by RFA record access mode has been deleted.
RMS$_DEV	000184C4	Bad device or inappropriate device type for operation.
RMS$_DIR	000184CC	Error in directory name.
RMS$_DME	000184D4	Dynamic memory exhausted; occurs only if the related I/O segment in the control region is full and the file is either a direct access process permanent file, or the user has disallowed the use of the program region for I/O buffers to VAX-11 RMS.
RMS$_DNA	000184DC	Error detected in the default file specification string.
RMS$_DNF	0001826A	Directory not found.
RMS$_DNR	00018272	Device not ready.
RMS$_DPE	0001C03A	Device positioning error; applies only to magnetic tape.
RMS$_DVI	000184F4	Invalid device identification in NAM block.
RMS$_ENT	0001C01A	Error during file enter service.
RMS$_ENV	00018724	Environment error; the code necessary to support the file organization or facility was not selected at system generation.
RMS$_EOF	0001827A	End of file.
RMS$_ESA	000184FC	Invalid expanded string area in NAM block.
RMS$_ESL	00018714	Invalid expanded string length in NAM block.
RMS$_ESS	00018504	Expanded string area too short.
RMS$_EXP	000182C2	File expiration date not yet reached.
RMS$_EXT	0001C022	File extend error.
RMS$_FAB	0001850C	Invalid FAB; block indentifier field incorrect.

Symbolic	Hex Value	Meaning
RMS$_FAC	00018514	Operation not allowed by the value set in the file access field of the FAB.
RMS$_FEX	00018282	File already exists.
RMS$_FLK	0001828A	File is locked and therefore not available.
RMS$_FNA	00018524	Invalid file specification string address in FAB.
RMS$_FND	0001C02A	Files-11 find function failed.
RMS$_FNF	00018292	File not found.
RMS$_FNM	0001852C	Syntax error in file name.
RMS$_FOP	0001853C	Invalid file processing options.
RMS$_FSZ	00018534	Invalid fixed control area size in FAB (equal to 1 for print files).
RMS$_FUL	00018544	Device full; cannot create or extend file.
RMS$_IFA	0001C124	Illegal file attributes; file header corrupted.
RMS$_IFI	00018564	Invalid internal file identifier in FAB; must be zero.
RMS$_IMX	0001856C	More than one XAB of the same type is present for the file.
RMS$_IOP	00018574	Illegal operation attempted: 1. block I/O when not block I/O access. 2. record I/O when block I/O access. 3. rewind of process permanent file. 4. inappropriate device type or file organization.
RMS$_IRC	0001857C	Illegal record in sequential file; invalid count field.
RMS$_ISI	00018584	Invalid internal stream identifier in RAB.
RMS$_KBF	0001858C	Invalid key buffer address; not in access limits.
RMS$_KEY	00018594	Invalid record key for random operation to a relative file.
RMS$_KSZ	000185A4	Key size not equal to 4 (relative file).
RMS$_LNE	000185BC	Logical name error; resulted in duplicates.
RMS$_MBC	00018734	Invalid multi-block count; must not be greater than 127.
RMS$_MKD	0001C032	Files-11 ACP could not mark file for deletion.
RMS$_MRN	000185CC	Illegal value for maximum record number.
RMS$_MRS	000185D4	Illegal value for maximum record size.
RMS$_NAM	000185DC	Invalid NAM block.
RMS$_NEF	000185E4	Attempt to use the put service to a sequential file when not positioned to end of file.
RMS$_NMF	000182CA	No more files for a search operation.
RMS$_NOD	000185F4	Node name error.
RMS$_ORG	0001860C	Illegal file organization.
RMS$_PBF	00018614	Invalid prompt buffer address.

Symbolic	Hex Value	Meaning
RMS$_PLG	0001861C	Error in file prologue; file is corrupted.
RMS$_PLV	0001872C	Prologue version unsupported.
RMS$_PRV	0001829A	Privilege violation; access denied.
RMS$_QUO	00018634	Error in quoted string.
RMS$_RAB	0001863C	Not a valid RAB; block identifier field incorrect.
RMS$_RAC	00018644	Illegal value in record access mode field of RAB.
RMS$_RAT	0001864C	Record attributes invalid in FAB.
RMS$_RBF	00018654	Invalid record address.
RMS$_RER	0001C0F4	File read error.
RMS$_REX	000182A2	Record already exists; in a random access mode operation to a relative file, a record was found in the target record cell.
RMS$_RFA	0001865C	Invalid record's file address contained in RAB.
RMS$_RFM	00018664	Illegal record format.
RMS$_RHB	0001866C	Invalid record header buffer.
RMS$_RLF	00018674	Invalid related file.
RMS$_RLK	000182AA	Record locked by another task.
RMS$_RMV	0001C0FC	Files-11 remove function failed.
RMS$_RNF	000182B2	Record not found.
RMS$_RNL	000181A0	Record not locked.
RMS$_RPL	0001C104	Error while reading prologue.
RMS$_RSA	0001868C	Record stream active; an attempt was made to issue a record operation request in an asynchronous environment to a record stream that has a request outstanding.
RMS$_RSL	0001873C	Resultant string length field of NAM block invalid.
RMS$_RSS	00018694	Resultant string area size field of NAM block is too small.
RMS$_RST	0001869C	Invalid resultant string area.
RMS$_RSZ	000186A4	Illegal record size.
RMS$_RTB	000181A8	Record too large for user buffer.
RMS$_SHR	000186B4	Invalid value in the file sharing field of FAB.
RMS$_SQO	000186C4	Operation not sequential.
RMS$_SYN	000186D4	Syntax error in file specification.
RMS$_SYS	0001C10C	Error in system QIO directive.
RMS$_TMO	000181B0	Time-out period expired.
RMS$_TYP	000186E4	Error in file type.
RMS$_UBF	000186EC	Invalid user record area address.
RMS$_USZ	000186F4	Invalid user record area size.
RMS$_VER	000186FC	Error in version number.

Symbolic	Hex Value	Meaning
RMS$_WER	0001C114	File processor write error.
RMS$_WLK	000182BA	Device is not write-locked.
RMS$_WPL	0001C11C	Error while writing prologue.
RMS$_WSF	0001871C	Working set full.
RMS$_XAB	0001870C	Not a valid XAB.

Valid Returns for Success Messages:

RMS$_CONTROLC	00010651	Operation completed under Control C.
RMS$_CONTROLO	00010609	Operation completed under Control O.
RMS$_CONTROLY	00010611	Operation completed under Control Y.
RMS$_CREATED	00010619	File was created; not opened; used in conjunction with the CIF option.
RMS$_KFF	00018031	Known file found.
RMS$_NORMAL	00010001	Operation successful (synonym for RMS$_SUC).
RMS$_OK_ALK	00018039	Record already locked.
RMS$_OK_DEL	00018041	Deleted record accessed correctly.
RMS$_OK_RLK	00018021	Record locked but read anyway; locked set RLK bit in ROP field.
RMS$_OK_RNF	00018049	Non-existent record accessed correctly.
RMS$_PENDING	00018009	Asynchronous operation not yet completed.
RMS$_SUC	00010001	Operation successful (synonym for RMS$_NORMAL).
RMS$_SUPERSEDE	00010631	Created file superseded an existing version.

Chapter 5
Improving SORT Efficiency

Users who have special sorting requirements such as very large files, storage media contraints, and processing time restrictions can modify SORT's behavior for optimum performance. Your ability to improve SORT's performance depends on your understanding of SORT's operational characteristics described in this chapter.

This chapter discusses:

- How the SORT program functions in each phase of operation, and what sequence of events occur during a sort run
- How a user can improve SORT's efficiency through the use of tuning procedures

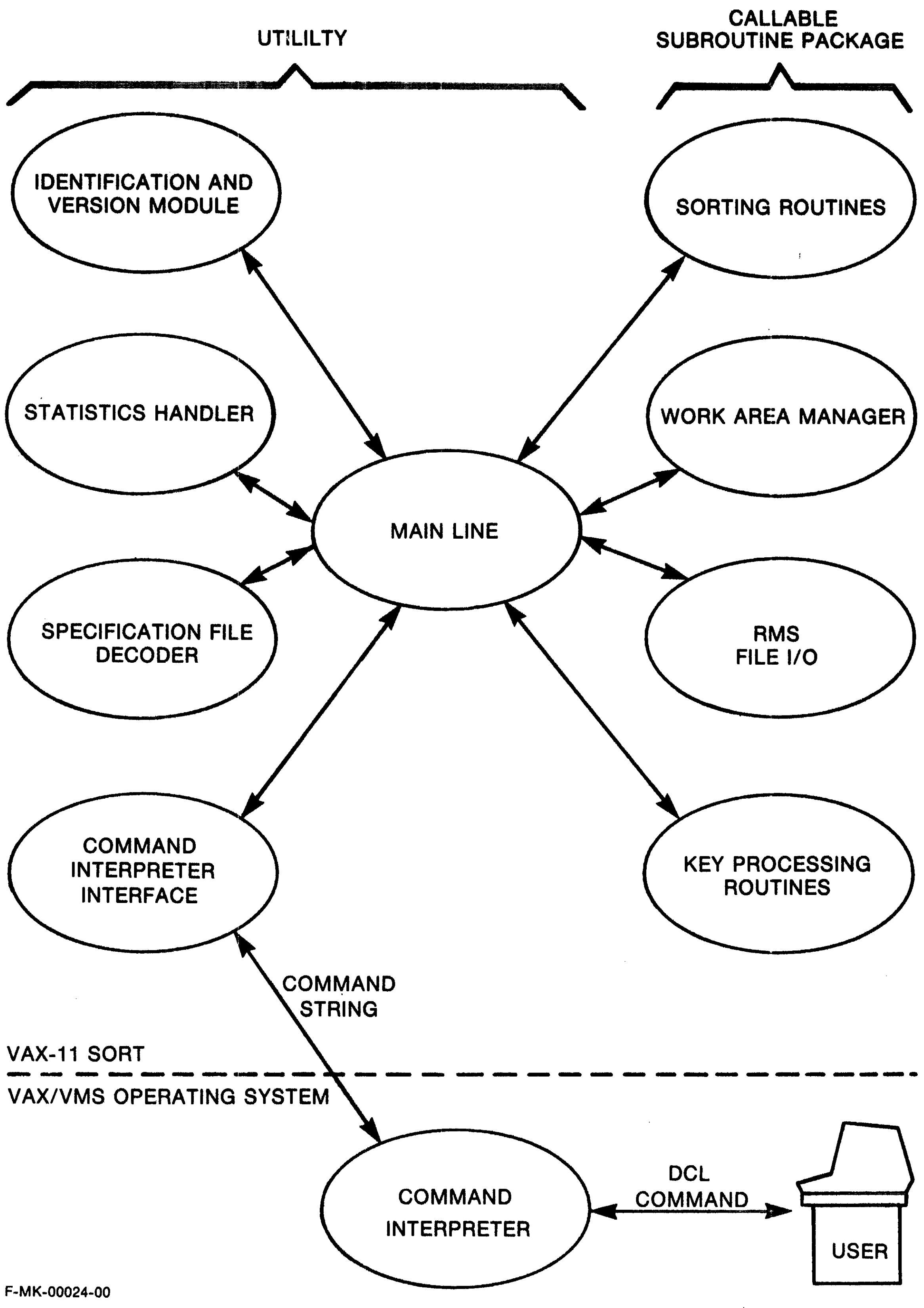
5.1 Functional Description

The SORT program consists of two basic parts: a control program called the utility and a callable subroutine package (see Figure 5-1). The utility directs the overall processing. The callable subroutine package serves as a collection of subroutines that the utility uses during its processing. You can write your own control program to take advantage of SORT's callable subroutines (see Chapter 3).

There are eight phases of operation in the SORT utility. These are described in more detail in Section 5.1.2. A sort run breaks down into three tasks.

First, SORT reads the command string and the specification file, if present, decodes them, and then stores the qualifier values and parameters. Any errors in the command string or specification file are reported at this point.

Figure 5-1: VAX-11 SORT Architecture, Main Functional Components



Second, SORT begins the pre-sort operation. The control program calls routines to open and read the input file and establish the keys. Then the SORT subroutine package is called to begin the initial sorting process. At this point, the amount of available internal storage space becomes important to the efficiency of the sort. If that space is not sufficient to hold all the records, SORT builds strings of sorted records and transfers them to work files on temporary storage devices (disk). The SORT program normally provides for a default of two work files. A qualifier in the command string can increase the number of work files used.

Third, SORT rebuilds the intermediate work files into a merged file. If the process is tag sort, another subroutine reads the records in the proper sequence. The records are then written in the output file. If there are no work files to merge because main memory was sufficient to hold all the records, the sorted records are written directly into the output file. After the last record is written, the control program cleans up the work files and exits; SORT is then ready to accept another job.

5.1.1 Sorting Processes

All four sorting processes can sort records of fixed or variable length, VFC, or any valid VAX-11 RMS. *Stream format is not supported.* The size of the records on a fixed-length format file is determined when the file is created. The first word of a variable-length format record contains the size of the record in bytes. This first word is used by the file system and is transparent to SORT.

5.1.1.1 Record Sort — Record sort outputs all data records in a specified sorted sequence. Each record is kept intact throughout the entire sorting process. Since this process moves the whole record, it is relatively slow and may require considerable main memory or external storage work space for large files.

5.1.1.2 Tag Sort — Tag sort produces the same kind of output file as record sort, but it only handles record pointers and key fields. Since this process moves a smaller amount of data than record sort, it may perform a faster sort than record sort. The input file must be randomly re-accessed to create the entire output file, which may be a lengthy process for large files.

Input Data Files

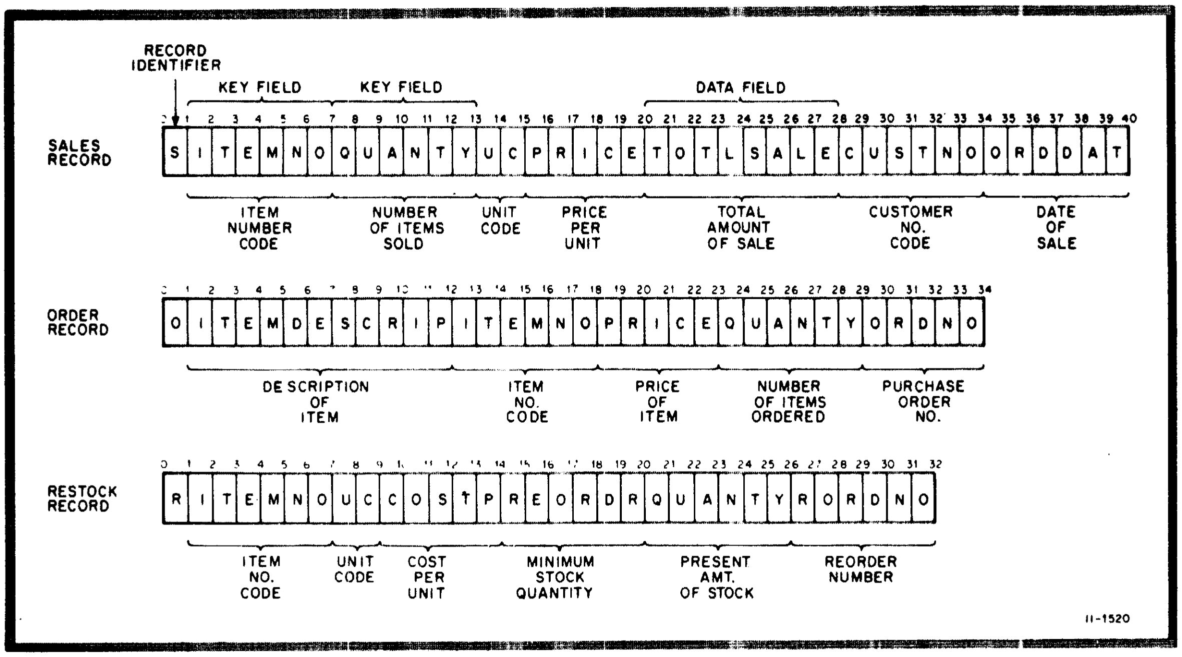
A record is usually divided into several logical areas called data fields. The data in each field may or may not be relevant to SORT. Each field may be interpreted as a record identifier, key data, or general data related to the logical content of the record and not relevant to the sorting process. SORT uses record identifiers to distinguish the various types of records in a file. SORT uses the key fields in each record to reorder an input file. Any other data field in a record may be retained in the output file or ignored by SORT.

Figure 5-2 shows three different types of input records, each with a different format. The record identifiers are the letters in position 1: S means sales record, O means order record, and R means restock record. In this case, the keys chosen for sorting the sales record types are the "item number code" in positions 2 to 7, and the "number of items sold" in positions 8 to 13. The "total amount of sale" is an example of a data field not relevant to the sorting process.

If you request a sort in ascending order on the sales records as shown in Figure 5-2, the sort is based on the item number code first and then on the number of each item sold within that item number. In order of decreasing significance, the keys are:

1. Item number
2. Number of items sold

Figure 5-2: Sample Record Types



Output Data Files

The output file contains all sales records in the order shown in Table 5–1.

Table 5–1: Sorted Output File

Major Key: Item Number	Minor Key: Quantity
Lowest item no.	Lowest quantity
	Next higher quantity
.	.
.	.
.	.
.	.
Lowest item no.	Highest quantity
Next higher item no.	Lowest quantity
	Next higher quantity
.	.
.	.
.	.
.	.
Next higher item no.	Highest quantity
Highest item no.	Lowest quantity
	Next higher quantity
.	.
.	.
.	.
.	.
Highest item no.	Highest quantity

5.1.1.3 Address Sort — Address sort produces address files, which consist of record's file addresses (RFAs), beginning at 1, and written in binary words. These files can be used as a special index file to access randomly the data in the original file. It is possible to maintain only one data file, but several different index files as needed. Like tag sort, this process uses the minimum amount of data necessary in the sorting process. Once the input phase is completed, the input file is not read again. This means that address sort is the fastest sorting method of the four SORT types.

NOTE:

Do not transfer an address index file to a device that cannot handle binary data, such as a printer or terminal.

The address sort produces an output file consisting of record indices. Each record index occupies one 6–byte record in the output file. Assume that you are sorting a file consisting of six records using the address sort process. If the sequence of record indices corresponding to the sorted records is 5,1, 6,3,4,2 then the output file can be represented as shown in Figure 5–3.

Figure 5–3: Sample Address Sort Output File

	Record's File Address (RFA)		
	Block Number		
RECORD NUMBER	LOW (2 bytes 16 bits)	HIGH (2 bytes 16 bits)	BYTE-IN-BLOCK (2 bytes 16 bits)
1	000001	000000	000162
2	000001	000000	000000
3	000001	000000	000236
4	000001	000000	000042
5	000001	000000	000132
6	000001	000000	000026

Number of Blocks per File

Number of Contiguous Bytes per Block

Note:

Byte and Block numbers are shown here in hexadecimal, they are written to the actual output file in binary.

5.1.1.4 Index Sort — Index sort produces an address file consisting of records file addresses (RFAs) in binary, and key fields in original form. This makes it slightly slower than address sort. During processing this sort handles only the RFAs and two forms of the key fields. One form is used for sorting and the other is left as it was in the original data.

Index sort produces an output file consisting of record indices plus keys in original form. Each record in the output file consists of a 6–byte record index plus the key field.

NOTE:

Do not transfer an index sort output file to a device that cannot handle binary data, such as a printer or terminal.

Assume that you are sorting a file consisting of six records using Index sort process, and you are using a key size of four characters (bytes). The sequence of record indices corresponding to the sorted record, is 5,1,6,3,4,2 as shown in Figure 5–4.

Figure 5–4: Sample Index Sort Output File

	Record's File Address (RFA)						
	Block Number						
RECORD NUMBER	**LOW (2 bytes 16 bits)**	**HIGH (2 bytes 16 bits)**	**BYTE-IN-BLOCK (2 bytes 16 bits)**	**KEY IN ORIGINAL FORM**			
1	000001	000000	000162	A	B	C	D
2	000001	000000	000000	A	B	C	D
3	000001	000000	000236	A	B	C	D
4	000001	000000	000042	A	B	C	D
5	000001	000000	000132	A	B	C	D
6	000001	000000	000026	A	B	C	D

Notes:

1. ABCD represents the sorting keys in original format.
2. Byte and Block numbers are shown here in hexadecimal, they are written to the actual output file in binary.

5.1.2 Internal Organization

SORT operates in eight phases (phase 0 through 7). Figure 5–5 summarizes these phases.

Phase	**Function**
0	Decode command line and specification file
1	Initialize SORT
2	Get records
3	Sort records
4	Initialize merge
5	Merge records
6	Output records
7	End SORT

The VAX/VMS command interpreter calls the SORT utility at its main entry point and Phase 0 is initiated. The initial process statistics are acquired from the system and stored in a table. SORT calls the command interpreter to parse and validate the command line. Then SORT validates this information and stores it in various tables and buffers. If a specification file is present it is opened, the records are read, the information validated and stored in various tables and buffers, and the file is closed. Any errors up to this point are reported by signaling the command interpreter. SORT opens the input file, and creates the output file.

SORT begins Phase 1. The sorting process is initialized by filling in the key comparison information, allocating the space needed for input and output buffers and the sort tree, creating the work files and initializing the sort tree.

Phase 2 begins the sort proper. SORT either reads records from the input file, or receives them from the caller.

At Phase 3, SORT builds the key from the record and inserts each record into the tree by key. This process repeats until the sort tree is full or there are no more records. SORT then outputs the records to the work files as a variable number of strings each of which is a set of sorted records. Each time a record is output from the tree a new one is input until there are no more records. The rest of the records in the tree are output and that ends the initial sorting phase (phases 1 through 3).

Phase 4 starts SORT's internal merging operation. The memory is redivided at this point for the merge phase into one to ten input buffers and one output buffer, depending on the number of initial strings. A different string is read into each input buffer and the records are merged together into one string and output to a work file. This process is repeated until the total number of strings is less than ten.

Phase 5 performs the final merge pass and outputs the remaining string of records, which is the final sorted file, to either the output file or the caller.

Phase 6 closes the input and output files, closes and deletes the work files, and returns the memory.

Phase 7 acquires the final statistics and prints them, then exits SORT back to the VAX/VMS command interpreter.

Notes:

1. Phases 0 and 7 are part of the utility only.
2. The last part of phase 0 (opening the input file and creating the output file) and phase 6 are used only by the file I/O subroutine package and the utility.
3. Phases 1 through 5 are used by the utility and both the file I/O and record I/O subroutine packages.

4. Errors during phases 1 through 6 are signaled to the VAX/VMS command interpreter if the utility is running, or returned as a status code to the caller if the subroutine package is running.

5. All signaled errors produce messages at the command interpreter level (see Chapter 4).

Figure 5-5: VAX-11 SORT Operating Phases

PHASE 0

DCL SORT Command (via VAX/VMS command interpreter).
Entry point to SORT.
Get initial process statistics.
Call command line processor to decode command line.
If it was specified, call specification file decoder.
Open the input file and create the output file.

PHASE 1

Set up the key comparison buffer and validate key information.
Get memory for sorting initial phase, input and output buffers, and set up to read input.
Create work files and initialize the sort.

PHASE 2

Read or get record from user and build key.

PHASE 3

Insert record by key into sort tree.
Is sort tree full? (YES, continue /NO, go back to phase 2)
Output records to work files as a number of strings of sorted records.
Output and input until end of file and all records are output.

PHASE 4

Read in strings from work file and merge them until there are 10 or fewer strings left in the work files.

PHASE 5

Set up to output records to user or output file.
Do final merge pass outputting records to user files, not work files.

PHASE 6

Delete work files, return memory, close output and input files.

PHASE 7

Print statistics and exit.

5.1.3 Buffer Allocation and Work Areas

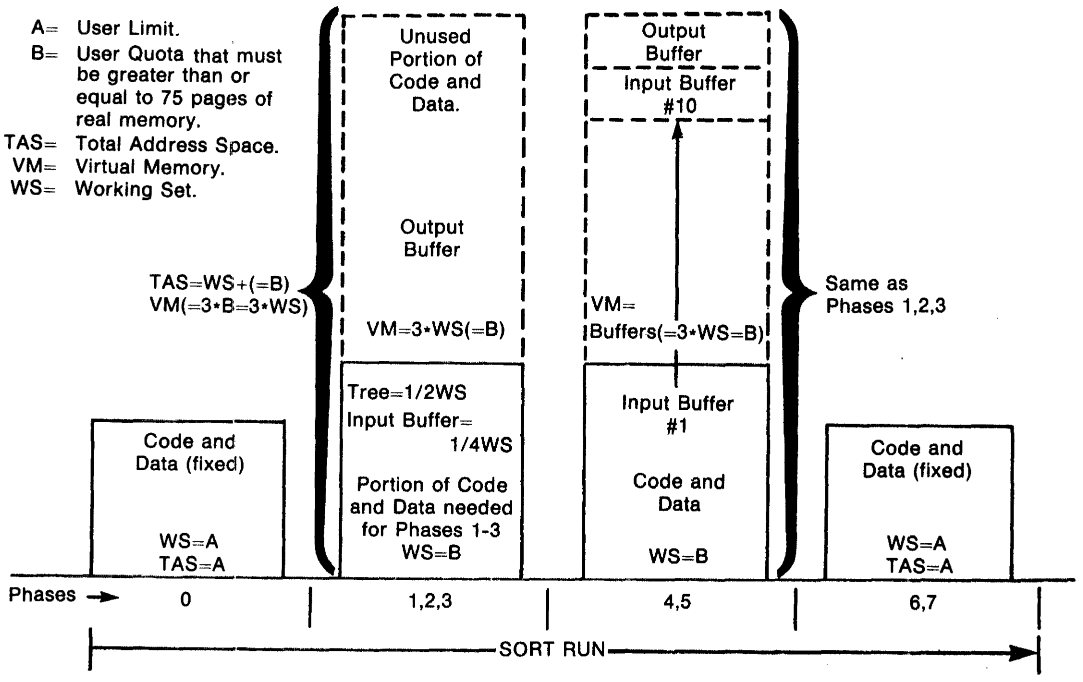
The SORT utility and subroutine package are initially linked with a minimum of space allocated. When SORT is initialized, the work area manager assigns as much virtual memory as the process needs, and adjusts the working set size to the process maximum. This allows SORT to minimize page faults during the sorting, and maximize the order of the merge. At the end of the sort operation the limits are restored to the size they were at the entry to SORT, returning the additional virtual memory to the system. SORT requires a minimum of 75 pages of memory for the working set.

The VAX/VMS memory management system service, create and map section, allows a user to specify that a particular span of virtual addresses in the program should be read from and written into a particular set of virtual blocks within a file on disk, when referenced or paged-out by the entry of another page. The actual I/O to and from the disk is all handled by the pager. SORT maps the individual virtual addresses representing the work area onto specific blocks within a work file. When a particular buffer is then referenced within the work area, the pager automatically brings the correct blocks from the work file into real memory, and writes the existing blocks back to the correct place in the file.

5.1.4 Dynamic Memory Usage

Figure 5-6 shows the total address space used by SORT during each of its eight phases of operation.

Figure 5-6: SORT Dynamic Memory Usage



H-MK-00025-00

5.1.5 I/O Considerations

The input and output files I/O and the specification file I/O are all performed under the control of VAX-11 RMS record I/O facilities. Multi-buffering is used together with read ahead on unit record devices to optimize the I/O operations. The work files are processed by the VAX/VMS memory management system service, create and map section.

Various devices can be used for input and output files. Figure 2-1 shows which devices are allowed for each of the four sorting techniques. Use Figure 2-1 to match the sorting process with the devices that best suit your processing environment. Data may be stored in binary, ASCII, decimal, packed or zoned.

5.2 Tuning Procedure

All generalized sorts consider several factors such as: the memory environment (large, small, virtual memory capability or not); the I/O devices to be used for work files and their characteristics (speed, arm movement, seek time, public or private units); type of files and data most likely to be sorted (large or small files, large or small records and keys, random or ordered partially, characters or numbers).

The algorithm must be very good for the cases occurring most often, and reasonable on all other cases. VAX-11 SORT is designed for an environment of: fairly large files, virtual memory capability, random access disk devices, public and private, larger random character data files, medium size records and keys.

There are three components of a sort that account for the majority of the processing time:

- The number of key comparisons per record per sort.
- The number of merge passes needed to complete the sort.
- The amount of time spent waiting for and/or doing I/O to work files.

5.2.1 User Performance Considerations

This section discusses how you can determine the most efficient values for the following SORT performance parameters.

- Working set quota
- Work file devices
- Number of work files
- Type of sort (process)
- File size

- Record size
- Key size
- System load
- System process parameters

5.2.1.1 Working Set Quota — For SORT to work efficiently the most important parameter is the working set quota (or size) the user decides to choose. The optimum working set quota is the smallest one for which the data can be sorted in memory, that is with 0 merge passes.

To compute the appropriate working set quota size, perform the following procedure:

Step 1. For any sort, take the size of the key fields added together in bytes.

Step 2. Then add 20.

Step 3. If the sorting process is record sort, add the number of bytes in the longest record; otherwise add 6. Then multiply by the number of records in the file. This is the total amount of data you have in bytes.

Step 4. Divide that number by 512 to get the amount in blocks.

Step 5. Multiply by 2 to get the size of the working set quota you should start with.

For most larger files the number computed will be much to large to actually use as a quota. In such cases, the largest reasonable size based on the system load and scheduling considerations is the correct size to use. An individual user's authorized quota is generally the largest reasonable size for the particular system.

For example:

> To sort a 1000 record file with 80–byte records and a total key field size of 80 bytes using the record process, compute the following:
>
> 1000 X (80+20+80) = 180,000 bytes of data
>
> 180,000/512 = 352 blocks of data
>
> 352 X 2 = 704 block working set quota
>
> Answer: start with a working set quota of 700.

However, if the same type of a file contained 40,000 records, the total amount of data would be 14,063 blocks. For most systems a quota of 28,000 blocks (pages), or even 14,063 is unacceptable. Here the largest reasonable quota should be used; for example 1024 pages.

5.2.1.2 Work File Devices — Another important parameter is where the work files are placed. The fuller the disk and the more activity on the disk containing the work files, the less efficient SORT will be. The optimum configuration would be to have each work file, and the input and output file all on separate empty disks which are only being used by SORT during the sorting process. However, this is seldom possible, so the next best configuration is to place work files on available disks having the lowest activity. See Section 2.7, Setting up the Work Files.

5.2.1.3 Number of Work Files — Because SORT does not depend on the number of work files used to determine the order of the merge like SORT–11, the advantage of using more than the default number of work files is limited. There are two reasons for using more than the default of two work files: 1) to spread the work files between more than two disks and/or 2) to have each individual file be a smaller size in order to fit onto a smaller or fuller disk.

If you are using three or four disks, it will help the sort performance to use three or four work files, one on each disk as discussed above. For example, if you have a 100,000 block file to sort, using two work files would create two 150,000 block files. But, using four work files would create four 75,000 block files that could be placed on disks with less free space.

5.2.1.4 Type of Sort — Although the type of sort used is often dictated more by functionality required than performance there are significant differences between the sorts.

Address sort is the fastest and uses the least temporary disk space.

Index sort is only slightly slower than address sort but uses more tempory storage.

Tag sort uses the same temporary storage as address sort, but is significantly slower. For large records with small keys it is faster than record sort in smaller memory sizes if the file is not large.

Record sort uses a larger amount of temporary storage and is the slowest.

5.2.1.5 Using SORT's Statistics — Analyze the sort statistics (Section 2.2.3) to determine how to improve the sort's performance. The number of records in, out, sorted if not all equal indicates that there were input or output errors, or that there are null records in the file (that is, the number of records read was greater than the number sorted or the number output). This condition can also be caused by some records containing invalid data in the key fields (if less than ten records are in error SORT will continue, otherwise SORT will stop executing).

Longest record length value is obtained from either RMS or the user and can be used to make sure the RMS value is correct.

The multi block and buffer counts indicate the amount of I/O optimization on the input and output file. The larger the working set quota the more optimization possible. No optimization would show all these counts as 1. This should not occur unless the file is huge compared to the working set quota. If it does, raise the quota if possible.

The order of the merge is the number, less one, of merge buffers that the working set is divided into for the merge phase.

Number of merge passes and **the number of initial runs** shows you how close the data is to fitting in memory. The higher these numbers are, particularly the number of passes, the longer SORT takes and the further away the working set size is from containing the data.

Virtual memory added is the amount of virtual memory SORT used for the data.

Elapsed time is the total wall clock time in hours, minutes, seconds, and 1/100 seconds from start to end for the sort run.

The total of the two I/O counts are the number of disk hits to get and write data and these will be higher if the multi block and buffer counts are lower. The lower the better.

CPU time is the time spent actually processing data minus all I/O time. The closer to the elapsed time the better optimization you are seeing in I/O.

Page faults are also a good indication of how well the data did or did not fit into memory. The higher the number of page faults, the less efficient the sort is.

5.2.2 System Manager Performance Considerations

The system manager can determine the following SORT performance parameter values based on the overall system usage: number of users, types of process most commonly run, and the amount of real memory available.

- System per process working set quota (WSMAX)
- System per process virtual page count (VIRTUALPAGECNT)
- System per process section count (PROCSECTCNT)
- System modified page writer cluster factor (MPW_WRTCLUSTER)

The values recommended are based solely on sort considerations; it is up to the system manager to integrate other system considerations with these in determining the appropriate final values.

5.2.2.1 Working Set Quota — The maximum for this value should be set to the largest size any sort job would ever require. For very large files, working sets of 500 to 1000 pages are not at all unreasonable, provided the system has enough physical memory to accommodate them. Individual maximums, to prevent users from monopolizing real memory, can be set on a per user basis by using the authorization file. For information on how to determine an appropriate working set for a particular sort job see Section 5.2.1.1. The general rule is, the smaller the working set, relative to the files to be sorted, the slower the sort.

5.2.2.2 Virtual Page Count — For this parameter the current value as well as the maximum value should be set to a minimum of 3 to 4 times the value of the working set quota maximum. When SORT initially starts executing it will request 2 and 1/2 to 3 times the working set quota of virtual memory from the system. If this value is too low SORT will be unable to run in certain cases.

5.2.2.3 Process Section Count — For working set quota maximums of 500 or less this parameter may stay at a minimum level. However, for working set quotas greater than 500 to 1000 a current value of 10 or greater is necessary. If this parameter is set too low, SORT will be unable to run in larger working sets due to internal mapping failures. The value should be increased as the working set quota maximum increases.

5.2.2.4 Modified Page Writer Cluster Factor — The value of this parameter will never cause SORT to fail, however it can cause a large difference in performance. For any larger sorts (that is, using working sets of 250 pages or greater) the larger this parameter, the better. Values of 64 and up are not too large. Be sure to adjust MPW_HILIM and MPW_LOLIMIT accordingly. For more information refer to the SYSGEN procedures in the *VAX-11 Software Installation Guide.*

Glossary

Alphanumeric Characters

The entire set of 128 ASCII characters (see Appendix B).

ASCII Character Set

The set of 128 eight-bit American Standard Code for Information Interchange characters (see Appendix B).

Batch

A mode of processing in which all commands to be executed by the operating system and, optionally, data to be used as input to the commands are placed in a file or punched onto cards and submitted to the system for execution.

BLISS

A high-level system implementation programming language. VAX-11 SORT is written in BLISS.

Block

The smallest addressable unit of data that the specified device can transfer in an I/O operation (512 contiguous bytes for most disk devices).

Bucket

See File Bucket.

Buffer

A temporary data storage area in a process address space used when performing input or output operations.

Byte

The smallest addressable unit of information; eight bits. For example, an ASCII character requires a single byte (see Appendix C for further definitions).

Call

The operation of invoking a procedure.

Caller

The procedure that invoked this procedure by a Call. At the time of procedure invocation, the invoking procedure is said to be the caller, and the invoked procedure is the callee. Contrast with User.

Character

The smallest addressable unit of usable data (byte). It is also a single letter, numeral, punctuation mark, or other symbol (such as $ or %), and is represented within the computer as a unique combination of bits. Typically, a character code consists of eight bits.

Character String Descriptor

A quadword data structure used for describing character data (strings). The first word of the quadword contains the length of the character string. The second word can contain type information. The remaining longword contains the address of the string.

CPU

The Central Processor Unit portion of a computer system.

Collating Sequence

The order into which characters are sorted based upon numeric values assigned to each.

Command

An instruction, generally an English word, typed by the user at a terminal or included in a command file, which requests the software monitoring a terminal or reading a command file to perform some well-defined activity. For example, typing the SORT command request the system to invoke the SORT utility.

Command File

A file containing command strings. See also Command Procedure.

Command Interpreter

Procedure-based system code that executes in supervisor mode in the context of a process to receive, syntax check, and parse commands typed by the user at a terminal or submitted in a command file.

Command Parameter

The positional operand of a command delimited by spaces, such as a file specification, option, or constant.

Command Procedure

A file containing commands and data that the command interpreter can accept in lieu of the user typing the commands individually on a terminal.

Command String

A line (or set of continued lines), normally terminated by typing the carriage return key, containing a command and, optionally, information modifying the command. A complete command string consists of a command, its qualifiers, if any, and its parameters (file specifications, for example), if any, and their qualifiers, if any.

Compatibility Mode

A mode of execution that enables the central processor to execute non-privileged PDP-11 instructions. The operating system supports compatibility mode execution by providing an RSX-11M programming environment for an RSX-11M task image. The operating system compatibility mode procedures reside in control region of the process executing a compatibility mode image. The procedures intercept calls to the RSX-11M executive and convert them to the appropriate operating system functions.

Contiguous Blocks

Physically adjacent and/or consecutively numbered blocks of data.

Data File Record

A record containing user data.

Data Structure

Any table, list, array, queue, or tree whose format and access conventions are well defined for reference by one or more images.

Data Type

In general, the way in which bits are grouped and interpreted. In reference to the processor instructions, the data type of an operand identifies the size of the operand and the significance of the bits in the operand. Operand data types include: byte, word, longword, and quad-word integer, floating and double floating, character string, packed decimal string, and variable-length bit field (see Appendix C).

DCL

Digital Command Language (DCL) is a set of English- like statements that a user types to initiate and control system operations.

Default

An assumed value supplied to the system when a command qualifier does not specifically override the normal command function; fields in a file specification that the system fills in when the specification is not complete.

Descriptor

See Character String Descriptor.

Device

The general name for any physical terminus or link connected to the processor that is capable of receiving, storing, or transmitting data. Card readers, line printers, and terminals are examples of record-oriented devices. Magnetic tape devices and disk devices are examples of mass storage devices. Terminal line interfaces and interprocessor links are examples of communications devices.

Directory

A file used to locate files on a volume that contains a list of file names (including extension and version number) and their unique internal identifications.

Directory Name

The field in a file specification that identifies the directory file in which a file is listed. The directory name is enclosed in brackets ([] or <>).

Field

A logically distinguishable area within a record. Usually a logical unit of data.

File

A logically related collection of data on a volume such as disk or magnetic tape. A file can be referenced by a name assigned by the user. A file normally consists of one or more logical records.

File Bucket

Within the RMS Relative File organization, a bucket is a storage structure of one to 32 blocks of data.

File Header

A block in the index file describing a file on a FILES-11 disk structure. The file header identifies the locations of the file's extents. There is a file header for every file on the disk.

File Organization

The particular file structure used to record the data constituting a file on a mass storage medium. RMS file organizations are: Sequential, Relative, and Indexed.

File Prologue

The first block in a relative or indexed file which contains header information for the file.

File Specification

A unique name for a file on a mass storage medium. It identifies the node, the device, the directory name, the file name, and the version number under which a file is stored (see Appendix D for additional information).

File Structure

The way in which the blocks forming a file are distributed on a disk or magnetic tape to provide a physical accessing technique suitable for the way the data in the file is processed.

File System

A method of recording, cataloging, and accessing files on a volume.

File Type

The field in a file specification that is preceded by a period or dot(.) and consists of a zero-to three-character type identification. By convention, the type identifies a generic class of files that have the same use or characteristics, such as ASCII text files, binary object files, etc.

Files–11

The standard physical disk structure used by VAX–11 RMS.

Filespec

File Specification that uniquely identifies a file by physical location (see Appendix D).

File, Input

See Input File.

File, Output

See Output File.

File, Work

See Work File.

Fixed Control Area

An area associated with a variable length record available for controlling or assisting record access operations. Typical uses include line numbers and printer format control information.

Fixed Position Field

An area associated with character position (or column numbers). Used in SORT–11 Specification Files.

Fixed Length Record Format

A file format in which all records have the same length.

Format

The arrangement of any record or file; the order in which fields reside in a record.

Free Fields

Logically positioned fields separated by commas. Contrast with fixed position fields.

Home Block

A block in the index file that contains the volume identification, such as volume label and protection.

Image

A file consisting of procedures and data that have been bound together by the linker. There are three types of images: Executable, Shareable, and System.

Indexed File Organization

A file organization in which a file contains records and a primary key index (and optionally one or more alternate key indices) used to process the records sequentially by index or randomly by index.

Index File

The file on a Files-11 volume that contains the access information for all files on the volume and enables the operating system to identify and access the volume.

Index File Bit Map

A table in the index file of a Files-11 volume that indicates which file headers are in use.

Index File Record

A record of file system data that is invisible to the user.

Input File

The file containing the records you wish to sort.

Key, Key Field

The data field containing the values chosen from a record to control the sort (see section 2.6).

Key, Major

The most important field in the total key. If you were sorting a list by department, salary and name, department would be the major key.

Key, Minor

The least significant field in the total key. In the preceding example name is the minor key.

Library

A collection of commonly used files.

Line

In this document, a line generally refers to a line in the SORT specifications form or a record in the specification file.

Logical Name

A user-specified name for any portion or all of a file specification. For example, the logical name INPUT can be assigned to a terminal device from which a program reads data entered by a user. Logical name assignments are maintained in logical name tables for each process, each group, and the system. A logical name can be created and assigned a value permanently or dynamically.

Longword

Four contiguous bytes (32 bits) starting on an addressable byte boundary (see Appendix C).

LRL

Longest Record Length (LRL) specified in bytes.

Merge

A process by which two or more ordered groups of records are put together record-by-record into a single identically ordered group.

Native Mode

The processor's execution mode, in which the programmed instructions are interpreted as byte-aligned, variable length instructions that operate on byte, word, longword, quadword integer, floating and double floating, character string, packed decimal, and variable length bit field data. The instruction execution mode other than compatibility mode.

Node

An individual computer system in a network.

Output File

The file created by running SORT. The output file may be either a data file or an address file.

Page

1). A set of 512 contiguous byte locations used as the unit of memory mapping and protection. 2). The data between the beginning of file and a page marker, between two markers, or between a marker and the end of a file.

Page Fault

An exception generated by a reference to a page which is not mapped into a working set.

Pager

A set of kernel mode procedures that executes as the result of a page fault. The pager makes the page for which the fault occurred available in physical memory so that the image can continue execution. The pager and the image activator provide part of the operating system's memory management functions.

Paging

The action of bringing pages of an executing process into physical memory when referenced. When a process executes, all of its pages are said to reside in virtual memory. Only the actively used pages, however, need to reside in physical memory. In this system, a process is paged only when it references more pages than it is allowed to have in its working set. When a process refers to a page not in its working set, a page fault occurs. This causes the operating system's pager to read in the referenced page fault if it is on disk (and optionally, other related pages depending on a cluster factor), replacing the least recently faulted pages as needed. This system only pages a process against itself.

Packed Decimal

A method for compact storage of numeric data; two digits are stored in each 8-bit byte and the sign resides in the last byte of the low-order digit.

Parse

To break down into individual parts from a whole.

Physical Memory

The memory modules contained within the CPU. Also called main memory.

Procedure

A routine that follows the VAX-11 calling sequence standard. A procedure may return values via the argument list and/or the standard value return registers. Contrast with routine.

Process

The basic entity scheduled by the system software that provides the context in which an image executes. A process consists of an address space and both hardware and software context.

Program

A program is the basic entity that is executed by the processor. Each program consists of a set of procedures and its execution represents a distinct activity that is potentially concurrent with others in the system.

Prologue

See File Prologue.

Quadword

Eight contiguous bytes (64 bits) starting on an addressable byte boundary. See Appendix C.

Qualifier

A portion of a command string that modifies a command verb or command parameter by selecting one of several options. A qualifier, if present, follows the command verb or parameter to which it applies and is in the format: "/qualifier=option". For example, in the command string "PRINT filename/ COPIES=3", the COPIES qualifier indicates that the user wants three copies of a given file printed.

Random access by record's file address

The retrieval of a record by its unique address, which is provided to the program by RMS. The method of access can be used to randomly access a sequentially organized file containing variable length records.

Random access by relative record number

The retrieval or storage of a record by specifying its position relative to the beginning of the file.

Real Memory

See Physical Memory.

Record

The unit of information in a file; a group of related fields treated as a logical unit.

Record Cell

A fixed length area in a relatively organized file that is used to contain one record.

Record Management Services (RMS)

A set of system procedures in the operating system that are called by programs to process files and records within files. RMS allows programs to issue GET and PUT requests at the record level (record I/O) as well as read and write blocks (block I/O). RMS is an integral part of the system software. RMS procedures run in executive mode.

Record-oriented Device

A device such as a terminal, line printer, or card reader, on which the largest unit of data that a program can access is the device's physical record.

Record's File Address (RFA)

The unique address of a record in a file that allows records to be accessed randomly regardless of file organization.

Record, Data File

See Data File Record.

Record, Field Specification

See Section 2.5.2

Record, Header

See Section 2.5.2

Record, Index File

See Index File Record.

Relative File Organization

A file organization in which the file contains fixed length record cells. Each cell is assigned a consecutive number that represents its position relative to the beginning of the file. Records within each cell can be the same length or smaller than the cell. Relative file organization permits sequential record access, random record access by record number, and random record access by record's file address.

RMS

See Record Management Services (RMS).

Routine

A sequence of instructions that performs a well defined action. It may have multiple entry points. For example, the SIN routine has SIN and COS entry points. A routine that follows the VAX-11 calling sequence standard is termed a procedure.

Sequential File Organization

A file organization in which records appear in the order in which they were originally written. The records can be fixed length or variable length. Although one does not speak of record cells with sequentially organized files, for purposes of comparison with relatively organized files one can say that the record itself is the same as its record cell, and its record number is the same as its relative cell number. Sequential file organization permits sequential record access and random record access by record's file address. Sequential file organization with fixed length records also permits random access by relative record number.

SORT Utility

A processing program that can be used to sort records by keys into a prescribed sequence. To segregate items into groups according to some definite rules.

Sort Tree

A data structure used to keep order of records by sort.

Subroutine

A procedure that does not return a known value in the value registers. If values are returned, they are returned via the argument list. By convention, the function "value" is unpredictable.

System Device

The device on which the Executive resides.

Terminal

The general name for those peripheral devices that have keyboards and video screen or printers. Under program control, a terminal enables people to type commands and data on a keyboard and receive messages on a video screen or printer. Examples of terminals are the LA36 DECwriter (hard-copy terminal) and the VT52 video display terminal (soft-copy terminal).

Unit Record Device

See Record-oriented Device.

User

The person who is directly using the computer, either via terminal or batch input. Contrast with Caller.

Variable-length Record

A record format in which records need not be the same length.

Variable with fixed-length control (VFC) record

A record format in which records of variable length contain an additional fixed-length control area. The control area may be used to contain file line numbers and/or print format control characters.

Virtual Address

A 32-bit integer identifying a byte "location" in virtual address space. The memory management hardware translates a virtual address to a physical address. The term virtual address may also refer to the address used to identify a virtual block on a mass storage device.

Virtual Memory

The set of storage locations in physical memory and on disk that are referred to by virtual addresses. From the programmer's viewpoint, the secondary storage locations appear to be locations in physical memory. The size of virtual memory in any system depends on the amount of physical memory available and the amount of disk storage used for non-resident virtual memory.

Volume

A mass storage medium such as a disk pack or reel of magnetic tape.

Wild card

The asterisk character when used as a substitute parameter in file specification indicates "all" for a given field.

Word

Two contiguous bytes (16 bits) starting on an addressable byte boundary (see Appendix C).

Work File

A collection of sorted records created during the processing cycle and released after the sort is finished. (Sometimes called Scratch Files.)

Working Set

The set of pages in process address space to which an executing process can refer without incurring a page fault. The working set must be resident in memory for the process to execute. The remaining pages of that process, if any, are either in memory and not in the process working set or they are on secondary storage.

Zoned Numeric Format

A specific ASCII coded decimal data type where the number sign and the least significant digit are combined into a single hexadecimal code (see Appendix C).

Appendix A
Octal/Hexadecimal/Decimal Conversion

A.1 Octal/Decimal Conversion

To convert a number from octal to decimal, locate in each column of the table the decimal equivalent for the octal digit in that position. Add the decimal equivalents to obtain the decimal number.

To convert a decimal number to octal:

1. locate the largest decimal value in the table that will fit into the decimal number to be converted,
2. note its octal equivalent and column position,
3. find the decimal remainder.

Repeat the process on each remainder. When the remainder is 0, all digits will have been generated.

	8^5	8^4	8^3	8^2	8^1	8^0
0	0	0	0	0	0	0
1	32,768	4,096	512	64	8	1
2	65,536	8,192	1,024	128	16	2
3	98,304	12,228	1,536	192	24	3
4	31,072	16,384	2,048	256	32	4
5	163,840	20,480	2,560	320	40	5
6	169,608	24,576	3,072	384	48	6
7	229,376	28,672	3,584	448	56	7

A.2 Powers of 2 and 16

Powers of 2

2**n	n
256	8
512	9
1024	10
2048	11
4096	12
8192	13
16384	14
32768	15
65536	16
131072	17
262144	18
524288	19
1048576	20
2094304	21
4194304	22
8388608	23
16777216	24

Powers of 16

16**n	n
1	0
16	1
256	2
4096	3
65536	4
1048576	5
16777216	6
268435456	7
4294967296	8
68719476736	9
1099511627776	10
17592186044416	11
281474976710656	12
4503599627370496	13
72057594037927936	14
1152921504606846976	15

A.3 Hexadecimal to Decimal Conversion

For each integer position of the hexadecimal value, locate the corresponding column integer and record its decimal equivalent in the conversion table A.5. Add the decimal equivalent to obtain the decimal value.

Example:

```
D0500AD0 (16)  =       ?(10)

D0000000       =  3,489,660,928
  500000       =      5,242,880
     A00       =          2,560
      D0       =            208

D0500AD0       =  3,494,904,576
```

A.4 Decimal to Hexadecimal Conversion

1. Locate in the conversion table A.5 the largest decimal value that does not exceed the decimal number to be converted.
2. Record the hexadecimal equivalent followed by the number of zeros (0) that corresponds to the integer column minus one.
3. Subtract the table decimal value from the decimal number to be converted.
4. Repeat steps 1–3 until the subtraction balance equals zero (0). Add the hexadecimal equivalents to obtain the hexadecimal value.

Example:

```
22,466 (10)        =  ?(16)

20,480             =  5000       22,466
 1,792             =   700      -20,480
   192             =    C0      --------
     2             =     2        1,986
------             =  ----      - 1,792
22,466             =  57C2      --------
                                    194
                                -   192
                                --------
                                      2
                                -     2
                                --------
                                      0
```

A.5 Hexadecimal Integer Columns

8		7		6		5		4		3		2		1	
HEX	**DEC**	**HEX**	**DEC**	**HEX**	**DEC**	**HEX**	**DEC**	**HEX**	**DEC**	**HEX**	**DEC**	**HEX**	**DEC**	**HEX**	**DEC**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	268,435,456	1	16,777,216	1	1,048,576	1	65,536	1	4,096	1	256	1	16	1	1
2	536,870,912	2	33,554,432	2	2,097,152	2	131,072	2	8,192	2	512	2	32	2	2
3	805,306,368	3	50,331,648	3	3,145,728	3	196,608	3	12,288	3	768	3	48	3	3
4	1,073,741,824	4	67,108,864	4	4,194,304	4	262,144	4	16,384	4	1,024	4	64	4	4
5	1,342,177,280	5	83,886,080	5	5,242,880	5	327,680	5	20,480	5	1,280	5	80	5	5
6	1,610,612,736	6	100,663,296	6	6,291,456	6	393,216	6	24,576	6	1,536	6	96	6	6
7	1,879,048,192	7	117,440,512	7	7,340,032	7	458,752	7	28,672	7	1,792	7	112	7	7
8	2,147,483,643	8	134,217,728	8	8,338,608	8	524,288	8	32,768	8	2,048	8	128	8	8
9	2,415,919,104	9	150,994,944	9	9,437,184	9	589,824	9	36,864	9	2,304	9	144	9	9
A	2,684,354,560	A	167,772,160	A	10,485,760	A	655,360	A	40,960	A	2,560	A	160	A	10
B	2,952,790,016	B	184,549,376	B	11,534,336	B	720,896	B	45,056	B	2,916	B	176	B	11
C	3,221,225,472	C	201,326,592	C	12,582,912	C	786,432	C	49,152	C	3,072	C	192	C	12
D	3,489,660,928	D	218,103,808	D	13,631,488	D	851,968	D	53,248	D	3,328	D	208	D	13
E	3,758,096,384	E	234,881,024	E	14,680,064	E	917,504	E	57,344	E	3,584	E	224	E	14
F	4,026,531,840	F	251,658,240	F	15,728,640	F	983,040	F	61,440	F	3,840	F	240	F	15

BYTE BYTE BYTE BYTE

WORD WORD

LONGWORD

Appendix B
The ASCII Character Set Collating Sequence

ASCII Character	Hexadecimal Number	ASCII 8-Bit Octal	Decimal	ASCII Character	Hexadecimal Number	ASCII 8-Bit Octal	Decimal
NUL	00	000	0	FS	1C	034	28
SOH	01	001	1	GS	1D	035	29
STX	02	002	2	RS	1E	036	30
ETX	03	003	3	US	1F	037	31
EOT	04	004	4	SP	20	040	32
ENQ	05	005	5	!	21	041	33
ACK	06	006	6	"	22	042	34
BEL	07	007	7	#	23	043	35
BS	08	010	8	$	24	044	36
HT	09	011	9	%	25	045	37
LF	0A	012	10	&	26	046	38
VT	0B	013	11	′	27	047	39
FF	0C	014	12	(	28	050	40
CR	0D	015	13	)	29	051	41
SO	0E	016	14	*	2A	052	42
SI	0F	017	15	+	2B	053	43
DLE	10	020	16	,	2C	054	44
DC1	11	021	17	–	2D	055	45
DC2	12	022	18	.	2E	056	46
DC3	13	023	29	/	2F	057	47
DC4	14	024	20	0	30	060	48
NAK	15	025	21	1	31	061	49
SYN	16	026	22	2	32	062	50
ETB	17	027	23	3	33	063	51
CAN	18	030	24	4	34	064	52
EM	19	031	25	5	35	065	53
SUB	1A	032	26	6	36	066	54
ESC	1B	033	27	7	37	067	55

(continued on next page)

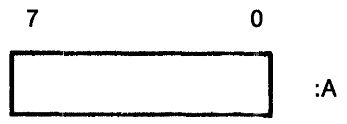
ASCII Character	Hexadecimal Number	ASCII 8-Bit Octal	Decimal	ASCII Character	Hexadecimal Number	ASCII 8-Bit Octal	Decimal
8	38	070	56	\	5C	134	92
9	39	071	57	]	5D	135	93
:	3A	072	58	∧	5E	136	94
;	3B	073	59	_	5F	137	95
<	3C	074	60	`	60	140	96
=	3D	075	61	a	61	141	97
>	3E	076	62	b	62	142	98
?	3F	077	63	c	63	143	99
@	40	100	64	d	64	144	100
A	41	101	65	e	65	145	101
B	42	102	66	f	66	146	102
C	43	103	67	g	67	147	103
D	44	104	68	h	68	150	104
E	45	105	69	i	69	151	105
F	46	106	70	j	6A	152	106
G	47	107	71	k	6B	153	107
H	48	110	72	l	6C	154	108
I	49	111	73	m	6D	155	109
J	4A	112	74	n	6E	156	110
K	4B	113	75	o	6F	157	111
L	4C	114	76	p	70	160	112
M	4D	115	77	q	71	161	113
N	4E	116	78	r	72	162	114
O	4F	117	79	s	73	163	115
P	50	120	80	t	74	164	116
Q	51	121	81	u	75	165	117
R	52	122	82	v	76	166	118
S	53	123	83	w	77	167	119
T	54	124	84	x	78	170	120
U	55	125	85	y	79	171	121
V	56	126	86	z	7A	172	122
W	57	127	87	{	7B	173	123
X	58	130	88	|	7C	174	124
Y	59	131	89	}	7D	175	125
Z	5A	132	90	~	7E	176	126
[	5B	133	91	DEL	7F	177	127

Appendix C
Data Types

The *data type* refers to the way in which *bits* are grouped and interpreted. In reference to the processor instructions, the data type of an operand identifies the size of the operand and the significance of the bits in the operand. Data types applicable to SORT and its associated VAX/VMS programs are: separated into three classes; character, binary, and decimal. These classes can be subdivided into data types of different sizes and formats such as; byte, word, longword, quadword, floating, double floating, character string, packed decimal string, and variable-length bit field.

C.1 Byte

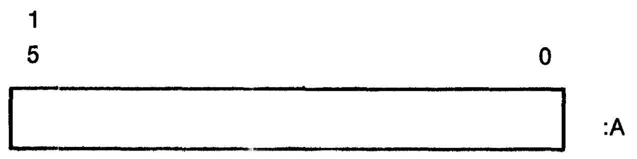
A *byte* is 8 contiguous bits starting on an addressable byte boundary. The bits are numbered from the right 0 through 7:



A byte is specified by its address A. When interpreted arithmetically, a byte is a twos complement integer with bits of increasing significance going 0 through 6 and bit 7 the sign bit. The value of the integer is in the range -128 through 127. For the purposes of addition, subtraction, and comparison, VAX-11 instructions also provide direct support for the interpretation of a byte as an unsigned integer with bits of increasing significance going 0 through 7. The value of the unsigned integer is in the range 0 through 255.

C.2 Word

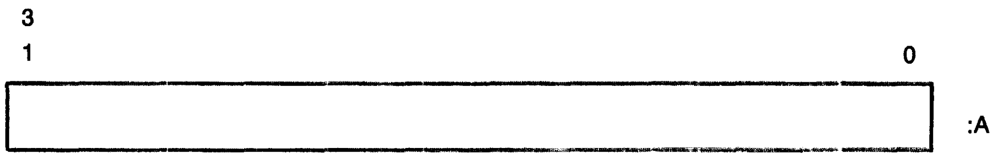
A *word* is 2 contiguous bytes starting on an arbitrary byte boundary. The bits are numbered from the right 0 through 15:



A word is specified by its address A, the address of the byte containing bit 0. When interpreted arithmetically, a word is a twos complement integer with bits of increasing significance going 0 through 14 and bit 15 the sign bit. The value of the integer is in the range -32,768 through 32,767. For the purposes of addition, subtraction and comparison, VAX-11 instructions also provide direct support for the interpretation of a word as an unsigned integer with bits of increasing significance going 0 through 15. The value of the unsigned integer is in the range 0 through 65,535.

C.3 Longword

A *longword* is 4 contiguous bytes starting on an arbitrary byte boundary. The bits are numbered from the right 0 through 31:

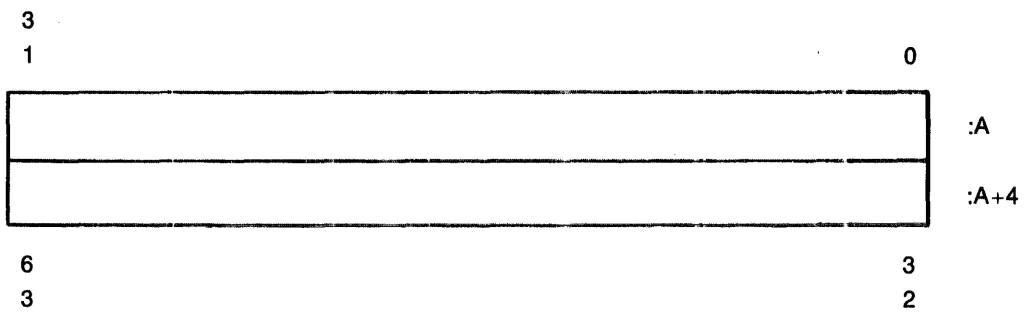


A longword is specified by its address A, the address of the type containing bit 0. When interpreted arithmetically, a longword is a twos complement integer with bits of increasing significance going 0 through 30 and bit 31 the sign bit. The value of the integer is in the range -2,147,483,648 through 2,147,483,647. For the purposes of addition, subtraction, and comparison, VAX-11 instructions also provide direct support for the interpretation of a longword as an unsigned integer with bits of increasing significance going 0 through 31. The value of the unsigned integer is in the range 0 through 4,294,967,295.

Note that the longword format is different from the longword format defined by the PDP-11 FP-11. In that format, bits of increasing significance go from 16 through 31 and 0 through 14. Bit 15 is the sign bit. Most DIGITAL software and in particular PDP-11 FORTRAN and COBOL use the VAX-11 longword format.

C.4 Quadword

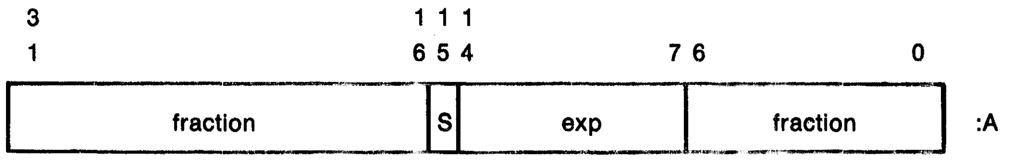
A *quadword* is 8 contiguous bytes starting on an arbitrary byte boundary. The bits are numbered from the right 0 through 63:



A quadword is specified by its address A, the address of the byte containing bit 0. When interpreted arithmetically, a quadword is a twos complement integer with bits of increasing significance going 0 through 62 and bit 63 the sign bit. The value of the integer is in the range -2**63 to 2**63-1. The quadword data type is not fully supported by VAX-11 instructions.

C.5 Floating

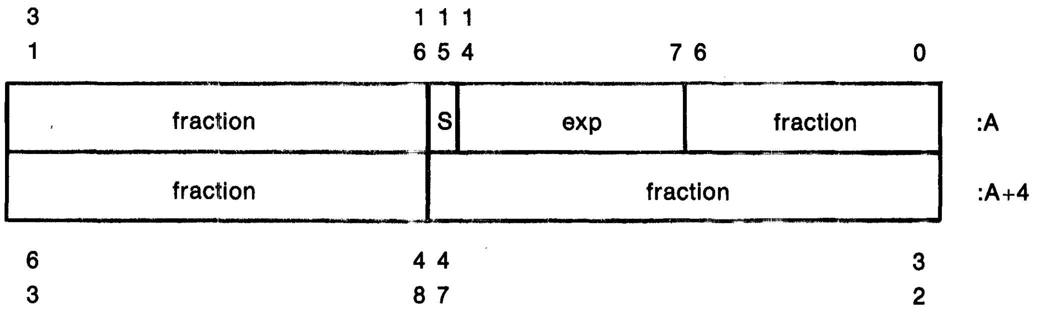
A *floating* datum is 4 contiguous bytes starting on an arbitrary byte boundary. The bits are labelled from the right 0 through 31.



A floating datum is specified by its address A, the address of the byte containing bit 0. The form of a floating datum is sign magnitude with bit 15 the sign bit, bits 14:7 an excess 128 binary exponent, and bits 6:0 and 31:16 a normalized 24-bit fraction with the redundant most significant fraction bit not represented. Within the fraction, bits of increasing significance go from 16 through 31 and 0 through 6. The 8-bit exponent field encodes the values 0 through 255. An exponent value of 0 together with a sign bit of 0, is taken to indicate that the floating datum has a value of 0. Exponent values of 1 through 255 indicate true binary exponents of −127 through +127. An exponent value of 0, together with a sign bit of 1, is taken as reserved. Floating point instructions processing a reserved operand take a reserved operand fault (See Chapter 4 and 6). The value of a floating datum is in the approximate range .29*10**−38 through 1.7*10**38. The precision of a floating datum is approximately one part in 2**23, that is, typically 7 decimal digits.

C.6 Double Floating

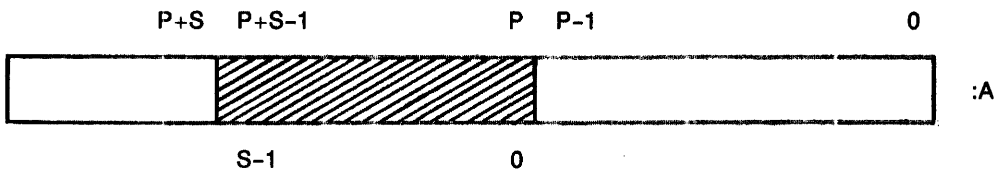
A *double floating* datum is 8 contiguous bytes starting on an arbitrary byte boundary. The bits are labelled from the right 0 through 63:



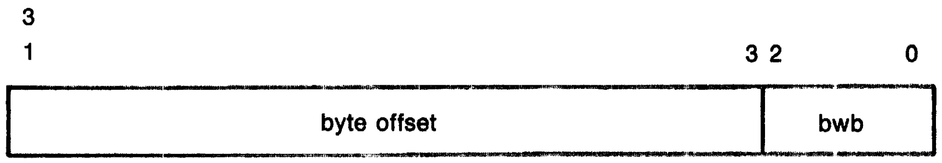
A double floating datum is specified by its address A, the address of the byte containing bit 0. The form of a double floating datum is identical to a floating datum except for an additional 32 low significance fraction bits. Within the fraction, bits of increasing significance go 48 through 63, 32 through 47, 16 through 31, and 0 through 6. The exponent conventions, and approximate range of values is the same for double floating as floating. The precision of a double floating datum is approximately one part in 2**55, that is, typically 16 decimal digits.

C.7 Variable Length Bit Field

A *variable bit field* is 0 to 32 contiguous bits located arbitrarily with respect to byte boundaries. A variable bit field is specified by 3 attributes: the address A of a byte, a bit position P which is the starting location of the field with respect to bit 0 of the byte at A, and a size S of the field. The specification of a bit field is indicated by the following where the field is the shaded area.



The position is in the range -2**31 through 2**31-1 and is conveniently viewed as a signed 29-bit offset and a 3-bit bit-within-byte (BWB) field:

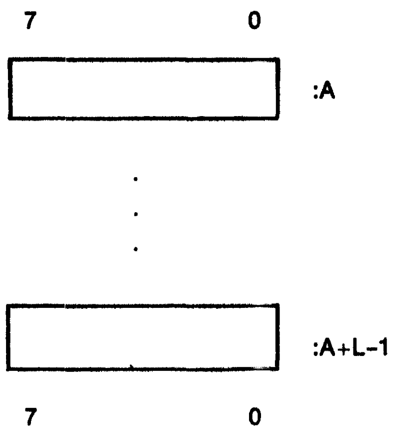


The sign extended 29-bit byte offset is added to the address A and the resulting address specifies the byte in which the field begins. The 3-bit bit-within-byte field encodes the starting position (0 through 7) of the field within that byte. The VAX-11 field instructions provide direct support for the interpretation of a field as a signed or unsigned integer. When interpreted as a signed integer, it is twos complement with bits of increasing significance going 0 through S-2; bit S-1 is the sign bit. When interpreted as an unsigned integer, bits of increasing significance go from 0 to S-1. A field of size 0 has a value identically equal to 0.

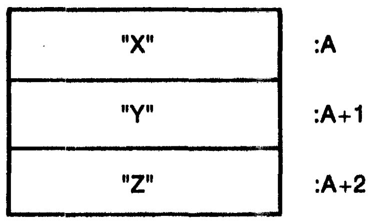
A variable bit field may be contained in 1 to 5 bytes. From a memory management point of view, only the minimum number of bytes necessary to contain the field is actually referenced.

C.8 Character String

A *character string* is a contiguous sequence of bytes in memory. A character string is specified by 2 attributes: the address A of the first byte of the string, and the length L of the string in bytes. Thus the format of a character string is:



The address of a string specifies the first character of a string. Thus "XYZ" is represented:



The length L of a string is in the range 0 through 65,535.

C.9 Trailing Numeric String

A *trailing numeric string* is a contiguous sequence of bytes in memory. The string is specified by 2 attributes: the address A of the first byte (most significant digit) of the string, and the length L of the string in bytes.

All bytes of a trailing numeric string, except the least significant digit byte, must contain an ASCII decimal digit character (0–9). The representation for the high order digits is:

digit	decimal	hex	ASCII character
0	48	30	0
1	49	31	1
2	50	32	2
3	51	33	3
4	52	34	4
5	53	35	5
6	54	36	6
7	55	37	7
8	56	38	8
9	57	39	9

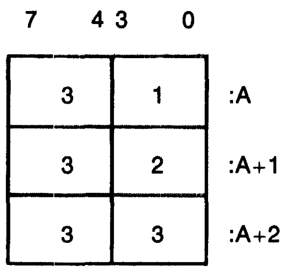
The highest addressed byte of a trailing numeric string represents an encoding of both the least significant digit and the sign of the numeric string. The VAX numeric string instructions support any encoding; however, there are 3 preferred encodings used by DIGITAL software. These are (1) unsigned numeric in which there is no sign and the least significant digit contains an ASCII decimal digit character, (2) zoned numeric, and (3) overpunched numeric. Because the overpunch format has been used by compilers of many manufacturers over many years, and because various card encodings are used, several variations in overpunch format have evolved. Typically, these alternate forms are accepted on input. The valid representations of the digit and sign in each of the later two formats is:

Representation of Least Significant Digit and Sign

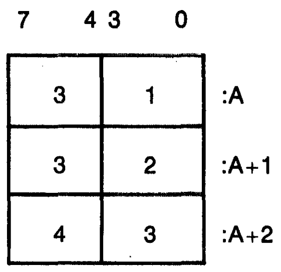
	Zoned Numeric Format			**Overpunch Format**			
			ASCII			**ASCII char**	
digit	**decimal**	**hex**	**char**	**decimal**	**hex**	**norm**	**alt.**
0	48	30	0	123	7B	{	[?
1	49	31	1	65	41	A	a
2	50	32	2	66	42	B	b
3	51	33	3	67	43	C	c
4	52	34	4	68	44	D	d
5	53	35	5	69	45	E	e
6	54	36	6	70	46	F	f
7	55	37	7	71	47	G	g
8	56	38	8	72	48	H	h
9	57	39	9	73	49	I	i
-0	112	70	p	125	7D	}	] ! :
-1	113	71	q	74	4A	J	j
-2	114	72	r	75	4B	K	k
-3	115	73	s	76	4C	L	l
-4	116	74	t	77	4D	M	m
-5	117	75	u	78	4E	N	n
-6	118	76	v	79	4F	O	o
-7	119	77	w	80	50	P	p
-8	120	78	x	81	51	Q	q
-9	121	79	y	82	52	R	r

The length L of a trailing numeric string must be in the range 0 to 31 (0 to 31 digits). The value of a 0 length string is identically 0. The address A of the string specifies the byte of the string containing the most significant digit. Digits of decreasing significance are assigned to increasing addresses. Thus "123" is represented:

Zoned Format or Unsigned

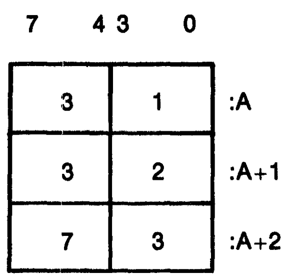


Overpunch Format

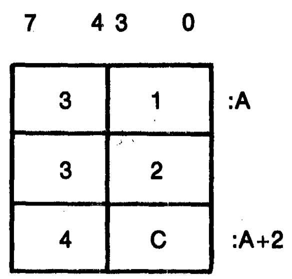


and "-123" is represented:

Zoned Format



Overpunch Format



C.10 Leading Separate Numeric String

A *leading separate numeric string* is a contiguous sequence of bytes in memory. A leading separate numeric string is specified by 2 attributes: the address A of the first byte (containing the sign character), and a length L, which is the length of the string in digits and NOT the length of the string in bytes. The number of bytes in a leading separate numeric string is L+1.

The sign of a separate leading numeric string is stored in a separate byte. Valid sign bytes are:

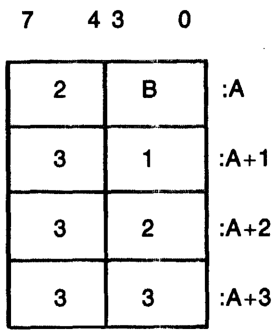
Sign	decimal	hex	ASCII character
+	43	2B	+
+	32	20	<blank>
–	45	2D	–

The preferred representation for "+" is ASCII "+". All subsequent bytes contain an ASCII digit character:

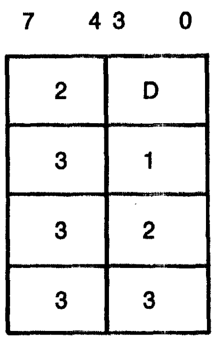
digit	decimal	hex	ASCII character
0	48	30	0
1	49	31	1
2	50	32	2
3	51	33	3
4	52	34	4
5	53	35	5
6	54	36	6
7	55	37	7
8	56	38	8
9	57	39	9

The length L of a leading separate numeric string must be in the range 0 to 31 (0 to 31 digits). The value of a 0 length string is identically 0.

The address A of the string specifies the byte of the string containing the sign. Digits of decreasing significance are assigned to bytes of increasing addresses. Thus "+123" is:



and "-123" is:



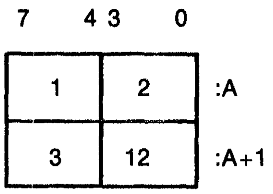
C.11 Packed Decimal String

A *packed decimal string* is a contiguous sequence of bytes in memory. A packed decimal string is specified by 2 attributes: the address A of the first byte of the string and a length L which is the number of digits in the string and NOT the length of the string in bytes. The bytes of a packed decimal string are divided into 2 4-bit fields (nibbles) which must contain decimal digits except the low nibble (bits 3:0) of the last (highest addressed) byte which must contain a sign. The representation for the digits and sign is:

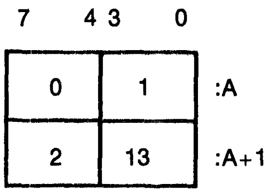
digit or sign	decimal	hex
0	0	0
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	8	8
9	9	9
+	10,12,14, or 15	A,C,E, or F
–	11 or 13	B, or D

The preferred sign representation is 12 for "+" and 13 for "-". The length L is the number of digits in the packed decimal string (not counting the sign) and must be in the range 0 through 31. When the number of digits is odd, the digits and the sign fit in L/2 (integer part only) + 1 bytes. When the number of digits is even, it is required that an extra "0" digit appear in the high nibble (bits 7:4) of the first byte of the string. Again the length in bytes of the string is L/2 + 1.

The address A of the string specifies the byte of the string containing the most significant digit in its high nibble. Digits of decreasing significance are assigned to increasing byte addresses and from high nibble to low nibble within a byte. Thus "+123" has length 3 and is represented:



and "-12" has length 2 and is represented:



Appendix D
Data Structures and Basic Concepts

This Appendix provides beginning users with additional information regarding the following topics:

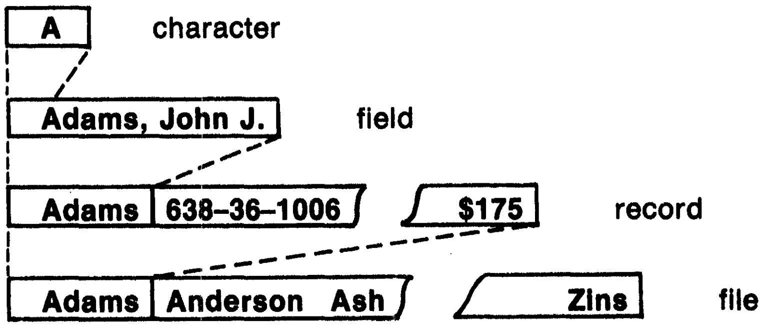
- VAX-11 RMS data files, records, and structures
- File specification parameters
- Programming languages

D.1 Data Files, Records, and Structures

D.1.1 Data Hierarchy

Four level data hierarchy (character, field, record, file) is shown in Figure D.1:

Figure D.1: Data Hierarchy

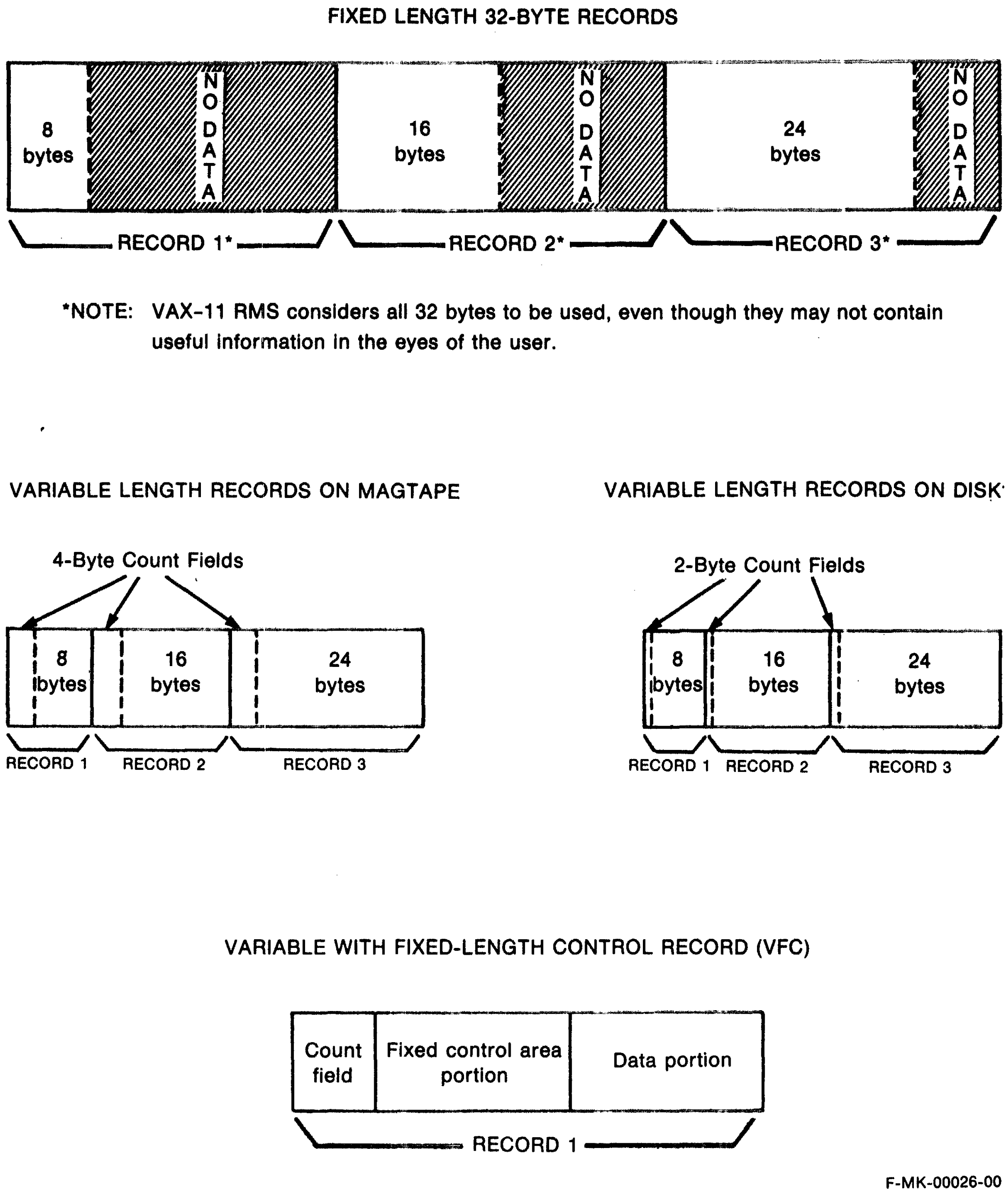


A *file* is a collection of related information. For example, a file might contain a company's personnel information (employee names, addresses, and job titles). Within this file, the information is divided into *records*. All the information on a single employee might constitute a single record. Each record in the personnel file would be subdivided into discrete pieces of information known as *fields*. By specifying *key fields* (*Keys*) in a particular order, you can sort entire records into any order. Using VAX-11 SORT you can retrieve records in ascending or descending order by ordered key fields (that is, create sorted data files); and you can create sorted address files for random record retrieval by user programs.

D.1.2 Record Types

VAX-11 SORT processes three different types of records: Fixed, Variable, and Variable with fixed-length control (VFC). Figure D.2 summarizes these record types.

Figure D.2: Record Types (fixed, variable, VFC)


FIXED LENGTH 32-BYTE RECORDS
8 bytes
NO DATA
16 bytes
NO DATA
24 bytes
NO DATA
RECORD 1*
RECORD 2*
RECORD 3*
*NOTE: VAX-11 RMS considers all 32 bytes to be used, even though they may not contain useful information in the eyes of the user.
VARIABLE LENGTH RECORDS ON MAGTAPE
4-Byte Count Fields
8 bytes
16 bytes
24 bytes
RECORD 1
RECORD 2
RECORD 3
VARIABLE LENGTH RECORDS ON DISK
2-Byte Count Fields
8 bytes
16 bytes
24 bytes
RECORD 1
RECORD 2
RECORD 3
VARIABLE WITH FIXED-LENGTH CONTROL RECORD (VFC)
Count field
Fixed control area portion
Data portion
RECORD 1
F-MK-00026-00

D.1.3 VAX–11 RMS File Organizations

1. **Sequential Files** (see Figure D.3) Sequential files may contain the following record types:

   a. Fixed Length Records

   b. Variable Length Records

   c. Variable with Fixed-length Control (VFC) Records

   The order of the records in a sequential file is determined by the order in which the records were originally written to the file. The first record in the file is the first record read; the second record next, and so on.*Sequential files are the only files permitted for magnetic tape and unit record devices.* They are also permitted for disk.

2. **Relative Files** – Records may be any type (that is, fixed, variable, or VFC) as long as the maximum record length is specified. Each record is numbered 1 to n relative to the beginning of file (as shown in Figure D.4).

   A relative file consists of record areas (cells) that are identified by relative record numbers. The first record area in the file is record number 1, the second is 2, and so on. Empty or null records are permitted. *Relative files can reside only on disk.*

   Relative file considerations:

   - Most efficient random access in terms of speed and storage space overhead.
   - Addresses of records (relative record numbers) must be known to process file randomly.
   - Requires storage space to contain all record positions from record number one to highest record number stored in file.
   - Records can span blocks, but cannot span buckets.
   - Can be write shared.

3. **Indexed Files** – Contain one or more indices, as well as data records. Records can be of any type (that is, fixed, variable, or VFC) as long as the maximum record length is specified. To retrieve information, you ask for the proper record by primary or alternate key. The system looks up the key in the appropriate index and retrieves the record using the record pointer associated with the key. *Indexed files can reside only on disk.*

   The location of records in the indexed file organization is transparent to the program. RMS completely controls the placement of records in an indexed file. The presence of keys in the records of the file governs this placement.

A key is a field present in every record of an indexed file. The location and length of this field are identical in all records. When creating an indexed file, the user decides which field or fields in the file's records are to be a key. Selecting such fields indicates to RMS that the contents (that is, key value) of those fields in any particular record written to the file can be used by a program to identify that record for subsequent retrieval.

At least one key must be defined for an indexed file: the primary key. Optionally, additional keys or alternate keys can be defined. An alternate key value can also be used as a means of identifying a record for retrieval.

As programs write into an indexed file, RMS builds a tree-structured table known as an index. An index consists of a series of entries containing a key value copied from a record that a program wrote into the file. Stored with each key value is a pointer to the location in the file of the record from which the value was copied. RMS builds and maintains a separate index for each key defined for the file. Each index is stored in the file. Thus, every indexed file contains at least one index, the primary key index. Figure D.5 shows an RMS indexed file organization with a primary key. When alternate keys are defined, RMS builds and stores an additional index for each alternate key.

Index file considerations:

- Multi-key indexed sequential capability.
- Most flexible in terms of how a record is accessed.
- A record is addressed by the contents of a field in the record (the key field).
- Records can be retrieved sequentially in a collated order by key field.
- Requires the most storage overhead (that is, the RMS index tree structure).
- Index records consist of block numbers, byte-in-block numbers and key.
- Can be write shared.

All VAX-11 RMS files have two additional blocks in the directory. These additional blocks contain information relating the type of RMS file and the record length.

Figure D.3: Sequential Files

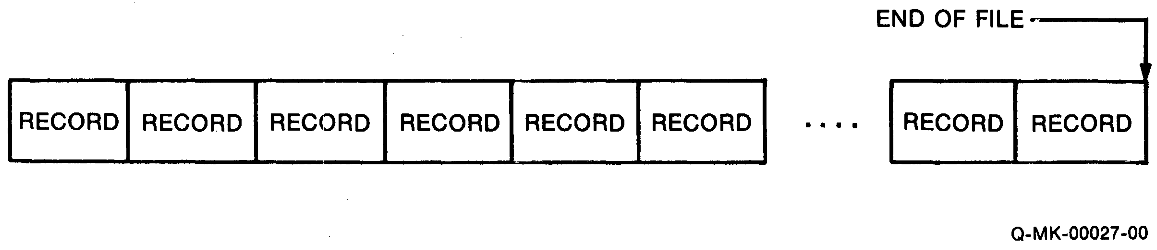


Figure D.4: Relative Files

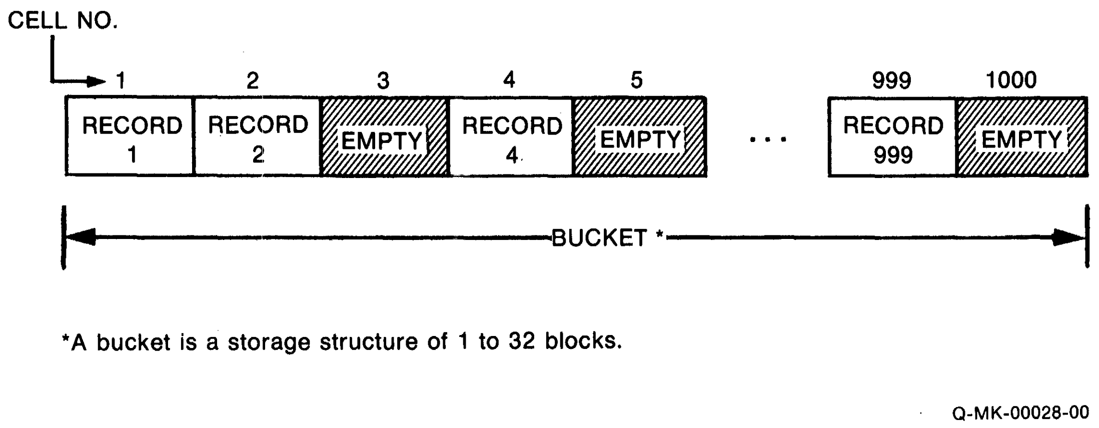
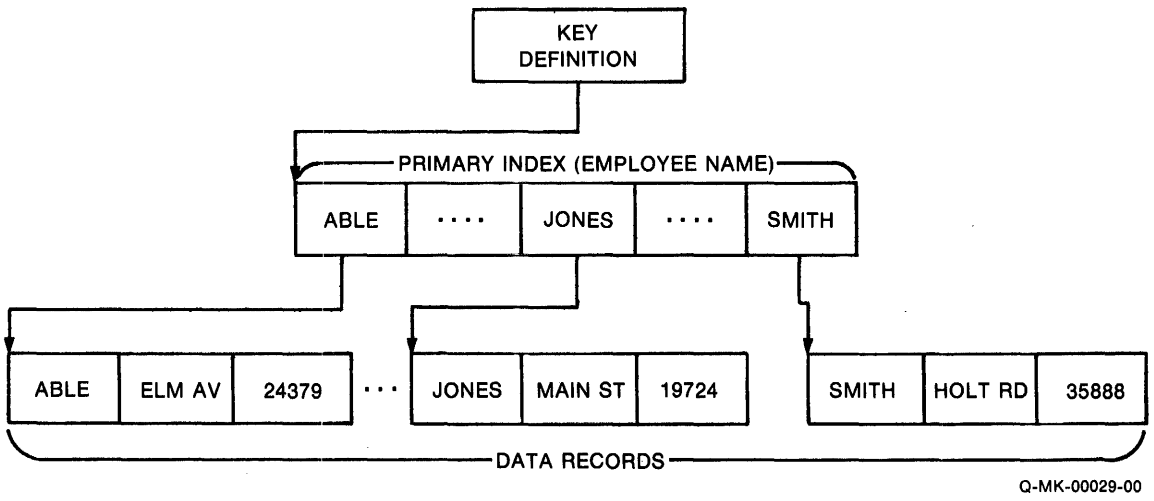


Figure D.5: Indexed Files



D.2 Input and Output File Specification

An input or output file specification uniquely identifies a file by indicating its physical location and a directory in which it is cataloged, as well as providing a unique filename for that particular file within the directory. However, it is not necessary to supply the physical location and directory for the file, since the system uses the defaults set up during the log-in procedure when these components are omitted from a file specification.

This section is only a summary. For a full description of defaults, wild cards, logical names, and subdirectories, see the *VAX/VMS Command Language User's Guide*.

The format of a file specification representing a physical file or device is:

```
node-name::device-name:[directory]filename.file-type;file-version
```

Node-name:: The individual computer system (or node) name within a network consists of 1–6 alphanumeric characters.

Example: BOSTON::

Device-name: The device name consists of three components: device type [controller] [unit-number]:

The maximum length of the device type and controller specification is 15 characters. The maximum unit number is 65535. The default value for controller is A, and the default value for unit is 0.

Physical device names are:

Mnemonic	**Device**
CR	Card Reader
DB	RP04, RP05, RP06 Disk
DM	RK06 Disk
DR	RM03 Disk
DX	Floppy Disk
LP	Line Printer
MB	Mailbox
MT	TE16 Magnetic Tape
NET	Network Communication Device
TT	Interactive Terminal
XM	DMC-11

Example: DB: is actually device name DBA0: by default.

[directory] The directory name or names must be inclosed in either square brackets ([]) or angle brackets (<>). A directory without a directory name (for example, [ ]) is not valid. The directory types are:

- A 1- to 9-alphanumeric character string
- A two-part number in the format of a user identification code (UIC)

- As subdirectories, in the format of name.name.name where each name can consist of up to 9 alphanumberic characters; each name represents a diretory level.

Filename.	The file name is limited to nine ASCII characters.
File type	The file type is limited to three ASCII characters. Some commonly used file types are:

File type	Contents
B2S	Input source file for the PDP-11 BASIC-PLUS-2/VAX compiler
CMD	Compatibility mode indirect command file
COM	Command procedure file to be executed with the @ (Execute Procedure) command, or to be submitted for batch execution with the SUBMIT command
COB	Input file containing source statements for the PDP-11 COBOL-74/VAX compiler
DAT*	Input or output data file
DIF	Output listing created by the DIFFERENCES command
DIR	Directory file
DMP	Output listing created by the DUMP command
EXE	Executable program image
FOR	Input file containing source statements for the VAX-11 FORTRAN-IV-PLUS compiler
LIB	Library file
LIS	Listing file created by a language compiler or assembler; default input file type for PRINT and TYPE commands
LOG	Batch job output file
LST	Compatibility mode listing file
MAC	MACRO-11 source file
MAP	Memory allocation map created by the linker
MAR	VAX -11 MACRO source file
MLB	Macro library
OBJ	Object file created by a language compiler or assembler
ODL	Overlay description file
OLB	Object module library
OPT	Options file for input to the LINK command
STB	Symbol table file created by the linker
TSK	Compatibility mode task image

;File-version	The file version number is automatically updated by the system each time the file is changed. Commands may optionally use a period to delimit the file version number, but the documentation will use a semicolon.

* indicates default file type for input files. Default file type for output files is whatever the input file type is.

D.3 Programming Languages Supported

The following compilers produce native mode programs that can use VAX-11 SORT:

- VAX-11 FORTRAN IV-PLUS
- VAX-11 MACRO
- VAX-11 BLISS
- VAX-11 COBOL-74

VAX-11 FORTRAN IV-PLUS

FORTRAN IV-PLUS is an especially complete version of the leading language for scientific and engineering computation. It is a high-performance superset of the American National Standard Institute's (ANSI) 1966 FORTRAN. It also implements many of the anticipated features of the forthcoming ANSI standard.

FORTRAN IV-PLUS supports character data types, an IF-THEN-ELSE statement, long variable names, and the standard CALL facility for calling system services.

The FORTRAN IV-PLUS compiler first optimizes user source code, then translates it to take advantage of the VAX-11 instruction set, which can compile whole FORTRAN IV-PLUS statements into single instructions. An interactive symbolic debugger allows source-level debugging of FORTRAN IV-PLUS programs.

VAX-11 MACRO

The VAX-11 MACRO assembly language allows the programmer to write 32-bit machine language instructions for special efficiency. The symbolic debugger can also be used with VAX-11 MACRO.

VAX-11 BLISS

BLISS is a medium level language designed for building system software; such as compilers, real-time processors, and utilities.

VAX-11 COBOL-74

The VAX-11 COBOL-74 language is based on the 1974 ANSI standard.

Index

VAX-11
SORT
User's Guide
Order No. AA-D113A-TE

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify) ______

Name ______ Date ______

Organization ______

Street ______

City ______ State ______ Zip Code or Country ______

— — — Do Not Tear - Fold Here and Tape —

digital

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

RT/C SOFTWARE PUBLICATIONS TW/A14
DIGITAL EQUIPMENT CORPORATION
1925 ANDOVER STREET
TEWKSBURY, MASSACHUSETTS 01876

— — Do Not Tear - Fold Here — — — — — — — — — — — — — — — — — — —