UNIVAC 1106/1108

MULTI-PROCESSOR SYSTEMS

EXEC 8 SORT/MERGE

PROGRAMMER REFERENCE

SPERRY UNIVAC
COMPUTER SYSTEMS

This document contains the latest information available at the time of publication. However, the Univac Division reserves the right to modify or revise its contents. To ensure that you have the most recent information, contact your local Univac Representative.

PRINTED IN U.S.A.

UNIVAC 1106/1108 Multi-Processor System
EXEC 8 Sort/Merge Programmer Reference
UP-7621 Rev. 1

May 1973

UPDATING PACKAGE A

File pages as specified below

SECTION	DESTROY FORMER PAGES NUMBERED	FILE NEW PAGES NUMBERED
Front Cover and Disclaimer	†	†
Page Status Summary	N. A.	PSS-1A**
Contents	1 and 2	1A and 2A
	3 and 4	3A and 4
Section 2	3 and 4	3 and 4A
	7 thru 10	7A thru 10A
	13 and 14	13 and 14A
	15 and 16	15 and 16A
	17 and 18	17 and 18A
	N. A.	19A**
Section 3	3 and 4	3 and 4A
Section 6	1 and 2	1 and 2A
	3 and 4	3A and 4A
	5 and 6	5A and 6
Appendix A	3 thru 6	3A thru 6A
	7 and 8	7A and 8
	9 and 10	9A and 10
Index	N. A.	1A** thru 4A**

†Destroy old cover and file new cover
**These are new pages

All the technical changes in an update are denoted by an arrow (→) in the margin. A downward pointing arrow (↓) next to a line indicates that technical changes begin at this line and continue until an upward pointing arrow (↑) is found. A horizontal arrow (→) pointing to a line indicates a technical change in only that line. A horizontal arrow located between two consecutive lines indicates technical changes in both lines or deletions.

PAGE STATUS SUMMARY

ISSUE: UP–7621 Rev. 1 Update A

Section	Page Number	Update Level	Section	Page Number	Update Level	Section	Page Number	Update Level
Cover/Disclaimer		A						
PSS	1	A						
Contents	1 thru 3 4	A Orig.						
1	1 and 2	Orig.						
2	1 thru 3 4 5 and 6 7 thru 10 11 thru 13 14 15 16 17 18 and 19	Orig. A Orig. A Orig. A Orig. A Orig. A						
3	1 thru 3 4 5 thru 12	Orig. A Orig.						
4	1 and 2	Orig.						
5	1 thru 4	Orig.						
6	1 2 thru 5 6 thru 13	Orig. A Orig.						
Appendix A	1 and 2 3 thru 7 8 9 10	Orig. A Orig. A Orig.						
Index	1 thru 4	A						
UCS								

CONTENTS

FIGURES

TABLES

1. INTRODUCTION

1.1. GENERAL

This manual describes the architecture and implementation of the UNIVAC 1106/1108 Sort/Merge package and the manner in which it operates with and is linked to the UNIVAC EXEC 8 Operating System. The reader is expected to be familiar with the UNIVAC EXEC 8 Operating System and the UNIVAC 1106/1108 assembler and compiler languages; none of this information is repeated in this manual. The UNIVAC 1106/1108 Sort/Merge package basically consists of a sort subroutine and a merge subroutine, which are independent. The information concerning these subroutines is presented in the sections of this manual as listed below:

Section 1. INTRODUCTION. Gives a brief functional description of subroutine operation.

Section 2. PARAMETER SPECIFICATION. Describes the sort parameter table and the method of entering information into the table.

Section 3. PROGRAM LINKAGES. Describes the required program linkages needed to create a sort program using the sort subroutine.

Section 4. CONTINUATION OF INTERRUPTED SORT PROGRAMS. Describes the method of restarting interrupted sort programs and how worker tapes can be recreated.

Section 5. MERGE SUBROUTINE. Describes the method of creating a merge program using the merge subroutine. Includes information on parameter specification and program linkages.

Section 6. OPERATING. Contains operating information and the format and meaning of diagnostic messages.

1.2. SORT SUBROUTINE – FUNCTIONAL OPERATION

The sort subroutine must be activated by a user program. It makes no assumptions as to the source or format of the original input file. The use of the subroutine approach provides the user with complete flexibility in tape file formats and the choice of original data file storage media. Since the sort subroutine is reusable at object time, the sort run can be executed many times without reloading the program. This provides obvious advantages in small volume sorts which are part of much larger programs. More conventional sort runs, in which sorting is the primary function, are easily constructed by combining input and output programs with the sort subroutine. The sort subroutine accepts data from own code, one record at a time, and returns the sorted data to own code in the same manner. Own code controls the disposition of the final output file.

When own code initiates the sort subroutine, it supplies the subroutine with the address of a parameter table. This table contains information concerning record size, key definitions, hardware facilities, and other information required by the sort. The user must specify the requirements for main storage and mass storage, and the number of tape units available for use by the subroutine. The subroutine reacts to these parameters at initiation, selecting the optimum configuration for performing the sort run. When initialization is completed, the subroutine is ready to accept and sort the data.

Own code enters the sort subroutine each time a record is available in main storage. The address of that record is supplied to the subroutine. When all the data has been entered, the subroutine sorts the records in accordance with the key definitions in the parameter table. It then returns the sorted records to own code (in main storage) by supplying the address of a record each time own code requests the next record in the output file. Own code and the sort subroutine communicate by means of standard UNIVAC 1106/1108 assembler-type program linkages.

The sort subroutine can accept any volume of data in a particular sort run. An automatic sort/merge feature enables the user to sort extremely large volumes of data under complete control of the sort subroutine. Instructions for demounting, labelling, and remounting tape reels are provided by the subroutine. Alternatively, the user may break up a large volume sort run into two or more parts to be executed separately. The sort subroutine is not re-entrant.

1.3. MERGE SUBROUTINE

An independent merge subroutine is provided as part of the UNIVAC 1106/1108 Sort/Merge package. It is capable of performing a single internal merge of 2 to 26 pre-sorted input files in accordance with specified key definitions. Like the sort subroutine, the merge subroutine makes no assumptions as to the format and source of the original input files, thereby allowing complete flexibility in user formats and choice of original data file storage media.

Input and output control for the merge must be provided by a user program. The merge subroutine accepts data from own code, one record at a time, and returns the merged records in the same manner.

Communication between the user program and the merge subroutine is made by way of three program linkages. Own code initializes the merge subroutine by supplying the address of a parameter table which contains the key definitions for the desired merge. Own code initiates merge selection by releasing the first record from each input file to the merge subroutine. After all input files have been initiated, own code requests records from the merge, releases records to the merge to replace the records returned, and informs the merge when an input file is exhausted.

Although the merge subroutine performs a single level merge, the subroutine may be executed several times in one run. This is useful, for example, if a sufficient number of tape units is not available to merge all the tape input files in a single merge. The merge subroutine is not re-entrant.

2. PARAMETER SPECIFICATION

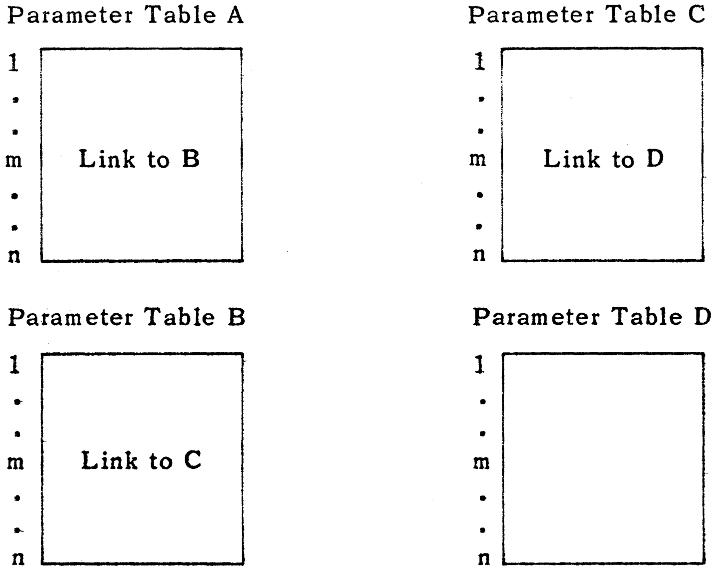
2.1. THE PARAMETER TABLE

The sort subroutine is activated at object time by a linkage which initializes the subroutine. At this time, the user program supplies the sort subroutine with the address of a parameter table containing all the information required for the particular sort run. The assembler, constants, and COBOL can be used to construct a parameter table. A user who desires to acquire a sort program using COBOL will have the table compiled as necessary. For the user of the UNIVAC 1106/1108 Assembler, the parameter table could be constructed as a series of constants. However, to facilitate construction of the table by means of the assembly language, a procedure directive called R$FILE is available. This directive accepts the desired parameter table entries on the PROC line in mnemonic form and constructs the internal format of the parameter table.

Whatever method is used at assembly time to construct the parameter table, the user can alter the contents of the table prior to transferring control to the sort subroutine for initialization. This feature is especially useful in creating package and utility sorts that can react at run time to simple parameter changes.

The parameter table is of free-form construction and consists of a variable number of words. The first character of every word is a fixed code, defined by the sort subroutine, which classifies the contents of the remainder of the word. The first word of the table contains a header code in the first character. Similarly, the last word of the table is indicated by a sentinel character code in the word. When the sort subroutine desires information from the parameter table, it searches the table from top to bottom for the code which indicates the information desired.

Although the parameter table is often contained in contiguous memory locations, a composite parameter table consisting of several different tables can be used. Within each table there is a code which links the tables and directs the search operation. Any number of parameter tables may be connected in this manner. Figure 2-1 illustrates how four tables are joined and searched.



These four tables, in different areas of storage, are searched in the following sequence to produce a single parameter table.

A1 to Am
B1 to Bm
C1 to Cm
D1 to Dn
C(m+1) to Cn
B(m+1) to Bn
A(m+1) to An

Figure 2–1. Linking Parameter Tables

2.2. PARAMETER TABLE ENTRIES

The types of information that may be present in the parameter table presented to the sort subroutine at initialization time are as follows:

- **Data**
  Describes the types and sizes of records, key description, and collating sequence **description.**
- Hardware facilities
  Specifies the main storage available for use by the sort subroutine, and the external-file-names assigned to the flying-head drum, FASTRAND drum, and/or tape files which may be available for use by the sort subroutine.
- Return points and exits
  In order for the user program to regain control during periods when the subroutine is operating, the addresses of some lines within the user program appear in the parameter table. At the appropriate times, control is transferred from the sort subroutine to these addresses within the user program.
- Special
  Two special codes may appear in the table indicating:
  (1) A branch to another table
  (2) This word contains no information (reserves area for user information)

The remaining information types apply only to large volume sorts:

- Type run
  Four different codes concern large volume sorts. One code instructs the sort subroutine to perform an automatic sort on a large volume of data, that is, proceeding through Parts A, B, and C. The other three codes allow for the nonautomatic operation of a large volume sort, in which case the user may instruct the subroutine to perform Part A, Part B, or Part C.
- Large volume control
  Certain types of information are required to control such things as time between checkpoints and output labels for the intermediate tapes of part B. Other items of information such as block size, input labels, and input units are required in the nonautomatic control of parts A, B, and C.
- Rerun control
  There are two operational types of rerun:
  (1) A sort run may be interrupted and then continued from the point of interruption to its normal completion; or
  (2) A previously produced tape may be recreated.

These two examples require different control information, which must be entered in the parameter table when necessary.

2.2.1. R$FILE

The R$FILE procedure call format is as follows:

label *R$FILE* *list-1* *list-2* *list-3 . . . list-n*

The call line *label* connects this set of parameters to the sort subroutine by way of program linkages.

The operand field consists of a variable number of lists separated by spaces. Each list contains at least two fields separated by commas. The first field of each list is a literal which defines that list; the remaining fields specify pertinent parameters. The lists may be written in any order; however, fields within a list must be written in a particular order.

The 26 alphanumeric constants, which are used to define lists of parameter information, are described in detail in the following paragraphs. Table 2-1 summarizes them, with their parameters, for ready reference.

In a parameter table, *one and only one* of the following alphanumeric lists is required: 'RSZ', 'RSZW', and 'VRSZ' or 'VRSZW'. In addition, at least *one* of the following is also required: 'KEY', 'KEYW', or 'COMP'.

Lists 15 through 24 are used only when sorting very large volumes of data.

2.2.1.1. 'RSZ'

Format: 'RSZ', *no.-of-char-in-record*

- no.-of-char-in-record – decimal number specifying record size in characters for fixed-length records.

2.2.1.2. 'RSZW'

Format: 'RSZW', *no.-of-words-in-record*

- no.-of-words-in-record – decimal number specifying record size in words for fixed-length records.

2.2.1.3. 'VRSZ'

Format: 'VRSZ', *max-no.-char-in-record, min-no.-char-in-link,*
no.-char-in-record-size-i, no.-of-records-of-record-size-i,
..., no.-char-in-record-size-n, no.-of-records-of-record-size-n

- max-no.-char-in-record – decimal number specifying largest record size, in characters, that will be encountered in set of variable-length records being sorted.
- min-no.-char-in-link – decimal number specifying minimum link size in characters. This must be equal to or greater than the highest 'KEY' in character position because the first link must contain all key fields.
- no.-char-in-record-size-i – decimal number specifying record size in characters for record size i.
- no.-of-records-of-record-size-i – decimal number specifying number of records of record size i.

To conserve main storage and provide optimum speed during the first data pass, variable-length records are divided into fixed-length links in main storage. For record sizes greater than 60 characters (or 10 words) in length, link size should not be specified as less than 30 characters (or 5 words). For record sizes less than or equal to 60 characters, link size should be specified as record size regardless of *actual* minimum link size. This prevents an inefficient utilization of the main storage allocation and decreases the probability of a Sort B5 error. In this list, *no.-char-in-record-size-i* and *no.-of-records-of-record-size-i* specify the distribution of record sizes for the input. The user may specify all record sizes or none. These parameters are optional.

2.2.1.4. 'VRSZW'

Format: 'VRSZW', *max-no.-words-in-record, min-no.-words-in-link,*
no.-words-in-record-size-i, no.-of-records-of-record-size-i,
..., no.-words-in-record-size-n, no.-of-records-of-record-size-n

- max-no.-words-in-record – decimal number specifying largest record size, in words, that will be encountered in set of variable-length records being sorted.
- no.-words-in-link – decimal number specifying minimum link size in words.
- no.-words-in-record-size-i – decimal number specifying record size in words for record size i.
- no.-of-records-of-record-size-i – decimal number specifying number of records of record size i.

In this list, *no.-words-in-record-size-i* and *no.-of-records-of-record-size-i* serve the same function as *no.-char-in-record-size-i* and *no.-of-records-of-record-size-i* in the 'VRSZ' list (see 2.2.1.3). Also this list may be used instead of the 'VRSZ' list for variable-size records.

LISTS		PARAMETERS									
NO.	LITERAL	p1	p2	p3	p4	p5	p6	p7	p8	p(n-1)	pn
1	‘RSZ’	No. of char. in record									
2	‘RSZW’	No. of words in record									
3	‘VRSZ’	Max. no. of char. in record	Min. no. of char. in link	No. of char. in variable record size 1	No. of records of variable record size 1			No. of char. in variable record size n	No. of records of variable record size n		
4	‘VRSZW’	Max. no. of words in record	Min. no. of words in link	No. of words in variable record Size 1	No. of records of variable record size 1			No. of words in variable record size n	No. of records of variable record size n		
5	‘KEY’	Char. position	No. of char.	Type	Order	Field no.					
6	‘KEYW’	Word position	Bit no. position	No. of bits	Type	Order	Field no.				
7	‘SEQ’	C1	C2	C3	C4	C5	C6	C7	C8	C(n-1)	Cn
8	‘FILES’	External file name 1	External file name 2			External file name n					

Table 2-1. Sort Parameter Lists
(Part 1 of 3)

LISTS		PARAMETERS									
No.	LITERAL	p1	p2	p3	p4	p5	p6	p7	p8	p(n-1)	pn
9	'CHECK'	'D' indicates omit drum checksum	'T' indicates omit tape checksum	'F' indicates omit FASTRAND checksum							
10	'CORE'	No. of words									
11	'DROC'	Address									
12	'COMP'	Address									
13	'FPOC'	Address									
14	'LPOC'	Address									
15	'FINAL'	Address									
16	'SMRG'	Output label prefix	Records per cycle	Reels per cycle							
17	'PARTA'	Output label prefix	Records per cycle	Reels per cycle	Max. output block size						
18	'PARTB'	Output label prefix	External file name for output	External file name for input tape 1	External file name for input tape 2				External file name for input file n		
19	'PARTC'	External file name for input file 1	External file name for input file 2				External file name for input file n				

Table 2-1. Sort Parameter Lists
(Part 2 of 3)

LISTS		PARAMETERS									
No.	LITERALS	p1	p2	p3	p4	p5	p6	p7	p8	p(n-1)	pn
20	'REDOA'	Cycle no.	From record no.	To record no.							
21	'REDOB'	Tape label	Reel no.								
22	'CONTA'	Cycle no.	From record no.								
23	'CONTB'	Tape label	Reel no.								
24	'CONTC'	Address									
25	'COPY'	Address									
26	'PAD'	No. of words									

Table 2–1. Sort Parameter Lists
(Part 3 of 3)

2.2.1.5. 'KEY'

Format: 'KEY', *char-pos, no.-of-char, type, order, field-no.*

- char-pos – decimal number specifying character position within record which contains leftmost character of key field. The character positions of a record are numbered from left to right beginning with 1.
- no.-of-char – decimal number specifying key field length in characters.
- type – alphabetic character specifying the key field format:
  - 'A' for alphanumeric format
  - 'B' for binary 1106/1108 format
  - 'D' for signed decimal format
  - 'M' for binary IBM 7090 format
  - 'P' for overpunched Fieldata decimal format
  - 'Q' for overpunched ASCII decimal format
  - 'R' for signed ASCII decimal format
  - 'S' for table translated ASCII format
  - 'U' for unsigned binary format

- order – alphabetic character specifying desired sorting sequence of the key field:
  - 'A' (or blank) for ascending field
  - 'D' for descending field
- field-no. – decimal number indicating significance of key fields from major to minor. The major key field is number 1, the next most significant field is number 2, etc. If the *field-no.* field is omitted from the 'KEY' lists, the first 'KEY' list encountered is assumed to describe the major field, the next list encountered is assumed to describe the next most significant field, etc. If the *field-no.* field is omitted from any list, it must be omitted from all lists.

2.2.1.6. 'KEYW'

Format: 'KEYW', *word-pos, bit-pos, no.-of-bits, type, order, field-no.*

- word-pos – decimal number specifying the word number within the record containing the most significant bit of the key field. Words within a record are numbered from left to right beginning with 1.
- bit-pos – a decimal number from 0 to 35 indicating the bit position within the word which is the most significant bit of the key field. Bit positions within a word are numbered from right to left beginning with 0.
- no.-of-bits – decimal number specifying key field length in bits. The remaining fields are the same as for the 'KEY' list (see 2.2.1.5). The 'KEYW' list merely provides an alternative to the 'KEY' list for specifying key fields. In any call on the R$FILE procedure, 'KEY' and 'KEYW' lists may be intermixed.

  As many as forty key fields may be specified. Each field requires a 'KEY' or a 'KEYW' list. These lists may be in any order in the operand field of the R$FILE procedure call line.

2.2.1.7. 'SEQ'

Format: 'SEQ', c_1, c_2, c_3, . . . , c_{64} *(or:* c_{128}*)*

- c_i – octal number specifying change in collating sequence. If the desired collating sequence is not 000–077 (or 000–177 for ASCII), then the entire new collating sequence must be specified with a 'SEQ' list. The character which is to have the ith position in ascending order is specified by c_i. The presence of a 'SEQ' list *temporarily* transforms all alphanumeric key fields ('A' type) or table translated ASCII ('S' type) keys to their 'SEQ' list equivalents for internal sorting processes. The fields are re-translated to their original numeric representation before return to the user.

2.2.1.8. 'FILES'

Format: 'FILES', '*external-file-name-1*', '*external-file-name-2*', ..., '*external-file-name-n*'.

- 'external-file-name-1' through 'external-file-name-n' – series of external-file-names which may be assigned to the drum, FASTRAND, or tape files used by the sort subroutine as scratch files.

The user must assign, either dynamically or by means of @ASG cards, the external-file-names which are available to the sort subroutine. A maximum of two mass storage files and 24 tape files may be assigned. All external files assigned for use by the subroutine must be temporary files. Tape files must not be assigned with data conversion or even parity options. An external-file-name may not exceed 12 characters and, since all sort files are temporary, a qualifier should not be specified. The *project* field from the @RUN statement is used as the qualifier.

The FACIL$ linkage determines if an external-file-name specified in the 'FILES' list has been assigned and, if assigned, to determine whether it is a mass storage or tape file. The sort subroutine equates the external-file-names assigned to sort internal-file-names by way of the USE statement.

The term ''mass storage '' includes all types of magnetic drums (FH-432, FH-880, or FASTRAND drum). A maximum of two mass storage files may be assigned. The internal-file-name $A is used only when one mass storage file is assigned. If two mass storage files are assigned, internal-file-name $A is assigned to the file containing the smaller number of words and internal-file-name $B is assigned to the file containing the larger number of words.

Magnetic tape files may be assigned to UNISERVO VIII-C, VI-C, IV-C, III-A, or II-A tape units. If tapes are assigned in a single-cycle sort run, a minimum of two tape units must be assigned. Tape units assigned in a single-cycle sort run may be a mixture of tape unit types. If the 'PARTA' parameter list (see 2.2.1.17) is specified, at least one tape file must be assigned. If more than one tape file is assigned for 'PARTA', at least two additional tape files must be assigned.

These tape files may be a mixture of any type of tape unit; the intermediate output file(s) are written on the first assigned tape file in the 'FILES' list. Whenever the 'SMRG' parameter list (see 2.2.1.16) is used, at least three tape files must be assigned. The tape units for an automatic, large-volume sort/merge ('SMRG') must all be of the same type. A maximum of 24 tape files may be assigned for use by the sort subroutine in any sort run. The internal-file-names $C, $D, ..., $Z are used for sort tape files.

An element of standard external-file-names (XA, XB, ..., XZ) is in a parameter table which is available to the user. This element may be used in place of the 'FILES' list. One of these external-file-names must be assigned for each file used by the sort subroutine. An 1106/1108 assembler user may incorporate this table with 'COPY', RSTD$ as a parameter to the R$FILE procedure. This table is automatically included in all COBOL sorts.

2.2.1.9. 'CHECK'

Format: 'CHECK', p_1, p_2, p_3

- p1, p2, p3 – specifies the type of hardware for which the checksum option is to be omitted. The omit checksum codes are:
  - – 'D' omit drum checksum
  - – 'F' omit FASTRAND drum checksum
  - – 'T' omit tape checksum

Any one or any combination of checksum omissions may be specified. A checksum word is calculated and written for each output block written and verified for each input block read. While this check can be advantageous in detecting otherwise undetectable hardware errors, it may be too time consuming in some multiprogramming environments.

2.2.1.10. 'CORE'

Format: 'CORE', *no.-of-words*

- no.-of-words – decimal number specifying the amount of main storage available for sorting.

The sort subroutine reserves an area of main storage equal to the number of words specified. If the 'CORE' list is omitted from the parameter table, the user may specify a working storage area by way of an @ASG statement. The @ASG statement is

```
@ASG,T R$CORE, F///n
```

where n is the number of words, in thousands, of storage to be used by the sort. This is a "dummy" FASTRAND assign to notify Sort of working core allocation.

The sort subroutine obtains n by FACIL$ of file R$CORE and requests n x 1000 words of storage by way of the MCORE$ linkage.

2.2.1.11. 'DROC'

Format: 'DROC', *address*

- address – starting address of own code that is executed every time the sort subroutine determines that two records being compared have equal keys.

2.2.1.12. 'COMP'

Format: 'COMP', *address*

- address – starting address of own code that is executed each time the sort subroutine needs to know which of two records is to precede the other.

2.2.1.13. 'FPOC'

Format: 'FPOC', *address*

- address – starting address of own code that is executed after the sort subroutine has been initialized by way of program linkage. After return to 'FPOC', the user own code supplies records to the sort subroutine.

2.2.1.14. 'LPOC'

Format: 'LPOC', *address*

- address – starting address of own code that is executed when the sort subroutine is ready to return records to the user in final sorted sequence.

2.2.1.15. 'FINAL'

Format: 'FINAL', *address*

- address – starting address of own code that is executed after part A or part B of a large volume sort.

Most sort programs do not need this list. If this list is not specified, the sort subroutine terminates the activity by way of an ER EXIT$.

2.2.1.16. 'SMRG'

Format: 'SMRG', *'output-label-prefix', records/cycle, reels/cycle*

- 'output-label-prefix' – two-character label prefix placed on every output tape of a sort run. Thus, unique output labels can be produced for different sort runs. In addition, the output tape label blocks contain fields showing merge level, file number within the level, and reel number within the file.

All intermediate output files produced by 'SMRG' or 'PARTA' are written on the first tape-assigned file in the 'FILES' list.

- records/cycle – decimal number specifying the maximum number of input records to be sorted in any cycle. The sort subroutine uses this decimal value only when it is less than the physical limit of the number of records which can be handled with the hardware assigned.
- reels/cycle – decimal number specifying the maximum number of output reels produced in any cycle.

The 'SMRG' list is used in automatic sorting of large volumes of data or when it cannot be determined in advance if a sort run will require operator intervention. In this list, *records/cycle* and *reels/cycle* are optional parameters.

2.2.1.17. 'PARTA'

Format: 'PARTA', *'output-label-prefix', records/cycle, reels/cycle, block-size*

- block-size – decimal number which places an upper limit on the block size in which the sorted output tapes will be written. Generally, the maximum tape block size is 1000 words; however, this parameter, if less than 1000, overrides this limit. This is particularly advantageous when main storage is limited and a large number of tape units will be available during the merging operation of part B. In such a case, the smaller block size enables the limited main storage to accommodate all of the required merge input buffer areas.

The other fields in this list are the same as for the 'SMRG' list (see 2.2.1.16). This list is used to run part A of a large-volume, nonautomatic sort. A 'PARTA' sort run can operate with only one tape file assigned. At least two additional tape files must be assigned if tape is used for intermediate storage during the run.

2.2.1.18. 'PARTB'

Format: 'PARTB', *'output-label-prefix', 'external-file-name-output-unit', 'external-file-name-input-file-1', ..., 'external-file-name-input-file-n'*

- 'output-label-prefix' – same as for 'SMRG' list (see 2.2.1.16).
- 'external-file-name- output-unit' – external-file-name specified on @ASG card which assigns output unit. This must be assigned as a temporary file. The external-file-name may be a maximum of 12 characters and may not have a qualifier.

- 'external-file-name-input-file-i' – external-file-name specified on @ASG card which assigns the input unit for input-i. The external-file-name must be the six-character label which the sort subroutine produced for this file.

The 'PARTB' list is used to run part B of a large-volume, nonautomatic sort.

2.2.1.19. 'PARTC'

Format: 'PARTC', *'external-file-name-input-file-i'*, ..., 'external-*file-name-input-file-n'*

- 'external-file-name-input-file-i' – same as for the 'PARTB' list (see 2.2.1.18).

The 'PARTC' list is used to run part C of a large-volume, nonautomatic sort.

2.2.1.20. 'REDOA'

Format: 'REDOA', *cycle-no., from-record-no., to-record-no.*

- cycle-no. – decimal number assigned to the first cycle output after restart.
- from-record-no. – decimal number specifying the record number of the first record of the set to be resorted when recreating the output of a particular cycle.
- to-record-no. – decimal number specifying the record number of the last record of the set to be resorted when recreating the output of a particular cycle.

This list is used to rerun a portion of a sort.

2.2.1.21. 'REDOB'

Format: 'REDOB', *'tape-label', reel-no.*

- 'tape-label' – label of reel containing rerun information for reproducing a reel originally created during part B of a sort run.
- reel-no. – the decimal number of the above reel.

This list is used to rerun a portion of a sort.

2.2.1.22. 'CONTA'

Format: 'CONTA', *cycle-no., from-record-no.*

- cycle-no. – decimal number assigned to the first cycle output after restart.
- from-record-no. – decimal number specifying the record number of the first record released to sort after restart.

This list is used to continue an interrupted sort program at some later time.

2.2.1.23. 'CONTB'

Format: 'CONTB', *'tape-label', reel-no.*

- 'tape-label' – label assigned to the first output reel written after restart.
- reel-no. – decimal number of above reel.

This list is used to continue a sort run interrupted during part B.

2.2.1.24. 'CONTC'

Format: 'CONTC', *address*

- address – address of the first word of a table containing information needed to continue a sort run from a point in part C.

2.2.1.25. 'COPY'

Format: 'COPY', *name*

- name – label of another parameter table.

This list affects the structure of the assembled parameter table. In response to the 'COPY' list, R$FILE produces a reference to *name*. If *name* is an externally defined label in a Program File (TPF$), this reference causes the collector to include that element in this program. In any case, while the sort is scanning the assembled parameter table at object time, it treats the reference to *name* as a branch in the table. R$FILE produces the sentinel word . . .

71 0000 address of table

at the end of the assembly table. While the sort subroutine is scanning a branch in the assembled parameter table at object time, this sentinel word restarts the scan at the word following the reference to *name*.

It is expected that for short, one-time sorting jobs, an installation will standardize on certain sort parameters such as facility assignments. The programmer need then be concerned only with basic logical parameters such as key location and obtaining the standard set of parameters by way of the 'COPY' list.

2.2.1.26. 'PAD'

Format: 'PAD', *no.-of-words*

- no.-of-words – decimal number which reserves a specified number of words in the parameter table. These locations can then be filled with meaningful information at the time of sort execution but before the ROPN$ linkage is executed.

2.2.2. Format of Parameter Table Entries

The parameter table may be prepared by means of the R$FILE procedure (see 2.2.1), or by the assembler, constant, or COBOL sort method. Table 2-2 summarizes the formats of all parameter table entries. The most significant character (two octal digits) of each entry is a code specifying the type of information contained in the remaining portion of the word. For example, code 00 is a table header and code 71 is the end-of-table sentinel; each of these two entries includes the table address. Code 77 specifies that the current entry is a continuation of the previous one. It is used when more than one word is required to store the detailed information related to a given entry. For this reason, the interpretation of the field for code 77 entries varies according to the preceding entry.

Table 2-3 provides detailed descriptions of the various parameter table entry formats. Formats are listed in the same order as they appear in Table 2-2.

Character Positions					
6	5	4	3	2	1
35			Bit Positions		0
00			Address of the first word of this table		
01			Words per record (octal)		
02			Words per link (octal)		
05	No. words in var. size record		No. of records this size		
03	Type of Key Field **	Ordering Sequence **	Position of MS. bit	Significance of key field	
77	Word Number		Number of bits in key field (octal)		
04			Address of start of translate table		
20			Number of words of working storage (octal)		
77			Starting address of working storage		
23			Address of External File Name		
40			Address of DROC		
41			Address of own code compare subroutine		
42			Address of FPOC		
43			Address of LPOC		
44			Address of final return		
30	* Indicates automatic large volume sort				
31	* Indicates part A only				
32	* Indicates part B only				
33	* Indicates part C only				
34	* Recreates tape within part A				
35	* Recreates tape within part B				
36	* Continues sort from within part A				
37	* Continues sort from within part B				
47	* Continues sort from within part C				
10	Maximum number of records per cycle				
11	Maximum reels per cycle (octal)				
12				Label prefix for merge output	
13				Maximum words per block on output tape (octal)	
14			Address of External File Name for 'Part B' or 'Part C' Input		
15			Address of External File Name for 'Part B' Output		
50	Number of first record on an output tape (octal)				
51	Number of last record on an output tape (octal)				
52	Cycle number for output tape (octal)				
53			Address of 2-word packet containing label and reel number		
54			Address of tape repositioning table		
60	Delete drum checksum	Delete tape checksum	Delete FASTRAND checksum		
06				Bias factor	
61				Minimum drum block size	
62				Minimum FASTRAND block size	
63	*Indicates deletion of console display of end-of-sort messages				
64			Volume to be sorted in thousands of records		
70			Address of table linked to this one		
72	Null				
71			Address of first word of table		

* Defines meaning of first 2 digits. The remainder of the word is void.

** Fieldata

Table 2–2. Summary of Parameter Table Entries

ENTRY FORMAT (IN OCTAL)	FUNCTION OR USE OF ENTRY	INTERPRETATION OF SPECIAL FIELDS*
00 00 00 *xx xx xx*	Table header word	*xxxxxx* field - specifies starting address of parameter table
01 00 00 *xx xx xx*	Required when sorting fixed size records	*xxxxxx* field - specifies record size in words
02 00 00 *xx xx xx*	Required when sorting variable size records	*xxxxxx* field - specifies mm - links size in words
05 *yy yy xx xx xx*	Required when sorting variable size records	*yyyy* field - specifies variable-record size in words *xxxxxx* field - specifies number of variable records of size yyyy
03 *tt oo bb ff ff* 77 *ww ww nn nn nn*	03 code specifies that entry is a key field description. 77 code indicates continuation of information for key field description.	*tt* field - Fieldata character indicating type of key field. Field must contain one of the following codes: 06 key field type A (alphanumeric) 07 key field type B (UNIVAC signed binary field) 11 key field type D (Fieldata signed, decimal field) 22 key field type M (IBM† signed binary field) 32 key field type U (unsigned binary field) *oo* field - Fieldata character specifying sorting sequence. Field must contain one of the following codes: 00 or 06 specifies ascending sequence 11 specifies descending sequence *bb* field - specifies bit position, within a word, of most significant bit of this key field. The most significant bit position within a word is number 1; the least significant bit position is number 36. *ffff* field - specifies significance of key field. If *ffff* is zero for any key field, it must be zero for all key fields. In this instance, key field significance depends on sequence of entries in parameter table. Thus, the first 03 entry encountered is the most significant and all succeeding 03 entries represent succeedingly less significant key fields. If some other sequence is desired, *ffff* is set to one and the remaining fields are numbered consecutively according to their relative importance. Higher numbers designate less significant fields. In this case, the set of values for *ffff* must include all values from 1 through n where n is the total number of key fields specified. *wwww* field - specifies word number, within the record, of the word containing most significant bit of key field. First word of record is number zero. *nnnnnn* field - specifies number of bits in key field
04 00 00 *xx xx xx*	Points to translation table	*xxxxxx* field - starting address of 64-word translation table. Each entry in the translation table takes the form: 00 00 00 00 00 *yy* Each *yy* is the octal representation of a character, and its position in the table represents the desired relationship of this character to entire character set. For example, the letters A and B in octal notation are represented by 06 and 07 respectively. In the normal collating process, the letter B tests greater than A, and all B's appear after A's in the final sorted output. If it were desired to have B's appear before A's, the translation table would be as follows: 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00 00 00 02 00 00 00 00 00 03 00 00 00 00 00 04 00 00 00 00 00 05 00 00 00 00 00 07 00 00 00 00 00 06 00 00 00 00 00 10 . . . 00 00 00 00 00 77
20 00 00 *xx xx xx* 77 00 00 *yy yy yy*	Defines working storage	*xxxxxx* field - specifies number of words of working storage *yyyyyy* field - starting address of working storage area
23 00 00 *xx xx xx*	N/A	*xxxxxx* field - address of external-file-name
40 00 00 *xx xx xx*	N/A	*xxxxxx* field - address of DROC (Data Reduction Own Code)
41 00 00 *xx xx xx*	N/A	*xxxxxx* field - address of own code compare subroutine
42 00 00 *xx xx xx*	N/A	*xxxxxx* field - address to which sort subroutine transfers control upon completion of ROPN $ linkage
43 00 00 *xx xx xx*	N/A	*xxxxxx* field - address to which sort subroutine returns control at completion of RSORT$ linkage
44 00 00 *xx xx xx*	N/A	*xxxxxx* field - address to which sort subroutine returns control at completion of part A or part B of large-volume sort
30 00 00 00 00 00	Indicates an automatic large volume sort is to be executed	N/A
31 00 00 00 00 00	Indicates that part A of nonautomatic large volume sort is to be executed	N/A

Table 2–3. Parameter Entry Table Formats
(Part 1 of 2)

ENTRY FORMAT (IN OCTAL)	FUNCTION OR USE OF ENTRY	INTERPRETATION OF SPECIAL FIELDS*
32 00 00 00 00 00	Indicates that part B of nonautomatic large volume sort is to be executed	N/A
33 00 00 00 00 00	Indicates that part C of nonautomatic large volume sort is to be executed	N/A
34 00 00 00 00 00	Indicates that a part A tape of a large volume sort is to be recreated	N/A
35 00 00 00 00 00	indicates that a part B tape of a large volume sort is to be recreated	N/A
36 00 00 00 00 00	Indicates that sort is being continued from within part A of a large volume sort	N/A
37 00 00 00 00 00	Indicates that sort is being continued from within part B of a large volume sort	N/A
47 00 00 00 00 00	Indicates that sort is being continued from within part C of a large volume sort	N/A
10 *xx xx xx xx xx*	N/A	Specifies maximum number of records to be sorted in any one cycle
11 *xx xx xx xx xx*	N/A	Specifies maximum number of output reels per cycle
12 00 00 00 *11 11*	N/A	*1111* field – user-supplied, two-character label for intermediate merging
13 00 00 00 *xx xx*	N/A	*xxxx* field – specifies maximum output block size in words
14 00 00 *xx xx xx*	N/A	*xxxxxx* field – address of external-file-name for an input file
15 00 00 *xx xx xx*	N/A	*xxxxxx* field – address of external-file-name for an output file
50 *xx xx xx xx xx*	N/A	*xxxxxxxxxx* field – specifies first record which is to appear on a recreated output tape
51 *xx xx xx xx xx*	N/A	*xxxxxxxxxx* field – specifies last record which is to appear on a recreated output tape
52 00 00 00 0*x xx*	N/A	*xxx* field – specifies cycle number for recreated output tape
53 00 00 *xx xx xx*	Specifies tape containing rerun information	*xxxxxx* field – address of two-word packet containing label and reel number of tape containing rerun information for reproducing a tape which was originally created during part B of a large volume sort Packet format is as follows: Word 1 *yyyyyyyyyyyy* Word 2 00000000*zzzz* where: *yyyyyyyyyyyy* is the label *zzzz* is the reel number
54 00 00 *xx xx xx*	N/A	*xxxxxx* field – address of tape repositioning table
60 *dd tt ff* 00 00	Specifies omission of checksums	*dd* field – if nonzero, drum checksum is omitted *tt* field – if nonzero, tape checksum is omitted *ff* field – if nonzero, FASTRAND checksum is omitted
06 00 00 00 *xx xx*	Specifies bias of input	*xxxx* field – specifies bias factor times 10
61 00 00 00 *xx xx*	Indicates that a check is to be made on drum block size	*xxxx* field – specifies minimum drum block size in words
62 00 00 00 *xx xx*	Indicates that a check is to be made on FASTRAND block size	*xxxx* field – specifies minimum FASTRAND block size in words
63 00 00 00 00 00	Specifies omission of end-of-sort messages from console	N/A
64 00 00 *xx xx xx*	Specifies volume to be sorted	*xxxxxx* field – specifies number of records in thousands
70 00 00 *xx xx xx*	N/A	*xxxxxx* field – address of another parameter table that is linked to this one
72 00 00 00 00 00	Used to reserve a word in the table for the user	N/A
71 00 00 *xx xx xx*	End-of-table sentinel	*xxxxxx* field – starting address of this parameter table

* Numerical representation in special fields is in octal notation unless specified otherwise.

† Registered Trademark of International Business Machines Corporation

Table 2–3. Parameter Entry Table Formats
(Part 2 of 2)

2.2.3. Key Fields and Their Translation

The UNIVAC 1106/1108 compare instructions (test greater than, equal to, and less than) operate on the ordered set of 36-bit signed binary numbers of the word format described in the following paragraphs. To obtain maximum efficiency from the key comparison routine, all key fields are translated as they enter the sort file so that the correct ordering may be obtained by the key comparison routines in the minimum amount of time. The translation varies for each type of key field. Just before the item is returned from the sort file, the key fields are translated to their original form. The key field formats and their translations are given in the following paragraphs. In addition to the indicated translations, all bits of key fields which fall into bit position 35 of a word are complemented.

- Unsigned binary, 'U' – key field contains the magnitude of a number in binary notation and no translation is made.
- Signed binary, UNIVAC 1106/1108 format 'B' – key field contains a number in UNIVAC 1106/1108 fixed-point (or single-precision floating point) notation. A positive number is represented by a 0 in the sign bit (leftmost bit) position and the magnitude in binary notation in the remaining bit positions. A negative number is represented by a 1 in the sign bit position and the ones complement of the binary representation of a positive number of the same magnitude. On translation, the sign (leftmost) bit is complemented.
- Signed binary, IBM format 'M' – key field contains a number in IBM 7090 fixed-point binary representation. The leftmost bit is the sign bit; 0 indicates positive, 1 indicates negative. The remaining bit positions contain the magnitude in binary notation. If the key field contains a positive number, only the sign bit is complemented, if the key field contains a negative number, the entire field is complemented.
- Alphanumeric 'A' – key field contains alphanumeric characters represented by six-bit bytes which are ordered as if they were unsigned binary numbers. If other than the normal ordering is desired, the different ordering must be specified using 'SEQ' parameter list. This field must start within a word at bit positions 5, 11, 17, 23, 29, or 35. No translation is made unless a 'SEQ' list has been specified. In this case, the key field is translated in six-bit bytes. Each byte is translated to n, where the value of the byte equals the nth field following 'SEQ' in the 'SEQ' list.
- Signed decimal, Fieldata format 'D' – key field contains a signed number represented in Fieldata characters. The first character is the sign and must be a space or plus for positive numbers and a minus for negative numbers. The remaining characters can be any of the Fieldata integers 0 through 9. The field must begin within a word at bit positions 5, 11, 17, 23, 29, or 35. If the first key field character is a minus sign, the entire key field, with the exception of the sign character, is complemented. Any other character in the first character position of the field is changed to a plus sign and is translated to a space just before being returned from the sort subroutine. Thus, a key field is returned to its original form only if a positive number is represented by a space as its first character.

- Overpunched Fieldata decimal 'P' – key field contains a decimal number represented in Fieldata characters with the sign represented in the low order character of the field by an overpunch character if the field is signed (negative or positive). Start bits possible for the field are the same as for alphanumeric 'A'. The "overpunch" characters are explained fully in UP-7923 (American National Standard COBOL), Page 3–45.

- Overpunched ASCII decimal 'Q' – key field contains a decimal represented in ASCII characters with the sign (if present) represented by an overpunch in the low order ASCII character field. The field must start in bits 8, 17, 26, or 35 of the word. See UP-7923, Page 3–45, for discussion of "overpunch" characters.

- Signed ASCII decimal 'R' – key field contains a signed number represented in ASCII characters. The first character of the field is the sign and must be an ASCII minus (–) for negative numbers. Any other character is assumed to indicate a positive number and is changed to an ASCII plus sign (+) for sorting. If the key field has an ASCII minus sign in the first character, the entire key field is complemented. Plus sign fields are translated to an ASCII blank in the leading character before return to the user. Field start bits are on quarter word boundaries (bits 8, 17, 26, or 35).

- Table translated ASCII 'S' – key field is ASCII characters with character translation requested via a 'SEQ' parameter entry in the parameter table. If 'SEQ' is specified, the quarter word fields of the key field are altered to the quarter word specified in the 'SEQ' list for ASCII characters. Field start bits are the same as for the 'R' field type.

2.3. DATA REDUCTION – OWN CODE

If a 'DROC' parameter list (see 2.2.1.11) appears in the set of parameters of a sort program, the sort subroutine transfers control to the *address* parameter in this list when two records with equal keys are found. The addresses of the two records are in registers A0 and A1. Own code may combine the two records to form a single record in the main storage area, addressed by A0, in which case the sort subroutine must be re-entered by J 1,X11. Upon further inspection of the contents of the records, if own code elects not to combine the two records, the sort subroutine must be re-entered, by a J 0,X11. The procedures R$DLT and R$NDLT (for Delete and No Delete) generate the instructions J 1,X11 and J 0,X11, respectively. Own code must not alter the contents of A0 or A1. Furthermore, own code must not alter the key fields of the two records if not combined, nor the key field of the record addressed by A0 if they are combined. In some cases, the sort subroutine translates the key fields. If these fields are to be inspected by own code, the author of own code must follow the translation rules given in 2.2.3.

Data reduction own code may use registers A2, A3, A4, A5, R1, R2, and R3 without saving and restoring them. Data reduction own code may not be used with variable size items.

2.4. OWN CODE COMPARISON

If a 'COMP' parameter list (see 2.2.1.12) appears in the set of parameters for a sort program and it is necessary to decide which of two records is to appear first in the final sorted sequence, the sort subroutine transfers control to the *address* parameter specified in this list.

The addresses of the two records are in registers A0 and A1. If the record whose address is in A0 is to precede, own code should re-enter the sort subroutine by a J 0,X11. If the record whose address is in A 1 is to precede, own code should re-enter the sort subroutine by a J 1,X11. If the final order is immaterial, as in the case of equal keys, own code should re-enter the sort subroutine by a J 2,X11. Jump instructions with these addresses are generated by the R$A0, R$A1, R$AE procedures, respectively. Data reduction own code, if present, is executed (when appropriate) only if re-entry from own code comparison is by a J 2,X11. Own code must not alter the contents of A0 or A1.

If 'COMP' is not present, a comparison subroutine is generated by way of the 'KEY' list. If the 'KEY' or 'KEYW' list and 'COMP' are both present, the sort subroutine observes the 'KEY' or 'KEYW' lists by translating key fields; however, the own code comparison subroutine is used by the sort subroutine. If key translation is utilized, the author of own code should follow the translation rules given in 2.2.3.

Own code comparison may not destroy the contents of A0, A1, A2, A3, or X11.

3. PROGRAM LINKAGE

3.1. SMALL VOLUME SORTS AND THEIR PROGRAM LINKAGES

A small volume of data is defined as the amount of data that can be sorted without operator intervention. Minimally, this is the amount of data that can be contained on a single reel of tape. However, if adequate drum storage is available, it could include several reels of tape; therefore, a small volume of data can range from 3 to 15 million words.

3.1.1. ROPN$

- The linkage

  label J ROPN$

  initializes the sort subroutine. When this line is executed, control register 15 (A3) must contain the address of the first word of the parameter table.

  This linkage can be generated by the procedure call:

  R$OPN *parameter-table-name* (this call prepares register A3)

  If the R$FILE procedure was used to construct the parameter table, *parameter-table-name* will be the label assigned to the R$FILE procedure call line.

- Exit Conditions

  The sort subroutine returns control to the address specified in the 'FPOC' parameter list (2.2.1.13) and saves and restores the contents of all A registers, X registers, and registers R1, R2, R3.

3.1.2. RREL$

- The linkage

  label LMJ 11, RREL$

  releases a record to the sort subroutine. When this line is executed, control register 12 (A0) must contain the address of the first word of the record to be sorted. If variable length records are being sorted, then the most significant half of register 12 (A0) must contain the length of the record.

  This linkage can be generated by the procedure call:

  R$REL

  This call does not prepare register 12 (A0).

- Exit Conditions

  The sort subroutine returns control to the line immediately following the call on R$REL and saves and restores the contents of all A registers, X registers, and registers R1, R2, and R3.

3.1.3. RSORT$

- The linkage

```
J       RSORT$
```

informs the sort subroutine that no more records are to be released.

This linkage can be generated by the procedure call:

R$SORT

- Exit Conditions

The sort subroutine returns control to the address specified in the 'LPOC' parameter list (see 2.2.1.14) and saves and restores the contents of all registers which it uses.

3.1.4. RRET$

- The linkage

label LMJ   11, RRET$

label + *1* *at-end-address*

requests the sorted output records from the sort subroutine. This linkage can be generated by the procedure call:

R$RET *at-end-address*

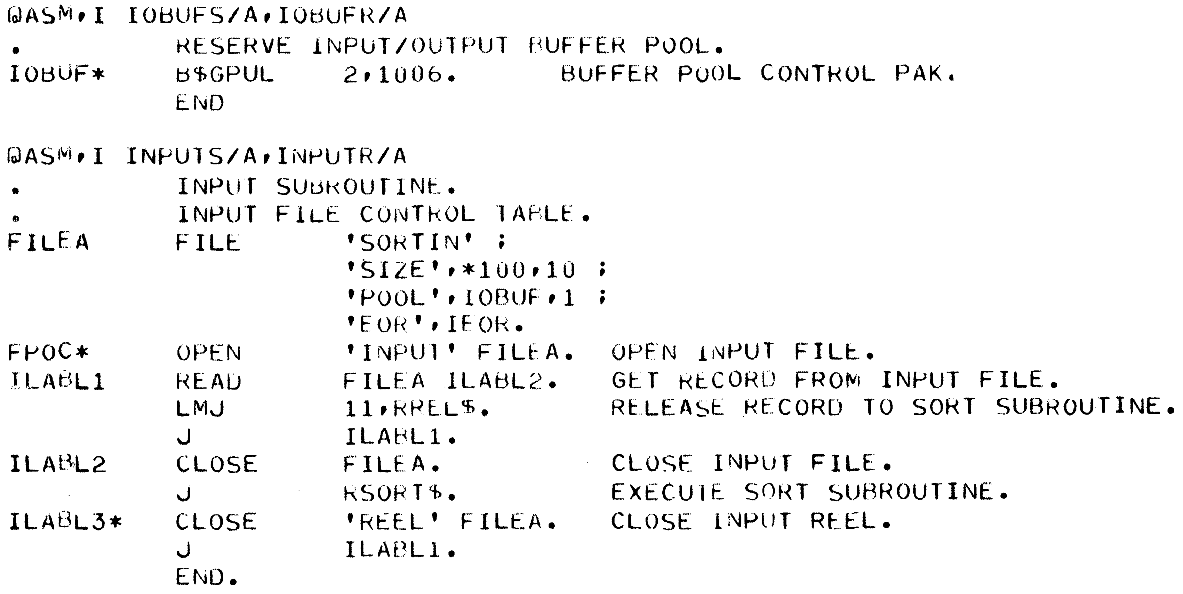
- Exit Conditions

Normal return is made to line *label* + *2*. Control register 12 (A0) contains the address of the first word of the record being returned. If variable length records are being sorted, then the most significant half of register 12 (A0) specifies the length of the record in words (binary number). No other registers are changed. If no record remains when this linkage is executed, return is made to the address specified by *at-end-address*.

3.1.5. Preparing a Small Volume Sort Program

As an example, assume a sort program is needed to sort one reel of data. The data consists of ten-word records with a two-word alphanumeric key field in the first two words. The hardware available for sorting consists of 150,000 words of FH-880 Magnetic Drum storage, 25 FASTRAND storage positions, and 16,000 words of main storage.

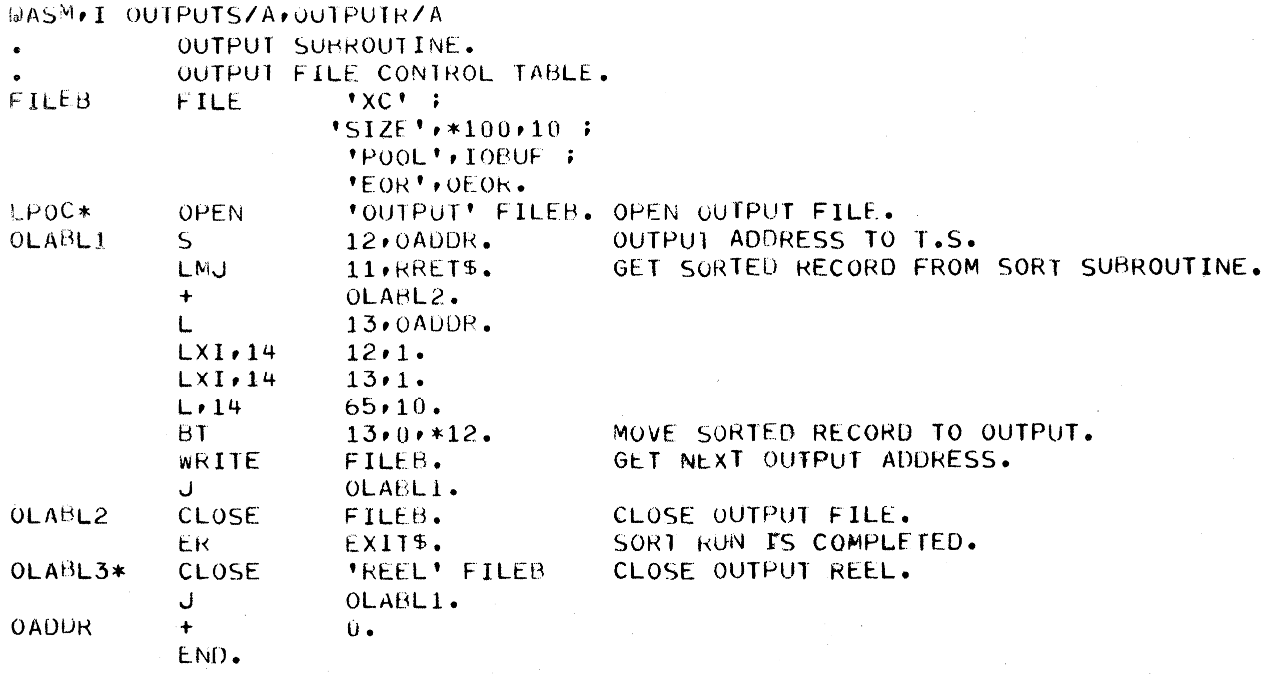
Figures 3-1 through 3-4 illustrate the coding for a set of cards which will assemble and execute the program.



```
@ASM,I IOBUFS/A,IOBUFR/A
.          RESERVE INPUT/OUTPUT BUFFER POOL.
IOBUF*     B$GPUL     2,1006.       BUFFER POOL CONTROL PAK.
           END

@ASM,I INPUTS/A,INPUTR/A
.          INPUT SUBROUTINE.
.          INPUT FILE CONTROL TABLE.
FILEA      FILE       'SORTIN' ;
                      'SIZE',*100,10 ;
                      'POOL',IOBUF,1 ;
                      'EOR',IEOR.
FPOC*      OPEN       'INPUT' FILEA.  OPEN INPUT FILE.
ILABL1     READ       FILEA ILABL2.   GET RECORD FROM INPUT FILE.
           LMJ        11,RREL$.       RELEASE RECORD TO SORT SUBROUTINE.
           J          ILABL1.
ILABL2     CLOSE      FILEA.          CLOSE INPUT FILE.
           J          RSORT$.         EXECUTE SORT SUBROUTINE.
ILABL3*    CLOSE      'REEL' FILEA.   CLOSE INPUT REEL.
           J          ILABL1.
           END.
```

Figure 3–1. Sample Input Routine for a Sort Program



```
@ASM,I OUTPUTS/A,OUTPUTR/A
.          OUTPUT SUBROUTINE.
.          OUTPUT FILE CONTROL TABLE.
FILEB      FILE       'XC' ;
                     'SIZE',*100,10 ;
                      'POOL',IOBUF ;
                      'EOR',OEOR.
LPOC*      OPEN       'OUTPUT' FILEB. OPEN OUTPUT FILE.
OLABL1     S          12,OADDR.       OUTPUT ADDRESS TO T.S.
           LMJ        11,RRET$.       GET SORTED RECORD FROM SORT SUBROUTINE.
           +          OLABL2.
           L          13,OADDR.
           LXI,14     12,1.
           LXI,14     13,1.
           L,14       65,10.
           BT         13,0,*12.       MOVE SORTED RECORD TO OUTPUT.
           WRITE      FILEB.          GET NEXT OUTPUT ADDRESS.
           J          OLABL1.
OLABL2     CLOSE      FILEB.          CLOSE OUTPUT FILE.
           ER         EXIT$.          SORT RUN IS COMPLETED.
OLABL3*    CLOSE      'REEL' FILEB    CLOSE OUTPUT REEL.
           J          OLABL1.
OADDR      +          0.
           END.
```

Figure 3–2. Sample Output Routine for a Sort Program

```
@ASM,I PARAM1S/A,PARAM1R/A
.          STANDARD PARAMETERS FOR SAMPLE SORTS.
PARAM1*    R$FILE      'RSZW',10 ;                . RECORDSIZE.
                       'KEYW',1,35,72,'A','A'. KEY FORMAT.
           END.

@ASM,I PARAM2S/A,PARAM2R/A
.          STANDARD PARAMETERS FOR SMALL VOLUME AND AUTOMATIC LARGE VOLUME
.          SAMPLE SORTS
PARAM2*    R$FILE      'FPOC',FPOC ;             . FIRST PASS OWN CODE ADDRESS.
                       'LPOC',LPOC ;             . LAST PASS OWN CODE ADDRESS
                       'COPY',RSTD$ ;            . STANDARD SORT FILE NAMES.
                       'COPY',PARAM1.
           END.

@ASM,I SORTSVS/A,SORTSVR/A
IEOR*      EQU         ILABL3.       END INPUT REEL ADDRESS.
OEOR*      EQU         OLABL3.       END OUTPUT REEL ADDRESS.
START      L,14        15,PARAM2.    PARAMETER TABLE ADDRESS TO A3.
           J           ROPN$.        OPEN SORT SUBROUTINE
           END         START.
```

Figure 3–3. Coding for Small Volume Sort

```
@PREP
@MAP,I   D,SR1
IN       SORTSVR
@ASG,T   R$CORE,F///16.        SPECIFY CORE AREA SIZE.
@ASG,T   SORTIN,8C,INPUT.      ASSIGN INPUT FILE.
@ASG,T   XC,C,SCRTCH.          ASSIGN OUTPUT FILE.
@ASG,T   XA,D/150000.          ASSIGN DRUM AREA TO BE USED BY SORT.
@ASG,T   XB,F//POS/25.         ASSIGN FASTRAND AREA TO BE USED BY SORT.
@XQT     SR1
```

Figure 3–4. Control Cards for a Small Volume Sort

```
@ASM,I PARAM3S/A,PARAM3R/A
.          PARAMETER TABLE FOR AUTOMATIC LARGE VOLUME SORT.
PARAM3*    R$FILE      'SMRG','AA'
                       'COPY',PARAM2.
           END.

@ASM,I SRTLVS/A,SRTLVR/A
IEOR*      LMJ         11,RENCY$.    AT END OF INPUT REEL, END PART A CYCLE.
           J           ILABL3.
OEOR*      LMJ         11,IPURI$.    AT END OF OUTPUT REEL, PUNCH RESTART
.                      INFORMATION.
           J           OLABL3.
START      L,14        15,PARAM3.    PARAMETER TABLE ADDRESS TO A3.
           J           ROPN$.        OPEN SORT SUBROUTINE.
           END         START.
```

Figure 3–5. Coding for a Large Volume Automatic Sort

3.2. LARGE VOLUME SORTS

A large volume sort is one for which sufficient hardware is not available to complete the sort without operator intervention. The small volume sort does not require operator intervention and, therefore, uses only a subset of the subroutine elements required for the large volume sort.

A large volume sort requires three separate phases:

- Part A – sort groups of records into many output files;
- Part B – merge small output files into larger output files; and
- Part C – merge output to a single file.

These three operations can be done either automatically with the sort subroutine controlling the entire operation or nonautomatically (one part at a time).

Typically, the sequence of operations for a large volume sort starts with the assignment of a number of tape units for temporary storage as well as areas of main and mass storage. In part A, the sort subroutine accepts input data from the user program one record at a time and periodically produces a sorted output file for part of the data. This process continues until all the input data have been sorted onto output tapes and the tapes have been systematically labeled by the operator. The portion of the sort subroutine which produces one output file is called a cycle.

During part B, the output tapes must be mounted on available tape units and merged into larger files. The number of such merge operations during part B depends on the number of files produced by part A and the number of tape units available. Part B ends when the number of sorted files is less than the number of tape units available. In part C, the sorted files are merged into a single output file.

3.2.1. Automatic Large Volume Sorts

This is the most common method of sorting large volumes of data. The sort subroutine controls demounting, labeling, and remounting of tapes through console messages to the operator.

3.2.1.1. Parameters

The 'SMRG' parameter list (see 2.2.1.16) is required. The *output-label* field in this list is used to keep track of intermediate output tapes from different sort programs. The *records/cycle* and *reels/cycle* fields are optional. They may be used to control the length of the sort cycle.

3.2.1.2. Sort Subroutine Linkages

The required sort subroutine linkages are: ROPN$, RREL$, RSORT$, and RRET$ (see 3.1.1 through 3.1.4). The following three linkages are optional:

- RENCY$

  – The linkage

  label LMJ 11, RENCY$
  may be used during 'FPOC' to cause the sort subroutine to terminate cycles at specific points in the routine. It is generated by the procedure call:

  R$ENCY

  – Exit Condition

  The sort subroutine returns control to line *label + 1* preserving the contents of all A registers, X registers, and registers R1, R2, R3.

- RINFO$

  – The linkage

  label LMJ 11, RINFO$
  may be used during 'LPOC' to request the address of a table containing rerun information. This table is required in order to continue an interrupted sort from some point in part C of a large volume sort.

  – Exit Conditions

  The sort subroutine returns control to location *label + 1*. Control register A0 contains the table address and control register A1 contains the length of the table in words (binary number). The contents of R1, A2, A3, A4, and A5 are destroyed.

- IPURI$

  – The linkage

  label LMJ 11, IPURI$
  may be used during 'LPOC' instead of the RINFO$ linkage. This linkage punches the rerun information on cards. These cards also contain an identifier for user convenience. The output on the printer provides the user with the cycle number and number of cards punched each time IPURI$ is entered. The cards produced by IPURI$ can be processed by the parameter card routine should restart become necessary during part C.

  – Exit Conditions

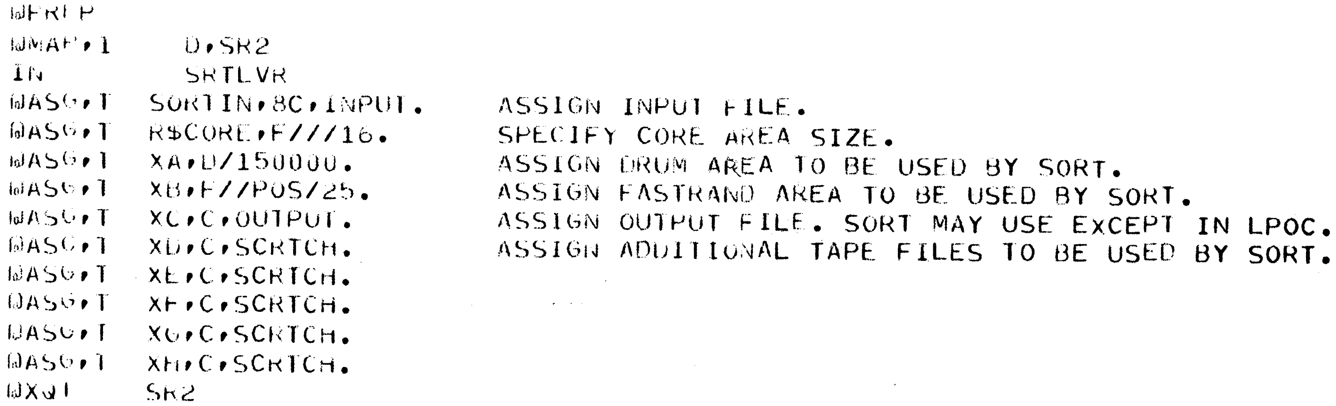
  The sort subroutine returns control to location *label + 1*. The contents of R1, A2, A3, A4, and A5 are destroyed.

3.2.1.3. Preparing a Large Volume Automatic Sort Program

As an example, assume a sort program is needed to sort an unknown volume of data. The data consists of ten-word records with a two-word alphanumeric key field in the first two words. The hardware available for sorting consists of 150,000 words of FH-880 Magnetic Drum storage, 25 FASTRAND storage positions, and 16,000 words of main storage.

Since, with exception of the amount of data, this example is the same as for the small volume sort, we will assume that all elements assembled in 3.1.5 are available in the program file. The input and output routines may then be used. Linkages for the user program to control rerun points in part A and part C can be made at the end-of-reel addresses-IEOR for end-of-input-reel and OEOR for end-of-output-reel. The only additional parameter list required is 'SMRG'; therefore, the 'COPY' list may be used to avoid recording the parameter lists already included in the element PARAM2R.

Figure 3-5 illustrates the assembly of the parameter table and the linkage necessary to open the sort. Figure 3-6 illustrates the control cards needed to assign the hardware facilities and execute the program.



```
@PREP
@MAP,I      D,SR2
IN          SRTLVR
@ASG,T   SORTIN,8C,INPUT.       ASSIGN INPUT FILE.
@ASG,T   R$CORE,F///16.         SPECIFY CORE AREA SIZE.
@ASG,T   XA,D/150000.           ASSIGN DRUM AREA TO BE USED BY SORT.
@ASG,T   XB,F//POS/25.          ASSIGN FASTRAND AREA TO BE USED BY SORT.
@ASG,T   XC,C,OUTPUT.           ASSIGN OUTPUT FILE. SORT MAY USE EXCEPT IN LPOC.
@ASG,T   XD,C,SCRTCH.           ASSIGN ADDITIONAL TAPE FILES TO BE USED BY SORT.
@ASG,T   XE,C,SCRTCH.
@ASG,T   XF,C,SCRTCH.
@ASG,T   XG,C,SCRTCH.
@ASG,T   XH,C,SCRTCH.
@XQT     SR2
```

Figure 3–6. Control Cards for a Large Volume Automatic Sort

3.2.2. Nonautomatic Sorting of Large Volumes of Data

The user may choose to execute a large volume sort in a number of separate runs. These runs are designated as part A, part B, and part C. Part A and part C are each executed once for every large volume sort. The number of times part B must be executed is dependent upon the number of intermediate output files produced in part A. Part B may not be executed at all.

3.2.2.1. Part A

- Parameters
  - The 'PARTA' parameter list (see 2.2.1.17) is required. The 'FINAL' parameter list (see 2.2.1.15) is optional, and if not specified, control passes to EXIT$ at the conclusion of part A.

- Sort Subroutine Linkages
  - ROPN$ – (see 3.1.1)
  - RREL$ – (see 3.1.2)
  - RSORT$ – This linkage is similar to that described in 3.1.3 except that the sort subroutine returns control to the address specified in the 'FINAL' parameter list (see 2.2.1.15) or, if not specified, to EXIT$.
  - RENCY$ – (see 3.2.1.2.)

3.2.2.2. Part B

- Parameters
  - The 'PARTB' parameter list (see 2.2.1.18) is required. The 'FINAL' parameter list (see 2.2.1.15) is optional and, if not specified, control is passed to EXIT$ at the conclusion of part B.
- Sort Subroutine Linkages
  - ROPN$ – This linkage is similar to that described in 3.1.1 except that the sort subroutine returns control to the address specified in the 'FINAL' parameter list or, if not specified, to EXITS$.

3.2.2.3. Part C

- Parameters
  - The 'PARTC' parameter list (2.2.1.19) is required.
- Sort Subroutine Linkages
  - ROPN$ – This linkage is similar to that described in 3.1.1 except that the sort subroutine returns control to the address specified in the 'LPOC' parameter list.
  - RRET$ – (see 3.1.4)
  - RINFO$ – (see 3.2.1.2)

    or

  - IPURI$ – (see 3.2.1.2)

3.2.2.4. Preparing a Large Volume Nonautomatic Sort Program

As an example, the sort program described in 3.2.1.3 could be divided into three separate runs as illustrated in Figures 3-7 through 3-9. Assume that the volume of data to be sorted was such that part A produced seven output files. The labels on these files would be 'AAA001' through 'AAA007'.

Part B would be performed once producing a single file labeled 'AAB001'.

The input to part C would then consist of tapes labeled 'AAA004' through 'AAA007' and 'AAB001'.

```
@ASM,I PARAM4S/A,PARAM4R/A
.            PARAMETER TABLE FOR PART A ONLY.
PARAM4*      R$FILE     'PARTA','AA' ;
                        'FPOC',FPOC ;
                        'COPY',RSTD$ ;
                        'COPY',PARAM1.
             END.

@ASM,I PARTAS/A,PARTAR/A
IEOR*        LMJ        11,RENCY$.  AT END OF INPUT REEL, END PART A CYCLE.
             J          ILABL3.
START        L,14       15,PARAM4.  PARAMETER TABLE ADDRESS TO A3.
             J          ROPN$.      OPEN SORT SUBROUTINE.
             END        START.

@PREP
@MAP,I   D,SR3
IN       PARTAR
@ASG,T   SORTIN,C,INPUT.            ASSIGN INPUT FILE.
@ASG,T   R$CORE,F///16.             SPECIFY CORE AREA SIZE.
@ASG,T   XA,D/150000.               ASSIGN DRUM AREA TO BE USED BY SORT.
@ASG,T   XB,F//POS/25.              ASSIGN FASTRAND AREA TO BE USED BY SORT.
@ASG,T   XC,C,OUTPUT.               ASSIGN OUTPUT FILE FOR SORT INTERMEDIATE OUTPUT
@XQT     SR3
```

Figure 3–7. Part A of a Large Volume Nonautomatic Sort

```
@ASM,I PARAM5S/A,PARAM5R/A
.            PARAMETER TABLE FOR PART B ONLY.
PARAM5*      R$FILE     'PARTB','AA','XC','AAA001','AAA002','AAA003' ;
                        'COPY',PARAM1.
             END.

@ASM,I PARTBS/A,PARTBR/A
START        L,14       15,PARAM5.  PARAMETER TABLE ADDRESS TO A3.
             J          ROPN$.      OPEN SORT SUBROUTINE.
             END        START.

@PREP
@MAP,I   D,SR4
IN       PARTBR
@ASG,T   R$CORE,F///8.              SPECIFY CORE AREA SIZE.
@ASG,T   XC,C,OUTPUT.               ASSIGN OUTPUT FILE FOR SORT INTERMEDIATE OUTPUT
@ASG,T   AAA001,C,INPUT.            ASSIGN TAPE FILES FOR PART B INPUT.
@ASG,T   AAA002,C,INPUT.
@ASG,T   AAA003,C,INPUT.
@XQT     SR4
```

Figure 3–8. Part B of a Large Volume Nonautomatic Sort

```
@ASM,I PARAM6S/A,PARAM6R/A
.         PARAMETER TABLE FOR PART C ONLY.
PARAM6*   R$FILE    'PARTC','AAA004','AAA005','AAA006','AAA007','AAB001' ;
                    'LPOC',LPOC ;
                    'COPY',PARAM1.
          END.

@ASM,I PARTCS/A,PARTCR/A
OEOR*     LMJ       11,IPURI$.   AT END OF OUTPUT REEL, PUNCH RESTART INFORMATION.
          J         OLABL3.
START     L,14      15,PARAM6.   PARAMETER TABLE ADDRESS TO A3.
          J         ROPN$.       OPEN SORT SUBROUTINE.
          END       START.

@PREP
@MAP,I   D,SR5
IN       PARTCR
@ASG,T   R$CORE,F///10.          SPECIFY CORE AREA SIZE.
@ASG,T   XC,C,OUTPUT.            ASSIGN OUTPUT FILE FOR FINAL OUTPUT.
@ASG,T   AAA004,C,INPUT.         ASSIGN TAPE FILES FOR PART C INPUT.
@ASG,T   AAA005,C,INPUT.
@ASG,T   AAA006,C,INPUT.
@ASG,T   AAA007,C,INPUT.
@ASG,T   AAB001,C,INPUT.
@XQT     SR5
```

Figure 3-9. Part C of a Large Volume Nonautomatic Sort

3.2.3. Tape Labeling

During part A of an automatic large volume sort, a number of intermediate output files are produced, each file consisting of one or more reels. As each reel is completed, information is printed out which instructs the operator to demount and label the reel. When all input records have been delivered to the sort and all intermediate output files have been produced in part A, one or more merge runs is required depending upon the number of intermediate files and the number of tape units available for merging. If the number of tape units exceeds the number of files, only one merge run is necessary and the routine proceeds directly from part A to part C. During part C, sorted data is returned to the user program one record at a time.

If the number of files to be merged equals or exceeds the number of tape units available, one or more merge runs is required prior to the final merge. These intermediate merges constitute part B of the automatic large volume sort. The intermediate merge runs continue until the number of files is one less than the number of tape units available. As in part A, the program types out instructions for labeling and mounting tape files.

The labels used for output files during both part A and part B consist of one word in the following format:

USER'S SORT RUN IDENTIFICATION	MERGE LEVEL	OUTPUT LEVEL NUMBER
35 24	23 18	17 0

BITS	USE
0-17	Output level number - output file number of specified merge starting at 001 for each merge level.
18-23	Merge level - during part A; this is always A. For part B, it advances from B to C to D, and so on, for successive merge levels.
24-35	User's sort run identification - two-character, user-supplied identification. Characters may be any one of the allowable character set.

All merging in part B is to the maximum way of merge using all available tape units with the possible exception of the first merge. If necessary, the first merge reduces the total number of files to a point where the tape units assigned can be used to the fullest extent.

During an automatic sort, the level of the first merge is automatically calculated. In nonautomatic sorting, the user must calculate the first merge level himself using the following two expressions:

$K^n \leq T < K^{n+1}$ and

First merge level = $\frac{T - K^n}{K - 1}$

where:

T = the number of files to be merged
K = the number of tape units assigned for input

Example:

Let T = 21 files and K = 4 tape units.

$4^n \leq 21 < 4^{n+1}$

$n = 2$

Merge level = $\frac{T - K^n}{K - 1} = \frac{21 - 16}{4 - 1} = \frac{5}{3}$

If the remainder is 0, the first merge level equals the number of tape units; if the remainder is not 0, the first merge level equals the remainder plus 1 and, in this case, 3.

Based on this information, at the completion of part A, the sort subroutine lists all labels for part B specifying the internal file name for the tape unit that is to be used for each. This listing designates exactly how the merging is to proceed in part B.

Table 3–1 is a sample listing for the preceding example. The user label is UL. The first merge used three input files as calculated on the preceding page, and the remaining merges use four input files. The sixth merge starts merging B-level output files (ULB001, etc.) to produce C-level files (ULC001) which are finally merged to final output. During the run, the program types out instructions to the operator which specify when to mount and demount tapes and how to label the reels.

	INPUT FILES		OUTPUT FILE	
	UNIT	LABEL	UNIT	LABEL
MRG NO. 1	$D	ULA001		
	$E	ULA002		
	$F	ULA003	$C	ULB001
MRG NO. 2	$D	ULA004		
	$E	ULA005		
	$F	ULA006		
	$G	ULA007	$C	ULB002
MRG NO. 3	$D	ULA008		
	$E	ULA009		
	$F	ULA010		
	$G	ULA011	$C	ULB003
MRG NO. 4	$D	ULA012		
	$E	ULA013		
	$F	ULA014		
	$G	ULA015	$C	ULB004
MRG NO. 5	$D	ULA016		
	$E	ULA017		
	$F	ULA018		
	$G	ULA019	$C	ULB005
MRG NO. 6	$D	ULA020		
	$E	ULA021		
	$F	ULB001		
	$G	ULB002	$C	ULC001
MRG NO. 7	$D	ULB003		
	$E	ULB004		
	$F	ULB005		
	$G	ULC001	FINAL OUTPUT	

Table 3–1. Label Listing for Merging 21 Files Using Five Tape Units

4. CONTINUATION OF INTERRUPTED SORT PROGRAMS

4.1. INTRODUCTION

As a large volume sort is being executed, messages supply the operator with labels, cycle numbers, and reel numbers of tapes as they are produced. This information allows the user to continue an interrupted sort run without restarting at the beginning. If a tape produced during part A or part B is unreadable, the sort may be interrupted and the tape be recreated. The following paragraphs describe how to continue a sort run from an interrupted point and how to recreate tapes produced during part A or part B.

4.1.1. Continuing From Part A

If the sort program is interrupted during part A, it may be continued by first adding the 'CONTA' parameter list (see 2.2.1.22) to the set of parameters originally prepared for this sort program and then executing the original program. In the 'CONTA' list, *cycle-number* is the number assigned to the output of the first cycle after restart. *From-record-number* is the number of the first record in the input file entered into the sort after restart.

4.1.2. Continuing From Part B

If a sort program is interrupted during part B, it may be continued by first adding the 'CONTB' parameter list (see 2.2.1.23) to the set of parameters originally prepared for this sort program and then executing the original program. In the 'CONTB' list, *tape-label* is the label of the first output tape written after restart and *reel-no.* is the number of the tape.

4.1.3. Continuing From Part C

If a sort program is interrupted during part C, it may be continued provided that the RINFO$ linkage or the IPURI$ linkage (see 3.2.1.2) was executed in the original program. If the RINFO$ linkage was used, the program may be restarted by adding the 'CONTC' parameter list (see 2.2.1.24) to the parameter table originally prepared for the sort and re-executing the original program. If the IPURI$ linkage was used, the program may be restarted by submitting the restart deck and re-executing the original program.

4.1.4. Recreating a Part A Tape

A tape produced during part A of a sort can be recreated, if necessary. This can be done only with *multicycle* sorts. At the end of each cycle, as each n items are sorted, the operator is informed that the current output reel(s) contains items x through y of the input data. To reproduce the output of a given cycle, the cycle number and the record numbers x and y which comprise the output of that cycle must be supplied to the sort subroutine. The sort subroutine then starts from the beginning, bypassing all records until the first record requested is reached. When the last record requested is reached, the entire group is sorted. There is no need for own coding to keep any special counts of items deleted or added during own coding.

To initiate this rerun of a cycle, the 'REDOA' parameter list (see 2.2.1.20) must be added to the set of parameters originally prepared for this sort program and the program must be re-executed. This run terminates when the set of records has been re-sorted.

4.1.5. Recreating a Part B Tape

A tape produced during part B of a sort may also be recreated. At the end of each output reel, the program types out a label for the output. Rerun information for this output reel is written in the first block of the next output reel of this intermediate merge.

To reproduce a given output reel, the output reel containing the rerun information for the reel to be reproduced must be mounted on the output unit and its label must be given to the sort subroutine. After the input reels are mounted, the sort subroutine positions all inputs and merges them until all items which were on the reel to be reproduced have been merged. The reproduced reel *does not* contain the rerun information for the previous output reel.

To initiate this rerun of a portion of part B, the 'REDOB' list (see 2.2.1.21) must be added to the set of parameters originally prepared for this sort program and the program must be re-executed. This run terminates when the output reel has been recreated.

5. MERGE SUBROUTINE

5.1. INTRODUCTION

This section provides the programmer with all the information he needs to create a merge program using the merge subroutine portion of the UNIVAC 1106/1108 Sort/Merge package.

5.2. PARAMETER SPECIFICATION

The merge subroutine is initialized by use of a program linkage which supplies the subroutine with the address of a parameter table. This parameter table contains all the information necessary for a particular merge run. The format of the parameter table is the same as for the sort subroutine. It may be constructed by use of the R$FILE procedure (see 2.2.1) or as a series of constants. The merge subroutine parameter lists are a subset of the sort subroutine parameter lists.

5.2.1. Required Parameters

At least one of the following parameter lists is required:

'KEY' (see 2.2.1.5)
'KEYW' (see 2.2.1.6)
'COMP' (see 2.2.1.12)

These parameters specify the comparisons necessary for merge selection. When records are found to have equal keys, the records are returned to the user program in the order in which the first record from each input file was released to the merge subroutine.

5.2.2 Optional Parameters

'SEQ' (see 2.2.1.7)
'COPY' (see 2.2.1.25)
'PAD' (see 2.2.1.26)
'RSZ', 'RSZW', 'VRSZ', or 'VRSZW' (see 2.2.1.1 through 2.2.1.4)

If 'RSZ', 'RSZW', 'VRSZ', or 'VRSZW' is specified, a sequence check is performed to ensure that the input files are in sequence.

Entries in the parameter table other than these required and optional parameters are ignored by the merge subroutine.

5.3. MERGE SUBROUTINE LINKAGES

The three program linkages required to create a merge program using the merge subroutine are as follows:

- RMGOPN
  - The linkage

    label SLJ RMGOPN

initializes the merge subroutine. When this line is executed, control register 12 (A0) must contain the parameter table address.

– Exit Conditions

The merge subroutine returns control to line *label* + *1* and restores the contents of all registers which it uses.

- RMGREL

– The linkage

label SLJ RMGREL

must be executed once for each input file. It is used to release the first record from each input file to the merge subroutine in order that merge selection can be initiated. When this line is executed, control register 12 (A0) must contain the address of the first record from an input file. The record area addressed by control register 12 is released to the merge subroutine, and the data contained in this area may not be altered by the worker program until the record is returned by the merge. Control register 13 (A1) must contain an 18-bit identifier for the input file. Each input file should have a unique identifier. This identifier is returned by the merge subroutine with a selected record to inform the user program from which input file the record originated.

– Exit Conditions

The merge subroutine returns control to line *label* + *1* and restores the contents of all registers which it uses.

- RMGREQ

– The linkage

label SLJ RMGREQ

requests a record from the merge subroutine. Except for the first execution of this linkage, control register 12 (A0) must contain the address of the next input record from the file which contained the last record returned by the merge subroutine. The data in the area addressed by control register 12 may not be altered by own code until the record is returned. When all records from an input file have been released, control register 12 should contain negative 0 instead of a record address. This informs the merge that the input file, which contained the last record returned, is exhausted.

– Exit Conditions

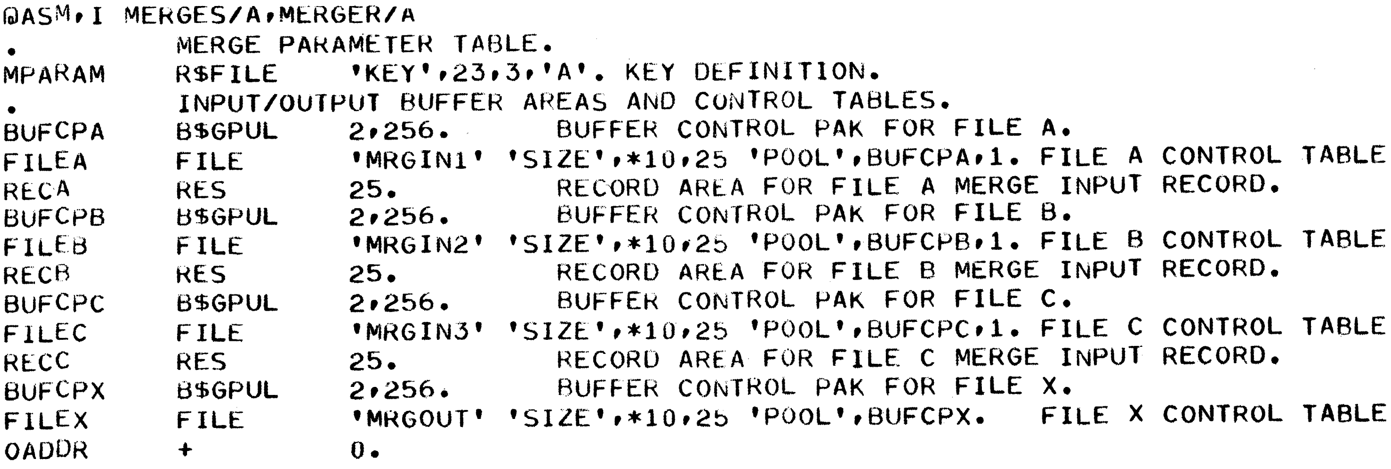
The merge subroutine returns control to line *label* + *1*. Control register 12 (A0) contains the address of the first word of the record being returned. Control register 13 (A1) contains the 18-bit file identifier for the record addressed by A0. If all records have been returned, register 12 contains negative 0. All other registers used by the merge subroutine are saved and restored.

When the address of a record is released to the merge subroutine, the left half of register 12 (A0) may contain other information. The whole word is returned unchanged when the record is returned to the user program. These positions may be useful for storing information such as record length when merging variable length records.

5.4. PREPARING A MERGE PROGRAM

For our example, a merge program is needed to merge three tape files. The data consists of 25-word records with a three-character alphanumeric key field in the least significant half of word five. The input files are written in standard conventions, ten records per block, and no label block. The output file is to be written on tape in the same format.

Figures 5–1 through 5–3 show the coding for a set of cards which will assemble and execute the program. FILE, OPEN, READ, WRITE, and CLOSE are procedures defined in the *UNIVAC 1108 Multi-Processor System Operating System EXEC 8 Programmers Reference, UP-4144* (current version). Figure 5–1 shows the parameter table that is required. It also includes the file control tables and storage that are required by own code. Figure 5–2 illustrates the usage of the merge linkages in the worker program. Figure 5–3 illustrates the control cards that are necessary to execute the program.



```
@ASM,I MERGES/A,MERGER/A
.           MERGE PARAMETER TABLE.
MPARAM      R$FILE      'KEY',23,3,'A'. KEY DEFINITION.
.           INPUT/OUTPUT BUFFER AREAS AND CONTROL TABLES.
BUFCPA      B$GPUL      2,256.      BUFFER CONTROL PAK FOR FILE A.
FILEA       FILE        'MRGIN1' 'SIZE',*10,25 'POOL',BUFCPA,1. FILE A CONTROL TABLE
RECA        RES         25.         RECORD AREA FOR FILE A MERGE INPUT RECORD.
BUFCPB      B$GPUL      2,256.      BUFFER CONTROL PAK FOR FILE B.
FILEB       FILE        'MRGIN2' 'SIZE',*10,25 'POOL',BUFCPB,1. FILE B CONTROL TABLE
RECB        RES         25.         RECORD AREA FOR FILE B MERGE INPUT RECORD.
BUFCPC      B$GPUL      2,256.      BUFFER CONTROL PAK FOR FILE C.
FILEC       FILE        'MRGIN3' 'SIZE',*10,25 'POOL',BUFCPC,1. FILE C CONTROL TABLE
RECC        RES         25.         RECORD AREA FOR FILE C MERGE INPUT RECORD.
BUFCPX      B$GPUL      2,256.      BUFFER CONTROL PAK FOR FILE X.
FILEX       FILE        'MRGOUT' 'SIZE',*10,25 'POOL',BUFCPX.   FILE X CONTROL TABLE
OADDR       +           0.
```

Figure 5–1. Parameter Table, File Control Tables, and Storage Area for a Merge Program

```
START    L,14     12,MPARAM.    MERGE PARAMETER TABLE ADDRESS TO A0.
         SLJ      RMGOPN.       OPEN MERGE SUBROUTINE.
         OPEN     'INPUT' FILEA. OPEN INPUT FILE A.
         LMJ      11,READA.     GET FIRST RECORD FROM INPUT FILE A.
         L,14     13,READA.     FILE A ID (FILE A ITEM ADVANCE ADDRESS) TO A1.
         SLJ      RMGREL.       RELEASE FIRST RECORD FROM FILE A TO MERGE.
         OPEN     'INPUT' FILEB. OPEN INPUT FILE B.
         LMJ      11,READB.     GET FIRST RECORD FROM INPUT FILE B.
         L,14     13,READB.     FILE B ID (FILE B ITEM ADVANCE ADDRESS) TO A1.
         SLJ      RMGREL.       RELEASE FIRST RECORD FROM FILE B TO MERGE.
         OPEN     'INPUT' FILEC. OPEN INPUT FILE C.
         LMJ      11,READC.     GET FIRST RECORD FROM INPUT FILE C.
         L,14     13,READC.     FILE C ID (FILE C ITEM ADVANCE ADDRESS) TO A1.
         SLJ      RMGREL.       RELEASE FIRST RECORD FROM FILE C TO MERGE.
         OPEN     'OUTPUT' FILEX. OPEN OUTPUT FILE.
         S        12,OADDR.     OUTPUT BUFFER ADDRESS TO T.S.
RETREC   SLJ      RMGREQ.       GET MERGED RECORD FROM MERGE SUBROUTINE.
         JN       12,CLOSEX.    HAVE ALL RECORDS BEEN RETURNED?
         L        14,OADDR.
         LXI,14   12,1.
         LXI,14   14,1.
         L,14     65,25.
         BT       14,0,*12.     MOVE SELECTED RECORD TO OUTPUT.
         WRITE    FILEX         GET NEXT OUTPUT ADDRESS.
         S        12,OADDR.     OUTPUT BUFFER ADDRESS TO T.S.
         LMJ      11,0,13.      GET RECORD FROM MERGE WINNER INPUT FILE.
         J        RETREC.       RELEASE RECORD (A0) FROM WINNER FILE TO MERGE.
READA    READ     FILEA  0,RECA CLOSEA. GET RECORD FROM INPUT FILE A.
         L,14     12,RECA.      RECORD ADDRESS TO A0.
         J        0,11.
READB    READ     FILEB  0,RECB CLOSEB. GET RECORD FROM INPUT FILE B.
         L,14     12,RECB.      RECORD ADDRESS TO A0.
         J        0,11.
READC    READ     FILEC  0,RECC CLOSEC. GET RECORD FROM INPUT FILE C.
         L,14     12,RECC.      RECORD ADDRESS TO A0.
         J        0,11.
CLOSEA   CLOSE    FILEA.        CLOSE INPUT FILE A.
         J        ENDSEL.       END MERGE SELECTION FOR FILE A.
CLOSEB   CLOSE    FILEB.        CLOSE INPUT FILE B.
         J        ENDSEL.       END MERGE SELECTION FOR FILE B.
CLOSEC   CLOSE    FILEC.        CLOSE INPUT FILE C.
ENDSEL   LN,14    12,0.         SENTINEL RECORD TO A0
         J        0,11.         END MERGE SELECTION FOR EXHAUSTED INPUT FILE.
CLOSEX   CLOSE    FILEX.        CLOSE OUTPUT FILE.
         ER       EXIT$.        MERGE RUN IS COMPLETED.
         END      START.
```

Figure 5–2. Coding for a Merge Program

```
@PREP
@MAP,I D,MR.
IN      MERGER.
@ASG,C  MRGIN1,C,NN.  ASSIGN INPUT FILE A.
@ASG,C  MRGIN2,C,NN.  ASSIGN INPUT FILE B.
@ASG,C  MRGIN3,C,NN.  ASSIGN INPUT FILE C.
@XQT    MR.
```

Figure 5–3. Control Cards for Execution of a Merge Program

6. OPERATING

6.1. PARAMETER CARD PROCESSING

It is possible to enter certain parameter table entries and rerun information from cards. The parameters accepted and the entry routine are described in the following paragraphs.

6.1.1. Parameters Accepted

The ability to process certain sort parameter lists from parameter cards submitted at execution time is provided to facilitate the incorporation of optional information into the parameter table.

The following parameter lists are accepted from parameter cards:

'SMRG'

'PARTA'

'PARTB'

'PARTC'

'REDOA'

'REDOB'

'CONTA'

'CONTB'

'CHECK'

'VRSZ'

'VRSZW'

These lists delete all 01, 02, and 05 entries that are already in the parameter table.

In addition to the above parameters, the parameter card routine accepts the following parameters which are not recognized by the R$FILE procedure:

'BIAS'

'DELCON'

'LIMDRM'

'LIMFST'

'PSORT'

'VOL'

6.1.1.1. 'BIAS'

Format: 'BIAS', *bias factor*

- bias factor – a decimal number of up to four digits which indicates the expected bias or degree of ordering which exists in the input data. A decimal point is implied one digit from the right of the number.

The bias of the input data can be defined as the ratio of the number of records (n) in the input file to the number of natural sequences (f) existing in the data. That is:

$$\text{bias} = \frac{n}{f}$$

For example, in the following three lists of numbers, the bias of each is:

List A	10/4 = 2.5
List B	10/10 = 1
List C	10/1 = 10

List A:		List B:	List C:
4	1	9	1
3	2	8	2
5		7	3
6		6	4
7		5	5
2	3	4	6
1	4	3	7
8		2	8
9		1	9
10		0	10

If the bias of the input data was known, sorting speed could, in most cases, be improved. Therefore, at the conclusion of each sort, the subroutine displays a message indicating the actual bias of the input data (see 6.2.1). This bias factor may then be utilized as a parameter in subsequent sorts of similar input data to improve sorting efficiency. Note that although a decimal point is printed in the bias message, it should not appear on the 'BIAS' parameter card. For example, a bias of 2.0 must be specified as 20.

This parameter *should* be entered whenever it is known. Sort assumes a value of 1.5 in the absence of this information. If the *actual* value is closer to 1.0 or greater than 2.0 (random data), the result is very poor utilization of equipment resources, often resulting in a drastic reduction of *effective* Sort scratch storage available. This can cause a Sort B5 error.

6.1.1.2. 'DELCON'

Format: 'DELCON'

This parameter card is used to suppress the printing of the end-of-sort messages (see 6.2.1) of the console. The messages will still be printed in the log and present on the listing of the run.

6.1.1.3. 'LIMDRM'

Format: 'LIMDRM', *no-of-words*

- no-of-words – a decimal number that specifies the minimum drum block size to be used.

If the ratio of main storage and drum assigned does not permit a block of at least the specified size, the run is aborted. The user must then resubmit the run assigning either a larger amount of main storage or a smaller amount of drum.

6.1.1.4. 'LIMFST'

Format: 'LIMFST', *no-of-words*

- no-of-words – a decimal number that specifies the minimum FASTRAND block size to be used.

If the ratio of main storage to FASTRAND mass storage, or if drum is assigned – the drum to FASTRAND mass storage ratio, is such that a FASTRAND block of at least the specified size cannot be obtained, the run is aborted. The user must resubmit the run assigning either a larger amount of main storage or drum, or a smaller amount of FASTRAND mass storage.

6.1.1.5. 'PSORT'

Format: 'PSORT'

This parameter card is used to delete all parameter table entries generated by the 'SMRG' list – 30, 10, 11, 12, and 13 – from the parameter table. This provision applies only to COBOL users, since COBOL sorts automatically specify the 'SMRG' list.

6.1.1.6. 'VOL'

Format: 'VOL', *no-of-records-in-thousands*

- no-of-records-in-thousands – a decimal number that specifies the volume to be sorted in thousands of records. Specifying the volume to be sorted will, in some cases, permit the sort routine to utilize facilities more efficiently.

6.1.2. Parameter Card Routine

In addition to the parameters specified above, the parameter card routine also processes the rerun information deck punched by the IPURI$ linkage (see 3.2.1.2) should restart become necessary during part C (see 4.1.3). The following general rules are applicable whenever any parameter is entered by way of parameter cards:

- Only one list may appear on any one card, but each list may use as many cards as are needed for the required information. A semicolon must be punched on cards to indicate list continuation on the succeeding card.

- The @XQT card must contain a P option to inform the sort that parameter cards are to be read.

- An @EOF~~bb~~A card should follow the last parameter card. This terminates the reading of parameter cards.

- If card input files are read during a sort program, which requires parameter cards, the user must exercise caution as to the relative placement of the data cards and the sort parameter cards with the execution of the statement opening the user card file and the statement opening the sort.

6.2. OPERATING INSTRUCTIONS

Merge and small volume sort programs require a minimum of operator intervention and only the basic operating instructions are pertinent to these programs. For automatic and nonautomatic large volume sort programs, the information given in 6.2.2 is applicable in addition to the basic operating instructions.

6.2.1. Basic Operating Instructions

The user must assign either dynamically or by way of @ASG cards the external-file-names which are available to the sort subroutine. If tape files are assigned, a blank tape must be mounted on each tape unit assigned for use by the sort subroutine. Before the sort subroutine returns control to *at-end-address* (see 3.1.4), the following messages are printed:

SORT: TIME PER RECORD = *ttt.t* msec

where: *ttt.t* is the processing time per record for this run; therefore, it does not indicate the total sort time per record when executing 'CONT A', 'CONTB', 'CONTC', or 'PARTC'.

IC = *iiiiiii*

where: *iiiiiii* is the input record count for this run; therefore, it does not specify total input record count when executing 'CONTA', 'CONTB', 'CONTC', or 'PARTC'.

OC = *oooooooo*

where: *oooooooo* is the output record count for this run; therefore, it does not specify total output record count when executing 'CONTC'.

BIAS IS *bbb.b*
or
BIAS IS SEQ

where: *bbb.b* = is the bias factor (degree of ordering in the input data file). For a file in random order, the bias factor is approximately 2. Bias increases the more the data is in sequence, but approaches 1 as the data approaches inverse sequence. If the input data is completely or almost completely in order, the bias is: SEQ. The bias factor may be utilized in subsequent sorts as described in 6.1.1.1.

M1 BLOCK SIZE: *xxxx*   MERGE POWER: *yyy*
M2 BLOCK SIZE: *xxxx*   MERGE POWER: *yyy*

where: M1 and M2 = specify the two sort modules which utilize mass storage devices. If only one mass storage file is assigned, module M1 is used but M2 is not. If two mass storage files are assigned, M1 is used for the smaller file (preferably fast drum) and M2 for the larger file (preferably FASTRAND mass storage).

xxxx = the block size used in reading and writing the assigned mass storage file.

yyy = the maximum number of strings it is possible to merge from the mass storage file using that block size.

The block size and merge power are displayed to help the user determine if the facility assignments he made were satisfactory. The sort subroutine calculates merge power and block size based on the amount of main storage, fast drum, and FASTRAND mass storage assigned. In general, assigning too small an amount of main storage or fast drum in proportion to FASTRAND mass storage causes the sort subroutine to use small blocks. This usage of short blocks results in a slow sort. To improve efficiency, assign more main storage or fast drum. Appendix A provides additional information on assigning facilities for efficient drum sorting.

6.2.2. Large Volume Sorts – Operating Instructions

A large volume sort program produces a number of intermediate output reels which must be labeled, demounted, and mounted by the operator. A number of messages instructing the operator must, therefore, be written on the console typewriter or the PAGEWRITER printer. A large volume sort is composed of three distinct phases: part A, part B, and part C. The operation of a large volume sort may be automatic or nonautomatic. An automatic sort proceeds from one phase to another under the control of the sort subroutine. A nonautomatic sort executes each phase as separate runs.

6.2.2.1. Part A of Large Volume Sort

This phase accepts input data from the user program and periodically produces a sorted output file for a part of the input data.

(1) After the label block for an output reel has been written, the following message is printed out:

cc/*uu* LABEL: REEL *n* OF *uuA*ccc

where: cc/*uu* is the channel/unit.
n is the reel number.
*uuA*ccc is the label.

with: *uu* is the user label prefix.
A denotes part A output.
ccc is the cycle number.

The operator should label the designated reel. No response is required for this message.

(2) After an end-of-reel or end-of-file sentinel block has been written on an output reel, the output reel rewinds with interlock and a tape unit swap is performed. The following message is printed out:

cc/*uu* DISMOUNT REEL *n* OF *uuA*ccc AND MOUNT BLANK

where: cc/*uu* is the channel/unit.
n is the reel number.
*uuA*ccc is the label.

The operator should demount the output reel and mount a blank tape on the designated unit. A response of Y is required for this message.

(3) After all reels of an output file are written, the following message is printed out:

CYC NO. *ccc* RECORD NOS. *yyyyyy* – *zzzzzz*

where: *ccc* is the cycle number.
yyyyyy is the lowest input record number in this output file.
zzzzzz is the highest input record number in this output file.

This message requires no response and the message should be retained in case rerun becomes necessary.

(4) After all the input data has been released to the sort and sorted into intermediate output files, the following message is printed out:

END PART A

No response is necessary. If this is an automatic sort, the sort produces a listing of the merges needed to complete this run. If this is a nonautomatic sort, the sort terminates unless the user program has requested that control be returned to it.

6.2.2.2. Part B of Large Volume Sort

This phase reads intermediate output files produced by the sort and merges them into larger intermediate output files. An automatic sort performs the number of merges needed to reduce the number of input files to one less than the number of tape files assigned. The output unit is then available to the user program after part B has been completed. A nonautomatic sort requires a separate run for each intermediate merge performed.

(1) If an automatic sort is being executed, the following message is printed out at the beginning of each merge:

START MRG NO. *ppp*

where: *ppp* is the merge number as shown on the printer listing.

No response is required for this message.

(2) The following message is printed out after the label block for each output reel has been written:

cc/*uu* LABEL: REEL *n* OF *uuxmmm*

where: *cc*/*uu* is the channel/unit.
n is the reel number.
uuxmmm is the label.

with: *uu* is the user label prefix.
x is the merge level (B-Z).
mmm is the merge number within the level.

The operator labels the designated reel and this label block is now available for use should rerun become necessary. No response is required for this message.

(3) An input reel is requested by the following message printed out:

cc/uu MOUNT REEL *n* OF *uuxxxx*

where: *cc/uu* is the channel/unit.
n is the reel number.
uuxxxx is the label.

After the input reel is mounted, this message must be answered with a Y.

(4) A label check is performed on the input reel just mounted. If the label and/or reel number read does not agree with the label and reel number requested, the error reel is rewound with interlock and the following message is printed out:

cc/uu LABEL ERROR: REEL *n* OF *xxxxxx*

where: *cc/uu* is the channel/unit.
n is the reel number read.
xxxxxx is the label read.

This message requires no response and the message to mount the requested reel is repeated. The error reel should be demounted and the correct reel mounted.

(5) When an end-of-reel or end-of-file sentinel block is detected on an input reel, the input reel is rewound with interlock and a tape unit swap is performed. The following message is printed out:

cc/uu DISMOUNT REEL *n* OF *uuxxxx*

where: *cc/uu* is the channel/unit.
n is the reel number.
uuxxxx is the label.

The operator should demount the designated reel. No response is required for this message.

(6) After an end-of-reel or end-of-file sentinel block is written on an output reel, the output reel rewinds with interlock, a tape unit swap is performed, and the following message is printed out:

cc/uu DISMOUNT REEL *n* OF *uuxmmm* & MOUNT BLANK

where: *cc/uu* is the channel/unit.
n is the reel number.
uuxmmm is the label.

The operator should DEMOUNT the output reel and mount a blank tape on the designated output unit. This message requires a response of Y.

(7) After all intermediate merges have been performed for an automatic sort or a single merge has been performed for a nonautomatic sort, the following message is printed out:

END PART B

No response is required for this message.

6.2.2.3. Part C of Large Volume Sort

This phase merges intermediate output files, which contain all of the data, into a single file that is returned to the user program one record at a time.

Part C requests input reels to be mounted, performs label checks and tells the operator to demount exhausted input reels by way of the same messages as part B (see 6.2.2.2). The disposition of the final output is under the control of the user program. During part C, the user program has the option of requesting the sort subroutine to periodically punch cards containing information for input repositioning in case restart becomes necessary. The following message is printed out each time this information is punched.

CONTC RERUN INFORMATION ccc CYCLE *nnn* CARDS

where: ccc is the deck number.
nnn is the number of cards.

No response is required for this message. The operator should remove the cards from the punch output hopper and mark them with their cycle number.

6.2.2.4. Rerun Capabilities

Two types of rerun are provided in large volume sorts:

(1) continuing an interrupted sort using the 'CONTA', 'CONTB', and 'CONTC' parameter lists; and (2) recreating an intermediate output file or reel using the 'REDOA' and 'REDOB' parameter lists. The parameter cards needed for rerun are read using the parameter card routine (see 6.1.2).

(1) 'CONTA'

If a sort is interrupted in part A, it can be continued at some later time by adding the 'CONTA' parameter list (see 2.2.1.22) to the parameter table and re-executing the original run.

This list gives the sort subroutine the input record number with which it should resume sorting. The input file is read from the beginning; however, the sort subroutine does not process the input until it receives the first record for the cycle with which it is to start. The sort then continues until normal completion of the run.

(2) 'CONTB'

If a sort is interrupted in part B, it can be continued at some later time by adding the 'CONTB' parameter list (see 2.2.1.23) to the parameter table and re-executing the original run.

The label block for each reel produced during part B contains the position of all input files at the beginning and end of the previous part B output reel. The label block can then be used to reproduce the previous part B output reel or to continue the sort from the point where the label block was written.

The label and reel number of the first output reel to be written after restart is the 'CONTB' parameter list. This is generally the reel which contains the last label block written before the sort was interrupted. This must be a part B output reel.

(a) The reel which contains the restart information is requested by the program by way of the message:

cc/uu MOUNT REEL *n* OF *uuxmmm*

where: *cc/uu* is the channel/unit.
n is the reel number.
uuxmmm is the label.

This message requires a response of Y after the requested reel is mounted.

(b) A label check is performed on the reel mounted. If the reel mounted is not the one requested, the error reel is rewound with interlock and the following message is printed out:

cc/uu LABEL ERROR: REEL *n* OF *xxxxxx*

where: *cc/uu* is the channel/unit.
n is the reel number read.
xxxxxx is the label read.

No response is required. The message requesting the reel specified in the parameter list to be mounted is repeated. The error reel is then demounted and the requested reel mounted.

(c) After the restart information has been read from the label block, the reel is rewound with interlock and the following message is printed out:

cc/uu DISMOUNT REEL *n* OF *uuxmmm*
cc/uu MOUNT BLANK

where: *cc/uu* is the channel/unit.
n is the reel number.
uuxmmm is the label of the reel which contained the restart information.

If this is 'CONTB', the same reel may be used for the output reel. If this is 'REDOB', the reel should be demounted and a blank tape should be mounted. After the output reel is mounted, this message should be answered with a Y. A label block is written and the sort is continued from the point requested. The sort then continues until normal completion of the run for 'CONTB'.

(3) 'CONTC'

Since the user program controls the disposition of the final output, it is not possible for the sort subroutine to provide for a complete restart procedure during part C. The sort provides the user with information concerning the positioning of the input files. If requested, this information is punched on cards which can be read by the parameter card routine (see 6.1.2.). However, the user must provide the operator with a procedure for positioning the output, if required.

(4) 'REDOA',

If an output reel produced during part A cannot be read, the output file which contained the reel can be recreated by adding a 'REDOA' parameter list (see 2.2.1.20) to the parameter table and re-executing the original run. The entire file must be recreated because part A output files are created from random input data. The 'REDOA' parameter list provides the sort subroutine with the cycle which is to be redone. The input file is read from the beginning; however, the sort does not process the input until it receives the first record for the cycle being recreated. The sort will not accept any more input after it has received the highest record number in the cycle to be redone. After the output file is recreated, the following message is printed out:

```
END REDO A
```

and the sort subroutine terminates the run, unless the user program has requested that control be returned to it. No response is required for this message.

(5) 'REDOB'

If an output reel produced during part B cannot be read, it can be recreated by adding a 'REDOB' parameter list (see 2.2.1.21) to the parameter table and re-executing the original run.

The information for reproducing a part B output reel is contained in the label block written subsequent to it. The label and reel number of the tape containing the redo information is supplied to the sort subroutine in the 'REDOB' parameter list. This must be a part B output reel and cannot be a reel produced by 'REDOB'.

'REDOB' requests that the reel, containing the redo information, be mounted by way of the same messages as for 'CONTB' (see step 2 of 6.2.2.4).

After the requested reel is recreated, this message is printed out:

```
END REDO B
```

and the sort terminates the run, unless the user program has requested that control be returned to it. No response is needed for this message.

6.3. DIAGNOSTIC MESSAGES

The following paragraphs provide information describing the diagnostic messages that can result from various errors occurring during the execution of sort and merge subroutines.

6.3.1. Recoverable Errors – Sort Subroutine

The sort subroutine can recover from certain errors that may occur during execution. These errors are as follows:

- Tape read errors

  The sort will attempt recovery on all tape read errors. The tape read error diagnostic message is as follows:

  cc/uu ss SORT ERROR CODE *xx* (NO OR GO)

  where: *cc/uu* is the channel/unit.
  ss is the status returned by 1108 EXEC.
  xx is the error indicator. This indicator can be any of the following:

  A2 – word count error
  A3 – checksum error
  A5 – read error detected by 1108 EXEC
  A7 – block count error

  The above error message requires a response. Recovery is attempted if the operator responds GO. The sort is terminated by way of an ER ERR$ if the operator responds NO.

- Parameter card routine errors

  If no EOF card is detected in the parameter card deck submitted to the sort, the following diagnostic message is printed out:

  MISSING EOF CARD

  The sort will continue.

6.3.2. Unrecoverable Errors – Sort Subroutine

The errors indicated below are unrecoverable and the sort subroutine will terminate by way of an ER ERR$ linkage.

- General sort errors

  Unrecoverable errors are denoted by the following diagnostic messages:

  SORT ERROR CODE *xx*

  where: *xx* is the error indicator. This indicator can be any of the following:

  (1) B1 – no record size specified in the parameter table
  (2) B2 – sort parameter table format error
  (3) B3 – no main storage allocation specified in parameter table and no R$CORE assignment
  (4) B5 – capacity of sort exceeded, single cycle
  (5) B6 – inconsistency in control parameters specified in parameter table. May specify only one of a set:
  (a) 'SMRG', 'PARTA', 'PARTB', 'PARTC',
  (b) 'REDOA', 'REDOB', 'CONTA', 'CONTB', 'CONTC'
  Set (a) must be consistent with set (b).
  (6) B7 – record size too large or 0 for variable length record
  (7) B8 – facility code error for scratch file assigned to sort
  (8) P0 – sequence error in core module. Usually caused by bad 'KEY' parameter
  (9) D4 – sequence error on drum module
  (10) F4 – sequence error in FASTRAND module
  (11) A1 – sequence error on tape

(12) D5 – read or write error on drum

(13) F5 – read or write error on FASTRAND

(14) D6 – checksum error on drum

(15) F6 – checksum error on FASTRAND

(16) D7 – block count error on drum

(17) F7 – block count error on FASTRAND

(18) D0 – logical error in drum module

(19) F0 – logical error in FASTRAND module

(20) M0 – illegal 'KEY' type (not 'A', 'B', 'D', 'M', or 'U')

(21) M1 – no 'COMP' list and/or no 'KEY' or 'KEYW' list specified

(22) M2 – starting bit or number of bits in key is invalid

(23) M3 – number of keys specified exceeds limit of 40

(24) M4 – invalid translation table (64 unique codes are required)

(25) M5 – key falls outside record

(26) M6 – key number is duplicated or missing

(27) E0 – insufficient main storage assigned to merge or specified block size is too small

(28) E1 – insufficient main storage allocated to read merge input tapes

(29) E2 – physical output tape for 'REDOB' cannot contain all data which was on reel being recreated

(30) E3 – 'REDOB' is impossible. Rerun reel specified is a part A output reel or was created by a 'REDOB' and does not contain rerun information for preceding reel

(31) E4 – insufficient number of tape files assigned for merge

(32) E5 – incorrect address in parameter table for 'CONTC' information table

(33) K9 – incorrect cycle number or sequence error in 'CONTC' input card deck.

(34) S0 – sequence error in input file to merge subroutine

- Tape errors

Unrecoverable errors are indicated by the following diagnostic message printed out:

cc/uu ss SORT ERROR CODE *xx*

where: *cc/uu* is the channel/unit.
ss is the status returned by the 1108 EXEC.
xx is the error indicator. This indicator may be any of the following:

A0 – physical end of tape; scratch tape too short
A4 – uncorrectable read error; operator responded NO
A6 – uncorrectable write error
A8 – error occurred while attempting to position a tape after a correctable read error

- Parameter card routine errors

Either of the two following diagnostic messages may be printed out on the console typewriter. The error indicated by the second diagnostic message occurs when the parameters submitted exceed the parameter table area reserved by the sort subroutine.

PARAMETER CARD INCORRECT

PARAMETER TABLE OVERFLOW

- Linkage use errors

  The following diagnostic message indicates that the sort subroutine linkage is executed at an inproper point in the program:

  xxxxxx SORT LINKAGE USED AT WRONG TIME

  where: *xxxxxx* is the linkage

  if ROPN$ – may not be used after sort has been initially opened, until sort has returned control to either the *at-end-address* or RRET$, or the 'FINAL' address specified in the parameter table.

  if RREL$ or RENCY$ – may not be used before ROPN$ has been executed or after RSORT$ has been executed.

  if RSORT$ – may be executed only once for each time the sort is opened.

  if RRET$ – may be used repeatedly after the sort has returned control to 'LPOC' address, but not after control has been returned to *at-end-address*.

APPENDIX A. ASSEMBLER, COBOL, AND DRUM SORT/MERGES

A.1. SORT/MERGE ELEMENTS AND SPACE REQUIREMENTS

The following paragraphs discuss those elements which are needed for the execution of a sort/merge and the space requirements of each element. It shows those elements which are required for a normal Assembler-coded sort, the additional elements needed when performing a COBOL sort, and the elements necessary for a merge-only program. These elements are shown in Table A-1.

ELEMENT NAME/V	DESCRIPTION	MAIN STORAGE	
		I BANK	D BANK
BPARPC/V1* PROC	Process sort parameter table entries; R$FILE	233	
BSRTPC/V1* PROC	Register definitions and procedure calls	136	
ARINFO/A* PROC	Procedure call	3	
BCONSG/N	Control segment	1053	171
PCORE/V3	Tournament processing	498	
PRATTS	Selection and preselection	79	
RDFM$/N	Drum/FASTRAND module	389	70
ATMRG/B	Tape module	1614	523
BKEYS/N	Key translation	1166	
RSTD$	Hardware file assignments		49
AINFO/A	Prepare 'CONTC' rerun information	14	
REBD$	Binary-to-decimal conversion	13	
KRFIND/V1	Search sort parameter table	28	2
KPARCD/V2	Parameter card routine	432	170
KPHRUN/V1	Punch 'CONTC' return information	58	34
TOTAL		4551	1036

*Procedures which must be present at assembly time.

Table A–1. Required Sort/Merge Elements

(1) When COBOL sorts are run, the following two elements must be included with those elements shown in Table A–1:

KSTINT/X1	COBOL-SORT INTERFACE	166	191
KTCORE/V1	Assign Core Area for COBOL Sort		15000

(2) 15,000 words of main storage are currently reserved for the COBOL sort. A user may change this to suit his own needs by reassembling the element and reserving the main storage area required:

```
7/8 ASM,S          KTCORE/V1, KTCORE, KTCORE/V1
-1, 1
TCORE$*            R$FILE 'CORE', #WORDS OF CORE
```

(3) The following element must be used in conjunction with BPARPC/V1, BSRTPC/V1, PRATTS, KRFIND, BKEYS, and REBD$ for the execution of a merge-only program.

KMERGE/X1	Merge Control	54	1060

(4) Element STERCD/V1 is present but not required in any sort run. This element contains all error codes which may occur during the running of a sort program.

(5) Depending upon the hardware features which may be required in the running of a sort program, certain sort elements may be deleted which correspond to those hardware features that will not be used.

In the situation where a user desires only a main and mass storage sort, the following elements may be deleted to provide more main storage area:

ARINFO/A   ATMRG/B   AINFO/A   KPHRUN/V1

However, when these elements are deleted, the user must externally define certain labels in his own code to satisfy labels referenced by the required sort elements. No instructions are necessary; it is used only to satisfy references to these labels contained in required elements. These labels are as follows:

RTMIN$	RTPT$	RTMT$
RTMAX$	RTST$	RLBLK$

Similar element deletion and label redefinition may be done if only main storage and tapes are necessary for a sort program. The element RDFM$/N (mass storage) element can be deleted with the following labels externally defined in the user code:

RDRMT$ RFSTR$

If only a main storage is desired, both the tape and mass storage modules may be deleted with the above mentioned labels from both modules externally defined in the user program.

The amount of main storage required by a sort/merge may be reduced further by deleting the element KPARCD/V2 provided that no parameter card processing is required.

If no parameter card processing is necessary, KPARCD/V2 may be deleted and replaced by the following lines of coding:

```
$(1)        S           01, RNEW$       .
ROPN$*      LMJ         01, RSTRG$      .
            'ROPN$Δ'                    .
            J           RBPN$           .
```

A.2. DRUM ONLY SORTS

The sort subroutine is designed to operate efficiently with different levels of storage (excluding tapes, which will not be discussed here) namely:

- Main storage
- Relatively fast drums: FH-432, FH-1782, FH-880
- Relatively slow drums: FASTRAND II

In addition, the fast drums may be used as word addressable or simulated FASTRAND drum, depending upon options exercised at systems generation time and at sort run time.

A.2.1. Utilization

For optimal utilization, both fast drum and FASTRAND drum should be assigned. To achieve this, there are two modules within the sort subroutine which are prepared to use drums. These modules are referred to as M1 and M2. Because of the options exercised at system generation time, by way of @ASG cards or by allocation of UNIVAC EXEC 8 (dependent on run time conditions), a UNIVAC 1106/1108 Sort can logically engage M1 and M2 as follows:

- If only one type of drum, assigned by way of an @ASG card, is used, M1 is used but M2 is not. The drum type may be:
  - word addressable fast drum,
  - simulated FASTRAND drum,
  - FASTRAND drum, or
  - 8414 disc (Not recommended for one-level sorts)

  The amount of drum assigned must be at least three times as great as the working main storage available; otherwise, the drum assigned is not used.

- If two types of drum, assigned by way of @ASG cards, are used, M1 is used for the smaller file and M2 for the larger file. M1 and M2 (individually) may use:
  - word addressable fast drum,
  - simulated FASTRAND drum,
  - FASTRAND drum, or
  - 8414 disc (Not recommended for M1 storage)

  However, M1 must be ≥ three times main storage and M2 must be ≥ two times M1.

The 8414 disc storage should not be used for the M1 (lower-volume) storage. The large sector size (128 words) of the 8414 disc can cause problems and inefficiencies.

A.2.2. Sort Operation Using Mass Storage Files

Paragraph A.3 contains procedures that help the user allocate his facilities when using mass storage files. To make use of these facilities, it is helpful to know how the sort operates when modules M1 and M2 are employed.

As the records are released to the sort, they are ordered into strings in main storage* and then they are written onto the next larger storage device assigned, usually fast drum. When either the drum is filled or the predicted merge limit is reached, these strings are merged simultaneously employing main storage and the resultant long string is written onto FASTRAND mass storage.** This process continues until (a) all of FASTRAND mass storage is filled, or (b) when the predicted merge limit is reached, or (c) input is exhausted. The strings on the FASTRAND mass storage are then merged simultaneously in main storage and they are then either written on tape or transferred to Last Pass Own Code as final output.

When merging n strings, n+1 input buffers are needed so that at least one block from each string can be in main storage at the same time. Three output buffers are also needed to write the merged output onto the next storage device. If the output is being sent to tape, only two buffers are needed; if the output is going to Last Pass Own Code, no buffers are needed.

The efficiency of the sort is greatly dependent upon the size of the blocks used to transfer the strings during the sorting process. Block size depends on the predicted number of strings that will be merged simultaneously and the amount of main storage available for the merging process.

The number of strings that must be merged simultaneously depends somewhat on the bias of the input data, but more so on the ratio between the amounts of main storage, fast drum, and FASTRAND mass storage assigned. Paragraph A.3 provides the formulae and tables needed by the programmer to determine the correct mix of main storage, fast drum, and FASTRAND mass storage so as to obtain an efficient assignment for a given sort.

A.3. FACILITY ASSIGNMENT FOR EFFICIENT DRUM SORTING

The most efficient sort is one in which:

- the fastest storage devices available are assigned;
- the number of passes over the data are reduced to a minimum; and
- the input and output channel time in each pass are completely overlapped.

In some instances, the strings generated are slightly smaller than the amount of main storage (C) assigned; however, the input data is generally in random order and the strings will be approximately 2C in length.

***Ideally, the strings on FASTRAND mass storage will approach the amount of drum (D) assigned in length.*

The fastest storage devices available for sorting are mass storage devices. If possible, the user should assign enough mass storage to contain the entire file, thus eliminating costly tape passes. If FASTRAND mass storage is required to accommodate the file, sufficiently large amounts of fast drum and main storage should be assigned to ensure that the block sizes which are used are large enough to achieve overlap.

A.3.1. Facility Assignment – Formulae

The following rule of thumb guidelines are provided to help the user choose the most efficient combination of facilities for his sorting requirements. Tables are also provided in A.3.2 as an alternate method of determining an optimal facility allocation. For variable length record sorting, the sort process requires that one word per record be added to carry the record size. This results in an increase in the volume (V) to be sorted. For example, a sort of 10,000 10-word records and 10,000 15-word records does not have a volume (V) of 250,000 words (10 x 10,000 plus 15 x 10,000) as would be assumed. The actual volume (V) is (11 x 10,000 plus 16 x 10,000) or 270,000 words. When sorting variable length records, the calculation of V for the following equations should be made with this in mind.

A.3.1.1. Fast Drum Sorting Only

If the data volume (V) can be contained on fast drum exclusively, assign just enough fast drum space (D) to hold the data file, plus a safety factor.*

$$D = 1.1\,(V)$$

Using the table in A.3.1.3, choose the appropriate minimum value for drum buffer size (d).

Substitute the amount of fast drum (D) and the drum buffer size (d) into the following equation to obtain the approximate amount of main storage (C) to assign.

$$D /\!\!/ C + 1 = \left| \frac{C}{d} \right|$$

A.3.1.2. Fast Drum and FASTRAND Mass Storage Sorting

If available, assign just enough FASTRAND space (F) to hold the data, plus a safety factor.*

$$F = 1.1\,(V)$$

Note the values for FASTRAND buffer size (f) and fast drum buffer size (d) shown in the table in A.3.1.3.

Substitute values for fast drum (D) and main memory (C) into the following equations until both f and d are at least as large as shown in the table in A.3.1.3.

* If variable length records are being sorted, this safety factor should be increased to 15 percent. Do *not* use more than the necessary volume plus the recommended safety factor; this would be self-defeating in most cases.

$$F /\!/ D + 1 = \left| \frac{C-2t}{f} \right|$$

where: t = 0 if tape is not required; t = 1000 if tape is required.

$$D /\!/ C + 1 = \left| \frac{C-3f}{d} \right|$$

A.3.1.3. Block Sizes

The block sizes shown in the following table are such that the effective transfer rate for the specified device will be equal to that of a UNISERVO VIII-C tape blocked at 1000 words. Assuming a 1:1 interlace, the block size required to achieve this balance is:

DEVICE	WORDS/BLOCK
FH-432 drum	125
FH-880 drum	500
FH-1732 drum	500
FASTRAND II mass storage	2000
FASTRAND III mass storage	1500

Larger block sizes may not improve the efficiency of the sort, but total systems efficiency may improve because the number of accesses to the device will be reduced.

The user must also keep in mind that a data block must be large enough to accommodate at least one data record. If the size of the record to be sorted is larger than the block size selected from the above table, use record size as block size in place of the value selected. If the records are of variable length, use the size of the largest record as block size.

NOTE: The operation a//b is the covered quotient operation, a//b = (a + b – 1)/b. The operation $|a/b|$ means the integer value of a/b.

A.3.1.4. Example

Assume 1,500,000 words of data are to be sorted and 15,000 words of main storage are available. Generous amounts of both FASTRAND II mass storage and FH-1732 drum are also available. The amount of FASTRAND II mass storage required to accommodate the file is:

$$F = 1.1\ (1{,}500{,}000)$$

$$F = 1{,}650{,}000$$

To determine an optimal assignment of FH-1732 drum, let f = 2000, as shown in A.3.1.3.

$$\frac{1{,}650{,}000}{D} + 1 = \left| \frac{15{,}000}{2000} \right|$$

$$\frac{1{,}650{,}000}{D} = 6 \qquad 275{,}000 \leq D < 330{,}000$$

To verify that an assignment of 275,000 words of drum will allow a drum buffer of sufficient size:

$$\frac{275,000}{15,000} + 1 = \left|\frac{9000}{d}\right|$$

$$19 + 1 = \left|\frac{9000}{d}\right| \qquad d = 450$$

Since d is an acceptable size for FH-1732 drum buffers, an efficient sort will be performed with F = 1,650,000 words, D = 275,000 words and C = 15,000 words.

If, in solving these equations, d should turn out to be far too small, the user should try to assign a larger amount of main storage (C). Or, if sufficient main storage is not available, try reducing fast drum. This would cause a reduction in FASTRAND buffer size (f), but an increase in drum buffer size (d).

A.3.2. Facility Assignment – Tables

The tables in the following paragraphs provide an alternate means for determining optimal facility allocation when using both fast drums and FASTRAND mass storage. The tables should be used in the following manner:

(1) Choose the appropriate table depending on the availability of FH-432, FH-880, or FH-1732 drum.

(2) Determine the volume (V) to be sorted by multiplying record size times the number of records, and add a 10 percent safety factor.

(3) Enter column 4, selecting the first number higher than V.

(4) Using this line entry, the proper assignment of facilities is as follows:

Column 1 = Main Storage

Column 3 = Fast Drum

Column 4 = FASTRAND Mass Storage

MAIN STORAGE	FASTRAND DRUM RATIO REQUIRED	MAXIMUM SIZE FH-880 or FH-1732	MAXIMUM SIZE FASTRAND II or III
Thousand Words		Thousand Words	Thousand Words
5	–	32	0
6	–	50	0
7	–	72	0
8	–	97	0
9	3	38	114
10	4	60	240
11	4	86	344
12	5	116	580
13	5	150	750
14	6	188	1128
15	6	240	1440
16	7	276	1932
17	7	327	2289
18	8	380	3040
19	8	438	3504
20	9	500	4500
22	10	636	6360
24	11	788	8668
26	12	956	11,472
28	13	1140	14,820
30	14	1340	18,760
32	15	1556	23,340
34	16	1788	28,608
36	17	2036	34,612
38	18	2300	41,400
40	19	2580	49,020
42	20	2876	57,520
44	21	3188	66,948
46	22	3516	77,352
48	23	3860	88,780
50	24	4220	101,280
52	25	4596	114,900

Not efficient to use both D and F

Table A–2. Sort Facility Allocation Chart for FASTRAND II or III Mass Storage and FH-880 or FH-1732 Drum

MAIN STORAGE	FASTRAND DRUM RATIO REQUIRED	MAXIMUM SIZE FH-432	MAXIMUM SIZE FASTRAND II or III
Thousand Words		Thousand Words	Thousand Words
5	–	179	0
6	–	263	0
7	–	363	0
8	–	480	0
9	3	198	594
10	4	298	1192
11	4	413	1652
12	5	545	2725
13	5	693	3465
14	6	857	5142
15	6	1037	6222
16	7	1232	8624
17	7	1444	10,108
18	8	1672	13,376
19	8	1916	15,328
20	9	2176	19,584
22	10	2743	27,430
24	11	3375	37,125
26	12	4070	48,840
28	13	4830	62,790
30	14	5654	79,156
32	15	6541	98,115
34	16	7493	119,888
36	17	8508	144,636
38	18	9588	172,584
40	19	10,732	203,908
42	20	11,939	238,780
44	21	13,211	277,431
46	22	14,546	320,012
48	23	15,946	366,758
50	24	17,410	417,840
52	25	18,937	473,425

} Not efficient to use both D and F

Table A–3. Sort Facility Allocation Chart for FASTRAND II or III Mass Storage and FH-432 Drum

A.3.2.1. Special Considerations

If the record size is large, the value of main storage obtained from column 1 should be increased by twice the record size. If the records are of variable length, add twice the size of the largest variable-length record. If the bias of the data is less than 1.5 (the Sort assumed value), it should be entered via a 'BIAS' parameter entry. If not known to be less, and not entered, an increase in core allotment (of up to 33%) will lessen the impact of badly biased (inverse) data. See 6.1.1.1 for 'BIAS'. Record sizes of fixed length greater than 150 words or less than 5 words put severe strain on core utilization. Variable length records with small link sizes (less than 5 words) or small fixed length records being sorted as variable length also cause adverse effects. For these cases, it is recommended that the core determined from the tables be increased from 20 to 50% if possible.

A.3.2.2. Systems Performance Improvement

It is sometimes desirable to decrease the number of accesses to a given device to improve overall systems efficiency. This can be accomplished by increasing the block size used to access the device. (Increasing block size will not, however, reduce the elapsed time of the sort.)

To decrease the number of drum accesses, allocate more main storage than is required and decrease the allocation of fast drum according to the drum/FASTRAND ratio given in column 2 of the appropriate table.

Example:

According to the table, sorting 5 million words of data requires 6,360,000 words of FASTRAND mass storage, 636,000 words of FH-1732 drum, and 22,000 words of main storage. However, if the amount of main storage is increased to 40,000 words, the number of drum accesses can be decreased by almost half. Continue to use 6,360,000 words of FASTRAND mass storage but, using the ratio of 1/19 (taken from column 2), assign 335,000 words of drum.

To decrease the number of FASTRAND accesses, allocate more main storage than is required and leave the FASTRAND mass storage and fast drum allocations unchanged.

Example:

According to the table, sorting 5 million words of data requires 6,360,000 words of FASTRAND mass storage, 636,000 words of FH-1732 drum and 22,000 words of main storage. If we increase the main storage assignment to 44,000 and leave the fast drum and FASTRAND assignments unchanged, we will have about half as many FASTRAND accesses and about half as many drum accesses.

To determine the number of accesses to a given device, use the formulae shown in A.3.1. By substituting values for C, D, and F, one obtains the values for d and f. Knowing these block sizes and the volume (V) to be sorted, one can calculate the expected number of drum (X) and FASTRAND (Y) accesses, as follows:

$$X = 2 \frac{V}{d}$$

$$Y = 2 \frac{V}{f}$$

INDEX

Comments concerning this manual may be made in the space provided below. Please fill in the requested information.

System: ____

Manual Title: ____

UP No: ____________ Revision No: ____________ Update: ____________

Name of User: ____

Address of User: ____

Comments:

CUT

FOLD

FIRST CLASS
PERMIT NO. 21
BLUE BELL, PA.

BUSINESS REPLY MAIL NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY

UNIVAC
P.O. BOX 500
BLUE BELL, PA. 19422

ATTN: SYSTEMS PUBLICATIONS DEPT.

FOLD

CUT