

HP 3000 Computer System

Sort/3000

Reference Manual



HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

HP 3000 Series II Computer System



Sort/3000

Reference Manual



5303 STEVENS CREEK BLVD., SANTA CLARA, CALIFORNIA, 95050

NOTICE

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

This document contains proprietary information which is protected by copyright. All rights are reserved. No part of this document may be photocopied, reproduced or translated to another program language without the prior written consent of Hewlett-Packard Company.

LIST OF EFFECTIVE PAGES

The List of Effective Pages gives the most recent date on which the technical material on any given page was altered. If a page is simply re-arranged due to a technical change on a previous page, it is not listed as a changed page. Within the manual, changes are marked with a vertical bar in the margin.

Title	Aug 1976	3-16	Apr 1975
ii to v	Jun 1976	3-17	Jun 1976
vi	Apr 1975	3-18	Apr 1975
1-1 to 1-3	Apr 1975	3-19	Jun 1976
1-4	Jun 1976	3-20	Apr 1975
2-1	Jun 1976	3-21	Jun 1976
2-2 to 2-3	Apr 1975	3-22	Apr 1975
2-4 to 2-6	Jun 1976	3-23	Jun 1976
2-7	Apr 1975	3-24 to 3-25	Apr 1975
2-8	Jun 1976	4-1	Jun 1976
2-9 to 2-11	Apr 1975	4-2	Apr 1975
3-1	Jun 1976	4-3	Jun 1976
3-2 to 3-4	Apr 1975	4-4 to 4-7	Apr 1975
3-5 to 3-6	Jun 1976	5-1	Jun 1976
3-7	Apr 1975	5-2 to 5-7	Apr 1975
3-8	Jun 1976	6-1 to 6-2	Apr 1975
3-9 to 3-12	Apr 1975	6-3	Jun 1976
3-13	Jun 1976	6-4 to 6-6	Apr 1975
3-14	Apr 1975	A-1	Apr 1975
3-15	Jun 1976	I-1 to I-3	Jun 1976

PRINTING HISTORY

New editions incorporate all update material since the previous edition. Update packages, which are issued between editions, contain additional and replacement pages to be merged into the manual by the customer. The date on the title page and back cover changes only when a new edition is published. If minor corrections and updates are incorporated, the manual is reprinted but neither the date on the title page and back cover nor the edition change.

First edition April 1975
Update Package #1 June 1976
Second edition August 1976

This publication is the reference manual for SORT/3000. SORT/3000 is a subsystem of the MPE/3000 Operating System and consists of two programs: a Sort program and a Merge program. The SORT/3000 subsystem *sorts* a file of records or *merges* multiple files of sorted records into a single file.

The Sort and Merge programs can be run as stand-alone programs controlled by direct user commands, or they can be called from user programs. Examples are provided throughout this manual which demonstrate how to run Sort and Merge as stand-alone programs and how to call them from SPL/3000 (Systems Programming Language for the HP 3000 Computer System) and FORTRAN/3000 (a version of FORTRAN IV for the HP 3000 Computer System). (The COBOL programmer uses the COBOL SORT verb to run SORT/3000.)

The content of this publication is:

Section I
introduces the SORT/3000 subsystem. The concepts of sorting and merging files are discussed, and the basic structure of SORT/3000 is explained.

Section II
provides instructions for executing the Sort program as a stand-alone program. Examples are provided which demonstrate running the Sort program in interactive and batch modes.

Section III
explains how to call Sort intrinsics from SPL/3000 and FORTRAN/3000 programs. Also provided are definitions of the Sort intrinsics, a discussion of the HP 3000 hardware data stack, and complete, operating programs in SPL/3000 and FORTRAN/3000.

Section IV
provides instructions for executing the Merge program as a stand-alone program. Examples are provided which demonstrate running the Merge program in interactive and batch modes.

Section V
explains how to call Merge intrinsics from SPL/3000 and FORTRAN/3000 programs. Operating programs are used as examples.

Section VI
provides error messages and recovery procedures.

Appendix A
contains a table of ASCII characters.

Index
contains an alphabetical listing of the main topics of the manual.

The following manuals are available for reference:

MPE Intrinsics Reference Manual (30000-90010)
MPE Commands Reference Manual (30000-90011)
FORTRAN Reference Manual (30000-90040)
System Programming Language Reference Manual (30000-90024)
COBOL/3000 Reference Manual (32213-90001)
System Reference Manual (30000-90020)

PREFACE (continued)

For 3000 systems which are not Series II, the following differences in manuals should be noted:

- Whenever the *MPE Commands Reference Manual* is referenced in this manual, use the *MPE/3000 Operating System Reference Manual* (32000-90002).
- Whenever the *FORTTRAN Reference Manual* is referenced in this manual, use the *FORTTRAN/3000 Reference Manual* (32102-90001).
- Whenever the *System Reference Manual* is referenced in this manual, use the *SPL/3000 Reference Manual* (03000-90002).
- Whenever the *System Reference Manual* is referenced in this manual, use the *HP 3000 Computer System, Reference Manual* (03000-90019).

<p>Section I Page</p> <p>INTRODUCING SORT/3000</p> <p>What is SORT/3000 1-1</p> <p>Purposes of SORT/3000 1-1</p> <p>Files, Records, and Data Fields 1-1</p> <p>Keys 1-2</p> <p>Sorting Order 1-2</p> <p>Inputs to SORT/3000 1-4</p> <p>Outputs from SORT/3000 1-4</p> <p>Section II Page</p> <p>RUNNING THE SORT PROGRAM AS A STAND-ALONE PROGRAM</p> <p>File Definitions 2-1</p> <p style="padding-left: 20px;">INPUT File 2-1</p> <p style="padding-left: 20px;">OUTPUT File 2-1</p> <p style="padding-left: 40px;">Output Options 2-2</p> <p style="padding-left: 20px;">LIST File 2-2</p> <p style="padding-left: 20px;">TEXT File 2-3</p> <p style="padding-left: 20px;">PROMPT File 2-3</p> <p style="padding-left: 20px;">SCRATCH File 2-3</p> <p>Sort Commands 2-3</p> <p style="padding-left: 20px;">INPUT Command 2-3</p> <p style="padding-left: 20px;">OUTPUT Command 2-3</p> <p style="padding-left: 20px;">KEY Command 2-4</p> <p style="padding-left: 20px;">RESET Command 2-5</p> <p style="padding-left: 20px;">VERIFY Command 2-5</p> <p style="padding-left: 20px;">END Command 2-6</p> <p>Statistics 2-6</p> <p>Control Y 2-6</p> <p>Running the Stand-Alone Sort Program in Interactive Session 2-6</p> <p>Running the Stand-Alone Sort Program in Batch Mode 2-10</p> <p>Section III Page</p> <p>RUNNING THE SORT PROGRAM FROM A USER PROGRAM</p> <p>SORTINITIAL Procedure 3-1</p> <p style="padding-left: 20px;">INPUTFILE Parameter 3-1</p> <p style="padding-left: 20px;">OUTPUTFILE Parameter 3-1</p> <p style="padding-left: 20px;">OUTPUTOPTION Parameter 3-2</p> <p style="padding-left: 20px;">RECLN Parameter 3-2</p> <p style="padding-left: 20px;">NUMRECS Parameter 3-2</p> <p style="padding-left: 20px;">NUMKEYS and KEYS Parameters 3-2</p> <p style="padding-left: 20px;">ERRORPROC Parameter 3-2</p> <p style="padding-left: 20px;">KEYCOMPARE Parameter 3-2</p> <p style="padding-left: 20px;">STATISTICS Parameter 3-3</p> <p style="padding-left: 20px;">FAILURE Parameter 3-3</p> <p>SORTINPUT Procedure 3-3</p> <p>SORTOUTPUT Procedure 3-3</p> <p>SORTEND Procedure 3-4</p> <p>SORTERRORMESS Procedure 3-4</p> <p>SORTTITLE Procedure 3-4</p>	<p>Calling the Sort Program from SPL/3000 3-4</p> <p>Treatment of the Stack 3-6</p> <p>Section IV Page</p> <p>RUNNING THE MERGE PROGRAM AS A STAND-ALONE PROGRAM</p> <p>File Definitions 4-1</p> <p style="padding-left: 20px;">INPUT Files 4-1</p> <p style="padding-left: 20px;">OUTPUT File 4-1</p> <p style="padding-left: 20px;">LIST File 4-2</p> <p style="padding-left: 20px;">TEXT File 4-2</p> <p style="padding-left: 20px;">PROMPT File 4-2</p> <p>Merge Commands 4-2</p> <p style="padding-left: 20px;">INPUT Command 4-2</p> <p style="padding-left: 20px;">OUTPUT Command 4-2</p> <p style="padding-left: 20px;">KEY Command 4-3</p> <p style="padding-left: 20px;">RESET Command 4-3</p> <p style="padding-left: 20px;">VERIFY Command 4-3</p> <p style="padding-left: 20px;">END Command 4-4</p> <p>Statistics 4-4</p> <p>Control Y 4-4</p> <p>Running the Stand-Alone Merge Program in Interactive Session 4-4</p> <p>Running the Stand-Alone Merge Program in Batch Mode 4-4</p> <p>Section V Page</p> <p>RUNNING THE MERGE PROGRAM FROM A USER PROGRAM</p> <p>MERGE Procedure 5-1</p> <p style="padding-left: 20px;">NUMINPUTFILES Parameter 5-1</p> <p style="padding-left: 20px;">INPUTFILES Parameter 5-1</p> <p style="padding-left: 20px;">OUTPUTFILE Parameter 5-1</p> <p style="padding-left: 20px;">KEYSONLY Parameter 5-1</p> <p style="padding-left: 20px;">NUMKEYS and KEYS Parameters 5-1</p> <p style="padding-left: 20px;">PREPROCESSOR Parameter 5-2</p> <p style="padding-left: 20px;">POSTPROCESSOR Parameter 5-2</p> <p style="padding-left: 20px;">ERRORPROC Parameter 5-2</p> <p style="padding-left: 20px;">KEYCOMPARE Parameter 5-2</p> <p style="padding-left: 20px;">STATISTICS Parameter 5-3</p> <p style="padding-left: 20px;">FAILURE Parameter 5-3</p> <p>MERGEERRORMESS Procedure 5-3</p> <p>MERGETITLE Procedure 5-3</p> <p>Calling the Merge Program from SPL/3000 5-3</p> <p>Calling the Merge Program from FORTRAN/3000 .. 5-3</p> <p>Section VI Page</p> <p>ERROR MESSAGES AND RECOVERY PROCEDURES</p> <p>Sort Program Error Messages 6-1</p> <p>Merge Program Error Messages 6-2</p> <p>Recovery Procedures 6-2</p> <p>APPENDIX A A-1</p> <p>INDEX I-1</p>
---	--

ILLUSTRATIONS

Title	Page	Title	Page
Record Example	1-1	Program 5 Output	3-14
Key Positions	1-2	Program 6. FORTRAN/3000 Program to Write Records into a File in Sorted Order	3-15
Key Formats	1-3	Program 6 Output	3-16
Major and Minor Key Examples	1-4	Program 7. FORTRAN/3000 Program to Read Records from a File for Processing	3-17
Using the :FILE Command to Specify INPUT and OUTPUT Files for the Stand-Alone Sort Program	2-7	Program 7 Output	3-18
Using the Sort Program INPUT and OUTPUT Com- mands to Specify INPUT and OUTPUT Files for the Stand-Alone Sort Program	2-8	Program 8. FORTRAN/3000 Program to Read Key Fields from a File in Sorted Order	3-19
Using the Terminal to Input Records to the Sort Program	2-9	Program 8 Output	3-20
Sorting a File Read In from a Card Reader	2-9	Program 9. FORTRAN/3000 Program to Create an Index File Consisting of Relative Record Numbers Only	3-21
Running the Stand-Alone Sort Program in Batch Mode	2-11	Program 9 Output	3-22
Program 1. SPL/3000 Program to Call Sort When Both INPUTFILE and OUTPUTFILE are Specified in the SORTINITIAL Call	3-5	Program 10. FORTRAN/3000 Program to Read Key Field and Its Record Number	3-23
Program 1. Output	3-6	Program 10 Output	3-24
Program 2. SPL/2000 Program to Call Sort When INPUTFILE is Specified in the SORTINITIAL Call but OUTPUTFILE is Not	3-7	Data Stack Layout During Execution of the Sort Program	3-25
Program 2 Output	3-8	Running the Stand-Alone Merge Program in Inter- active Mode	4-5
Program 3. SPL/3000 Program to Call Sort When OUTPUTFILE is Specified in the SORTINITIAL Call but INPUTFILE is Not	3-9	Use of \$STDLIST to List the Merged Records on the Terminal	4-6
Program 3 Output	3-10	Running the Stand-Alone Merge Program in Batch Mode	4-7
Program 4. SPL/3000 Program to Call Sort When Neither INPUTFILE nor OUTPUTFILE are Specified in the SORTINITIAL Call	3-11	Program 11. Calling the Merge Program from SPL/3000	5-4
Program 4 Output	3-12	Program 11 Output	5-5
Program 5. FORTRAN/3000 Program to Sort from One File to Another	3-13	Program 12. Calling the Merge Program from FORTRAN/3000	5-6
		Program 12 Output	5-7

TABLES

Title	Page	Title	Page
Units Digit Representation For DISPLAY Format ..	2-5	MERGELIB Error Messages	6-4
SORTLIB Error Messages	6-1	Merge Program Error Messages	6-5
Sort Program Error Messages	6-2		

INTRODUCING SORT/3000

1-1 WHAT IS SORT/3000?

SORT/3000 is a subsystem of the HP 3000 Multiprogramming Executive Operating System (MPE/3000). SORT/3000 consists of two programs: a Sort program and a Merge program.

The Sort program provides the capability of sorting a set of data records into a specified order. Sort may be used as a stand-alone program or the Sort intrinsic may be called from your own program. When used as a stand-alone program, the original records are taken from a file specified by you, and the sorted records are written to a similar file. When the Sort intrinsic is called from your program, the records to be sorted are passed programmatically between your program and Sort.

The Merge program provides the capability of merging several files, each of which has been sorted independently, and producing a single sorted file as the result. Merge may be used as a stand-alone program or the Merge intrinsic may be called from your own program.

1-2. PURPOSES OF SORT/3000

Organizing data can be a time consuming job if done manually. For example, imagine yourself as a high school administrator who is asked to present a list ranking all students in terms of their grades in math and English. The task of sorting manually through data on every student, arranging it in terms of the requirements, and preparing a list might take days, particularly if the school population is large.

Assume, however, that the necessary student records are available on punched cards, or stored on magnetic tape or a disc file. A program that can read these records, rearrange them according to the criteria of grades in English and math, and print the rearranged records (accomplishing all of this in a short amount of time), is a very valuable tool. SORT/3000 is such a program. Given an input file, SORT/3000 will rearrange the records in that file into any order prescribed by you. This rearranged file then can be combined with other files, each arranged in the same manner, to form one file.

The two separate programs of SORT/3000, then, operate as follows:

- The *Sort* program rearranges (or sorts) the records in a file.
- The *Merge* program takes any number of files, all sorted according to the same criteria, and combines (or merges) these files into one file. Thus, if each high school in a school district compiled a list like the one described above, all of the lists then could be combined into one file that represented all students in that district.

1-3. FILES, RECORDS, AND DATA FIELDS

A *file* is a collection of records. The individual records for all students in a high school collectively form a file. Each *record* in the file might contain such specific information as name, sex, age, address, grade level, and grades in individual courses. For example, an individual record could be as shown in Figure 1-1.

NAME	AGE	SEX	ADDRESS		
Markham, Alice B.	15	F	725 High St.		

LEVEL	ENGLISH	HISTORY	P.E.	MATH	ART
11	C	B	A	A	B

Figure 1-1. Record Example

Each item of information in a record is called a *data field*. Thus, HISTORY is a data field in the record shown in Figure 1-1. The record shown is part of a file containing many other such records, each having the same data fields.

SORT/3000 processes both fixed-length and variable-length records. (Fixed-length records are records which always occupy the same amount of space, for example, the 80 columns of a punched card. Variable-length records, as the name implies, consist of records of varying lengths. SORT/3000 can process both types of records because it sorts on fields, or *keys*, within each record. See paragraph 1-4 for a discussion of keys.) The size of a record is subject only to the amount of storage space available to the program, although it is *recommended* that the record size be kept at 256 bytes or less to obtain the most efficient operation.

1-4. KEYS

SORT/3000 rearranges *records* in a file on the basis of *keys*. A *key* consists of a group of characters contained in specific positions in a record. In Figure 1-2, for example, if the sort were to be performed on the basis of telephone numbers, then the key would be specified as positions 21 through 32. Thus, a key can be a data field in a record (the telephone numbers are separate data fields (positions 21 through 32) as are the first names (positions 1 through 10) and the last names (positions 11 through 20)). Also, a key can be any group of characters within a data field (the area codes of the telephone number field, for example), or a key can overlap data fields. For example, the key could be the whole record. The only restriction on the size of a key is that it must be contained within the size of the record.

Records are sorted according to the number of keys, and all of the data fields in a record can be keys. Keys can be contiguous or separated in a record, or they can overlap. They must, however, appear in the same relative position in each record of the file to be sorted. Thus, if the first five positions (or elements) in a record are specified as a key, then the first five elements in every record of the file will be

considered to be a key and must have the same data format. Similarly, keys in files to be merged must be in the same relative position and have the same data format in each record of each file.

Figure 1-3 shows three card records that illustrate various formats for keys in a record. Note that these records do not belong to the same file, as the locations of the keys are different in each record. (These files, even though each could be sorted separately, could not be merged since the keys are not in the same relative position in each record of each file.) Commands are used to inform SORT/3000 of the location of the keys in the file records. The most significant key is called the *major* key and is declared first in the command. Other keys are called *minor* keys and have significance according to their relative position following the major key in the command. Minor keys are *compared* only if a comparison of more significant keys results in an equal condition. Thus, if the major key is ENGLISH and the minor key is MATH, the data in the MATH fields is compared only if the data in the ENGLISH fields are the same.

1-5. SORTING ORDER

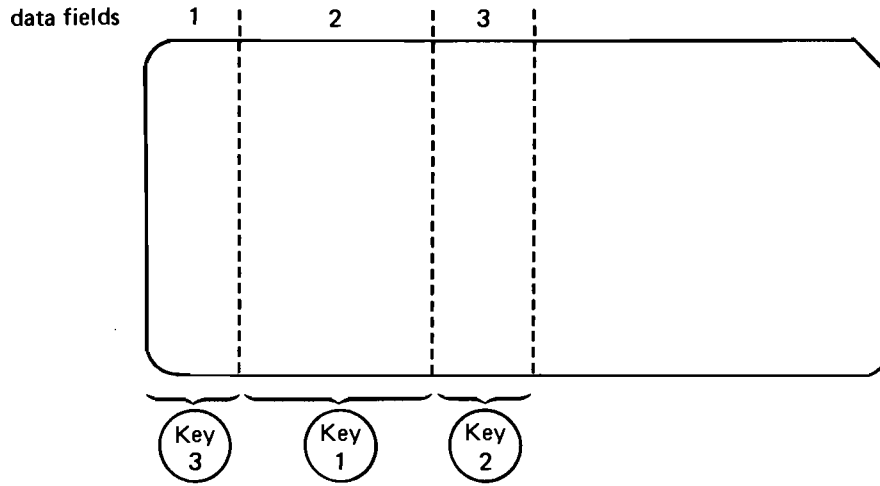
SORT/3000 sorts records in a file (according to the value of the data in the keys) into either *ascending* or *descending* order. Records are in *ascending* order when the key value of each record is greater than or equal to the preceding record. The series A, B, C, D, E and 1, 2, 3, 4, 5 are in *ascending* order. When each record is less than or equal to the preceding record, such as the series F, D, C, A and 10, 6, 5, 2, the records are in *descending* order.

You can specify the order in which records in a file are to be sorted. Thus, if the first record in a high school file is to be that of the student with the highest grades (A) in English and math, you would specify an ascending order. (Although A is the highest grade, it is the lowest value in a sort and thus would appear as the *first* record in an ascending sort.) If the student with the lowest grade (D) is to be first, you would specify a descending sort.

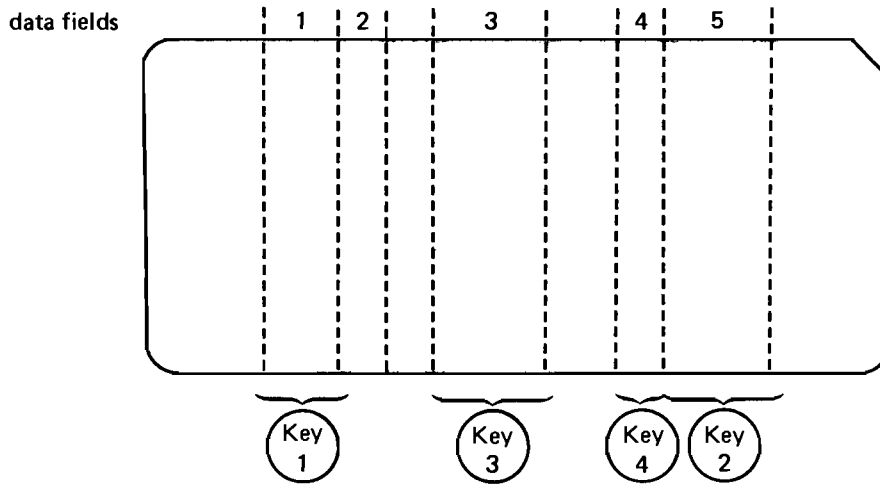
POSITIONS																																	
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
J	A	N	E							D	O	E									4	0	8	-	9	7	8	-	7	6	0	0	
J	A	M	E	S						D	O	U	G	H							9	7	8	-	4	0	8	-	9	7	0	0	
S	A	M	M	Y						D	O	U	G	H	E						7	1	9	-	7	6	2	-	7	4	2	1	

Figure 1-2. Key Positions

RECORD A – CONTIGUOUS KEYS



RECORD B – NONCONTIGUOUS KEYS



RECORD C – OVERLAPPING KEYS

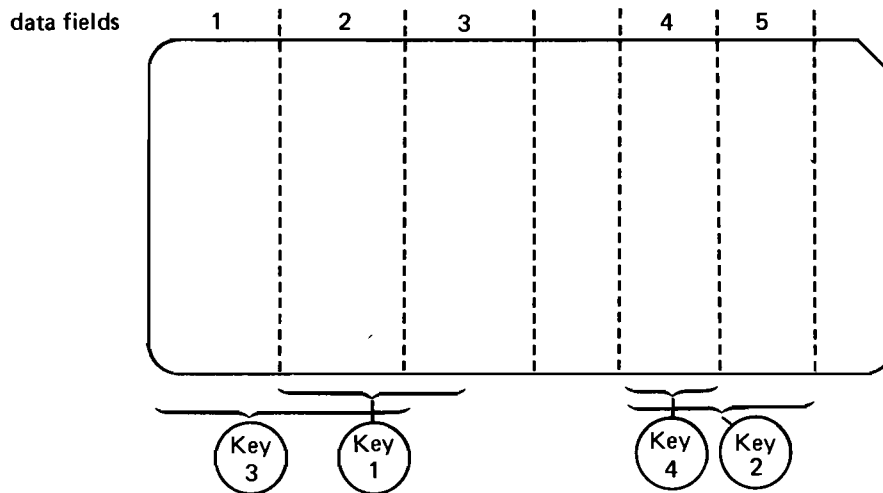


Figure 1-3 Key Formats

The sorting order is specified in the same commands that specify the sorting keys. An order is declared for each key. This order does not necessarily have to be the same for all the keys in a record. In other words, an ascending order can be specified for the major key and a descending order for the minor keys. Then if the major keys are equal, the record with the higher value in the minor key is selected.

For example, assume that you declare that the major key, ENGLISH, has an ascending order, and the minor key, MATH, has a descending order. Then, given two records as shown in Figure 1-4, the second record would appear in the list before the first, producing a list beginning with those students having the highest grades in English (A before B) but the lowest grades in math (D before C).

SORT/3000 correctly sorts keys that contain binary, ASCII, and EBCDIC data according to an 8-bit binary sequence (00000000 to 11111111). Other types of data (integer, real, etc.) are sorted according to standard arithmetic relational operators (e.g., 5×10^{-3} is greater than 2×10^{-5} and 2 is greater than -5). The type of data in a key also is specified in a user command.

1-6 INPUTS TO SORT/3000

SORT/3000 requires you to define the input and output files. The MPE/3000 :FILE command or the SORT/3000 INPUT and OUTPUT commands are used to accomplish this, as explained in Section II. (An *input* file is the file which is to be read and sorted, an *output* file is the rearranged file.) File references are made through standard MPE/3000 file designators. If run as a stand-alone program or called from a FORTRAN/3000 program, SORT/3000

opens all files. If SORT/3000 is called from an SPL/3000 program, you must open the files with the MPE/3000 FOPEN intrinsic. For further information concerning the file system, consult the *MPE Commands Reference Manual*.

An input file can be any file capable of sequential storage such as a file on magnetic tape, disc, or punched cards.

1-7 OUTPUTS FROM SORT/3000

After being processed by SORT/3000, records are written sequentially to the output file. You can specify the composition of the output file with the SORT/3000 OUTPUT command (explained in Section II).

A *sorted* output file can consist of:

- Records of the sorted input file.
- Key fields of the sorted input file.
- Relative addresses of records of the sorted input file.
- Relative addresses of records and key fields of the sorted input file.

A *merged* output file can consist of key fields only or records only.

Because it is possible to increase the number of records in an output file when the Merge program is run, you must ensure that the output file is capable of storing all incoming records. (Refer to Sections IV and V for discussions concerning how to run the Merge program.)

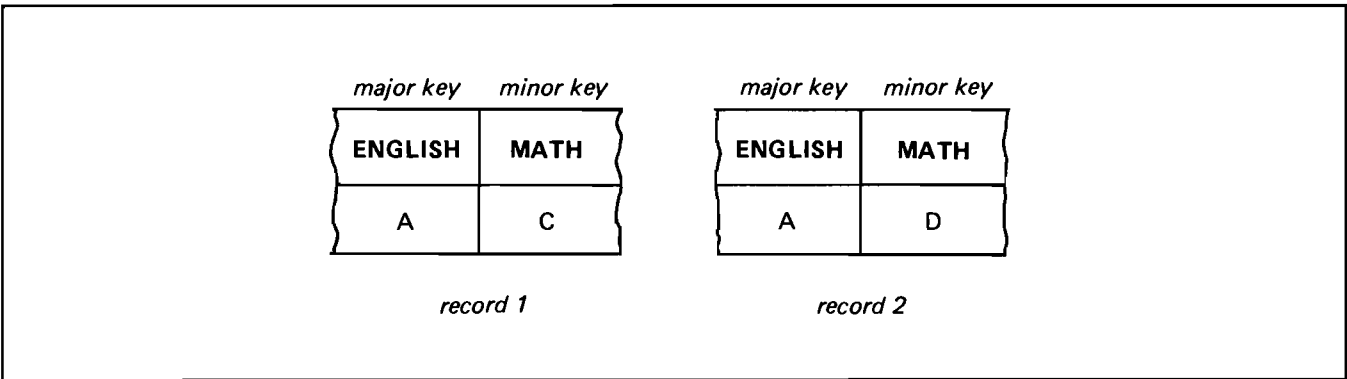


Figure 1-4. Major and Minor Key Examples

RUNNING THE SORT PROGRAM AS A STAND-ALONE PROGRAM

SECTION

II

The Sort program can be run as a stand-alone program during either a batch job or an interactive session. It may be more convenient, when large amounts of input and output are involved, to run the program in batch mode. You can, however, initiate the Sort program from an interactive terminal and can instruct the program to read input from the terminal, from a batch input device (such as a card reader), or from a disc file. The output file can be a line printer, punched cards, disc, tape, or the terminal itself.

Initiation of batch jobs and interactive sessions is covered in this section to the extent necessary for you to run the Sort program; however, more extensive descriptions of these procedures can be found in the *MPE Commands Reference Manual*.

The Sort program operates in the minimum HP 3000 memory size. The amount of memory available to Sort, however, is a critical factor in determining the performance and can affect both the time required to perform a sort and the amount of secondary storage which is required for a temporary file. The Sort program normally will attempt to make its stack as large as possible, within the limits established at the time the Sort program was installed on the system. You can override this limit and provide a larger stack size for the Sort program by appending the `MAX-DATA = segsize` parameter to the `RUN SORT.PUB.SYS` command. (This is not necessary unless the error message "INSUFFICIENT STACK SPACE" is received when you attempt to run the Sort program.) A `segsize` of 10,000 should be sufficient for sorting most files. If the error message occurs again, however, increase the `segsize` parameter.

2-1. FILE DEFINITIONS

The stand-alone Sort program references various files during execution. These files are described in the following paragraphs. For a complete understanding of the treatment of files by the operating system, read the *MPE Commands Reference Manual*.

2-2. INPUT FILE

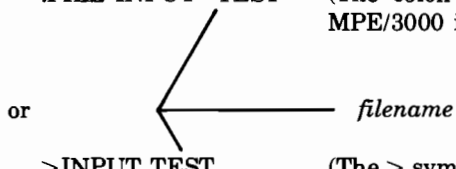
The INPUT file contains the records to be sorted.

When running the Sort program, you must equate the formal designator INPUT with the actual file designator (or *filename*). This is done with the MPE/3000 `:FILE` command or the Sort program INPUT command, as follows:

`:FILE INPUT=TEST` (The colon (:) is output by MPE/3000 in a session)

or

`>INPUT TEST` (The > symbol is output by the Sort program in a session)



The INPUT file is opened as follows:

Formal designator:
INPUT

Foptions:
Domain — old or oldtemp; others default in a session.

Aoptions:
Read-access only; others default.

All other parameters:
Default.

NOTE: The file is opened automatically by MPE/3000 and you need not be concerned with the file parameters. They are presented here (and in paragraph 2-3) for reference only.

2-3. OUTPUT FILE


The sorted records are written to the OUTPUT file.

As with the INPUT file, you must equate the file formal designator (OUTPUT) with the actual file designator (*filename*) with the MPE/3000 `:FILE` command or the Sort program OUTPUT command, as follows:

`:FILE OUTPUT=TEST1`

or

`>OUTPUT TEST1`



The OUTPUT file is opened as follows:

Formal designator:
OUTPUT

Foptions:
Domain = new; ASCII/BINARY same as INPUT file (but if output option (see paragraph 2-4) is *record*

number, BINARY is used); record format same as INPUT file (but if output option (see paragraph 2-4) is not default, *fixed* is used); others default.

Aoptions:

Write-access only; others default.

Recsize:

Depends on output option, as follows:

<u>Output Option</u>	<u>Recsize (in bytes)</u>
default	same as INPUT file
recnum only	4
keys only	sum of lengths of keyfields
recnum + keys	sum of lengths of keyfields + 4

Device:

DISC

Blockfactor:

If default output option, same as INPUT file; otherwise default.

Filesize:

Same as end-of-file of INPUT file.

Numextents:

If default option, same as INPUT file; otherwise default.

Initialloc:

All extents are initially allocated.

Filecode:

Same as INPUT file.

All other parameters:

Default.

- The output records will each consist of a double integer whose value is the original logical (relative) record number. (The MPE/3000 file system numbers records in a file beginning at zero, thus the file system record number of the fourth record in a file is three, whereas the logical (or relative) number is four.)
- The output records will consist of the key fields concatenated (butted) together from left to right.
- The output records will consist of both the original logical (relative) record number and the key fields concatenated (butted) together, with the logical record number on the left.

The OUTPUT file is closed with the option SAVE, and, if it was opened as a new file, space beyond the end-of-file is released. If the close fails due to another file having the same name, the interactive user is asked if the old file can be purged, as follows:

PURGE OLD OUTPUT FILE *filename*?

If the response is "Y", the old file will be purged. If this cannot be done (for example, you cannot purge a file which is not in your group account or a file which is being used concurrently by another user), or if the answer was "N", you will be asked for a new name for the OUTPUT file:

ENTER NEW NAME FOR OUTPUT FILE:

The OUTPUT file will be renamed using the name entered by you, and another attempt will be made to save it.

In batch mode, the foregoing sequence cannot take place. Instead, the OUTPUT file will be assigned an artificial name constructed by the Sort program and the OUTPUT file will be saved under that name.

The following message will be printed on the LIST file:

OUTPUT FILE SAVED WITH FILENAME
"filename"

Note: If * is specified in the OUTPUT command (meaning that the sorted records will be output to the standard list device), the OUTPUT file will not be saved. (See paragraph 2-11.)

2-4. OUTPUT OPTIONS. The format of the output records depends on the output option. There are four options, as follows:

- The output records will be the same as the input records.

2-5. LIST FILE

The LIST file is used by the Sort program to output information to you and to prompt for input (if in interactive session). The LIST file should not be confused with the OUTPUT file (which is used to output the sorted records). The LIST file normally is equated with \$STDLIST.

2-6. TEXT FILE

The TEXT file is used by the Sort program to read commands and other information from you. Normally, TEXT is equated with \$STDINX.

2-7. PROMPT FILE

The PROMPT file is used by the Sort program to prompt you for input when the TEXT file is the session terminal but the LIST file is not the session terminal. The prompt is a "greater than" (>) symbol.

2-8. SCRATCH FILE

The Sort program requires a SCRATCH file on disc to perform the sort. This file is not apparent to you but it may be necessary to know the name of the file in case of errors. The name of the SCRATCH file is SORTSCR.

2-9. SORT COMMANDS

The Sort program is controlled by commands which specify INPUT files, key fields, OUTPUT files, output options, etc. In batch mode, these commands are included with the :RUN SORT.PUB.SYS command for the Sort program; in interactive session, the Sort program outputs a prompt character (>) after it begins execution and the commands are typed in from the terminal.

2-10. INPUT COMMAND

The INPUT command specifies the file which contains the records to be sorted. The form of the INPUT command is

$$\text{INPUT } \left\{ \begin{array}{l} \textit{filename} \\ * \end{array} \right\} \left[\begin{array}{l} \textit{number of records} \\ \textit{record size} \end{array} \right]$$

Note The use of brackets [] throughout this manual indicates that the enclosed parameters are optional. Braces { } indicate that one of the enclosed parameters is required.

where

filename
is any legal formal designator.

*

specifies that the input records will be read from the TEXT file instead of from the INPUT file.

number of records
is a positive integer specifying the upper limit of the number of records to be sorted.

record size
is a positive integer specifying the number of characters in each record.

If you specified *filename* in the INPUT command instead of *, *filename* will be used as the formal designator of the INPUT file instead of the designator "INPUT". If * is specified, input records will be read from the TEXT file instead of from the INPUT file. Input records will be assumed to follow the END command (see paragraph 2-15) and continue until the end-of-file is reached. If the TEXT file is the session terminal and you specified *, you will be prompted with the character "?" for each record (end-of-file is signalled by typing :EOD). The OUTPUT file in this case will be opened with ASCII, fixed-length records, and default blockfactor, numextents, and filecode.

If you specify *number of records*, it is used as the upper limit of the number of records which are to be sorted. This should be specified only when the INPUT file is not a disc file. When the INPUT file is a disc file, its current end-of-file value will be used even if *number of records* is specified. If the INPUT file is not a disc file and *number of records* is not specified, a default value of 10,000 is assumed by the Sort program.

Record size may be used to override the record size of the TEXT file when the INPUT * option is used. *Record size* indicates the number of characters which are to be read, and must not be greater than the record size of the TEXT file. This parameter is ignored if the INPUT * option is not being used.

One or more INPUT commands may be entered at any time before the END command. If more than one INPUT command is used, only the new parameters which are specified have any effect.

Examples of the INPUT command are as follows:

```
INPUT TAPEX,50000
Sort file TAPEX, which has at most 50,000 records.
```

```
INPUT *,250,40
Sort up to 250 records from TEXT file, with a maximum of 40 characters to be read from each record.
```

2-11. OUTPUT COMMAND

The OUTPUT command specifies the file to which the sorted records will be written. The form of the OUTPUT command is

$$\text{OUTPUT } \left\{ \begin{array}{l} \textit{filename} \\ * \end{array} \right\} \left[\textit{NUM} \right] \left[\textit{KEY} \right]$$

where

filename
is any legal formal designator.

*

specifies that the sorted records will be sent to the LIST file.

NUM
specifies that the output records will consist of the original logical record numbers.

KEY

specifies that the output records will consist of the key fields.

If *filename* is specified in the OUTPUT command, it will be used as the formal designator instead of the designator "OUTPUT."

If * is specified, the sorted output records will be sent to the LIST file instead of the OUTPUT file. (The OUTPUT file will not be saved in this case.)

If NUM is specified and KEY is not, the output records will each consist of a double integer whose value is the original logical (relative) record number.

If KEY is specified and NUM is not, the output records will consist of the key fields concatenated together from left to right.

If NUM and KEY are both specified, the output records will consist of both the original logical record number and the key fields concatenated together, with the logical record number on the right.

If neither NUM nor KEY is specified, the output records are identical to the input records.

Examples of the OUTPUT command are as follows:

```
OUTPUT *
Output the sorted records to the LIST file.

OUTPUT *,KEY
Output the keys to the LIST file.

OUTPUT SEQFILE,NUM
Output the logical record numbers to the file SEQ-
FILE.
```

2-12. KEY COMMAND

The KEY command specifies the key fields to be used in sorting the input records. The form of the KEY command is

```
KEY { position } [length] [type]
  [DESC] [position] [length]
  [type] [DESC] . . .
```

where

position
is a positive integer.

length
is a positive integer.

type
is one of the following mnemonics: BYTE, INT, DOUBLE, REAL, LONG, PACKED, DISPLAY, or PACKED*. (See the meanings of the mnemonics below.)

DESC

indicates that the records are to be sorted in descending order.

As shown in the example of the KEY command form, each KEY command can specify one or more key fields, with the specifications being separated by semicolons. Multiple key fields also can be specified with several KEY commands (all the key fields do not have to be specified in one KEY command). In either case, the first key field specified is the most significant, or *major* key, with subsequent fields having less significance.

The *position* parameter indicates the number within the record of the first position of the key field (the first position of the record is numbered 1).

The *length* parameter indicates the number of bytes in the field. This parameter is required if *type* = BYTE, PACKED, DISPLAY, or PACKED*. It is optional in other cases. If not specified, the value assumed for length depends on the type: 2 if *type* = INT; 4 if *type* = DOUBLE or REAL; and 8 if *type* = LONG*.

The *type* parameter defines the type of data contained in the key field. If not specified, BYTE is assumed. The mnemonics used have the following meanings:

BYTE

a direct byte comparison is to be used. This specification should be used for ASCII, EBCDIC, or logical quantities. The *length* parameter is required.

INT

the key field contains a 2's complement number of the specified *length* (in bytes). If no *length* is given, 2 is assumed.

DOUBLE

the key field contains a 2's complement number of the specified *length* (in bytes). If no *length* is given, 4 is assumed.

REAL

the key field contains a floating-point number. Any *length* may be specified; if no *length* is specified, 4 is assumed.

LONG

the key field contains a floating-point number. Any *length* may be specified; if no *length* is specified, 8 is assumed.*

PACKED

the key field contains a packed decimal number. In this format, each byte except the last contains 2 digits, each of which occupies 4 bits, with digits 0 - 9 represented by 4-bit quantities. The last byte contains the least significant digit of the number in its leftmost 4 bits, and the sign of the number in its rightmost 4 bits. The sign is considered to be minus if it has the value 1101, and plus for any other value.

*For 3000 systems which are not Series II, the *type* = LONG has a *length* of 6.

DISPLAY

the key field contains a numeric display quantity. Numeric display items are represented as ASCII coded (8 bits) decimal digits (0 through 9) except for the units digit which carries the sign of the data item. The units digit, with the sign of its associated number being positive, negative, or no sign (absolute value) respectively, is represented in ASCII code as shown in Table 2-1. For example, to represent the digit 1 as positive, the ASCII code octal 101 (which is the code for the letter A) is used; to represent this digit as -1, octal 112 is used; and for no sign, octal 61. In this manner, the same eight bits can represent both the digit *and* the sign.

PACKED*

this format is identical to PACKED except that there are only an even number of digits plus a sign, whereas PACKED has an odd number of digits plus a sign. In PACKED* format, the highest order 4 bits are not considered to be part of the field.

The *DESC* option, if specified, indicates that the records are to be ordered in descending order with respect to this key field. If *DESC* is not specified, the records will be ordered in ascending order (the default condition).

Examples of the KEY command are:

KEY 10,5
 BYTE key of length 5 starting in position 10, to be sorted in ascending order.

KEY 20,REAL
 REAL key of length 4 starting in position 20, to be sorted in ascending order.

KEY 30,20,INT,DESC

20-byte INTEGER key starting in position 30, to be sorted in descending order.

KEY 10,5;20,REAL;30,20,INT,DESC

This last example is equivalent to the first three.

2-13. RESET COMMAND

The RESET command indicates that any KEY commands which have been entered thus far are to be ignored and that a new set will be entered. (This command is useful if you make a typing error.) The form of the RESET command is

RESET

2-14. VERIFY COMMAND

The VERIFY command causes a listing of the options in effect. The form of the VERIFY command is

VERIFY

The format of the listing is as follows:

INPUT FILE = FILEA

NUMBER OF RECORDS = 10,000

OUTPUT FILE = *,KEY

POSITION	LENGTH	TYPE	DESC	
50	5	BYTE	ASC	(MAJOR KEY)
1	10	DISPLAY	DESC	
21	8	PACKED	ASC	

Table 2-1. Units Digit Representation for DISPLAY Format

Units Digits	Internal Representation in ASCII (with its octal equivalent inside parentheses)		
	Positive	Negative	No Sign
0	{ (173)	} (175)	0 (60)
1	A (101)	J (112)	1 (61)
2	B (102)	K (113)	2 (62)
3	C (103)	L (114)	3 (63)
4	D (104)	M (115)	4 (64)
5	E (105)	N (116)	5 (65)
6	F (106)	O (117)	6 (66)
7	G (107)	P (120)	7 (67)
8	H (110)	Q (121)	8 (70)
9	I (111)	R (122)	9 (71)

2-15. END COMMAND

The END command indicates that there are no more commands and that the sort should begin. The form of the END command is

```
END
```

If INPUT * has been specified, the Sort program will output the character "?" and the input records are typed in from the terminal.

2-16. STATISTICS

When a sort is completed, the Sort program prints statistics on the standard list device as follows:

```
NUMBER OF RECORDS = 100,000
RECORD SIZE (IN BYTES) = 100
NUMBER OF INTERMEDIATE PASSES = 2
SPACE AVAILABLE (IN WORDS) = 18,521
NUMBER OF COMPARES = 1,734,091
NUMBER OF SCRATCHFILE IO'S = 9,921
CPU TIME (MINUTES) = 44.19
ELAPSED TIME (MINUTES) = 63.95
```

Parameters whose meanings are not self evident are:

```
NUMBER OF INTERMEDIATE PASSES
the number of passes over the data made by Sort, not
including the input and output of the records.

SPACE AVAILABLE
the number of words in the working space for the Sort
procedures.

NUMBER OF COMPARES
the number of comparisons made between records.

NUMBER OF SCRATCHFILE IO'S
the number of FREADs and FWRITEs performed
against the SORTSCR file.

CPU TIME
CPU time expended between the start and end of the
Sort program.

ELAPSED TIME
real time between start and end of the Sort program.
```

2-17. CONTROL Y

During the running of the stand-alone Sort program, it is possible to obtain the status of the sort by typing CON-

TROL Y. This feature is available only if the program is being run in the interactive mode. The format of the output depends on the phase (input, intermediate sort, or output) as follows:

```
INPUT PHASE:
1234 RECORDS HAVE BEEN INPUT
```

```
INTERMEDIATE SORT PHASE:
PASS 1 OF 3
```

```
OUTPUT PHASE:
1234 RECORDS HAVE BEEN OUTPUT
```

2-18. RUNNING THE STAND-ALONE SORT PROGRAM IN INTERACTIVE SESSION

The Sort program can be used to sort an existing file or the records to be sorted can be typed in from the terminal in interactive mode. In either case, the INPUT file must be specified.

When an existing file is to be sorted, the actual file designator (*filename*) must be equated with the formal designator "INPUT" through the use of the MPE/3000 :FILE command or with the Sort program > INPUT command.

For example:

```
:FILE INPUT = TEST
(Use of the MPE/3000 :FILE command to equate the
existing file TEST to "INPUT")
```

or

```
> INPUT TEST
(Use of the Sort program INPUT command to equate
the existing file TEST to "INPUT")
```

The INPUT * command is used to specify that the input records will be read from the TEXT file, as follows:

```
INPUT *
```

The OUTPUT file must be equated to the formal designator "OUTPUT" with the MPE/3000 :FILE command or the Sort program > OUTPUT command, as follows:

```
:FILE OUTPUT = OUTFILE
```

or

```
> OUTPUT OUTFILE
```

If the sorted records are to be sent to the LIST file, the OUTPUT * command is used, as follows:

```
> OUTPUT *
```

```

HELLO MANAGER.SCR
SESSION NUMBER = #S127
FRI, JAN 17, 1975, 2:42 PM
HP32000C.F0.25

:FILE INPUT =MAIL1
:FILE OUTPUT=TEST
:RUN SORT.PUB.SYS

HP32214B.00.00 SORT/3000 FRI, JAN 17, 1975, 2:43 PM

>KEY 11,9
>KEY 1,10
>VERIFY

INPUT FILE = INPUT
OUTPUT FILE = OUTPUT
KEY POSITION   LENGTH   TYPE   ASC/DESC
           11         9   BYTE   ASC   (MAJOR KEY)
           1         10  BYTE   ASC

>END

                STATISTICS

NUMBER OF RECORDS =                12
RECORD SIZE (IN BYTES) =            80
NUMBER OF INTERMEDIATE PASSES =      0
SPACE AVAILABLE (IN WORDS) =        13,346
NUMBER OF COMPARES =                 45
NUMBER OF SCRATCHFILE IO'S =         10
CPU TIME (MINUTES) =                  .01
ELAPSED TIME (MINUTES) =              .14

    END OF PROGRAM
:BYE

CPU (SEC) = 7
CONNECT (MIN) = 5
FRI, JAN 17, 1975, 2:46 PM
END OF SESSION

```

Figure 2-1. Using the :FILE Command to Specify INPUT and OUTPUT Files for the Stand-Alone Sort Program.

Figures 2-1 through 2-4 illustrate the use of the Sort program in the interactive mode. Note that the MAXDATA = *segsz* parameter is not used in the RUN SORT.PUB.SYS commands (which is normal unless exceptionally large files are being sorted).

In Figure 2-1, the :FILE command is used to specify the INPUT file (MAIL1) and OUTPUT file (TEST). Once the Sort program is accessed (with the RUN SORT.PUB.SYS command), only the key fields need be specified.

Figure 2-2 shows the use of the Sort program INPUT and OUTPUT command to specify the INPUT and OUTPUT files. The OUTPUT * command is used in the example, which causes the sorted records to be sent to the LIST file (the terminal in interactive session). (The OUTPUT file will not be saved in this case.)

Figure 2-3 illustrates how to input records from the terminal, have them sorted by the program, and listed on the terminal.

Figure 2-4 illustrates how to read the input file from a card reader, have it sorted by the program, and stored on disc. The data deck must be preceded by a :DATA card, as follows:

```

:DATA [jobname,] {username} [/upass]
      {acctname} [/apass] [filename]

```

where

jobname
is the name of the job or session that is to read the data. (Optional parameter.)

```

:RUN SORT.PUB.SYS

HP32214B.00.00 SORT/3000  FRI, JAN 17, 1975,  2:47 PM

>INPUT MAILLIST
>OUTPUT *
>KEY 11,9
>KEY 1,10
>VERIFY

INPUT FILE = MAILLIST
OUTPUT FILE = *
KEY POSITION   LENGTH   TYPE   ASC/DESC
      11         9   BYTE   ASC   (MAJOR KEY)
      1         10  BYTE   ASC

>END
LOIS      ANYONE   6190 COURT ST      METROPOLIS  NY 20115 619-732-4997
KING     ARTHUR   329 EXCALIBUR ST   CAMELOT     CA 61322 812-200-0100
ALI      BABA      40 THIEVES WAY     SESAME      CO 69142 NONE
JOHN     BIGTOWN   965 APPIAN WAY     METROPOLIS  NY 20013 619-407-2314
KNEE     BUCKLER   974 FISTICUFF DR   PUGILIST    ND 04321 976-299-2990
SWASH    BUCKLER   497 PLAYACTING CT  MOVIE TOWN  CA 61497 NONE
JAMES    DOE       4193 ANY ST        ANYTOWN     MD 00133 237-408-7100
JANE     DOE       3959 TREEWOOD LN   BIGTOWN     MA 21843 714-399-4563
JOHN     DOUGHE   239 MAIN ST        HOMETOWN    MA 26999 714-411-1123
JENNA    GRANDTR  493 TWENTIETH ST   PROGRESSIVE CA 61335 799-191-9191
KARISSA GRANDTR  7917 BROADMOOR WAY BIGTOWN     MA 21799 713-244-3717
SPACE    MANN     9999 GALAXY WAY    UNIVERSE    CA 61239 231-999-9999

                STATISTICS

NUMBER OF RECORDS =                12
RECORD SIZE (IN BYTES) =            72
NUMBER OF INTERMEDIATE PASSES =      0
SPACE AVAILABLE (IN WORDS) =        13,346
NUMBER OF COMPARES =                44
NUMBER OF SCRATCHFILE IO'S =         8
CPU TIME (MINUTES) =                 .01
ELAPSED TIME (MINUTES) =             1.62

END OF PROGRAM

```

Figure 2-2. Using the Sort Program INPUT and OUTPUT Commands to Specify INPUT and OUTPUT Files for the Stand-Alone Sort Program

username

is the user's name, as established in MPE/3000 by the user with Account Manager capability. (Required parameter.)

apass

the account password. (Required if account has a password.)

upass

the user password. (Required if user has a password.)

filename

an additional qualifying name that can be used by the job or session to access the data. (Optional parameter.)

acctname

the name of the account, as established by the user with System Manager capability. (Required parameter.)

See the *MPE Commands Reference Manual* for further descriptions of the :DATA command parameters.

```

:RUN SORT.PUB.SYS

HP32214B.00.00 SORT/3000  FRI, JAN 17, 1975,  2:53 PM

>INPUT *
>OUTPUT *
>KEY 1,5,INT
>END
?98765
?87654
?76543
?65432
?54321
?67890
?56789
?45678
?3456789
?234567
?12345
?:EOD
12345
234567
3456789
45678
54321
56789
65432
67890
76543
87654
98765

                STATISTICS

NUMBER OF RECORDS =                11
RECORD SIZE (IN BYTES) =           72
NUMBER OF INTERMEDIATE PASSES =     0
SPACE AVAILABLE (IN WORDS) =        13,349
NUMBER OF COMPARES =                34
NUMBER OF SCRATCHFILE IO'S =        8
CPU TIME (MINUTES) =                .01
ELAPSED TIME (MINUTES) =            1.06

END OF PROGRAM

```

Figure 2-3. Using the Terminal to Input Records to the Sort Program

```

:FILE IN:DEV=CARD
:RUN SORT.PUB.SYS

HP32214B.00.00 SORT/3000  THU, JAN 30, 1975,  1:16 PM

>INPUT IN
>OUTPUT MAIL1
>KEY 11,9
>KEY 1,10
>END
PURGE OLD OUTPUT FILE MAIL1.PUB.GOODWIN ? YES

                STATISTICS

NUMBER OF RECORDS =                25
RECORD SIZE (IN BYTES) =           80
NUMBER OF INTERMEDIATE PASSES =     0
SPACE AVAILABLE (IN WORDS) =        13,346
NUMBER OF COMPARES =                125
NUMBER OF SCRATCHFILE IO'S =        18
CPU TIME (MINUTES) =                .01
ELAPSED TIME (MINUTES) =            .18

END OF PROGRAM

```

Figure 2-4. Sorting a File Read In From a Card Reader

2-19 RUNNING THE STAND-ALONE SORT PROGRAM IN BATCH MODE

The stand-alone Sort program can be run in batch mode by specifying the INPUT and OUTPUT file by using the MPE/3000 :FILE command or by using the Sort program INPUT and OUTPUT commands.

For example, the cards could be as follows:

```
:JOB BATCH.JOB
:FILE INPUT=MAILLIST
:FILE OUTPUT=TEST
:RUN SORT.PUB.SYS
KEY 11,9
KEY 1,10
END
:EOJ
```

or

```
:JOB BATCH.JOB
:RUN SORT.PUB.SYS
INPUT MAILLIST
OUTPUT TEST
KEY 11,9
KEY 1,10
END
:EOJ
```

Figure 2-5 shows the printout resulting from a batch job which specified * as the INPUT file (the cards to be sorted were included with the :JOB deck), and * as the OUTPUT file, which outputs the sorted records to the LIST file (the line printer in batch mode).

Note: The OUTPUT file is not saved in this case.

The cards used were as follows:

```
:JOB MANAGER.SCR
:RUN SORT.PUB.SYS
INPUT *
OUTPUT *
KEY 11,9
KEY 1,10
END
.
.
.
CARDS CONTAINING RECORDS TO
BE SORTED
.
.
.
:EOD
:EOJ
```



```

:JOB  MANAGER,SCR, PUB
PRI= DS; INPRI= 13; TIME= ?
JOB NUMBER = #J14
WED, JAN 15, 1975, 1:30 PM
HP32000C.F0.25

```

```

:RUN SORT.PUB.SYS

```

```

HP32214R.00.00 SORT/3000 WED, JAN 15, 1975, 1:30 PM

```

```

INPUT *
OUTPUT *
KEY 11,9
KEY 1,10
END

```

PLAINS	ANTELOPE	201	OPENSACE AVE	BIGCOUNTRY	WY	49301	369-732-4821
BLACK	REAR	47	ALLOVER DR	ANYWHERE	US	00111	NONE
ANIMAL	CRACKERS	1000	ANYWHERE PL	ALLUVER	US	00001	001-100-1000
MULE	DEER	963	FOREST PL	NICECOUNTRY	CA	97643	493-900-9000
WHITETAIL	DEER	34	WOODSY PL	BACACOUNTRY	ME	01341	619-433-4333
PRAIRIE	DOG	493	ROLLINGHILLS DR	OPENSACE	ND	24321	992-419-4192
MALLARD	DUCK	79	MARSH PL	PUDDLEDUCK	CA	97432	492-492-4922
SNOWSHOF	HARE	742	FRIGID WAY	COLDSPOT	MN	37434	732-732-7320
MOUNTAIN	LION	796	KING DR	THICKET	NM	37643	712-712-7122
SWAMP	RARRIT	4444	DAMPPLACE RD	RAYOU	LO	79999	NONE
NASTY	RATTLER	243	DANGER AVE	DESERTVILLE	CA	87654	828-432-4321
BIGHORN	SHEEP	999	MOUNTAIN DR	HIGHPLACE	CO	34567	776-409-9040
GREY	SQUIRREL	432	PLEASANT DR	FALLCOLORS	MA	14321	619-619-6199

STATISTICS

```

NUMBER OF RECORDS = 13
RECORD SIZE (IN BYTES) = 80
NUMBER OF INTERMEDIATE PASSES = 0
SPACE AVAILABLE (IN WORDS) = 13,346
NUMBER OF COMPARES = 50
NUMBER OF SCRATCHFILE IO'S = 10
CPU TIME (MINUTES) = .01
ELAPSED TIME (MINUTES) = .01

```

```

END OF PROGRAM
:EOJ

```

```

CPU (SEC) = 5
ELAPSED (MIN) = 1
WED, JAN 15, 1975, 1:30 PM
END OF JOB

```

Figure 2-5. Running the Stand-Alone Sort Program in Batch Mode



RUNNING THE SORT PROGRAM FROM A USER PROGRAM

SECTION

III

You can run the Sort program from an SPL/3000 or FORTRAN/3000 program through the use of intrinsic calls. The arguments of the intrinsic calls provide the parameters needed by Sort to perform specific operations. With COBOL, the COBOL SORT verb is used to run the Sort program (see the *COBOL/3000, Reference Manual* for complete details).

To run Sort from your program, the Sort program intrinsics must be called in the correct order. There are six programmatically callable intrinsics, as follows:

SORTINITIAL
SORTINPUT
SORTOUTPUT
SORTEND
SORTERRORMESS
SORTTITLE

These intrinsics (in the form of SPL/3000 procedures) reside in the system segmented library (SL.PUB.SYS) in the segment SORTLIB. SORTINITIAL is always called first, followed by calls to SORTINPUT and SORTOUTPUT (unless SORTINITIAL has been instructed to perform input and/or output directly (see paragraphs 3-2 and 3-3)). Finally, SORTEND must be called to allow the Sort program to restore the data stack to its original state.

The procedures are described in the following paragraphs and their forms (in SPL/3000) are shown. It is necessary for you to know the forms of the various procedures so that you may provide arguments in the correct order (and of the correct type) in your intrinsic calls from SPL/3000 or FORTRAN/3000. See the *System Programming Language Reference Manual* for further descriptions of SPL/3000 procedures.

The Sort program operates in the minimum HP 3000 memory size. The amount of memory available to Sort, however, is a critical factor in determining the performance and can affect both the time required to perform a sort and the amount of secondary storage which is required for a temporary file. The Sort program normally will attempt to make its stack as large as possible. When you are calling the Sort intrinsics from your program, you must permit Sort to obtain this stack space by using the MAXDATA = *segsz* parameter when your program is being prepared. A *segsz* of 4000 was used in the :PREP commands in the examples shown in this section (see paragraphs 3-17 and 3-18).

3-1. SORTINITIAL PROCEDURE

The SORTINITIAL procedure is called to initiate the Sort program. The form of this procedure (in SPL/3000) is

```
PROCEDURE SORTINITIAL (INPUTFILE,  
                        OUTPUTFILE,OUTPUTOPTION,RECLN,  
                        NUMRECS,NUMKEYS,KEYS,  
                        ERRORPROC,KEYCOMPARE,STATISTICS,  
                        FAILURE);
```

```
VALUE INPUTFILE,OUTPUTFILE,  
        OUTPUTOPTION,RECLN,NUMRECS,  
        NUMKEYS;
```

```
INTEGER INPUTFILE,OUTPUTFILE,  
        OUTPUTOPTION,RECLN,NUMKEYS;
```

```
DOUBLE NUMRECS;
```

```
ARRAY KEYS,STATISTICS;
```

```
PROCEDURE ERRORPROC;
```

```
LOGICAL PROCEDURE KEYCOMPARE;
```

```
LOGICAL FAILURE;
```

```
OPTION VARIABLE,EXTERNAL;
```

The SORTINITIAL parameters are all optional, but, certain combinations are not permitted, as explained in the following paragraphs.

3-2. INPUTFILE PARAMETER

INPUTFILE is the MPE/3000 file number of the file which is to be sorted. If this parameter is specified in your call to the SORTINITIAL intrinsic, the input records are read directly from the file by the Sort program, and no calls must be made to SORTINPUT (see paragraph 3-12). If the parameter is not specified, the records are passed via SORTINPUT (which must be called). In addition, the RECLN parameter is mandatory in this case.

3-3. OUTPUTFILE PARAMETER

OUTPUTFILE is the MPE/3000 file number of the file to which sorted records are to be output. If specified in your call to the SORTINITIAL intrinsic, no calls to SORTOUTPUT must be made (see paragraph 3-13). If the parameter is not specified, the sorted records are output via SORTOUTPUT (which must be called).

3-4. OUTPUTOPTION PARAMETER

OUTPUTOPTION determines the format of the output records. There are four possibilities:

- 0 The output record is the same as the input record.
- 1 The output record is a double integer (4 bytes) whose value is the logical (relative) record number of the record.
- 2 The output record contains only the key fields, concatenated together with the major keys on the left, minor keys following.
- 3 The output record is the logical record number (as in 1) followed by the key fields (as in 2).

If OUTPUTOPTION is not specified, its value is defaulted to 0.

3-5. RECLLEN PARAMETER

RECLLEN is the maximum length of a record, in bytes. If RECLLEN is not specified, the record length is taken from INPUTFILE, which must be specified.

3-6. NUMRECS PARAMETER

NUMRECS is an upper bound on the number of records which are to be sorted. If this parameter is not specified, the end-of-file of INPUTFILE is used. If INPUTFILE is not specified, the value of 10,000 (double integer) is used.

3-7. NUMKEYS AND KEYS PARAMETERS

The parameters NUMKEYS and KEYS specify how the records are to be ordered. If either is specified, the other must be also. These parameters must be specified *if and only if* KEYCOMPARE is not. NUMKEYS is the number of keys to be used in the compare and must be at least one. KEYS is an array which specifies how the records are to be compared. It contains three words for each key field. These are defined as follows:

WORD 0 = position within input record of the first character of the key. (The first character of the record is position 1.)

WORD 1 = number of bytes in the key.

WORD 2 (bits 0 through 7) = 0 for ascending key, 1 for descending key.

WORD 2 (bits 8 through 15) gives type of data, as follows:

- 0 = logical or character
- 1 = two's complement (including integer and double integer)
- 2 = floating point (including real and long)
- 3 = packed decimal (see KEY command, Section II)
- 4 = numeric display (see KEY command, Section II)
- 5 = packed decimal with even number of digits (see KEY command, Section II)

3-8. ERRORPROC PARAMETER

The ERRORPROC is a user-supplied procedure and, if specified in your call to SORTINITIAL, must be called from your program and used in conjunction with the SORTERRORMESS procedure (see paragraph 3-15).

ERRORPROC, if specified, is called programmatically whenever a fatal error occurs in one of the Sort program procedures. The form of this procedure is

```
PROCEDURE ERRORPROC(ERRORCODE);
INTEGER ERRORCODE;
```

ERRORCODE is the Sort program error number which is passed to ERRORPROC when an error occurs.

If ERRORPROC is not specified, a default procedure is used. The default procedure simply prints an error message which corresponds to ERRORCODE. For a list of these errors, see Section VI.

3-9. KEYCOMPARE PARAMETER

The KEYCOMPARE parameter is a user-supplied logical procedure which must be specified in your call to SORTINITIAL if you did not specify NUMKEYS and KEYS. If KEYCOMPARE is specified, it will be called from your program whenever two records must be compared. The form of this procedure is

```
LOGICAL PROCEDURE KEYCOMPARE
(REC1,LEN1,REC2,LEN2);
```

```
BYTE ARRAY REC1,REC2;
```

```
INTEGER LEN1,LEN2;
```

REC1 and REC2 are pointers to the two records. LEN1 and LEN2 are their respective lengths in bytes. The procedure will return a *true* value if REC1 is to precede REC2, and a *false* value otherwise. *True* also will be returned in case of ties, to ensure that records with equal keys retain their original order.

3-10. STATISTICS PARAMETER

STATISTICS is an array which, if specified, is filled with the following data:

WORDS 0 and 1 =
number of records sorted (double integer)

WORD 2 =
number of intermediate passes

WORD 3 =
space available for sorting

WORDS 4 and 5 =
number of comparisons (double integer)

WORDS 6 and 7 =
number of scratchfile inputs/outputs (double integer)

WORDS 8 and 9 =
cpu time used (milliseconds) (double integer)

WORDS 10 and 11 =
elapsed time (milliseconds) (double integer)

3-11. FAILURE PARAMETER

FAILURE is a logical variable which is set to -1 (true) if a fatal error occurred, and to 0 (false) otherwise. If this parameter is specified, it also is set after each call to SORTINPUT and SORTOUTPUT and, in addition, the condition code is set.

Error conditions:

CCE
no error occurred (FAILURE set to false)

CCL
error occurred (FAILURE set to true)

3-12. SORTINPUT PROCEDURE

If INPUTFILE was not specified in your SORTINITIAL call, your program must pass the input records, one at a time, to the Sort program by calling the SORTINPUT procedure. (The SORTINPUT procedure must not be called, however, if the INPUTFILE parameter (see paragraph 3-2) was specified in the SORTINITIAL call.)

The form of the SORTINPUT procedure is

```
PROCEDURE SORTINPUT(RECORD,LENGTH);
VALUE LENGTH;
ARRAY RECORD;
INTEGER LENGTH;
OPTION EXTERNAL;
```

where:

RECORD
is an array containing a data record.

LENGTH
is the (positive) number of characters in the record. Length must be long enough to contain all the keys which were specified, and may not be longer than the record size (RECLLEN) which was specified in SORTINITIAL.

Error conditions:

CCE
no error occurred (FAILURE set to false)

CCL
error occurred (FAILURE set to true)

3-13. SORTOUTPUT PROCEDURE

After SORTINPUT has been used to pass all the input records to the Sort program, the SORTOUTPUT procedure is used to pass the records to the OUTPUT file. (The SORTOUTPUT procedure must not be called, however, if the OUTPUTFILE parameter (see paragraph 3-3) was specified in the SORTINITIAL call.)

The form of the SORTOUTPUT procedure is

```
PROCEDURE SORTOUTPUT(RECORD,
LENGTH);
ARRAY RECORD;
INTEGER LENGTH;
OPTION EXTERNAL;
```

where:

RECORD
is an array into which the next output record is deposited, in the format specified by the OUTPUTOPTION parameter of SORTINITIAL.

LENGTH

is set to the positive number of characters returned in the record. When no more records remain, LENGTH is set to -1.

Error conditions:

CCE

no error occurred (FAILURE set to false)

CCL

error occurred (FAILURE set to true)

3-14. SORTEND PROCEDURE

The SORTEND procedure is called after all calls to OUTPUTFILE are completed by the SORTINITIAL procedure (if OUTPUTFILE was specified), or after all calls to SORTOUTPUT are completed (if the SORTOUTPUT procedure was called). The SORTEND procedure allows the Sort program to perform certain cleanup functions such as closing the SCRATCH file and restoring the stack to its original state. (For more details on the operation of the stack, see paragraph 3-19).

The SORTEND procedure must be called from the same procedure which called SORTINITIAL, so that the stack is restored properly.

The form of the SORTEND procedure is

PROCEDURE SORTEND;

OPTION EXTERNAL;

Error conditions:

CCE

no error occurred during the sort (FAILURE set to false)

CCL

an error occurred during the sort (FAILURE set to true)

3-15. SORTERRORMESS PROCEDURE

The SORTERRORMESS procedure is used only when you provide your own ERRORPROC procedure. The form of the SORTERRORMESS procedure is

PROCEDURE SORTERRORMESS(ERRORCODE,
MESSAGE,LENGTH);

VALUE ERRORCODE;

INTEGER ERRORCODE,LENGTH;

BYTE ARRAY MESSAGE;

OPTION EXTERNAL;

where:

ERRORCODE

is the Sort program error number which is passed to ERRORPROC when an error occurs.

MESSAGE

is a byte array into which the message is placed. The array must be at least 72 bytes long.

LENGTH

is the (positive) length of the message, in bytes.

3-16. SORTTITLE PROCEDURE

The SORTTITLE procedure prints the version of the SORTLIB segment which is being used, along with the date and time as produced by the library procedure DATELINE. The form of the SORTTITLE procedure is

PROCEDURE SORTTITLE;

OPTION EXTERNAL;

The SORTTITLE message appears on the standard list device as follows:

HP32214B.00.0 SORT/3000 TUE, Nov 21, 1974, 7:24 AM

3-17. CALLING THE SORT PROGRAM FROM SPL/3000

Intrinsic calls to perform a sort from an SPL/3000 program are shown in Figures 3-1 through 3-8. Note that the MAX-DATA = 4000 parameter is appended to the :PREP commands.

The four cases shown are:

Both INPUTFILE and OUTPUTFILE are specified in the SORTINITIAL call (Figure 3-1).

INPUTFILE specified but not OUTPUTFILE in the SORTINITIAL call (Figure 3-3).

OUTPUTFILE specified but not INPUTFILE in the SORTINITIAL call (Figure 3-5).

Neither INPUTFILE nor OUTPUTFILE are specified in the SORTINITIAL call (Figure 3-7).

```

:SPL SPLTEST1

PAGE 0001  HP32100A-04.6B

00001000 00000 0 $CONTROL USLINIT
00002000 00000 0 << SPL EXAMPLE NO. 1 >>
00003000 00000 0 << INPUTFILE AND OUTPUTFILE BOTH SPECIFIED >>
00004000 00000 0 << SORT THE FILE MAILLIST INTO TEST >>
00005000 00000 0 << SORT ON FIRST NAME WITHIN LAST NAME >>
00006000 00000 0 BEGIN
00007000 00000 1 BYTE ARRAY MAILLIST(0:8):="MAILLIST ";
00008000 00006 1 BYTE ARRAY TEST(0:4):="TEST ";
00009000 00004 1 ARRAY ERROR(0:6):="ERROR IN SORT";
00010000 00007 1 ARRAY BUF(0:35);
00011000 00007 1 ARRAY KEYS(0:5);
00012000 00007 1 INTEGER OPIN, OPOUT, LEN;
00013000 00007 1 INTRINSIC FOPEN, FREAD, FPOINT, PRINT;
00014000 00007 1 INTRINSIC SORTINITIAL, SORTEND;
00015000 00007 1 << OPEN FILES >>
00016000 00007 1 OPIN:=FOPEN(MAILLIST, %605, %305);
00017000 00010 1 OPOUT:=FOPEN(TEST, %605, %305);
00018000 00020 1 << ESTABLISH KEYS >>
00019000 00020 1 << MAJOR AT 11 FOR 9 BYTES (LAST NAME) >>
00020000 00020 1 << MINOR AT 1 FOR 10 BYTES (FIRST NAME) >>
00021000 00020 1 KEYS(0):=11;
00022000 00023 1 KEYS(1):=9;
00023000 00026 1 KEYS(2):=0;
00024000 00031 1 KEYS(3):=1;
00025000 00034 1 KEYS(4):=10;
00026000 00037 1 KEYS(5):=0;
00027000 00042 1 << CALL SORTINITIAL - OUTPUT OPTION = 0 >>
00028000 00042 1 SORTINITIAL(OPIN, OPOUT, 0, , 2, KEYS);
00029000 00053 1 SORTEND;
00030000 00054 1 IF <> THEN GOTO SORTERR;
00030100 00055 1 << RESET OUTPUTFILE TO RECORD 1 >>
00030200 00055 1 FPOINT(OPOUT, 0D);
00031000 00060 1 DISPLAY;
00032000 00060 1 LEN:=FREAD(OPOUT, BUF, 36);
00033000 00066 1 IF > THEN GOTO STOP;
00034000 00067 1 PRINT(BUF, LEN, 0);
00035000 00073 1 GOTO DISPLAY;
00036000 00101 1 SORTERR:
00037000 00101 1 PRINT(ERROR, 7, 0);
00038000 00105 1 STOP;
00039000 00105 1 END.
PRIMARY DB STORAGE=%010; SECONDARY DB STORAGE=%00071
NO. ERRORS=000; NO. WARNINGS=000
PROCESSOR TIME=0:00:03; ELAPSED TIME=0:03:48

END OF COMPILE

```



Figure 3-1. Program 1. SPL/3000 Program to Call Sort When Both INPUTFILE and OUTPUTFILE are Specified in the SORTINITIAL Call

```

:BUILD PROG1;CODE=PROG
:PREP $OLDPASS,PROG1;MAXDATA=4000

  END OF PREPARE
:RUN PROG1

LOIS      ANYONE  6190 COURT ST      METROPOLIS  NY 20115 619-732-4997
KING      ARTHUR  329 EXCALIBUR ST   CAMELOT    CA 61322 812-200-0100
ALI       BABA    40 THIEVES WAY      SESAME     CO 69142 NONE
JOHN      BIGTOWN 965 APPIAN WAY      METROPOLIS NY 20013 619-407-2314
KNEE     BUCKLER 974 FISTICUFF DR    PUGILIST   ND 04321 976-299-2990
SWASH    BUCKLER 497 PLAYACTING CT   MOVIETOWN  CA 61497 NONE
JAMES    DOE      4193 ANY ST         ANYTOWN    MD 00133 237-408-7100
JANE     DOE      3959 TREEWOOD LN    BIGTOWN    MA 21843 714-399-4563
JOHN     DOUGHE  239 MAIN ST        HOMETOWN   MA 26999 714-411-1123
JENNA    GRANDTR 493 TWENTIETH ST    PROGRESSIVE CA 61335 799-191-9191
KARISSA  GRANDTR 7917 BROADMOOR WAY  BIGTOWN    MA 21799 713-244-3717
SPACE    MANN    9999 GALAXY WAY     UNIVERSE   CA 61239 231-999-9999

  END OF PROGRAM

```

Figure 3-2. Program 1 Output

3-18. CALLING THE SORT PROGRAM FROM FORTRAN/3000

Figures 3-9 through 3-20 contain FORTRAN/3000 programs which perform the following: (Note that the MAXDATA = 4000 parameter is appended to the :PREP commands.)

- Sort from one file to another (Figure 3-9).
- Write records into a file in sorted order (Figure 3-11).
- Read records from a file in sorted order (file did not have the records in sorted order) for processing (Figure 3-13).
- Read key fields from a file in sorted order (file was not in sorted order) for a report (Figure 3-15).
- Create an index file consisting only of relative record numbers. The index file is ordered by the Sort program (Figure 3-17).
- Read the key field *and* its record number so that the entire field may be read if the key field satisfies certain conditions (Figure 3-19).

3-19. TREATMENT OF THE STACK

Note: The following information concerning the stack is presented for reference in case it is needed for advanced projects. It is not necessary to read these paragraphs in order to use the Sort and Merge programs.

During execution of the various procedures which comprise the Sort program, each procedure must be able to access the data generated by the other procedures. Because the procedures are called separately, normally they would be unable to share data. This is because the Q register (which marks the starting location for the currently executing procedure's data) changes every time a procedure is called. MPE/3000 avoids this problem by placing a four-word stack marker on the stack whenever a procedure is called. This marker indicates the beginning of the stack section generated for the current procedure. It also enables the current procedure to return to its calling procedure and it resets the Q register to point to the calling procedure's stack marker.

Figure 3-21-A represents the stack just before SORTINITIAL is called. The S register (top-of-stack) is represented by S_0 and is set to the top-of-stack before the SORTINITIAL call. The Z register (stack limit) is represented by Z_0 . When SORTINITIAL returns to the calling program, the stack will appear as shown in Figure 3-21-B. The Q register will be as it was before the call, but S_1 and Z_1 will have been moved up as much as possible in order to obtain room for the Sort program data area. This area must not be modified by the calling program. SORTINITIAL will always leave at least 1000 words between S (top-of-stack) and Z (the stack limit) for processing by the calling program. The procedures SORTINPUT and SORTOUTPUT will not cause any further modification of the stack. When SORTEND is called, it will restore the stack to the way it was in Figure 3-11-A. Care must be exercised, therefore, to call SORTEND from the same procedure which called SORTINITIAL. SORTINPUT and SORTOUTPUT may be called from other procedures.

For a complete explanation of the hardware stack structure and its use, refer to *System Reference Manual*.


```

: SPL SPLTEST2

PAGE 0001   HP32100A.04.6B

00001000  00000 0  $CONTROL USLINIT
00002000  00000 0  << SPL EXAMPLE NO. 2 >>
00003000  00000 0  << INPUTFILE SPECIFIED BUT NOT OUTPUTFILE >>
00004000  00000 0  << SORT THE FILE MAILLIST INTO TEST >>
00005000  00000 0  << SORT ON PHONE NUMBER WITHIN STATE >>
00006000  00000 0  BEGIN
00007000  00000 1  BYTE ARRAY MAILLIST(0:8):="MAILLIST ";
00008000  00006 1  BYTE ARRAY TEST(0:4):="TEST ";
00009000  00004 1  ARRAY ERROR(0:6):="ERROR IN SORT";
00010000  00007 1  ARRAY BUF(0:35);
00011000  00007 1  ARRAY KEYS(0:5);
00012000  00007 1  INTEGER OPIN, OPOUT, LEN:=36;
00013000  00007 1  INTRINSIC FOPEN, FREAD, FPOINT, PRINT, FWRITE;
00014000  00007 1  INTRINSIC SORTINITIAL, SORTEND, SORTOUTPUT;
00015000  00007 1  << OPEN FILES >>
00016000  00007 1  OPIN:=FOPEN(MAILLIST, %605, %305);
00017000  00010 1  OPOUT:=FOPEN(TEST, %605, %305);
00018000  00020 1  << ESTABLISH KEYS >>
00019000  00020 1  << MAJOR AT 52 FOR 2 BYTES (STATE) >>
00020000  00020 1  << MINOR AT 61 FOR 12 BYTES (PHONE NO) >>
00021000  00020 1  KEYS(0):=52;
00022000  00023 1  KEYS(1):=2;
00023000  00026 1  KEYS(2):=0;
00024000  00031 1  KEYS(3):=61;
00025000  00034 1  KEYS(4):=12;
00026000  00037 1  KEYS(5):=0;
00027000  00042 1  << CALL SORTINITIAL - OUTPUT OPTION = 0 >>
00028000  00042 1  << OUTPUTFILE NOT SPECIFIED >>
00029000  00042 1  SORTINITIAL(OPIN,,0,,2,KEYS);
00030000  00053 1  IF <> THEN GOTO ENDSORT;
00031000  00054 1  << CALL SORTOUTPUT >>
00032000  00054 1  OUTPUT:
00033000  00054 1  SORTOUTPUT(BUF,LEN);
00034000  00057 1  IF <> THEN GOTO ENDSORT;
00035000  00060 1  IF LEN >=0 THEN
00036000  00063 1  BEGIN
00037000  00063 2  FWRITE(OPOUT, BUF, 36, 0);
00038000  00070 2  GOTO OUTPUT;
00039000  00076 2  END;
00039100  00076 1  << RESET OUTPUTFILE TO RECORD 1 >>
00039200  00076 1  FPOINT(OPOUT, 0D);
00040000  00101 1  ENDSORT:
00041000  00101 1  SORTEND;
00042000  00102 1  IF <> THEN GOTO SORTERR;
00043000  00103 1  DISPLAY:
00044000  00103 1  LEN:=FREAD(OPOUT, BUF, 36);
00045000  00111 1  IF > THEN GOTO STOP;
00046000  00112 1  PRINT(BUF, LEN, 0);
00047000  00116 1  GOTO DISPLAY;
00048000  00117 1  SORTERR:
00049000  00117 1  PRINT(ERROR, 7, 0);
00050000  00123 1  STOP:
00051000  00123 1  END.
PRIMARY DB STORAGE=%010;  SECONDARY DB STORAGE=%00071
NO. ERRORS=000;          NO. WARNINGS=000
PROCESSOR TIME=0:00:04;  ELAPSED TIME=0:05:02

END OF COMPILE

```

Figure 3-3. Program 2. SPL/3000 Program to Call Sort When INPUTFILE is Specified in the SORTINITIAL Call but OUTPUTFILE is Not

```

:BUILD PROG2; CODE=PROG
:PREP SOLDPASS,PROG2;MAXDATA=4000

  END OF PREPARE
:RUN PROG2

SPACE      MANN      9999 GALAXY WAY      UNIVERSE    CA 61239 231-999-9999
JENNA     GRANDTR  493 TWENTIETH ST    PROGRESSIVE CA 61335 799-191-9191
KING      ARTHUR   329 EXCALIBUR ST     CAMELOT     CA 61322 812-200-0100
SWASH     BUCKLER   497 PLAYACTING CT    MOVIE TOWN  CA 61497 NONE
ALI       BABA      40 THIEVES WAY       SESAME      CO 69142 NONE
KARISSA   GRANDTR  7917 BROADMOOR WAY   BIGTOWN     MA 21799 713-244-3717
JANE      DOE        3959 TREEWOOD LN     BIGTOWN     MA 21843 714-399-4563
JOHN      DOUGHE   239 MAIN ST          HOMETOWN    MA 26999 714-411-1123
JAMES     DOE        4193 ANY ST          ANYTOWN     MD 00133 237-408-7100
KNEE     BUCKLER   974 FISTICUFF DR     PUGILIST    ND 04321 976-299-2990
JOHN     BIGTOWN   965 APPIAN WAY       METROPOLIS  NY 20013 619-407-2314
LOIS     ANYONE    6190 COURT ST        METROPOLIS  NY 20115 619-732-4997

  END OF PROGRAM

```

Figure 3-4. Program 2 Output

```

:SPL SPLTEST3

PAGE 0001   HP32100A.04.6B

00001000  00000 0  $CONTROL USLINIT
00002000  00000 0  << SPL EXAMPLE NO. 3 >>
00003000  00000 0  << OUTPUTFILE SPECIFIED BUT NOT INPUTFILE >>
00004000  00000 0  << SORT THE FILE MAILLIST INTO TEST >>
00005000  00000 0  << SORT ON ZIP CODES WITHIN STATE >>
00006000  00000 0  BEGIN
00007000  00000 1  BYTE ARRAY MAILLIST(0:8):="MAILLIST ";
00008000  00006 1  BYTE ARRAY TEST(0:4):="TEST ";
00009000  00004 1  ARRAY ERROR(0:6):="ERROR IN SORT";
00010000  00007 1  ARRAY BUF(0:39);
00011000  00007 1  ARRAY KEYS(0:5);
00012000  00007 1  INTEGER OPIN, OPOUT, LEN;
00013000  00007 1  INTRINSIC FOPEN, FREAD, FPOINT, PRINT;
00014000  00007 1  INTRINSIC SORTINITIAL, SORTEND, SORTINPUT;
00015000  00007 1  << OPEN FILES >>
00016000  00007 1  OPIN:=FOPEN(MAILLIST, %605, %305);
00017000  00010 1  OPOUT:=FOPEN(TEST, %605, %305);
00018000  00020 1  << ESTABLISH KEYS >>
00019000  00020 1  << MAJOR AT 52 FOR 2 BYTES (STATE) >>
00020000  00020 1  << MINOR AT 55 FOR 5 BYTES (ZIP CODE) >>
00021000  00020 1  KEYS(0):=52;
00022000  00023 1  KEYS(1):=2;
00023000  00026 1  KEYS(2):=0;
00024000  00031 1  KEYS(3):=55;
00025000  00034 1  KEYS(4):=5;
00026000  00037 1  KEYS(5):=0;
00027000  00042 1  << CALL SORTINITIAL - OUTPUT OPTION = 0 >>
00028000  00042 1  << INPUTFILE NOT SPECIFIED >>
00029000  00042 1  SORTINITIAL(, OPOUT, 0, 80, , 2, KEYS);
00030000  00054 1  IF <> THEN GOTO ENDSORT;
00031000  00055 1  << READ RECORD FROM INPUT FILE >>
00032000  00055 1  INPUT:
00033000  00055 1  LEN:=FREAD(OPIN, BUF, -80);
00034000  00063 1  IF > THEN GOTO ENDSORT;
00035000  00064 1  BEGIN
00036000  00064 2  << CALL SORTINPUT >>
00037000  00064 2  SORTINPUT(BUF, LEN);
00038000  00067 2  IF <> THEN GOTO ENDSORT;
00039000  00070 2  GOTO INPUT;
00040000  00076 2  END;
00041000  00076 1  ENDSORT:
00042000  00076 1  SORTEND;
00043000  00077 1  IF <> THEN GOTO SORTERR;
00043100  00100 1  << RESET OUTPUTFILE TO RECORD 1 >>
00043200  00100 1  FPOINT(OPOUT, 0D);
00044000  00103 1  DISPLAY:
00045000  00103 1  LEN:=FREAD(OPOUT, BUF, 40);
00046000  00111 1  IF > THEN GOTO STOP;
00047000  00112 1  PRINT(BUF, LEN, 0);
00048000  00116 1  GOTO DISPLAY;
00049000  00117 1  SORTERR:
00050000  00117 1  PRINT(ERROR, 7, 0);
00051000  00123 1  STOP:
00052000  00123 1  END.
PRIMARY DB STORAGE=%010; SECONDARY DB STORAGE=%00075
NO. ERRORS=000; NO. WARNINGS=000
PROCESSOR TIME=0:00:04; ELAPSED TIME=0:05:15

END OF COMPILE

```

Figure 3-5. Program 3. SPL/3000 Program to Call Sort When OUTPUTFILE is Specified in the SORTINITIAL Call but INPUTFILE is Not

```

:BUILD PROG3;CODE=PROG
:PREP $OLDPASS,PROG3;MAXDATA=4000

END OF PREPARE
:RUN PROG3

SPACE      MANN      9999 GALAXY WAY      UNIVERSE    CA 61239 231-999-9999
KING       ARTHUR   329 EXCALIBUR ST      CAMELOT     CA 61322 812-200-0100
JENNA     GRANDTR  493 TWENTIETH ST      PROGRESSIVE CA 61335 799-191-9191
SWASH     BUCKLER   497 PLAYACTING CT     MOVIE TOWN  CA 61497 NONE
ALI       BABA      40 THIEVES WAY        SESAME      CO 69142 NONE
KARISSA   GRANDTR  7917 BROADMOOR WAY    BIGTOWN     MA 21799 713-244-3717
JANE      DOE       3959 TREEWOOD LN      BIGTOWN     MA 21843 714-399-4563
JOHN      DOUGHE   239 MAIN ST           HOMETOWN    MA 26999 714-411-1123
JAMES     DOE      4193 ANY ST           ANYTOWN     MD 00133 237-408-7100
KNEE     BUCKLER  974 FISTICUFF DR      PUGILIST    ND 04321 976-299-2990
JOHN     BIGTOWN  965 APPIAN WAY        METROPOLIS  NY 20013 619-407-2314
LOIS     ANYONE   6190 COURT ST        METROPOLIS  NY 20115 619-732-4997

END OF PROGRAM

```

Figure 3-6. Program 3 Output

```

:SPL SPLTEST4

PAGE 0001    HP32100A.04.6B
00001000  00000 0  $CONTROL USLINIT
00002000  00000 0  << SPL EXAMPLE NO. 4 >>
00003000  00000 0  << NEITHER INPUTFILE NOR OUTPUTFILE SPECIFIED >>
00004000  00000 0  << SORT THE FILE MAILLIST INTO TEST >>
00005000  00000 0  << SORT ON FIRST NAME WITHIN LAST NAME >>
00006000  00000 0  BEGIN
00007000  00000 1  BYTE ARRAY MAILLIST(0:8):='MAILLIST '
00008000  00006 1  BYTE ARRAY TEST(0:4):='TEST '
00009000  00004 1  ARRAY ERROR(0:6):='ERROR IN SORT';
00010000  00007 1  ARRAY BUF(0:39);
00011000  00007 1  ARRAY KEYS(0:5);
00012000  00007 1  INTEGER OPIN, OPOUT, LEN;
00013000  00007 1  INTRINSIC FOPEN, FREAD, FWRITE, FPOINT, PRINT;
00014000  00007 1  INTRINSIC SORTINITIAL, SORTEND;
00015000  00007 1  INTRINSIC SORTINPUT, SORTOUTPUT;
00016000  00007 1  << OPEN FILES >>
00017000  00007 1  OPIN:=FOPEN(MAILLIST, %605, %305);
00018000  00010 1  OPOUT:=FOPEN(TEST, %605, %305);
00019000  00020 1  << ESTABLISH KEYS >>
00020000  00020 1  << MAJOR AT 11 FOR 9 BYTES (LAST NAME) >>
00021000  00020 1  << MINOR AT 1 FOR 10 BYTES (FIRST NAME) >>
00022000  00020 1  KEYS(0):=11;
00023000  00023 1  KEYS(1):=9;
00024000  00026 1  KEYS(2):=0;
00025000  00031 1  KEYS(3):=1;
00026000  00034 1  KEYS(4):=10;
00027000  00037 1  KEYS(5):=0;
00028000  00042 1  << CALL SORTINITIAL - OUTPUT OPTION = 0 >>
00029000  00042 1  << INPUTFILE AND OUTPUTFILE NOT SPECIFIED >>
00030000  00042 1  SORTINITIAL(.,,0,80,,2,KEYS);
00031000  00053 1  IF <> THEN GOTO ENDSORT;
00032000  00054 1  << READ RECORD FROM INPUT FILE >>
00033000  00054 1  INPUT:
00034000  00054 1  LEN:=FREAD(OPIN, BUF, -80);
00035000  00062 1  IF > THEN GOTO OUTPUT;
00036000  00063 1  << CALL SORTINPUT >>
00037000  00063 1  BEGIN
00038000  00063 2  SORTINPUT(BUF, LEN);
00039000  00066 2  IF <> THEN GOTO ENDSORT;
00039100  00067 2  END;
00039200  00067 1  GOTO INPUT;
00040000  00075 1  << CALL SORTOUTPUT >>
00040100  00075 1  OUTPUT:
00041000  00075 1  BEGIN
00043000  00075 2  SORTOUTPUT(BUF, LEN);
00044000  00100 2  IF <> THEN GOTO ENDSORT;
00045000  00101 2  IF LEN >=0 THEN
00046000  00104 2  BEGIN
00047000  00104 3  FWRITE(OPOUT, BUF, 40, 0);
00048000  00111 3  GOTO OUTPUT;
00049000  00112 3  END;
00049100  00112 2  END;
00050000  00112 1  ENDSORT;
00051000  00112 1  SORTEND;
00052000  00113 1  IF <> THEN GOTO SORTERR;
00052100  00114 1  << RESET OUTPUTFILE TO RECORD 1 >>
00052200  00114 1  FPOINT(OPOUT, 0D);
00053000  00117 1  DISPLAY:
00054000  00117 1  LEN:=FREAD(OPOUT, BUF, 40);
00055000  00125 1  IF > THEN GOTO STOP;
00056000  00126 1  PRINT(BUF, LEN, 0);
00057000  00132 1  GOTO DISPLAY;
00058000  00133 1  SORTERR:
00059000  00133 1  PRINT(ERROR, 7, 0);
00060000  00137 1  STOP;
00061000  00137 1  END.
PRIMARY DB STORAGE=%10; SECONDARY DB STORAGE=%00075
NO. ERRORS=000; NO. WARNINGS=000
PROCESSOR TIME=0:00:05; ELAPSED TIME=0:06:07

END OF COMPILE

```

Figure 3-7. Program 4. SPL/3000 Program to Call Sort When Neither INPUTFILE nor OUTPUTFILE are Specified in the SORTINITIAL Call

```

: BUILD PROG4; CODE=PROG
: PREP SOLDPASS, PROG4; MAXDATA=4000

END OF PREPARE
: RUN PROG4

LOIS      ANYONE  6190 COURT ST      METROPOLIS NY 20115 619-732-4997
KING      ARTHUR  329 EXCALIBUR ST   CAMELOT    CA 61322 812-200-0100
ALI       BABA    40 THIEVES WAY      SESAME     CO 69142 NONE
JOHN      BIGTOWN 965 APPIAN WAY      METROPOLIS NY 20013 619-407-2314
KNEE      BUCKLER 974 FISTICUFF DR    PUGILIST   ND 04321 976-299-2990
SWASH     BUCKLER 497 PLAYACTING CT   MOVIE TOWN CA 61497 NONE
JAMES     DOE      4193 ANY ST         ANYTOWN    MD 00133 237-408-7100
JANE      DOE      3959 TREEWOOD LN   BIGTOWN    MA 21843 714-399-4563
JOHN      DOUGHE  239 MAIN ST        HOMETOWN   MA 26999 714-411-1123
JENNA     GRANDTR 493 TWENTIETH ST   PROGRESSIVE CA 61335 799-191-9191
KARISSA   GRANDTR 7917 BROADMOOR WAY BIGTOWN    MA 21799 713-244-3717
SPACE     MANN     9999 GALAXY WAY    UNIVERSE   CA 61239 231-999-9999

END OF PROGRAM
    
```

Figure 3-8. Program 4 Output

```

PAGE 0001  HP32102B.00.01  FORTRAN/3000  (C) HEWLETT-PACKARD CO.

00001000  SCONTROL USLINIT,INIT,FILE=21,FILE=22
00002000          PROGRAM XMPL1
00003000          CHARACTER*72 BUF
00004000          INTEGER KEYS(6),FNUM
00005000          LOGICAL FAILURE
00006000          SYSTEM INTRINSIC SORTINITIAL,SORTEND
00007000  C
00008000  C SORT THE FILE MAILLIST (FTN21) INTO TEST (FTN22)
00009000  C SORT ON PHONE NUMBERS WITHIN STATES
00010000  C
00011000  C ESTABLISH THE KEYS - MAJOR AT 52 FOR 2 BYTES (STATE)
00012000  C MINOR AT 61 FOR 12 BYTES (PHONE NO)
00013000  C
00014000          KEYS(1)=52
00015000          KEYS(2)=2
00016000          KEYS(3)=0
00017000          KEYS(4)=61
00018000          KEYS(5)=12
00019000          KEYS(6)=0
00020000  C
00021000  C INITIALIZE SORT - OUTPUT OPTION = 0
00022000  C
00023000          CALL SORTINITIAL(FNUM(21),FNUM(22),,,,
00024000          #2,KEYS,,,,FAILURE)
00025000          IF(FAILURE)STOP 10
00026000          CALL SORTEND
00027000          IF(FAILURE)STOP 20
00028000  C
00029000  C READ AND DISPLAY OUTPUT FILE (STATE AND PHONE NO. ONLY)
00030000  C
00031000          REWIND 22
00032000  30  READ(22,END=100)BUF
00033000          DISPLAY BUF(52:21," ",BUF(61:12)
00034000          GO TO 30
00035000  100  STOP
00036000          END

```

PROGRAM UNIT XMPL1 COMPILED

```

****      GLOBAL STATISTICS      ****
**** NO ERRORS, NO WARNINGS ****
TOTAL COMPILATION TIME  0:00:02
TOTAL ELAPSED TIME     0:00:09

```

Figure 3-9. Program 5. FORTRAN/3000 Program to Sort from One File to Another

```
: BUILD PROG5; CODE=PROG
: PREP SOLDPASS, PROG5; MAXDATA=4000
```

```
END OF PREPARE
: FILE FTN21=MAILLIST, OLD
: FILE FTN22=TEST, OLD
: RUN PROG5
```

```
CA 231-999-9999
CA 799-191-9191
CA 812-200-0100
CA NONE
CO NONE
MA 713-244-3717
MA 714-399-4563
MA 714-411-1123
MD 237-408-7100
ND 976-299-2990
NY 619-407-2314
NY 619-732-4997
```

```
END OF PROGRAM
```

Figure 3-10. Program 5 Output


```

PAGE 0001  HP32102B.00.01  FORTRAN/3000  (C) HEWLETT-PACKARD CO.

00001000  $CONTROL USLIMIT,INIT,FILE=21,FILE=22
00002000      PROGRAM XMPL2
00003000      CHARACTER*72 BUF
00004000      INTEGER KEYS(6),FNUM
00005000      LOGICAL FAILURE,LBUF(36)
00006000      EQUIVALENCE (LBUF,BUF)
00007000      SYSTEM INTRINSIC SORTINITIAL,SORTINPUT,SORTEND
00008000  C
00009000  C WRITE RECORDS INTO FILE IN SORTED ORDER
00010000  C FTN21=MAILLIST (INPUT) AND FTN22= TEST (OUTPUT)
00011000  C SORT ON FIRST NAME WITHIN LAST NAME
00012000  C
00013000  C ESTABLISH KEYS - MAJOR AT 11 FOR 9 BYTES (LAST NAME)
00014000  C MINOR AT 1 FOR 10 BYTES (FIRST NAME)
00015000  C
00016000      KEYS(1)=11
00017000      KEYS(2)=9
00018000      KEYS(3)=0
00019000      KEYS(4)=1
00020000      KEYS(5)=10
00021000      KEYS(6)=0
00022000  C
00023000  C INITIALIZE SORT - OUTPUT OPTION = 0
00024000  C OUTPUT RECORD = INPUT RECORD
00025000  C SORT WILL WRITE RECORDS TO OUTPUT FILE ONLY
00026000  C
00027000      CALL SORTINITIAL(,FNUM(22),,,,2,
00028000      #KEYS,,,,FAILURE)
00029000      IF (FAILURE)STOP 10
00030000  C
00031000  C READ THE INPUT FILE (MAILLIST) TO SIMULATE
00032000  C GENERATION OF RECORDS
00033000  C
00034000      50  READ(21,END=100)BUF
00035000  C
00036000  C CALL SORTINPUT TO INPUT RECORD TO THE SORTED FILE
00037000  C
00038000      CALL SORTINPUT(LBUF,72)
00039000      IF (FAILURE)STOP 20
00040000      GO TO 50
00041000      100 CONTINUE
00042000      CALL SORTEND
00043000      IF (FAILURE)STOP 30
00044000  C
00045000  C READ AND DISPLAY OUTPUT FILE (FIRST 54 CHARACTERS)
00046000  C
00047000      REWIND 22
00048000      60  READ(22,END=200)BUF
00049000      DISPLAY BUF[1:54]
00050000      GO TO 60
00051000      200 STOP
00052000      END

```

PROGRAM UNIT XMPL2 COMPILED

```

****      GLOBAL STATISTICS      ****
****      NO ERRORS,      NO WARNINGS      ****
TOTAL COMPILATION TIME  0:00:02
TOTAL ELAPSED TIME      0:00:12

```

Figure 3-11. Program 6. FORTRAN/3000 Program to Write Records into a File in Sorted Order

```

:BUILD PROG6;CODE=PROG
:PREP $OLDPASS,PROG6;MAXDATA=4000

END OF PREPARE
:FILE FTN21=MAILLIST,OLD
:FILE FTN22=TEST,OLD
:RUN PROG6

LOIS      ANYONE  6190 COURT ST      METROPOLIS  NY
KING      ARTHUR  329 EXCALIBUR ST     CAMELOT     CA
ALI       BABA    40 THIEVES WAY        SESAME      CO
JOHN     BIGTOWN  965 APPIAN WAY        METROPOLIS  NY
KNEE     BUCKLER  974 FISTICUFF DR      PUGILIST    ND
SWASH    BUCKLER  497 PLAYACTING CT     MOVIE TOWN  CA
JAMES    DOE      4193 ANY ST           ANYTOWN     MD
JANE     DOE      3959 TREEWOOD LN     BIGTOWN     MA
JOHN     DOUGHE  239 MAIN ST           HOMETOWN    MA
JENNA    GRANDTR  493 TWENTIETH ST     PROGRESSIVE CA
KARISSA  GRANDTR  7917 BROADMOOR WAY    BIGTOWN     MA
SPACE    MANN    9999 GALAXY WAY       UNIVERSE    CA

END OF PROGRAM
    
```

Figure 3-12. Program 6 Output

PAGE 0001 HP32102B.00.01 FORTRAN/3000 (C) HEWLETT-PACKARD CO.

```

00001000 $CONTROL USLINIT,INIT,FILE=21,FILE=22
00002000     PROGRAM XMPL3
00003000     CHARACTER*72 BUF
00004000     INTEGER KEYS(9),FNUM
00005000     LOGICAL FAILURE,LBUF(36)
00006000     EQUIVALENCE (LBUF,BUF)
00007000     SYSTEM INTRINSIC SORTINITIAL,SortOUTPUT,SortEND
00008000 C
00009000 C READ RECORDS FROM A FILE IN SORTED ORDER
00010000 C SORT ON FIRST NAME WITHIN LAST NAME WITHIN TOWN
00011000 C
00012000 C ESTABLISH KEYS - MAJOR AT 40 FOR 12 BYTES (TOWN)
00013000 C MINOR AT 11 FOR 9 BYTES (LAST NAME)
00014000 C MINOR AT 1 FOR 10 BYTES (FIRST NAME)
00015000 C
00016000     KEYS(1)=40
00017000     KEYS(2)=12
00018000     KEYS(3)=0
00019000     KEYS(4)=11
00020000     KEYS(5)=9
00021000     KEYS(6)=0
00022000     KEYS(7)=1
00023000     KEYS(8)=10
00024000     KEYS(9)=0
00025000 C
00026000 C INITIALIZE SORT - OUTPUT OPTION = 0
00027000 C SORT WILL READ THE INPUT FILE ONLY
00028000 C
00029000     CALL SORTINITIAL(FNUM(21),,,,,,
00030000     #KEYS,,,,FAILURE)
00031000     IF(FAILURE)STOP 10
00032000 C
00033000 C CALL SortOUTPUT TO DISPLAY THE SORTED RECORDS
00034000 C
00035000 50  CALL SortOUTPUT(LBUF,LEN)
00036000     IF(LEN.EQ.-1)GO TO 60
00037000     IF(FAILURE)STOP 20
00038000     DISPLAY BUF(1:59)
00039000     GO TO 50
00040000 60  CONTINUE
00041000     CALL SortEND
00042000     IF(FAILURE)STOP 30
00043000     STOP
00044000     END

```



PROGRAM UNIT XMPL3 COMPILED

```

****      GLOBAL STATISTICS      ****
**** NO ERRORS, NO WARNINGS ****
TOTAL COMPILATION TIME  0:00:02
TOTAL ELAPSED TIME      0:00:12

```

Figure 3-13. Program 7. FORTRAN/3000 Program to Read Records from a File for Processing

```
: BUILD PROG7; CODE=PROG
: PREP $OLDPASS, PROG7; MAXDATA=4000

END OF PREPARE
: FILE FTN21=MAILLIST, OLD
: FILE FTN22=TEST, OLD
: RUN PROG7

JAMES   DOE      4193 ANY ST      ANYTOWN   MD 00133
JANE    DOE      3959 TREEWOOD LN BIGTOWN   MA 21543
KARISSA GRANDTR  7917 BROADMOOR WAY BIGTOWN   MA 21799
KING    ARTHUR   329 EXCALIBUR ST  CAMELOT  CA 61322
JOHN    DOUGHE   239 MAIN ST      HOMETOWN MA 26999
LOIS    ANYONE   6190 COURT ST    METROPOLIS NY 20115
JOHN    BIGTOWN  965 APPIAN WAY   METROPOLIS NY 20013
SWASH   BUCKLER   497 PLAYACTING CT MOVIE TOWN CA 61497
JENNA   GRANDTR  493 TWENTIETH ST PROGRESSIVE CA 61335
KNEE    BUCKLER   974 FISTICUFF DR  PUGILIST ND 04321
ALI     BABA     40 THIEVES WAY   SESAME   CO 69142
SPACE   MANN     9999 GALAXY WAY  UNIVERSE CA 61239

END OF PROGRAM
```

Figure 3-14. Program 7 Output

```

PAGE 0001  HP32102B.00.01  FORTRAN/3000  (C) HEWLETT-PACKARD CO.

00001000  $CONTROL USLINIT,INIT,FILE=21,FILE=22
00002000      PROGRAM XMPL4
00003000      CHARACTER*9 BUF,NAME
00004000      INTEGER KEYS(3),FNUM
00005000      LOGICAL FAILURE,LBUF(5)
00006000      EQUIVALENCE (LBUF,BUF)
00007000      SYSTEM INTRINSIC SORTINITIAL,SORTOUTPUT,SORTEND
00008000      10  FORMAT(/T20," NAME",6X,"NUMBER"/)
00009000      20  FORMAT(T20,S,T30,I3)
00010000      30  FORMAT(/T25,"TOTAL = ",I3////)
00011000  C
00012000  C PRINT A SORTED REPORT OF ALL THE LAST NAMES IN MAILLIST
00013000  C THE NUMBER OF TIMES A NAME APPEARS
00014000  C AND THE TOTAL NUMBER OF NAMES IN THE FILE
00015000  C
00016000  C PRINT HEADING
00017000  C
00018000      WRITE(6,10)
00019000  C
00020000  C ESTABLISH KEY
00021000  C
00022000      KEYS(1)=11
00023000      KEYS(2)=9
00024000      KEYS(3)=0
00025000  C
00026000  C INITIALIZE SORT - OUTPUT OPTION = 2
00027000  C OUTPUT = KEY FIELD ONLY
00028000  C
00029000      CALL SORTINITIAL(FNUM(21),,2,,,
00030000      #1,KEYS,,,FAILURE)
00031000      IF(FAILURE)STOP 100
00032000      50  CALL SORTOUTPUT(LBUF,LEN)
00033000      IF(LEN.EQ.-1)GO TO 500
00034000      IF(FAILURE)STOP 200
00035000  C
00036000  C REPORT GENERATION SECTION
00037000  C
00038000      ITOTAL=ITOTAL+1
00039000      IF(BUF.EQ.NAME)GO TO 60
00040000      IF(ICTR.EQ.0)GO TO 70
00041000      WRITE(6,20)NAME,ICTR
00042000      70  NAME=BUF
00043000      ICTR=0
00044000      60  ICTR=ICTR+1
00045000      GO TO 50
00046000      500 WRITE(6,30)ITOTAL
00047000  C
00048000  C END OF REPORT GENERATION
00049000  C
00050000      CALL SORTEND
00051000      IF(FAILURE)STOP 300
00052000      STOP
00053000      END

```

PROGRAM UNIT XMPL4 COMPILED

```

****      GLOBAL STATISTICS      ****
****      NO ERRORS,   NO WARNINGS  ****
TOTAL COMPILATION TIME  0:00:02
TOTAL ELAPSED TIME     0:00:09

```

Figure 3-15. Program 8. FORTRAN/3000 Program to Read Key Fields from a File in Sorted Order

```
:BUILD PROG8; CODE=PROG
:PREP $OLDPASS, PROG8; MAXDATA=4000
```

```
END OF PREPARE
:FILE FTN21=MAILLIST, OLD
:FILE FTN22=TEST, OLD
:RUN PROG8
```

NAME	NUMBER
ANYONE	1
ARTHUR	1
BABA	1
BIGTOWN	1
BUCKLER	2
DOE	2
DOUGHE	1
GRANDTR	2

TOTAL = 12

END OF PROGRAM

Figure 3-16. Program 8 Output

```

PAGE 0001  HP32102B.00.01  FORTRAN/3000  (C) HEWLETT-PACKARD CO.

00001000  $CONTROL USLINIT,INIT,FILE=21,FILE=22
00002000      PROGRAM XMPL5
00003000      CHARACTER*72 BUF
00004000      INTEGER KEYS(6),FNUM
00005000      LOGICAL FAILURE
00006000      DOUBLE PRECISION DFLOAT
00007000      SYSTEM INTRINSIC SORTINITIAL,SORTEND
00008000  C
00009000  C BUILD AN INDEX FILE (FTN22) FROM MAILLIST (FTN21)
00010000  C CONSISTING OF DOUBLE-WORD RECORD POINTERS
00011000  C THE INDEX WILL BE SORTED ON ZIP CODES WITHIN STATES
00012000  C
00013000  C ESTABLISH THE KEYS - MAJOR AT 52 FOR 2 BYTES (STATE)
00014000  C MINOR AT 55 FOR 5 BYTES (ZIP CODE)
00015000  C
00016000      KEYS(1)=52
00017000      KEYS(2)=2
00018000      KEYS(3)=0
00019000      KEYS(4)=55
00020000      KEYS(5)=5
00021000      KEYS(6)=0
00022000  C
00023000  C INITIALIZE SORT - OUTPUT OPTION = 1
00024000  C OUTPUT RECORD = DOUBLE-WORD RECORD NUMBER ONLY
00025000  C SORT WILL READ/WRITE THE FILES
00026000  C
00027000      CALL SORTINITIAL(FNUM(21),FNUM(22),1,,,
00028000      #2,KEYS,,,,FAILURE)
00029000      IF(FAILURE)STOP 10
00030000      CALL SORTEND
00031000      IF(FAILURE)STOP 20
00032000  C
00033000  C CHECK INDEX FILE BY USING IT TO READ MAILLIST
00034000  C
00035000      REWIND 22
00036000      50  READ(22,END=100)DUMMY
00037000  C
00038000  C RECORD NUMBERS IN INDEX START AT ZERO
00039000  C FORTRAN/3000 ACCEPTS RECORD NUMBERS STARTING AT ONE
00040000  C THUS ADD ONE TO RECORD NUMBERS IN INDEX
00041000  C
00042000      READ(21@DFLOAT(\DUMMY\)+1.DO)BUF
00043000      DISPLAY BUF[1:19]," ",BUF[52:2]," ",BUF[55:5]
00044000      GO TO 50
00045000      100  STOP
00046000      END

```

PROGRAM UNIT XMPL5 COMPILED

```

****      GLOBAL STATISTICS      ****
****      NO ERRORS,      NO WARNINGS      ****
TOTAL COMPILATION TIME  0:00:02
TOTAL ELAPSED TIME      0:00:07

```

Figure 3-17. Program 9. FORTRAN/3000 Program to Create an Index File Consisting of Relative Record Numbers Only

```
: BUILD PROG9; CODE=PROG
: PREP $OLDPASS, PROG9; MAXDATA=4000

END OF PREPARE
: FILE FTN21=MAILLIST, OLD
: FILE FTN22=TEST, OLD
: RUN PROG9

SPACE      MANN      CA      61239
KING       ARTHUR   CA      61322
JENNA     GRANDTR   CA      61335
SWASH     BUCKLER    CA      61497
ALI       BABA      CO      69142
KARISSA   GRANDTR   MA      21799
JANE      DOE       MA      21843
JOHN     DOUGHE   MA      26999
JAMES    DOE       MD      00133
KNEE     BUCKLER   ND      04321
JOHN     BIGTOWN  NY      20013
LOIS     ANYONE   NY      20115

END OF PROGRAM
```

Figure 3-18. Program 9 Output

PAGE 0001 HP32102B.00.01 FORTRAN/3000 (C) HEWLETT-PACKARD CO.

```

00001000 $CONTROL USLIMIT,INIT,FILE=21,FILE=22
00002000     PROGRAM XMP6
00003000     CHARACTER*72 BUF,CITY*12
00004000     INTEGER KEYS(9),FNUM
00005000     LOGICAL LBUF(23),FAILURE
00006000     DOUBLE PRECISION DFLOAT
00007000     EQUIVALENCE (CITY,LBUF(3)),(DUMMY,LBUF(1))
00008000     SYSTEM INTRINSIC SORTINITIAL,SORTOUTPUT,SORTEND
00009000     10  FORMAT(//T11,"NAME",T40,"CITY",T57,"PHONE NO.");//
00010000     20  FORMAT(2X,S,T37,S,T54,S)
00011000     C
00012000     C PRINT A REPORT OF ALL THOSE LIVING IN
00013000     C METROPOLIS, PROGRESSIVE, AND BIGTOWN
00014000     C ORDER THE REPORT BY FIRST NAME WITHIN LAST NAME
00015000     C WITHIN CITIES
00016000     C
00017000     C PRINT HEADING
00018000     C
00019000     WRITE(6,10)
00020000     C
00021000     C ESTABLISH KEYS - MAJOR AT 40 FOR 12 BYTES (CITY)
00022000     C MINOR AT 11 FOR 9 BYTES (LAST NAME)
00023000     C MINOR AT 1 FOR 10 BYTES (FIRST NAME)
00024000     C
00025000     KEYS(1)=40
00026000     KEYS(2)=12
00027000     KEYS(3)=0
00028000     KEYS(4)=11
00029000     KEYS(5)=9
00030000     KEYS(6)=0
00031000     KEYS(7)=1
00032000     KEYS(8)=10
00033000     KEYS(9)=0
00034000     C
00035000     C INITIALIZE SORT - OUTPUT OPTION = 3
00036000     C OUTPUT RECORD = DOUBLE-WORD RECNUM + KEYS
00037000     C
00038000     CALL SORTINITIAL(FNUM(21),,3,,3,
00039000     #KEYS,,,FAILURE)
00040000     IF(FAILURE)STOP 100
00041000     50  CALL SORTOUTPUT(LBUF,LEN)
00042000     IF(LEN.EQ.-1)GO TO 60
00043000     IF(FAILURE)STOP 200
00044000     C
00045000     C PRINT DESIRED FIELDS IF THE CITY MATCHES
00046000     C
00047000     IF(CITY.NE."METROPOLIS ".AND.
00048000     # CITY.NE."PROGRESSIVE ".AND.
00049000     # CITY.NE."BIGTOWN ")GO TO 50
00050000     READ(21@DFLOAT(\DUMMY\)+1.DO)BUF
00051000     WRITE(6,20)BUF[1:19],BUF[40:12],BUF[61:12]
00052000     GO TO 50
00053000     60  CALL SORTEND
00054000     STOP
00055000     END

```

PROGRAM UNIT XMP6 COMPILED

```

**** GLOBAL STATISTICS ****
**** NO ERRORS, NO WARNINGS ****
TOTAL COMPILATION TIME 0:00:02
TOTAL ELAPSED TIME 0:00:08

```

Figure 3-19. Program 10. FORTRAN/3000 Program to Read Key Field and Its Record Number

```

: BUILD PROG10; CODE=PROG
: PREP $OLDPASS, PROG10; MAXDATA=4000

END OF PREPARE
: FILE FTN21=MAILLIST, OLD
: FILE FTN22=TEST, OLD
: RUN PROG10

      NAME                CITY                PHONE NO.

JANE   DOE                BIGTOWN                714-399-4563
KARISSA GRANDTR          BIGTOWN                713-244-3717
LOIS   ANYONE            METROPOLIS            619-732-4997
JOHN   BIGTOWN           METROPOLIS            619-407-2314
JENNA  GRANDTR           PROGRESSIVE           799-191-9191

END OF PROGRAM

```

Figure 3-20. Program 10 Output

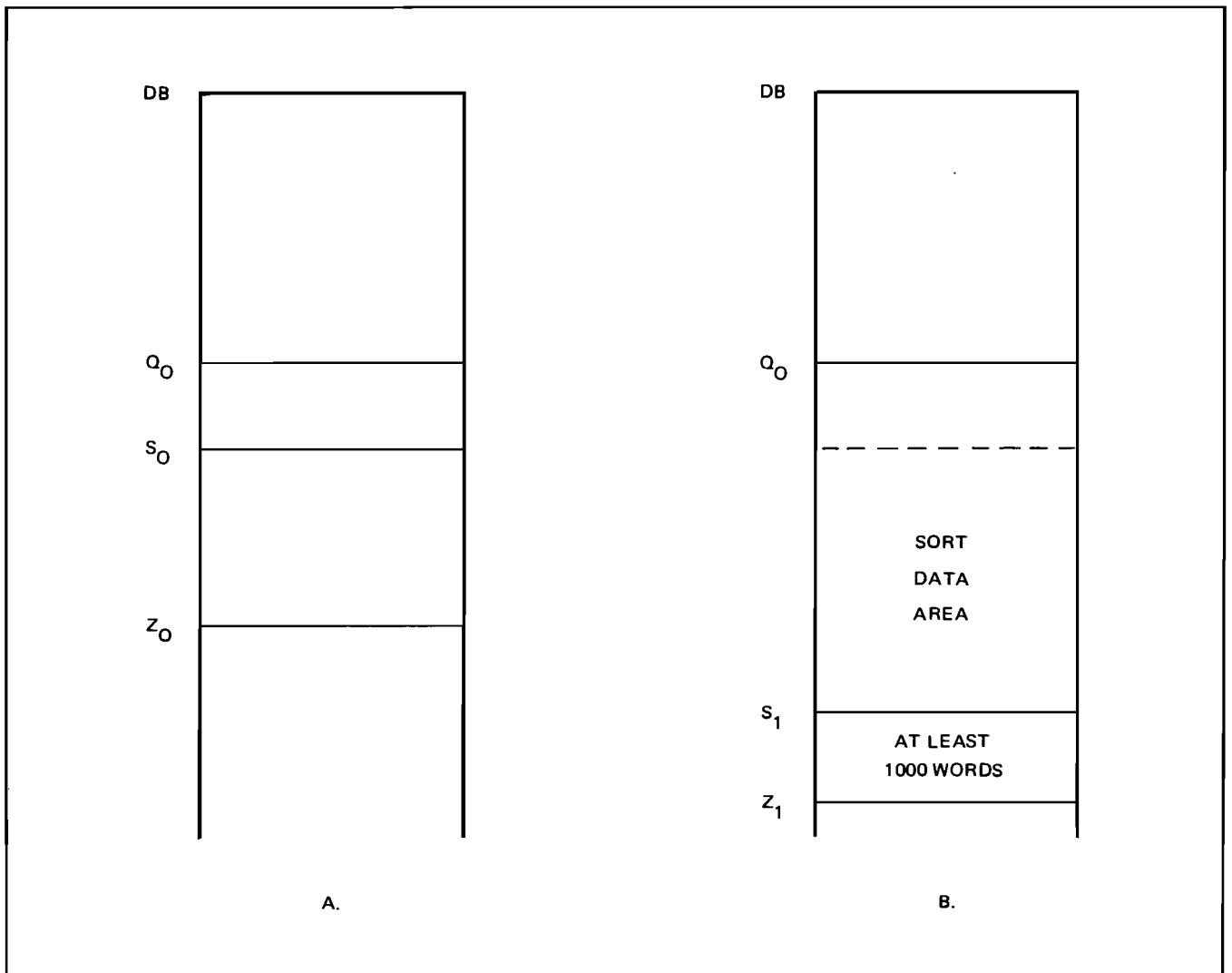


Figure 3-21. Data Stack Layout During Execution of the Sort Program



RUNNING THE MERGE PROGRAM AS A STAND-ALONE PROGRAM

SECTION

IV

The Merge program provides the capability of merging several files, each of which has been sorted independently, and producing a single sorted file. The Merge program can be run as a stand-alone program during a batch job or an interactive session.

The Merge program operates in the minimum HP 3000 memory size. Performance can be improved, however, by increasing the amount of memory available if the files being merged and/or the output file reside on the same device. A larger stack will enable Merge to read/write more records consecutively from/to the same file, thereby reducing the number of disc arm movements. You can increase the stack size by appending the `MAXDATA = segsize` parameter to the `RUN MERGE.PUB.SYS` command. (This is not necessary unless you receive the "INSUFFICIENT STACK SPACE" error message when you attempt to run the Merge program.) A `segsize` of 10,000 should be sufficient for merging most files. If the error message occurs again, however, increase the `segsize` parameter.

4-1. FILE DEFINITIONS

The stand-alone Merge program references various files during execution. These files are described in the following paragraphs. For a complete understanding of the treatment of files by the operating system, read the *MPE Commands Reference Manual*.

4-2. INPUT FILES

The INPUT files contain the records to be merged.

When running the Merge program, you must equate the formal designator INPUT with the actual file designator (*filename*). This is done with the Merge program INPUT command, as follows:

```
>INPUT MASTER,UPDATE
(The > symbol is output by the Merge program in a
session)
```

The INPUT files are opened as follows:

Formal designator:
INPUT

Foptions:
Domain = old or oldtemp; others default.

Aoptions:
Read-access only; others default.

All other parameters:
Default.

Note: The files are opened automatically by MPE/3000 and you need not be concerned with the file parameters. They are included here (and in paragraph 4-3) for reference only.

4-3. OUTPUT FILE

The merged records are written to the OUTPUT file. It is opened as follows:

Formal designator:
OUTPUT

Foptions:
Domain = new; ASCII/BINARY (same as the first INPUT file); record format is *fixed* if all the INPUT files are *fixed* and have the same record size or if the KEY option is specified in the OUTPUT command, otherwise *variable* is used; others default.

Aoptions:
Write-access only; others default.

Recsize:
If the KEY option is specified in the OUTPUT command, the recsize is the sum of the keyfield lengths; otherwise, the recsize is the maximum recsize of the INPUT files.

Device:
DISC

Blockfactor:
If all the INPUT files have the same blockfactor and the KEY option is not specified in the OUTPUT command, that blockfactor is used. Otherwise, the system default is used.

Filesize:
If all the INPUT files are disc files, the sum of their end-of-files is used. Otherwise, each of the INPUT files which are not disc files is assumed to have 10,000 records. The filesize parameter may be overridden by a Merge program OUTPUT command.

Initialloc:
All extents are initially allocated.

Filecode:
Same as the first INPUT file.

All other parameters:
Default.

The OUTPUT file can be specified by using an MPE/3000 :FILE command or by using a Merge program OUTPUT command, as follows:

```
:FILE OUTPUT=OUTFILE
(The colon is output by MPE/3000 in a session)
```

or

```
>OUTPUT OUTFILE
(The > symbol is output by the Merge program in a session)
```

The format of the output records depends on whether or not the KEY option is specified.

The OUTPUT file is closed with the option SAVE, and, if it was opened as a new file, space beyond the end-of-file is released. If the close fails due to another file having the same name, the interactive user is asked if the old file can be purged, as follows:

```
PURGE OLD OUTPUT FILE filename?
```

If the response is "Y", the old file will be purged. If this cannot be done (for example, you cannot purge a file which is not in your group account, or a file which is being used concurrently by another user), or if the answer was "N", you will be asked for a new name for the OUTPUT file:

```
ENTER NEW NAME FOR OUTPUT FILE:
```

The OUTPUT file will be renamed using the name entered by you, and another attempt will be made to save it under this name.

In batch mode, the foregoing sequence cannot take place. Instead, the OUTPUT file will be assigned an artificial name constructed by the Merge program and the OUTPUT file will be saved under that name. The following message will be printed on the LIST file:

```
OUTPUT FILE SAVED WITH FILENAME
"filename"
```

Note: If \$STDLIST is specified in the OUTPUT command (meaning that the merged records will be sent to the LIST file), the OUTPUT file will not be saved. (See paragraph 4-9.)

4-4. LIST FILE

The LIST file is used by the Merge program to output information to you and to prompt for input (if in interactive session). The LIST file should not be confused with the OUTPUT file (which is used to output the merged records). The LIST file normally is equated with \$STDLIST.

4-2

4-5. TEXT FILE

The TEXT file is used by the Merge program to read commands and other information from you. Normally, the TEXT file is equated with \$STDINX.

4-6. PROMPT FILE

The PROMPT file is used by the Merge program to prompt you for input when the TEXT file is the session terminal but the LIST file is not. The prompt is with a "greater than" (>) symbol.

4-7. MERGE COMMANDS

The Merge program is controlled by commands which specify INPUT files, key fields, OUTPUT files, output options, etc. In batch mode, these commands are entered with the RUN MERGE.PUB.SYS command for the Merge program; in interactive session, the Merge program outputs a prompt character (>) after it begins execution and the commands are typed in from the terminal.

4-8. INPUT COMMAND

The INPUT command is used to specify the names of the files which are to be merged. The form of the INPUT command is

```
INPUT { filename } [filename] [filename] [, . . .]
```

where

filename
is any legal formal designator.

Each of the files to be merged is assumed to be ordered correctly according to the specified keys. *The order in which the files are specified is relevant in that records with equal keys will be ordered according to files in which they appear.*

If more than one INPUT command is entered, only the last one entered will be significant. Thus, all files to be merged must be specified in a single INPUT command.

An example of the INPUT command is

```
INPUT MASTER,UPDATE
```

4-9. OUTPUT COMMAND

The OUTPUT command specifies the file to which the merged records will be written. The form of the OUTPUT command is

```
OUTPUT { filename } [,number of records] [,KEY]
```

where

filename

is any legal formal designator. Unlike the Sort program, the Merge program does not allow the use of an asterisk (*) to send the merged records to the LIST file. You can use the *filename* \$STDLIST, however, if you want the merged records listed on the output device. (The OUTPUT file will not be saved in this case.)

number of records

is a positive integer.

KEY

specifies that the output option will consist of the key fields.

If *number of records* is specified, this value will be used as the *filesize* parameter when opening the OUTPUT file, if any of the INPUT files are not disc files. This parameter is ignored if all INPUT files are disc files.

If **KEY** is specified, the output records will consist of the key fields only.

If more than one OUTPUT command is entered, only the last one entered will be significant.

Examples of the OUTPUT command are

```
OUTPUT NEWMASTER,KEY
```

```
OUTPUT MYFILE, 50000
```

4-10. KEY COMMAND

The KEY command specifies the key fields to be used when the input records are merged and is used in the same manner as with the Sort program (see paragraph 2-12). The form of a KEY command is

```
KEY { position } [,length] [,type] [,DESC]
    [;position [,length] [,type] [,DESC]]
```

where:

position

is a positive integer.

length

is a positive integer.

type

is one of the following mnemonics: BYTE, INT, DOUBLE, REAL, LONG, PACKED, DISPLAY, or PACKED*. (See the meanings of the mnemonics in Section II, paragraph 2-12.)

DESC

indicates that the records are to be merged in descending order.

As shown in the example of the KEY command form, each KEY command can specify one or more key fields, with the specifications being separated by semicolons. Multiple key fields also can be specified with several KEY commands. In either case, the first key field specified is the most significant, or *major* key, with subsequent fields having less significance.

The *position* parameter indicates the number within the record of the first position of the key field (the first position of the record is numbered 1).

The *length* parameter indicates the number of bytes in the field. This parameter is required if *type* = BYTE, PACKED, DISPLAY, or PACKED*. It is optional in other cases. If not specified, the value assumed for length depends on the *type*: 2 if *type* = INT; 4 if *type* = DOUBLE or REAL; and 8 if *type* = LONG.*

The *type* parameter defines the type of data contained in the key field. If not specified, BYTE is assumed. The types are explained in Section II, paragraph 2-12.

Examples of the KEY command are

```
KEY 10,5
```

BYTE key of length 5 starting in position 10, to be merged in ascending order.

```
KEY 20,REAL
```

REAL key of length 4 starting in position 20, to be merged in ascending order.

```
KEY 30,20,INT,DESC
```

20-byte INTEGER key starting in position 30, to be merged in descending order.

```
KEY 10,5;20,REAL;30,20,INT,DESC
```

This last example is equivalent to the first three.

4-11. RESET COMMAND

The RESET command indicates that any KEY commands which have been entered thus far are to be ignored and that a new set will be entered. (This command is useful if you make a typing error.) The form of the RESET command is

```
RESET
```

4-12. VERIFY COMMAND

The VERIFY command causes a listing of the options in effect. The form of the VERIFY command is

```
VERIFY
```

The format of the listing is as follows:

```
INPUT FILES = MASTER, UPDATE
```

```
OUTPUT FILE = NEWMASTER
```

*For 3000 systems which are not Series II, the *type* = LONG has a *length* of 6.

KEY POSITION	LENGTH	TYPE	ASC/DESC	
50	5	BYTE	ASC	(MAJOR KEY)
1	10	DISPLAY	DESC	
21	9	PACKED	ASC	

4-13. END COMMAND

The END command indicates that there are no more commands and that the merge should begin. The form of the END command is

```
END
```

4-14. STATISTICS

When a merge is completed, the Merge program prints statistics, as follows:

```
NUMBER OF INPUT FILES = 3
NUMBER OF RECORDS = 100,000
RECORD SIZE (IN BYTES) = 100
SPACE AVAILABLE (IN WORDS) = 15,325
NUMBER OF COMPARES = 167,012
CPU TIME (MINUTES) = 3.25
ELAPSED TIME (MINUTES) = 9.73
```

Parameters whose meanings are not self evident are:

SPACE AVAILABLE
the number of words in the working space for the Merge procedures.

NUMBER OF COMPARES
the number of comparisons made between records.

CPU TIME
CPU time expended between the start and end of the Merge program.

ELAPSED TIME
real time between start and end of the Merge program.

4-15. CONTROL Y

During the running of the stand-alone Merge program, it is possible to obtain the status of the merge by typing CONTROL Y. This feature is available only if the program is being run in interactive mode. The format of the output is as follows:

```
1234 RECORDS HAVE BEEN OUTPUT
```

4-16. RUNNING THE STAND-ALONE MERGE PROGRAM IN INTERACTIVE SESSION

Figures 4-1 and 4-2 illustrate how to run the stand-alone Merge program during an interactive session. Note that the MAXDATA = *segsiz*e parameter is not used in the :RUN MERGE.PUB.SYS commands (which is normal unless exceptionally large files are being merged).

In Figure 4-1, the INPUT files specified were MAIL1 and MAIL2 and the OUTPUT file was specified as MAIL3. The merged OUTPUT file (MAIL3) will be saved as MAIL3 on disc.

In Figure 4-2, \$STDLIST was specified as the OUTPUT file and the merged records were listed on the terminal (the merged file will not be saved in this case).

Note: The Merge program does not allow the use of the asterisk (*) to send the merged records to the LIST file. \$STDLIST must be used, therefore, if you want the merged output records listed on the terminal.

4-17. RUNNING THE STAND-ALONE MERGE PROGRAM IN BATCH MODE

Figure 4-3 demonstrates the use of the stand-alone Merge program in batch mode.

The commands used were as follows:

```
:JOB MANAGER.SCR
:RUN MERGE.PUB.SYS
INPUT MAIL1,MAIL2
OUTPUT $STDLIST
KEY 11,9
KEY 1,10
END
:EOJ
```



```
! RUN MERGE.PUB.SYS
HP32214B.00.00 MERGE/3000  FRI, JAN 17, 1975,  2:57 PM
> INPUT MAIL1,MAIL2
> OUTPUT MAIL3
> KEY 11,9;1,10
> END
PURGE OLD OUTPUT FILE MAIL3.PUB.GOODWIN ? YES

                STATISTICS

NUMBER OF INPUT FILES =                2
NUMBER OF RECORDS =                    25
RECORD SIZE (IN BYTES) =                72
SPACE AVAILAEL (IN WORDS) =            13,368
NUMBER OF COMPARES =                    21
CPU TIME (MINUTES) =                    .01
ELAPSED TIME (MINUTES) =                .07

END OF PROGRAM
```



Figure 4-1. Running the Stand-Alone Merge Program in Interactive Session

:RUN MERGE.PUB.SYS

HP32214B.00.00 MERGE/3000 TUE, JAN 21, 1975, 10:40 A4

>INPUT MAIL1,MAIL2
 >OUTPUT \$STDLIST
 >KEY 11,9
 >KEY 1,10
 >END

PLAINS	ANTELOPE	201 OPENSACE AVE	BIGCOUNTRY	WY	49301	369-732-4321
LOIS	ANYONE	6193 COURT ST	METROPOLIS	NY	20115	619-732-4997
KING	ARTHUR	329 EXCALIBUR ST	CAMELOT	CA	61322	812-200-0100
ALI	BABA	40 THIEVES WAY	SESAME	CO	69142	NONE
BLACK	BEAR	47 ALLOVER DR	ANYWHERE	US	00111	NONE
JOHN	BIGTOWN	965 APPIAN WAY	METROPOLIS	NY	20013	619-407-2314
KNEE	BUCKLER	974 FISTICUFF DR	PUGILIST	ND	04321	976-299-2990
SWASH	BUCKLER	497 PLAYACTING CT	MOVIETOWN	CA	61497	NONE
ANIMAL	CRACKERS	1000 ANYWHERE PL	ALLOVER	US	00001	001-100-1000
MULE	DEER	963 FOREST PL	HIGHCOUNTRY	CA	97643	900-493-9000
WHITETAIL	DEER	34 WOODSY PL	BACKCOUNTRY	ME	01341	619-433-4333
JAMES	DOE	4193 ANY ST	ANYTOWN	MD	00133	237-408-7100
JANE	DOE	3959 TREEWOOD LN	EIGTOWN	MA	21843	714-399-4563
PRAIRIE	DOG	493 ROLLINGHILLS DR	OPENSACE	ND	24321	992-419-4192
JOHN	DOUGHE	239 MAIN ST	HOMETOWN	MA	26999	714-411-1123
MALLARD	DUCK	79 MARSH PL	PUDDLEDUCK	CA	97432	492-492-4922
JENNA	GRANDTR	493 TWENTIETH ST	PROGRESSIVE	CA	61335	799-191-9191
KARISSA	GRANDTR	7917 BROADMOOR WAY	BIGTOWN	MA	21799	713-244-3717
SNOWSHOE	HARE	742 FRIGID WAY	COLDSPOT	MN	37434	732-732-7320
MOUNTAIN	LION	796 KING DR	THICKET	NM	37643	712-712-7122
SPACE	MANN	9999 GALAXY WAY	UNIVERSE	CA	61239	231-999-9999
SWAMP	RABBIT	4444 DAMPLACE RD	BAYOU	LO	79999	NONE
NASTY	RATTLER	243 DANGER AVE	DESERTVILLE	CA	87654	828-432-4321
BIGHORN	SHEEP	999 MOUNTAIN DR	HIGHPLACE	CO	34567	776-409-9040
GREY	SQUIRREL	432 PLEASANT DR	FALLCOLORS	MA	14321	619-619-6199

STATISTICS

NUMBER OF INPUT FILES =	2
NUMBER OF RECORDS =	25
RECORD SIZE (IN BYTES) =	72
SPACE AVAILABLE (IN WORDS) =	13,368
NUMBER OF COMPARES =	21
CPU TIME (MINUTES) =	.01
ELAPSED TIME (MINUTES) =	3.50

END OF PROGRAM

Figure 4-2. Use of \$STDLIST to List the Merged Records on the Terminal

```

:JOB   MANAGER.SCR, PUB
PRI= DS; INPRI= 13; TIME= ?
JOB NUMRER = #J15
WED, JAN 15, 1975,  1:31 PM
HP32000C.F0.25

:RUN MERGE.PUB.SYS

HP32214B.00.00 MERGE/3000  WED, JAN 15, 1975,  1:31 PM

INPUT MAIL1,MAIL2
OUTpUT $STDLIST
KEY 11,9
KEY 1,10
END

PLAINS   ANTELOPE 201 OPENSACE AVE   BIGCOUNTRY WY 49301 369-732-4821
LOIS     ANYONE  6190 COURT ST     METHOPOLIS NY 20225 619-732-4997
KING     ARTHUR  329 EXCALIBUR ST    CAMELOT    CA 61322 812-200-0100
ALI      BABA    40 THIEVES WAY        SESAME     CO 69142 NONE
BLACK   BEAR    47 ALLOVER DR        ANYWHERE   US 00111 NONE
JOHN    RIGTOWN 965 APPIAN WAY        METHOPOLIS NY 20113 619-407-2314
KNEE    BUCKLER 974 FISTICUFF DR     PUGILIST   ND 04321 976-299-2990
SWASH   BUCKLER 497 PLAYACTING CT    MOVIETOWN  CA 61497 NONE
ANIMAL  CRACKERS 1000 ANYWHERE PL     ALLOVER    US 00001 001-100-1000
MULE    DEER    963 FOREST PL        NICECOUNTRY CA 97643 493-900-9000
WHITETA DEER    34 WOODSY PL        BACKCOUNTRY ME 01341 619-433-4333
JAMES   DOE      4193 ANY ST          ANYTOWN    MD 00133 237-408-7100
JANE    DOE      3959 TREEWOOD LN    BIGTOWN    MA 21843 714-399-4563
PRAIRIE DOG     493 ROLLINGHILLS DR OPENSACE   ND 24321 992-419-4192
JOHN    DOUGHE  239 MAIN ST         HOMETOWN   MA 26999 714-411-1123
MALLARD DUCK    79 MARSH PL        PUDDLEDUCK CA 97432 492-492-4922
JENNA   GRANDTR 493 TWENTIETH ST    PROGRESSIVE CA 61335 799-191-9191
KARISSA GRANDTR 7917 BROADMOOR WAY  BIGLOWN    MA 21799 713-244-3717
SNOWSHOE HARE    742 FRIGID WAY     COLDSPOT   MN 37434 732-732-7320
MOUNTAIN LION   796 KING DR        THICKET    NM 37643 712-712-7122
SPACE   MANN    9999 GALAXY WAY     UNIVERSE   CA 61239 231-999-9999
SWAMP   RABBIT  4444 DAMPLPLACE RD  BAYUU     LO 79999 NONE
NASTY   RATTLER 243 DANGER AVE     DESERTVILLE CA 87654 828-432-4321
BIGHORN SHEEP   999 MOUNTAIN DR    HIGHPLACE  CO 34567 776-409-9040
GREY    SQUIRREL 432 PLEASANT DR    FALLCOLORS MA 14321 619-619-6199

          STATISTICS

NUMBER OF INPUT FILES =                2
NUMBER OF RECORDS =                    25
RECORD SIZE (IN BYTES) =                80
SPACE AVAILABLE (IN WORDS) =            13,368
NUMBER OF COMPARES =                    21
CPU TIME (MINUTES) =                    .01
ELAPSED TIME (MINUTES) =                .02

          END OF PROGRAM
          IEOJ

          CPU (SEC) = 5
          ELAPSED (MIN) = 1
          WED, JAN 15, 1975,  1:31 PM
          END OF JOB

```

Figure 4-3. Running the Stand-Alone Merge Program in Batch Mode



RUNNING THE MERGE PROGRAM FROM A USER PROGRAM

SECTION

V

You can run the Merge program from an SPL/3000 or FORTRAN/3000 program through the use of intrinsic calls. There are three programmatically callable intrinsics, as follows:

MERGE

MERGEERRORMESS

MERGETITLE

These intrinsics (in the form of SPL/3000 procedures) reside in the system segmented library (SL.PUB.SYS) in the segment MERGELIB.

The procedures are described in the following paragraphs and their forms (in SPL/3000) are shown. It is necessary for you to know the forms of the various procedures so that you may provide parameters in the correct order (and of the correct type) in your intrinsic calls from SPL/3000 and FORTRAN/3000. See the *System Programming Language Reference Manual* for further descriptions of SPL/3000 procedures.

The Merge program operates in the minimum HP 3000 memory size. Performance can be improved, however, by increasing the amount of memory available if the files being merged and/or the output file reside on the same device. A larger stack will enable Merge to read/write more records consecutively from/to the same file, thereby reducing the number of disc arm movements. When you are calling the Merge intrinsics from your program, you must ensure that Merge obtains sufficient stack space by using the MAX-DATA = *segsz* parameter when your program is being prepared. A *segsz* of 4000 was used in the examples provided in this section (see paragraphs 5-15 and 5-16).

5-1. MERGE PROCEDURE

The MERGE procedure is called to merge one or more files, each of which has been sorted previously. The form of the MERGE procedure is

```
PROCEDURE MERGE
  (NUMINPUTFILES,INPUTFILES,
   OUTPUTFILE,KEYSONLY,NUMKEYS,KEYS,
   PREPROCESSOR,POSTPROCESSOR,ERRORPROC,
   KEYCOMPARE,STATISTICS,FAILURE);
VALUE NUMINPUTFILES,OUTPUTFILE,
KEYSONLY,NUMKEYS;
INTEGER NUMINPUTFILES,OUTPUTFILE,
NUMKEYS;
LOGICAL KEYSONLY,FAILURE;
INTEGER ARRAY INPUTFILES,KEYS,STATISTICS;
```

```
PROCEDURE PREPROCESSOR,POSTPROCESSOR,
ERRORPROC;
LOGICAL PROCEDURE KEYCOMPARE;
OPTION VARIABLE, EXTERNAL;
```

The parameters for the MERGE procedure are described in the following paragraphs.

5-2. NUMINPUTFILES PARAMETER

NUMINPUTFILES is the number of input files which are to be merged. This parameter must be at least one, and is not optional.

5-3. INPUTFILES PARAMETER

INPUTFILES is an array which contains the MPE/3000 file numbers of the files to be merged. These file numbers appear in locations INPUTFILES(0) through INPUTFILES(NUMINPUTFILES - 1). This parameter is not optional.

5-4. OUTPUTFILE PARAMETER

OUTPUTFILE is the MPE/3000 file number of the file to which the merged records are to be written. If OUTPUTFILE is not specified in your MERGE call, the records are not written anywhere. In that case, the parameter POSTPROCESSOR (see paragraph 5-8) must be specified.

5-5. KEYSONLY PARAMETER

KEYSONLY indicates whether entire records or keyfields only will be output. If KEYSONLY is true, only the keyfields, concatenated together with the major key on the left, are output. Otherwise, the entire record is output. If KEYSONLY is true, the parameter KEYCOMPARE may not be specified. The default for KEYSONLY is false.

5-6. NUMKEYS AND KEYS PARAMETERS

NUMKEYS and KEYS specify how the records are to be ordered. If either is specified, the other must be also. These parameters must be specified *if and only if* KEYCOMPARE is not. NUMKEYS is the number of keys to be used in the compare. It must be at least one. KEYS is an array which specifies how the records are to be compared. It contains three words for each key field, as follows:

WORD 0 =
position within record of the first character of the key.
(The first character of the record is position 1.)

WORD 1 =
number of bytes in the key.

WORD 2 (bits 0 through 7) =
0 for ascending key, 1 for descending key.

WORD 2 (bits 8 through 15) =
gives type of data, as follows:

- 0 =
logical or character
- 1 =
two's complement (including integer and double)
- 2 =
floating point
- 3 =
packed decimal
- 4 =
numeric display
- 5 =
packed decimal with even number of digits

5-7. PREPROCESSOR PARAMETER

PREPROCESSOR is a procedure which, if specified, is called whenever a record is read from any of the input files. The form of this procedure is

```
PROCEDURE PREPROCESSOR(FILE,RECORD,
    LENGTH);

INTEGER FILE,LENGTH;

BYTE ARRAY RECORD;
```

where

FILE
is the index to the array INPUTFILES of the file from which the record was read. This index has a value between 0 and NUMINPUTFILES - 1.

RECORD
is the data record.

LENGTH
is the number of characters in the record.

5-8. POSTPROCESSOR PARAMETER

POSTPROCESSOR is a procedure which, if specified, is called before each record is sent to the OUTPUTFILE. Either this parameter or OUTPUTFILE (or both) must be specified. The form of this procedure is

```
PROCEDURE POSTPROCESSOR (RECORD,
    LENGTH);
INTEGER LENGTH;

BYTE ARRAY RECORD;
```

where

RECORD
is the data record.

LENGTH
is the number of characters in the record.

5-9. ERRORPROC PARAMETER

The ERRORPROC procedure is a user-supplied procedure, and, if specified in your call to MERGE, must be called from your program and used in conjunction with the MERGEERRORMESS procedure (see paragraph 5-13).

ERRORPROC, if specified, is called programmatically whenever a fatal error occurs in the MERGE procedure. The form of this procedure is

```
PROCEDURE ERRORPROC(ERRORCODE);

INTEGER ERRORCODE;
```

ERRORCODE is the Merge program error number which is passed to ERRORPROC when an error occurs. If ERRORPROC is not specified, a default procedure is used. The default procedure simply prints an error message which corresponds to ERRORCODE. For a list of these errors, see Section VI.

5-10. KEYCOMPARE PARAMETER

The KEYCOMPARE parameter is a user-supplied logical procedure which must be specified in your call to MERGE if you did not specify NUMKEYS and KEYS. If KEYCOMPARE is specified, it will be called from your program whenever two records must be compared. The form of this procedure is

```
LOGICAL PROCEDURE KEYCOMPARE
    (REC1,LEN1,REC2,LEN2);

BYTE ARRAY REC1,REC2;

INTEGER LEN1,LEN2;
```

where

REC1 and REC2
are pointers to the two records.

LEN1 and LEN2
are the lengths of the respective records in bytes.

The KEYCOMPARE procedure will return a true value if REC1 is to precede REC2, and a false value otherwise. True also will be returned in case of ties. This ensures that records from input files earlier in the list will precede those from input files later in the list.

5-11. STATISTICS PARAMETER

STATISTICS is an array which, if specified, is filled with the following data:

WORD 0 =
number of input files.

WORDS 1 and 2 =
number of records merged (double integer)

WORD 3 =
space available for merging.

WORD 4 and 5 =
number of comparisons (double integer).

WORDS 6 and 7 =
CPU time (milliseconds) (double integer).

WORDS 8 and 9 =
elapsed time (milliseconds) (double integer).

5-12. FAILURE PARAMETER

FAILURE is a logical variable which, if specified, is set to -1 (true) if a fatal error occurred, and to 0 (false) otherwise.

Error conditions:

CCE
no error occurred (FAILURE set to false)

CCG
an error occurred (FAILURE set to true)

5-13. MERGEERRORMESS PROCEDURE

The MERGEERRORMESS procedure is used to convert MERGE error codes into ASCII strings. This procedure normally is used only when you provide your own ERRORPROC procedure. The form of the MERGEERRORMESS procedure is

```
PROCEDURE MERGEERRORMESS
  (ERRORCODE,MESSAGE,LENGTH);
```

```
VALUE ERRORCODE;
```

```
INTEGER ERRORCODE,LENGTH;
```

```
BYTE ARRAY MESSAGE;
```

```
OPTION EXTERNAL;
```

where:

ERRORCODE

is the MERGE error number which is passed to ERRORPROC when an error occurs.

MESSAGE

is a byte array into which the message is placed. It must be at least 72 bytes long.

LENGTH

is the (positive) length of the message, in bytes.

5-14. MERGETITLE PROCEDURE

The MERGETITLE procedure prints the version of the MERGELIB segment which is being used, along with the date and time as produced by the library procedure DATELINE. The form of the MERGETITLE procedure is

```
PROCEDURE MERGETITLE;
```

```
OPTION EXTERNAL;
```

The MERGETITLE message appears on the job list device as follows:

```
HP32214B.00.00 MERGE/3000 WED,JAN 22, 1975, 8:43 AM
```

5-15. CALLING THE MERGE PROGRAM FROM SPL/3000

Figure 5-1 provides an example of how to call Merge from an SPL/3000 program. Note that the MAXDATA = 4000 parameter is appended to the :PREP command.

5-16. CALLING THE MERGE PROGRAM FROM FORTRAN/3000

When calling Merge from a FORTRAN/3000 program, the same conventions must be observed as explained in Section II, paragraph 3-18.

Figure 5-3 provides an example of how to call Merge from FORTRAN/3000. Note that the MAXDATA = 4000 parameter is appended to the :PREP command.

```

: SPL SPLTEST5

PAGE 0001  HP32100A.04.6B

00001000 00000 0 $CONTROL USLIMIT
00002000 00000 0 << SPL MERGE EXAMPLE >>
00003000 00000 0 << MERGE FILES MAIL1 AND MAIL2 >>
00004000 00000 0 << (WHICH HAVE BEEN SORTED) >>
00005000 00000 0 << INTO ONE FILE (TEST) >>
00006000 00000 0 BEGIN
00007000 00000 1 BYTE ARRAY MAIL1(0:5):="MAIL1 ";
00008000 00004 1 BYTE ARRAY MAIL2(0:5):="MAIL2 ";
00009000 00004 1 BYTE ARRAY TEST(0:4):="TEST ";
00010000 00004 1 ARRAY ERROR(0:6):="ERROR IN MERGE";
00011000 00007 1 ARRAY BUF(0:35);
00012000 00007 1 ARRAY KEYS(0:5);
00013000 00007 1 INTEGER ARRAY INFILES(0:1);
00014000 00007 1 INTEGER OPOUT,LEN;
00015000 00007 1 LOGICAL FAILURE;
00016000 00007 1 INTRINSIC FOPEN, FREAD, FPOINT, PRINT, MERGE;
00017000 00007 1 << OPEN FILES >>
00018000 00007 1 INFILES(0):=FOPEN(MAIL1, %605, %305);
00019000 00011 1 INFILES(1):=FOPEN(MAIL2, %605, %305);
00020000 00022 1 OPOUT:=FOPEN(TEST, %605, %305);
00021000 00032 1 << ESTABLISH KEYS >>
00022000 00032 1 << MAJOR AT 11 FOR 9 BYTES (LAST NAME) >>
00023000 00032 1 << MINOR AT 1 FOR 10 BYTES (FIRST NAME) >>
00024000 00032 1 KEYS(0):=11;
00025000 00035 1 KEYS(1):=9;
00026000 00040 1 KEYS(2):=0;
00027000 00043 1 KEYS(3):=1;
00028000 00046 1 KEYS(4):=10;
00029000 00051 1 KEYS(5):=0;
00030000 00054 1 << CALL MERGE >>
00031000 00054 1 MERGE(2, INFILES, OPOUT, , 2, KEYS);
00032000 00065 1 IF <> THEN GOTO MERGERR;
00033000 00066 1 << OUTPUT MERGED FILE >>
00033100 00066 1 << RESET OUTPUTFILE TO RECORD 1 >>
00033200 00066 1 FPOINT(OPOUT, 0D);
00034000 00071 1 DISPLAY:
00035000 00071 1 BEGIN
00036000 00071 2 LEN:=FREAD(OPOUT, BUF, 36);
00037000 00077 2 IF > THEN GOTO STOP;
00038000 00100 2 PRINT(BUF, LEN, 0);
00039000 00104 2 GOTO DISPLAY;
00040000 00114 2 END;
00041000 00114 1 MERGERR:
00042000 00114 1 PRINT(ERROR, 7, 0);
00043000 00120 1 STOP;
00044000 00120 1 END.
PRIMARY DB STORAGE=%012; SECONDARY DB STORAGE=%00074
NO. ERRORS=000; NO. WARNINGS=000
PROCESSOR TIME=0:00:04; ELAPSED TIME=0:04:51

END OF COMPILE

```

Figure 5-1. Program 11. Calling the Merge Program from SPL/3000


```

:BUILD PROG11;CODE=PROG
:PREP $OLDPASS,PROG11;MAXDATA=4000

END OF PREPARE
:RUN PROG11

PLAINS ANTELOPE 201 OPENSACE AVE BIGCOUNTRY WY 49301 369-732-4821
LOIS ANYONE 6190 COURT ST METROPOLIS NY 20115 619-732-4997
KING ARTHUR 329 EXCALIBUR ST CAMELOT CA 61322 812-200-0100
ALI BABA 40 THIEVES WAY SESAME CO 69142 NONE
BLACK BEAR 47 ALLOVER DR ANYWHERE US 00111 NONE
JOHN BIGTOWN 965 APPIAN WAY METROPOLIS NY 20013 619-407-2314
KNEE BUCKLER 974 FISTICUFF DR PUGILIST ND 04321 976-299-2990
SWASH BUCKLER 497 PLAYACTING CT MOVIE TOWN CA 61497 NONE
ANIMAL CRACKERS 1000 ANYWHERE PL ALLOVER US 00001 001-100-1000
MULE DEER 963 FOREST PL HIGHCOUNTRY CA 97643 900-493-9000
WHITETAIL DEER 34 WOODSY PL BACKCOUNTRY ME 01341 619-433-4333
JAMES DOE 4193 ANY ST ANYTOWN MD 00133 237-408-7100
JANE DOE 3959 TREEWOOD LN BIGTOWN MA 21843 714-399-4563
PRAIRIE DOG 493 ROLLINGHILLS DR OPENSACE ND 24321 992-419-4192
JOHN DOUGHE 239 MAIN ST HOMETOWN MA 26999 714-411-1123
MALLARD DUCK 79 MARSH PL PUDDLE DUCK CA 97432 492-492-4922
JENNA GRANDTR 493 TWENTIETH ST PROGRESSIVE CA 61335 799-191-9191
KARISSA GRANDTR 7917 BROADMOOR WAY BIGTOWN MA 21799 713-244-3717
SNOWSHOE HARE 742 FRIGID WAY COLDSPOT MN 37434 732-732-7320
MOUNTAIN LION 796 KING DR THICKET NM 37643 712-712-7122
SPACE MANN 9999 GALAXY WAY UNIVERSE CA 61239 231-999-9999
SWAMP RABBIT 4444 DAMPLACE RD BAYOU LO 79999 NONE
NASTY RATTLER 243 DANGER AVE DESERTVILLE CA 87654 828-432-4321
BIGHORN SHEEP 999 MOUNTAIN DR HIGHPLACE CO 34567 776-409-9040
GREY SQUIRREL 432 PLEASANT DR FALLCOLORS MA 14321 619-619-6199

END OF PROGRAM

```



Figure 5-2. Program 11 Output

```

:FORTRAN XMPL7
PAGE 0001   HP32102A.01.4

00001000  $CONTROL INIT,FILE=20,FILE=21,FILE=22
00002000      PROGRAM XMPL7
00003000      CHARACTER BUF*72
00004000      INTEGER KEYS(6),FNUM,INFILES(2)
00005000      LOGICAL FAILURE
00006000  C
00007000  C MERGE TWO SORTED FILES (MAIL1 (FTN20) AND MAIL2 (FTN21))
00008000  C INTO A THIRD FILE (MAIL3 (FTN22))
00009000  C
00010000  C ESTABLISH KEYS - MAJOR AT 11 FOR 9 BYTES (LAST NAME)
00011000  C MINOR AT 1 FOR 10 BYTES (FIRST NAME)
00012000  C
00013000      KEYS(1)=11
00014000      KEYS(2)=9
00015000      KEYS(3)=0
00016000      KEYS(4)=1
00017000      KEYS(5)=10
00018000      KEYS(6)=0
00019000  C
00020000  C ESTABLISH NUMBERS FOR INPUT FILES (MAIL1 AND MAIL2)
00021000  C
00022000      INFILES(1)=FNUM(20)
00023000      INFILES(2)=FNUM(21)
00024000  C
00025000  C INITIALIZE MERGE - 2 KEYS ARE SPECIFIED
00026000  C
00027000      CALL MERGE(\2\,INFILES,\FNUM(22)\,\0\,\2\,KEYS,
00028000      #\0\,\0\,\0\,\0\,\0\,\0\,FAILURE,\%7301\
00029000      IF(FAILURE)STOP 10
00030000  C
00031000  C READ AND DISPLAY OUTPUT FILE
00032000  C
00033000      REWIND 22
00034000  20  READ(22,END=30)BUF
00035000      DISPLAY BUF[1:72]
00036000      GO TO 20
00037000  30  STOP
00038000      END

**** NO ERRORS, NO WARNINGS; PROGRAM UNIT COMPILED ****
COMPILATION TIME  1.699 SECONDS  ELAPSED TIME  155.120 SECONDS
TOTAL COMPILATION TIME  0:00:02
TOTAL ELAPSED TIME  0:02:49

END OF COMPILE

```

Figure 5-3. Program 12. Calling the Merge Program from FORTRAN/3000

```

:BUILD PROG12;CODE=PROG
:PREP $OLDPASS,PROG12;MAXDATA=4000

```

```

END OF PREPARE
:FILE FTN20=MAIL1,OLD
:FILE FTN21=MAIL2,OLD
:FILE FTN22=MAIL3,OLD
:RUN PROG12

```

PLAINS	ANTELOPE	201	OPENSACE AVE	BIGCOUNTRY	WY	49301	369-732-4821
LOIS	ANYONE	6190	COURT ST	METROPOLIS	NY	20115	619-732-4997
KING	ARTHUR	329	EXCALIBUR ST	CAMELOT	CA	61322	812-200-0100
ALI	BABA	40	THIEVES WAY	SESAME	CO	69142	NONE
BLACK	BEAR	47	ALLOVER DR	ANYWHERE	US	00111	NONE
JOHN	BIGTOWN	965	APPIAN WAY	METROPOLIS	NY	20013	619-407-2314
KNEE	BUCKLER	974	FISTICUFF DR	PUGILIST	ND	04321	976-299-2990
SWASH	BUCKLER	497	PLAYACTING CT	MOVIETOWN	CA	61497	NONE
ANIMAL	CRACKERS	1000	ANYWHERE PL	ALLOVER	US	00001	001-100-1000
MULE	DEER	963	FOREST PL	HIGHCOUNTRY	CA	97643	900-493-9000
WHITETAIL	DEER	34	WOODSY PL	BACKCOUNTRY	ME	01341	619-433-4333
JAMES	DOE	4193	ANY ST	ANYTOWN	MD	00133	237-408-7100
JANE	DOE	3959	TREEWOOD LN	BIGTOWN	MA	21843	714-399-4563
PRAIRIE	DOG	493	ROLLINGHILLS DR	OPENSACE	ND	24321	992-419-4192
JOHN	DOUGHE	239	MAIN ST	HOMETOWN	MA	26999	714-411-1123
MALLARD	DUCK	79	MARSH PL	PUDDLEDUCK	CA	97432	492-492-4922
JENNA	GRANDTR	493	TWENTIETH ST	PROGRESSIVE	CA	61335	799-191-9191
KARISSA	GRANDTR	7917	BROADMOOR WAY	BIGTOWN	MA	21799	713-244-3717
SNOWSHOE	HARE	742	FRIGID WAY	COLDSPOT	MN	37434	732-732-7320
MOUNTAIN	LION	796	KING DR	THICKET	NM	37643	712-712-7122
SPACE	MANN	9999	GALAXY WAY	UNIVERSE	CA	61239	231-999-9999
SWAMP	RABBIT	4444	DAMPLACE RD	BAYOU	LO	79999	NONE
NASTY	RATTLER	243	DANGER AVE	DESERTVILLE	CA	87654	828-432-4321
BIGHORN	SHEEP	999	MOUNTAIN DR	HIGHPLACE	CO	34567	776-409-9040
GREY	SQUIRREL	432	PLEASANT DR	FALLCOLORS	MA	14321	619-619-6199

```

END OF PROGRAM

```



Figure 5-4. Program 12 Output



ERROR MESSAGES AND RECOVERY PROCEDURES

SECTION

VI

6-1. SORT PROGRAM ERROR MESSAGES

Table 6-1 contains a list of error numbers and the corresponding messages which are output by the SORTLIB segment of the system segmented library. Those messages which are marked by I/O in the second column of the table

also will result in a file information display. Those which contain LIB in the second column will not be printed by the stand-alone Sort program but will be printed (if an error occurs) if the Sort library intrinsics are being called programmatically. Each of the messages consists of the characters "SORTLIB:", followed by the remainder of the message.

Table 6-1. SORTLIB Error Messages

ERROR NUMBER	TYPE OF ERROR	MESSAGE
1	LIB	IF KEYCOMPARE IS SPECIFIED, KEYS AND NUMKEYS MUST NOT BE
2	LIB	IF KEYCOMPARE IS NOT SPECIFIED, KEYS AND NUMKEYS MUST BE
3	LIB	NO RECLN PARAMETER SPECIFIED OR ≤ 0
4	LIB	KEYCOMPARE MAY NOT BE SPECIFIED IF OUTPUTOPTION > 1
5	I/O	FREAD ERROR ON SCRATCHFILE
6	LIB	ILLEGAL OUTPUTOPTION
7	I/O	SCRATCH FILE CANNOT BE OPENED
8	LIB,I/O	FAILURE ON FGETINFO(INPUTFILE)
9	LIB	ILLEGAL NUMKEYS
10		KEYFIELD IS NOT WITHIN SPECIFIED RECORD LENGTH
11	LIB	ILLEGAL ASCENDING/DESCENDING CODE
12	LIB	ILLEGAL KEY CODE
13		INSUFFICIENT STACK SPACE
14		INPUT RECORD DOES NOT INCLUDE ALL KEY FIELDS
15	LIB	INPUT RECORD IS TOO LONG
16		TOO MANY INPUT RECORDS
17	I/O	FWRITE ERROR ON SCRATCHFILE
18	I/O	FREAD ERROR ON INPUTFILE
19	I/O	FWRITE ERROR ON OUTPUTFILE
20	I/O	FCLOSE ERROR ON SCRATCHFILE

Table 6-2 contains messages which occur in addition to the SORTLIB messages. Those containing I/O in the second column also will result in a file information display. Those marked with HARD in the second column cause the program to terminate. All others also cause program termination, unless the program is being run interactively, in which case you will be asked to enter the command again. The stand-alone Sort program commands listed in the fourth column of the table are those commands which can cause the error to occur.

6-2. MERGE PROGRAM ERROR MESSAGES

Table 6-3 contains a list of error numbers and the corresponding messages which are output by the MERGELIB segment of the system segmented library. Those messages which contain I/O in the second column of the table also will result in a file information display. Those which contain LIB in the second column will not be printed by the stand-alone Merge program but will be printed (if an error occurs) if the Merge library intrinsics are being called program-matically. Each of the messages consists of the characters "MERGELIB:", followed by the remainder of the message.

Table 6-4 contains a list of the error numbers and the corresponding messages which occur in addition to the

MERGELIB messages. Those containing I/O in the second column also will result in a file information display. Those marked with HARD in the second column cause the program to terminate. All others also cause program termination unless the program is being run interactively, in which case you will be asked to enter the command again. The stand-alone Merge program commands listed in the fourth column of the table are those commands which can cause the error to occur.

6-3. RECOVERY PROCEDURES

If you wish your program to continue when SORTLIB errors occur, your program must call SORTEND in order to restore the stack to its original condition. The remainder of your program then will continue to run.

When an error occurs in the MERGELIB procedures (i.e., when the program is being called from your program), no recovery is necessary since the procedure returns directly to your program.

Errors which occur when the stand-alone Sort or Merge programs are being run in batch mode are not recoverable and the programs will abort. In interactive mode, however, syntax errors are recoverable and you will be asked to enter the command again.

Table 6-2. Sort Program Error Messages

ERROR NUMBER	TYPE OF ERROR	MESSAGE	COMMAND
1	I/O,HARD	FAILURE ON FOPEN OF LIST FILE	
2	HARD	LIST FILE IS READ-ONLY	
3	I/O, HARD	FAILURE ON FOPEN OF TEXT FILE	
4	HARD	TEXT FILE IS WRITE-ONLY	
5		ILLEGAL COMMAND	
6		NO KEYS WERE SPECIFIED	END
7		FILENAME CANNOT EXCEED 35 CHARACTERS	INPUT, OUTPUT
8		MISSING COMMA	INPUT, OUTPUT, KEY
9		ILLEGAL NUMBER OF RECORDS	INPUT
10		NUMBER OF RECORDS TOO LARGE OR TOO SMALL	INPUT
11		ILLEGAL RECORD SIZE	INPUT

Table 6-2. Sort Program Error Messages (Continued)

ERROR NUMBER	TYPE OF ERROR	MESSAGE	COMMAND
12		RECORD SIZE TOO LARGE OR TOO SMALL	INPUT
13		TOO MANY PARAMETERS	INPUT, OUTPUT, KEY, RESET, VERIFY, END
14	I/O, HARD	FAILURE ON FOPEN OF INPUT FILE	END
15		MISSING NUM OR KEY	OUTPUT
16		ILLEGAL POSITION	KEY
17		POSITION OUT OF RANGE	KEY
18		MISSING PARAMETER	INPUT, OUTPUT, KEY
19		LENGTH OUT OF RANGE	KEY
20		LENGTH PARAMETER NOT AN INTEGER	KEY
21		LENGTH NOT SPECIFIED FOR TYPE BYTE, PACKED, DISPLAY	KEY
22		MISSING DESC	KEY
23	HARD	INPUT FILE IS WRITE-ONLY	END
24	I/O, HARD	FAILURE ON FOPEN OF OUTPUT FILE	END
25	HARD	OUTPUT FILE IS READ-ONLY	END
26	I/O	FAILURE ON FCLOSE OF OUTPUT FILE	
27		SUM OF KEYFIELD SIZES TOO LARGE	
28	I/O	FAILURE ON PURGE OF OLD OUTPUT FILE	
29	I/O	FAILURE ON FOPEN OF OLD OUTPUT FILE	
30	I/O	FAILURE ON FRENAME OF OUTPUT FILE	
31	I/O, HARD	FAILURE ON FWRITE OF PROMPT FILE	
32	I/O, HARD	FAILURE ON FREAD OF TEXT FILE	

Table 6-3. MERGELIB Error Messages

ERROR NUMBER	TYPE OF ERROR	MESSAGE
1	LIB	NO NUMINPUTFILES PARAMETER SPECIFIED
2	LIB	ILLEGAL NUMINPUTFILES
3	LIB	NO INPUTFILES PARAMETER SPECIFIED
4	LIB	NEITHER OUTPUTFILE NOR POSTPROCESSOR PARAMETER SPECIFIED
5	LIB	IF KEYCOMPARE IS SPECIFIED, KEYS AND NUMKEYS MUST NOT BE
6	LIB	IF KEYCOMPARE IS NOT SPECIFIED, KEYS AND NUMKEYS MUST BE
7	LIB	ILLEGAL NUMKEYS
8	LIB	KEYFIELD IS NOT WITHIN RECORD LENGTH OF EACH FILE
9	LIB	ILLEGAL ASCENDING/DESCENDING CODE
10	LIB	ILLEGAL KEY CODE
11	LIB, I/O	FAILURE ON FGETINFO(INPUTFILE)
12	I/O	FREAD ERROR ON INPUT FILE
13	I/O	FWRITE ERROR ON OUTPUT FILE
14	I/O	INPUT RECORD DOES NOT INCLUDE ALL KEY FIELDS
15	LIB	IF KEYCOMPARE IS SPECIFIED, KEYONLY MAY NOT BE
16		INSUFFICIENT STACK SPACE

Table 6-4. Merge Program Error Messages

ERROR NUMBER	TYPE OF ERROR	MESSAGE	COMMAND
1	I/O,HARD	FAILURE ON FOPEN OF LIST FILE	
2	HARD	LIST FILE IS READ-ONLY	
3	I/O, HARD	FAILURE ON FOPEN OF TEXT FILE	
4	HARD	TEXT FILE IS WRITE-ONLY	
5		ILLEGAL COMMAND	
6		NO KEYS WERE SPECIFIED	END
7		FILENAME CANNOT EXCEED 35 CHARACTERS	INPUT, OUTPUT
8		MISSING COMMA	INPUT, OUTPUT, KEY
9		MISSING PARAMETER	INPUT, OUTPUT, KEY
10		ILLEGAL NUMBER OF RECORDS	OUTPUT
11		NUMBER OF RECORDS TOO LARGE OR TOO SMALL	OUTPUT
12		TOO MANY PARAMETERS	OUTPUT, KEY RESET, VERIFY, END
13	HARD	INSUFFICIENT SPACE	
14	I/O, HARD	FAILURE ON FOPEN OF INPUT FILE	END
15		ILLEGAL POSITION	KEY
16		POSITION OUT OF RANGE	KEY
17		LENGTH OUT OF RANGE	KEY
18		LENGTH PARAMETER NOT AN INTEGER	KEY
19		LENGTH NOT SPECIFIED FOR TYPE BYTE, PACKED, DISPLAY	KEY
20		MISSING DESC	KEY
21	I/O, HARD	INPUT FILE IS WRITE-ONLY	END
22	I/O, HARD	FAILURE ON FOPEN OF OUTPUT FILE	END

Table 6-4. Merge Program Error Message (Continued)

ERROR NUMBER	TYPE OF ERROR	MESSAGE	COMMAND
23	I/O, HARD	OUTPUT FILE IS READ-ONLY	END
24		NO INPUTFILES WERE SPECIFIED	END
25		FAILURE ON FCLOSE OF OUTPUT FILE	
26		SUM OF KEYFIELD SIZES TOO LARGE OUTPUT FILE	
27	I/O	FAILURE ON PURGE OF OLD OUTPUT FILE	
28	I/O	FAILURE ON FOPEN OF OLD OUTPUT FILE	
29	I/O	FAILURE ON FRENAME OF OUTPUT FILE	
30	I/O, HARD	FAILURE ON FWRITE OF PROMPT FILE	
31	I/O, HARD	FAILURE ON FREAD OF TEXT FILE	

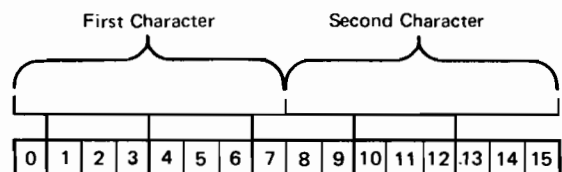
ASCII CHARACTER SET

APPENDIX

A

ASCII Character	First Character Octal Equivalent	Second Character Octal Equivalent
A	040400	000101
B	041000	000102
C	041400	000103
D	042000	000104
E	042400	000105
F	043000	000106
G	043400	000107
H	044000	000110
I	044400	000111
J	045000	000112
K	045400	000113
L	046000	000114
M	046400	000115
N	047000	000116
O	047400	000117
P	050000	000120
Q	050400	000121
R	051000	000122
S	051400	000123
T	052000	000124
U	052400	000125
V	053000	000126
W	053400	000127
X	054000	000130
Y	054400	000131
Z	055000	000132
a	060400	000141
b	061000	000142
c	061400	000143
d	062000	000144
e	062400	000145
f	063000	000146
g	063400	000147
h	064000	000150
i	064400	000151
j	065000	000152
k	065400	000153
l	066000	000154
m	066400	000155
n	067000	000156
o	067400	000157
p	070000	000160
q	070400	000161
r	071000	000162
s	071400	000163
t	072000	000164
u	072400	000165
v	073000	000166
w	073400	000167
x	074000	000170
y	074400	000171
z	075000	000172
0	030000	000060
1	030400	000061
2	031000	000062
3	031400	000063
4	032000	000064
5	032400	000065
6	033000	000066
7	033400	000067
8	034000	000070
9	034400	000071
NUL	000000	000000
SOH	000400	000001
STX	001000	000002
ETX	001400	000003
EOT	002000	000004
ENQ	002400	000005

ASCII Character	First Character Octal Equivalent	Second Character Octal Equivalent
ACK	003000	000006
BEL	003400	000007
BS	004000	000010
HT	004400	000011
LF	005000	000012
VT	005400	000013
FF	006000	000014
CR	006400	000015
SO	007000	000016
SI	007400	000017
DLE	010000	000020
DC1	010400	000021
DC2	011000	000022
DC3	011400	000023
DC4	012000	000024
NAK	012400	000025
SYN	013000	000026
ETB	013400	000027
CAN	014000	000030
EM	014400	000031
SUB	015000	000032
ESC	015400	000033
FS	016000	000034
GS	016400	000035
RS	017000	000036
US	017400	000037
SPACE	020000	000040
!	020400	000041
"	021000	000042
#	021400	000043
\$	022000	000044
%	022400	000045
&	023000	000046
'	023400	000047
(024000	000050
)	024400	000051
*	025000	000052
+	025400	000053
,	026000	000054
-	026400	000055
.	027000	000056
/	027400	000057
:	035000	000072
;	035400	000073
<	036000	000074
=	036400	000075
>	037000	000076
?	037400	000077
@	040000	000100
[055400	000133
\	056000	000134
]	056400	000135
Δ	057000	000136
-	057400	000137
,	060000	000140
{	075400	000173
	076000	000174
}	076400	000175
~	077000	000176
DEL	077400	000177





- Account name, 2-8
- Account password, 2-8
- Aoptions
 - Merge program, 4-1
 - Sort program, 2-1
- Ascending sort, 1-2
- ASCII Character Set, A-1

- Batch mode
 - Merge program, 4-4
 - Sort program, 2-10
- Byte data, 2-4

- Calling the Merge program
 - from SPL/3000, 5-3
 - from FORTRAN/3000, 5-3
- Calling the Sort program
 - from SPL/3000, 3-4
- Control Y
 - Merge program, 4-4
 - Sort program, 2-6
- CPU time
 - Merge Program, 4-4
 - Sort program, 2-6

- :DATA command, 2-7
- Data fields, definition of, 1-1
- Data stack layout, 3-25
- Data types
 - byte, 2-4
 - display, 2-5
 - double, 2-4
 - integer, 2-4
 - long, 2-4
 - packed, 2-4
 - packed*, 2-5
 - real, 2-4
- Descending sort, 1-2
- Display data, 2-5
- Double data, 2-4

- Elapsed time
 - Merge program, 4-4
 - Sort program, 2-6
- END command
 - Merge program, 4-4
 - Sort program, 2-6
- Error messages
 - MERGELIB, 6-4
 - Merge program, 6-2
 - SORTLIB, 6-1
 - Sort program, 6-1
- ERRORPROC parameter
 - Merge program, 5-2
 - Sort program, 3-2

- FAILURE parameter
 - Merge program, 5-3
 - Sort program, 3-3
- File definitions
 - Merge program, 4-1
 - Sort program, 2-1
- Filecode
 - Merge program, 4-1
 - Sort program, 2-2
- Files, definition of, 1-1
- Filesize
 - Merge program, 4-1
 - Sort program, 2-2
- Foptions
 - Merge program, 4-1
 - Sort program, 2-1
- Formaldesignator
 - Merge program, 4-1
 - Sort program, 2-1

- Initialloc
 - Merge program, 4-1
 - Sort program, 2-2
- INPUT command
 - Merge program, 4-2
 - Sort program, 2-3
- INPUT file
 - Merge program, 4-1
 - Sort program, 2-1
- INPUTFILE parameter
 - Merge program, 5-1
 - Sort program, 3-1
- Inputs to SORT/3000
 - Integer data, 2-4
- Interactive sessions
 - Merge program, 4-4
 - Sort program, 2-6
- Intermediate passes, 2-6

- Jobname, 2-7

- KEY command
 - Merge program, 4-3
 - Sort program, 2-4

KEYCOMPARE parameter
Merge program, 5-2
Sort program, 3-2
Key formats, 1-3
Keys, definition of, 1-2
KEYONLY parameter, 5-1
KEYS parameter
Merge program, 5-1
Sort program, 3-2

Length parameter, 2-4
LIST file
Merge program, 4-2
Sort program, 2-2
Long data, 2-4

Major and minor keys, 1-4
MAXDATA, 2-1
MERGEERRORMESS procedure, 5-3
MERGELIB error messages, 6-4
MERGE procedure, 5-1
Merge program
calling programmatically, 5-1
commands, 4-2
description, 1-1
error messages, 6-1
files, 4-1
stand-alone, 4-1
Merge program commands
END, 4-4
INPUT, 4-2
KEY, 4-3
OUTPUT, 4-2
RESET, 4-3
VERIFY, 4-3
Merge program files
INPUT, 4-1
LIST, 4-2
OUTPUT, 4-1
PROMPT, 4-2
TEXT, 4-2
MERGETITLE procedure, 5-3

Number of compares
Merge program, 4-4
Sort program, 2-6
Number of records, 2-3
NUMINPUTFILES parameter, 5-1
NUMKEYS parameter
Merge program, 5-1
Sort program, 3-2
NUM parameter, 2-3
NUMRECS parameter, 3-2

OUTPUT command
Merge program, 4-2
Sort program, 2-3

OUTPUT file
Merge program, 4-1
Sort program, 2-1
OUTPUTFILE parameter
Merge program, 5-1
Sort program, 3-1
OUTPUTOPTION parameter, 3-2
Output options, 2-2
Outputs from SORT/3000, 1-4

Packed data, 2-4
Packed* data, 2-5
Position parameter, 2-4
POSTPROCESSOR parameter, 5-2
PREPROCESSOR parameter, 5-2
PROMPT file
Merge program, 4-2
Sort program, 2-3

Q register, 3-6

Real data, 2-4
RECLLEN parameter, 3-2
Records
definition of, 1-1
example, 1-1
Record size, 2-3
Recovery procedures, 6-2
Recsize
Merge program, 4-1
Sort program, 2-2
RESET command
Merge program, 4-3
Sort program, 2-5
Running the stand-alone Merge program
in batch mode, 4-4
in interactive session, 4-4
Running the stand-alone Sort program
in batch mode, 2-10
in interactive session, 2-6

SCRATCH file, 2-3
Scratchfile I/O's, 2-6
SORTERRORMESS procedure, 3-4
Sorting order, 1-2
SORTINITIAL procedure, 3-1
SORTINPUT procedure, 3-3
SORTLIB error messages, 6-1
SORTOUTPUT procedure, 3-3
Sort program
calling programmatically, 3-1
commands, 2-3

- description, 1-1
- error messages, 6-1
- files, 2-1
- stand-alone, 2-1
- Sort program commands
 - END, 2-6
 - INPUT, 2-3
 - KEY, 2-4
 - OUTPUT, 2-3
 - RESET, 2-5
 - VERIFY, 2-5
- Sort program error messages, 6-1
- Sort program files
 - INPUT, 2-1
 - LIST, 2-2
 - OUTPUT, 2-1
 - PROMPT, 2-3
 - TEXT, 2-3
- SORTTITLE procedure, 3-4
- SORT/3000
 - description, 1-1
 - purposes, 1-1
- Space available
 - Merge program, 4-4
 - Sort program, 2-6
- S register, 3-6
- Stack limit, 3-6
- Statistics

- Merge program, 4-4
- Sort program, 2-6
- STATISTICS parameter
 - Merge program, 5-3
 - Sort program, 3-3

- TEXT file
 - Merge program, 4-2
 - Sort program, 2-3
- Treatment of the stack, 3-8

- Username, 2-8
- User password, 2-8

- VERIFY command
 - Merge program, 4-3
 - Sort program, 2-5

- Z register, 3-6

