WANG

INTEGRATED SUPPORT SYSTEM USER MANUAL

SYSTEM 2200

INTEGRATED SUPPORT SYSTEM USER MANUAL

WANG LABORATORIES, INC.
836 NORTH STREET, TEWKSBURY, MASSACHUSETTS 01876, TEL. (617) 851-4111, TWX 710 343-6769, TELEX 94-7421

WANG LABORATORIES, INC.

836 NORTH STREET, TEWKSBURY, MASSACHUSETTS 01876, TEL (617) 851 4111, TWX 710 343 6769, TELEX 94 7421

HOW TO USE THIS MANUAL

This manual is a guide to the use of the Integrated Support System, (ISS). There are two types of software in ISS: system access software and support software.

The system access software consists of CPU diagnostic and initialization routines, and a hierarchy of software menus. This ties together all the software components of a system, including user written application programs and the support software of ISS itself. The system access software is described in Part I of this manual.

The support software of ISS can be broken down into broad groups as follows:

1. General-purpose utility programs. These are described in Part II of this manual, "The ISS Utilities."

2. Disk Sort Systems: There are two complete disk sort systems in ISS. One of these is a stand-alone system that requires operator entry of sort parameters. It is described in Part III of this manual, "The ISS Disk Sort Utility." The other is a subsystem that is called by a user-written program. It is called SORT-3, and is described in Part V of this manual, "The Programming Aids."

3. Disk File Access Systems: There are two complete disk file access systems in ISS. They are similar except that one is designed for use in a multiplexed disk environment. They are described in Part IV, "KFAM".

4. Subroutines for Standard Programming Tasks: ISS includes a group of subroutines which perform the most commonly required program tasks of a disk-based 2200 system. These are described in Part V of this manual.

TABLE OF CONTENTS

PAGE

PART I OVERVIEW OF ISS. 1

CHAPTER 1 INTRODUCTION 2

CHAPTER 2 ISS SUPPORT SOFTWARE 3
2.1 Application Support Diskettes. 3
2.2 Programming Aids Diskette. 4

CHAPTER 3 ISS SYSTEM ROUTINES. 7
3.1 Overview of IPL and START. 7
3.2 IPL. 7
3.3 START. 8
3.4 Operating Instructions - IPL 10
3.5 The Change System Specs Routine. 12
3.6 Transferring to an Application Program 13
3.7 Transferring to Another ISS Diskette 14
3.8 Copying the ISS System to a Model 2230/2260 Disk Drive 14

CHAPTER 4 SYSTEM REQUIREMENTS FOR ISS OPERATION. 16
4.1 Hardware Requirements. 16
4.2 Software Requirements for Integrating Application Program into ISS 16

PART II THE ISS UTILITIES. 17

CHAPTER 5 INTRODUCTION TO THE ISS UTILITIES. 18

CHAPTER 6 THE COPY/VERIFY AND CREATE REFERENCE FILE UTILITIES. 20
6.1 Introduction 20
6.2 Copy/Verify Operating Instructions 20
6.3 Create Reference File. 24

CHAPTER 7 THE SORT DISK CATALOG UTILITY. 29
7.1 Introduction 29
7.2 Operating Instructions 29

CHAPTER 8 THE DISK DUMP UTILITY. 32
8.1 Introduction 32
8.2 Operating Instructions 32

CHAPTER 9 THE DECOMPRESS UTILITY 35
9.1 Introduction 35
9.2 Operating Instructions 35

CHAPTER 10 THE LIST/CROSS-REFERENCE UTILITY 39
10.1 Introduction 39
10.2 Operating Instructions 40

CHAPTER 11 THE COMPRESSION UTILITY. 43
11.1 Introduction 43
11.2 Operating Instructions 44

CHAPTER 12 THE LIST UTILITY 47
12.1 Introduction 47
12.2 Operating Instructions 47

CHAPTER 13 THE RECONSTRUCT INDEX UTILITY. 50
13.1 Introduction 50
13.2 Operating Instructions 50

PART III THE ISS DISK SORT UTILITY. 52

CHAPTER 14 OVERVIEW 53
14.1 System Summary 53
14.2 File Requirements. 54

CHAPTER 15 DETERMINE WORK FILE SIZE 56
15.1 Introduction 56
15.2 Operating Instructions 56

CHAPTER 16 EXECUTING THE DISK SORT UTILITY. 59
16.1 Introduction 59
16.2 Operating Instructions 60

CHAPTER 17 ISS DISK SORT UTILITY TIMINGS. 64

PART IV KFAM 65

CHAPTER 18 OVERVIEW OF THE KFAM SYSTEMS 66
18.1 What is KFAM?. 66
18.2 The Functional Components of KFAM-3 and KFAM-4 . . 67
18.3 How to Get Started with KFAM 68
18.4 Overview of KFAM-4 71

CHAPTER 19 KFAM REQUIREMENTS AND CONVENTIONS. 75
19.1 User File. 75
19.2 Key. 77
19.3 Key File 77
19.4 Device Addresses 78

CHAPTER 20 THE KFAM SET-UP UTILITIES. 79
20.1 Overview of Initialize KFAM File (KFAM-3 and KFAM-4). 79
20.2 Initialize KFAM File Operating Instructions (KFAM-3 and KFAM-4). 85
20.3 The Key File Creation Utility (KFAM-3 and KFAM-4). 91

CHAPTER 21 THE KFAM-3 SUBROUTINES 97
21.1 Overview of KFAM-3 Subroutines 97
21.2 Build Subroutine Module (KFAM-3) 99
21.3 Calling the Subroutines. 103
21.4 OPEN 106

21.5 DELETE. 107
21.6 FINDOLD 108
21.7 FINDNEW 108
21.8 FINDNEW(HERE) 109
21.9 FINDFIRST 110
21.10 FINDLAST. 110
21.11 FINDNEXT. 111
21.12 CLOSE 111

CHAPTER 22 THE KFAM-4 SUBROUTINES. 113
22.1 Programming with the KFAM-4 Subroutines 113
22.2 Build Subroutine Module (KFAM-4). 118
22.3 Calling the KFAM-4 Subroutines. 122
22.4 OPEN. 125
22.5 DELETE. 126
22.6 FINDOLD 127
22.7 FINDNEW 128
22.8 FINDNEW(HERE) 129
22.9 FINDFIRST 130
22.10 FINDLAST. 130
22.11 FINDNEXT. 131
22.12 RELEASE 132
22.13 CLOSE 132

CHAPTER 23 THE KFAM REORGANIZE UTILITIES (KFAM-3 AND KFAM-4) 133
23.1 The Reorganize Sub-system (KFAM-3 and KFAM-4) . 133
Overview. 133
Writing the Set-Up Module 134
Utility Operation and Error Messages. 138
23.2 The Reorganize KFAM File Utility (KFAM-3 and KFAM-4) 143
Overview. 143
Operating Instructions. 144

CHAPTER 24 THE ADJUST KFAM FILES UTILITIES 151
24.1 Reallocate KFAM File Space (KFAM-3 and KFAM-4). 151
Overview. 151
Operating Instructions. 153
24.2 Disk Copy and Reorganize (KFAM-3 and KFAM-4). . 158
Overview. 158
Operating Instructions. 159

CHAPTER 25 THE PRINT KEY FILE UTILITIES. 164
25.1 Print Key File KFAM-3 164
25.2 Print Key File KFAM-4 166

CHAPTER 26 THE RECOVERY UTILITIES. 168
26.1 The Key File Recovery Utility (KFAM-3 and KFAM-4) 168
Overview. 168
Operating Instructions. 169
26.2 The Reset Access Table Utility (KFAM-4) 172

CHAPTER 27 THE KFAM CONVERSION UTILITIES 176

27.1 The KFAM-3 Conversion Utilities 176
Overview. 176
Conversion Procedure. 177
27.2 The KFAM-4 Conversion Utility 181

CHAPTER 28 GENERAL TECHNICAL INFORMATION 184
28.1 Key File Record Layouts 184
28.2 Key File Structure. 188
28.3 Key File Recovery Information 193
28.4 FINDNEW with Blocked Files under KFAM-4 194
28.5 Compatibility between KFAM-1 and KFAM-3 195

CHAPTER 29 KFAM ADVANCED PROGRAMMING TECHNIQUES. 197
29.1 Eliminating the Printer in KFAM-3 and KFAM-4. . 197
29.2 Files too Large for One Platter in KFAM-3 and KFAM-4. 199
29.3 Reusing Deleted Space with FINDNEW(HERE). . . . 201
29.4 Multiple Key Files per User File. 203
29.5 Status of the KDR in KFAM-3 203
29.6 Status of the KDR in KFAM-4 206
29.7 File Names for the KFAM Utilities 207

PART V THE PROGRAMMING AIDS 208

CHAPTER 30 OVERVIEW. 209
30.1 Summary 209
30.2 How to Load the Subroutines 209

CHAPTER 31 THE SCREEN/DISK SUBROUTINES 211
31.1 Data Entry. 211
31.2 Free Unused Sectors 213
31.3 Allocate Data File Space. 214
31.4 Search Index. 214
31.5 Open/Close Output 215
31.6 Open/Close Input. 217
31.7 Position Cursor 218
31.8 Alphanumeric Input. 218
31.9 Numeric Input 219
31.10 Date Routines 219
31.11 Operator Wait 222

CHAPTER 32 TRANSLATION TABLE SUBROUTINES 223

CHAPTER 33 SORT-3. 224
33.1 Introduction. 224
33.2 Input File Requirements 225
33.3 Writing the Set-Up Module 227
33.4 Calculating the Exact Required Work File Size . 236
33.5 Normal Operating Procedure. 238
33.6 Writing a Special Input Procedure 239
33.7 Writing a Special Output Procedure. 241
33.8 Error Messages and Recovery Procedures. 245

APPENDIX A Conditions Governing Spurious Results from the List/Cross-Reference Utility. 251

APPENDIX B Sort-3 Variable Check-Off List. 253

PART I
OVERVIEW OF ISS

CHAPTER 1
INTRODUCTION

The Integrated Support System (ISS) is a collection of software designed to make it easier to create and run a disk-based Wang 2200 system. It provides utility programs to do some of the standard tasks of the system and subroutines to do some of the standard tasks of programs. It ties together support and application software. Under ISS, a common system setup tests the CPU, and then begins the processing day by requesting standard system data. Beneath this common setup, a hierarchy of accessways links all parts of the system and makes the standard system data universally available.

The Integrated Support System (ISS) resides on four functionally organized diskettes. Each contain the daily setup module, called Initial Program Load (IPL), the master transfer module, called START, as well as a file of routines.

The IPL module is executed once at the beginning of each day. It first performs a brief, 30-45 second, CPU diagnostic. If no hardware malfunction is detected, it begins the processing day by requesting keyboard entry of the date. The date is saved in Gregorian alphanumeric format and in Julian numeric format for use by any part of the system. IPL then passes control to START.

The START modules are the key links of the system. They offer easy access to the routines on their respective diskettes, as well as transfer routes to the other START modules and to application software. They execute an intra-system CPU initialization which makes standard system data available to all software in the system. At the end of execution, each user program or ISS routine offers a route back to a START module. START is a major system-integrating factor which puts all the elements of the system at the fingertips of the operator.

ISS provides for entry of the system CPU memory size and makes this information available for use by any part of the system.

ISS is designed so that application programs may be integrated into it with the minimum constraint of those programs. In addition, all four ISS diskettes may be copied to a single rigid disk. This automatically brings a new highest-level menu into the system which preserves the system software access routes.

CHAPTER 2
ISS SUPPORT SOFTWARE

2.1 THE APPLICATION SUPPORT DISKETTES

The Application Support Diskettes are so named since they provide transfer routes to an application program. In addition to this basic function, they contain ISS Utilities, the Disk Sort Utility, KFAM-3 and KFAM-4. The ISS Utilities and the Disk Sort Utility reside on Application Support Diskette #1; the KFAM systems occupy Application Support Diskettes #2 and #3. Each Application Support Diskette provides a routine for changing certain standard system information, such as the memory size and the ISS standard loading address.

ISS Utilities

The ISS Utilities are stand-alone routines designed to perform tasks frequently required in a disk-based data processing system. They reside on Application Support Diskette #1. Their functions are summarized below:

1) COPY/VERIFY - Copies files from one disk to another and verifies the copy. Additional sectors may be added to the copied files. Copied files may be renamed, or may replace existing files on the output disk. Files to be copied may be specified directly, during operation of the utility, or indirectly by means of a COPY/VERIFY Reference File.

2) SORT DISK CATALOG - Prints a disk catalog index, sorted alphabetically by file name or numerically by starting sector address.

3) DISK DUMP - Prints the hexadecimal and character equivalents of the contents of any disk file.

4) DECOMPRESS - Copies a program file and in doing so breaks up all multi-statement lines, assigning a unique line number to each BASIC statement.

5) LIST/CROSS REFERENCE - Prints a list of a program file with each BASIC statement printed on a separate line. For each input program file, it prints three cross-reference tables: one which associates referenced line numbers with the lines which refer to them, one which associates all variables with the lines in which they appear, and one which associates all DEFFN' subroutines with the lines which refer to them.

6) COMPRESSION - Reduces the size of source program files by eliminating REM lines, extra spaces, and inessential line numbers.

7) LIST - Prints a list of a program file with each BASIC statement on a separate line.

8) RECONSTRUCT INDEX - Reconstructs a disk catalog index in the event of its accidental destruction.

9) CREATE REFERENCE FILE - Creates a reference file for use by the COPY/VERIFY utility.

KFAM

KFAM is a software system designed to produce, search, and maintain an index to the records in a disk-based data file. The index is kept as a cataloged file on disk. KFAM includes subroutines which are incorporated into user written application programs. These subroutines perform all the routine operations on the index: random access search, sequential access search, adding and deleting entries. KFAM also includes utility programs that set-up a new KFAM index, and programs which carry out a variety of occasional maintenance tasks on a file.

There are two versions of KFAM in ISS. KFAM-3 is a powerful general purpose version for use when a file is to be accessed by only one CPU. KFAM-4 is designed for use in a multiplexed disk environment in which several CPU's may wish to access a file simultaneously.

The ISS Disk Sort Utility

The ISS Disk Sort Utility is a stand-alone system designed to rapidly sort the records in a cataloged disk data file. It resides on the ISS Application Support Diskette #1. It has two components. One component is used to calculate the size of the work file needed for execution; the other performs the actual sorting. For maximum efficiency the system uses the "extended" BASIC statements described in SORT STATEMENTS (Publication #700-3559A).

2.2 PROGRAMMING AIDS DISKETTE

The Programming Aids diskette offers the ISS programmer a variety of DEFFN' subroutines which may be incorporated into application programs. These subroutines are designed to perform standard tasks frequently required within application programs. In addition to these subroutines, it also offers SORT-3, a sub-system for sorting records on a disk file. SORT-3 is loaded via a user written set-up program that provides all the operational parameters for the sort. After completing its execution, SORT-3 optionally loads a user application program. SORT-3 is a fast and highly flexible sorting utility.

The Programming Aids diskette includes IPL and a START module and, therefore, may be used to begin the processing day.

Subroutines

There are two groups of DEFFN' subroutines on the programming aids diskette. They are the Screen/Disk subroutines and Translation Tables.

The Screen/Disk subroutines perform standard tasks relating to operator to CPU and disk to CPU interaction. They include the following:

1) Search Catalog Index: This subroutine examines the Disk Catalog Index to see if a particular file has been cataloged.

2) Allocate Data File Space: This subroutine opens a data file on any selected disk, and allocates to it the available sectors between the current end and the end of the cataloged area. It checks the index to ensure the uniqueness of the file name; it allows a minimum acceptable file size to be specified.

3) Free Unused Sectors: This subroutine examines the last file in a catalog area, de-allocates those sectors between the DATASAVE DC END trailer and the end of the file, and repositions the end of file control sector. The de-allocation may be restricted by specifying that a minimum number of extra sectors be maintained in the file area.

4) Data Entry: This subroutine accepts a keyboard entry, using the KEYIN statement, and checks the entry to ascertain whether it is within a specified range and whether its length, and number of places before and after the decimal, is acceptable. It also displays a prompt and an appropriate entry mask.

5) Open/Close Output: These subroutines open for output, or close, data files containing certain, special purpose, software header and trailer records.

6) Open/Close Input: These subroutines open for input, or close, data files containing certain, special purpose, software header and trailer records.

7) Alphanumeric Input: This subroutine displays a prompt on line 1 of the CRT, and a series of prompting dashes on line 2 indicating the maximum field size to be entered. The entered alphanumeric information replaces the prompting dashes.

8) Numeric Input: This subroutine displays a prompt on line 1, and, on line 2, a series of prompting dashes indicating the maximum number of digits to be entered before and after the decimal point. The entered numeric data replace the prompting dashes.

9) Position Cursor: This subroutine moves the cursor to any point on the CRT and, optionally, erases characters to the right of the new cursor position and lines below it.

10) Date: This is a group of routines which convert and manipulate dates in Gregorian and Julian form. It includes a routine for operator entry of the date.

11) Operator Wait: This subroutine displays the message "Key RETURN(EXEC) TO RESUME" and waits on an INPUT instruction for depression of RETURN(EXEC).

The Translation Table subroutines set up a table (an alphanumeric array) for use with the BASIC statement $TRAN. Four subroutines are provided which assign the proper hex codes for the following translations:

EBCDIC	TO	ASCII
ASCII	TO	EBCDIC
2200	TO	1200
1200	TO	2200

CHAPTER 3
ISS SYSTEM ROUTINES

3.1 OVERVIEW OF IPL AND START

IPL and START together link ISS software with user software, and produce an integrated system with common information available to all system components. In accomplishing this end, IPL and START maintain a data file called "MEMORY" on each of the ISS diskettes. Though, for the most part, the operator is never directly involved in accessing or altering it, "MEMORY" is an important component in the ISS system.

"MEMORY" is a two-sector cataloged data file maintained on each of the ISS diskettes. The first sector contains the following fields:

Field	Description	Type	Bytes	Comments
Field #1	System Size	Numeric	8	(8,12,16...32)
Field #2	Gregorian Date	Alphanumeric	8	MM/DD/YY
Field #3	Julian Date	Numeric	8	YYDDD
Field #4	Application Disk Menu Select #	Numeric	8	0 if uninitialized

All the fields are set to 0 on all ISS diskettes when they are first delivered. The second sector contains information used for ISS's internal control.

3.2 IPL

IPL is used to begin the processing day. The processing day is taken to include all types of system activity other than hardware diagnosis. IPL first loads "MEMORY" and checks for a valid memory size specification (8, 12, 16, 20, 24, 28, 32). If the memory size specification is invalid, as it is when IPL is performed with a new ISS diskette, IPL prompts for operator entry of a valid memory size. The entered size is saved in "MEMORY".

As soon as a valid memory size is available to IPL, it performs two CPU diagnostic tests. These tests last approximately 30-45 seconds, depending upon memory size.

The first test is the memory test. If this test finds a problem, the message STOP MEMORY ERROR appears. The Memory Diagnostic on the 2200 Hardware Diagnostic System must then be run to pinpoint the nature and location of the problem; a Wang Service Representative should be summoned.

The syntax check is the second diagnostic test. If a malfunction is encountered in this test, a "hardware" error of the form ERR XX, is signaled which specifies the malfunctioning instruction. For this type of problem, this test constitutes a complete specification. A Wang Service Representative should be summoned, but no further diagnostic need be performed.

> NOTE:
>
> Any IPL detected CPU problem prevents the processing day from beginning.

When the diagnostics are completed without error detection, the processing day begins. "Today's date" is requested, and, once entered, displayed for verification in Gregorian and Julian formats. (Detailed specifications of these formats can be found in Section 31.10.) Upon verification, the date information is saved in "MEMORY" and the START module is loaded.

> NOTE:
>
> Since IPL writes to the data file MEMORY, ISS diskettes must not have the write protect feature invoked (that is, on the 2240 series disk drives, the protect tab must not be affixed. On the 2270 models, the protect hole must be covered). It is therefore strongly recommended that a backup copy be maintained of each ISS diskette.

IPL should not be executed more than once in a single day. In the event of a system error from which no other recovery is possible, START can be loaded in the immediate mode by keying

```
SELECT DISK XYY
LOAD DC T "START"
```

where: XYY is the ISS standard loading address

Before this course is chosen, though, the full effects of the START intra-system initialization should be considered.

3.3 START

Master Menu

START is the entry point of a hierarchical system of "menus" which provides access to all the ISS services and to application software. The START module of each of the ISS diskettes displays the master menu for its diskette. The operation of START is identical on each of the ISS diskettes; only the menu content varies. If the ISS system is copied to a single rigid disk, the START master menu offers access to the software previously stored on each diskette.

A sample master menu is given below. It specifies the special function keys to be depressed for access to each of the listed routines.

WANG COMPUTER SYSTEMS
APPLICATION SUPPORT #1.

FN KEY	PROGRAM NAME	FN KEY	PROGRAM NAME
00	ISS UTILITIES	02	CHANGE SYSTEM SPECS
01	DISK SORT	03	APPLICATIONS
		15	NOT ON THIS VOL.

Intra-System Initialization

Transfer to START can be effected from four locations: from an application disk, from IPL, from another ISS START module, or from a subsidiary menu. Regardless of which of these effects the transfer, START executes the same intra-system initialization as follows:

All variables, including COM variables, are cleared. The disk device table is cleared and file number #0 is set to the ISS standard loading address. CI and INPUT class I/O parameters are set to address 001. CO, LIST, and PRINT class I/O parameters are set to address 005. COM variables Q1, Q1$, S and S1 are established and "MEMORY" is loaded into them as follows:

Q1$	- Gregorian Date
Q1	- Julian Date
S	- Memory Size in K
S1	- Menu Selection Number for Application Disk

The master menu is displayed.

Subsidiary Menus

In most cases an exit from the master menu to one of the ISS support services results in the display of another subsidiary menu. These subsidiary menus should not be confused with the master menu; the initialization procedures of START are not duplicated within these subsidiary menus.

The subsidiary menus all offer return routes to START. In general, the stand-alone routines, upon completion of their execution, return to the menu which provided their entry-point. This menu offers return to START via Special Function Key 15.

ISS Diskette Transfer

As a mnemonic convention in ISS, Special Function Key 15 has been adopted as the standard exit route from one START module to another or from a subsidiary menu to the next higher level menu. All the master menus offer the selection "NOT ON THIS VOLUME" at Special Function key 15.

When "NOT ON THIS VOLUME" is selected, a mount message is displayed and the operator mounts the desired ISS diskette. The contents of COM variables S, Q1$, and Q1 are then written to the "MEMORY" file on the newly loaded ISS diskette. Control is passed to the START module. In this manner, the continuity of the processing day is preserved across ISS diskette substitutions. IPL is not run a second time. (Note that the variable S1 is not passed during an ISS diskette substitution.)

Transfer to Applications

On the three Application Support Diskettes the master menus offer an exit to Applications. Selection of this route initiates a test on COM variable S1. If S1 is zero, the display prompts for selection of the disk device address of the application. Specified by selection numbers 1 through 6, the available addresses are 310, 320, 330, B10, B20, B30. Once an address has been selected, it is stored in the device table opposite file number #1. The menu selection number is stored in S1. The operator is prompted to mount the application program disk at the specified address. When the operator signals that the disk is mounted, ISS loads a file named START from the address of the application. If such a file is not in the catalog index of the application disk, the mount message reappears.

The design of the ISS system suggests that application programs be accessed via an application menu module, named "START", which is similar in function and appearance to the subsidiary menus within ISS. Such an application program menu can only be created by the application programmer. However, access operations under ISS only require that a program file named "START" reside on the application disk; this program need not be a menu program.

If application programs are always run from the same disk device address, the address entry operation may be bypassed by specifying the selection number as part of the ISS system specifications. (The selection number is the number that appears on the display, opposite the device address.) If the ISS system specifications in MEMORY include a non-zero selection number, the selection of "Applications" from a master menu immediately causes the mount message to appear. The CHANGE SYSTEM SPECS. routines can be used to initialize an ISS diskette "memory" file with a selection number.

If applications are processed from one disk address, without exception, then all three application support diskettes may be initialized to the proper selection number. If there is an occasional exception, one may be initialized for routine use, and the others left with the zero that causes the address entry operation to appear.

To recapitulate from the application program viewpoint, IPL and START function together to provide application programs with the Gregorian and Julian dates, the disk address selection number, and the available memory size. These data are passed to applications in COM variables Q1$, Q1, S1 and S, respectively. Should these be cleared during application processing, a return to an ISS START module automatically restores them.

3.4 OPERATING INSTRUCTIONS - IPL

DISPLAY	INSTRUCTIONS
1. READY :	1. Key CLEAR (EXEC). SELECT DISK xyy (EXEC) where: xyy is the ISS standard loading address.

2.	READY :	2.	Mount any desired ISS diskette at the ISS standard loading address.
3.	READY :	3.	Key LOAD DC T "IPL" (EXEC).
4.	READY LOAD DC T "IPL" :	4.	Key RUN (EXEC). If this is the first time that this ISS diskette has been mounted, go to step 5. Otherwise, skip to step 6.
5.	ENTER MEMORY SIZE IN K (i.e., 8, 12, 16...32)	5.	Enter the appropriate number for the memory size available.
6.	MEMORY SIZE (IN K) = XX LOOP T	6.	Temporary display appears during memory test. The memory size in K is given at XX. The incrementing of T indicates that processing is continuing. The test is complete when T = 10.

NOTE:

If the message STOP MEMORY ERROR appears, the diagnostic has detected a malfunction in memory. The processing day cannot begin. Load and execute the memory test of the 2200 Hardware Diagnostic System. Call your Wang Service Representative and report the problem.

7.	SYNTAX TEXT	7.	Temporary display appears during Syntax check.

NOTE:

If an automatically reported error, of the form ERR XX, appears together with a BASIC statement, a syntax failure has been detected. The processing day cannot begin. Call your Wang Service Representative to report the problem.

8.	CPU VERIFY TEST PASSED	8.	Temporary display appears signifying completion of IPL diagnostics without detection of malfunctions.
9.	ENTER TODAY'S DATE AS MM/DD/YY ----------	9.	Enter the date, with slashes (/). Leading zeros may be omitted.
10.	OK (Y OR N) MM/DD/YY YYDDD	10.	The entered date is displayed in Gregorian and Julian format. If it is correct, enter Y and go to the next step. Otherwise, enter N and go to step 9.
11.		11.	The START master menu for the resident ISS diskette is displayed.

3.5 CHANGE SYSTEM SPECS

Overview

All ISS diskette master menus offer the CHANGE SYSTEM SPECS routine. This routine permits the following standard system data to be changed.

1. The CPU memory size in K.
2. The application-program disk device selection number.
3. The ISS Standard Loading Address

Operating Instructions

1.		1.	From an ISS master menu access CHANGE SYSTEM SPECS via the specified Special Function Key.
2.	ENTER THE NUMBER OF THE FIELD TO CHANGE (0=END) ?_/ 1. MEMORY SIZE = 2. APPLICATION SELECTION NUMBER =	2.	Enter 1 to change the specified system memory size. Go to step 3. Enter 2 to select a different application disk device selection number. Go to step 4. Enter 0 if the displayed specifications are acceptable. Go to step 5.
3.	ENTER SYSTEM MEMORY SIZE (8, 12, 16,...32) ?__/	3.	Enter 8, 12, 16, 20, 24, 28 or 32. Go to step 2.

4. ENTER THE NUMBER OF THE DESIRED LOADING ADDRESS OF APPLICATIONS. ADDRESS AVAILABLE 0-VARIABLE 1-310 4-B10 2-320 5-B20 3-330 6-B30	4. Enter 1-6 to choose one of the displayed addresses as the standard application disk device address. Enter 0 if you wish the operator to enter the application address each time an exit to APPLICATIONS is made from this disk's ISS START module. Go to step 2.
5. DO YOU WANT TO CHANGE THE RUNNING ADDRESS OF ISS? (Y/N) ?	5. Enter Y if you wish to change the standard address from which the ISS system is loaded. Go to Step 6. Enter N if you do not wish to change the standard address from which the ISS system is loaded. Go to step 8.
6. ENTER THE DESIRED LOADING ADDRESS OF ISS. ?---	6. Enter the desired ISS standard loading address. Go to step 7. If RE-ENTER is displayed the entered address is either invalid, or there is no disk drive in the system with that address.
7. MOUNT DISK TO BE MODIFIED AT NEW ISS ADDRESS (0=END) KEY RETURN(EXEC) TO RESUME? NEW ISS LOADING ADDRESS=XYY	7. Successively mount all ISS disks that are to be loaded from the new address, at the new address. Key (EXEC) after mounting each one. After all desired ISS disks have been mounted at the new address in this fashion, enter zero to return to ISS master menu, at the old address.
8.	8. The system returns to the ISS master menu at the address from which CHANGE SYSTEM SPECS was loaded.

3.6 TRANSFERRING TO AN APPLICATION PROGRAM

To access an application program from a master menu, follow the instructions given below.

DISPLAY	INSTRUCTIONS
1. (Application Support Diskette master menu)	1. Depress the Special Function Key stipulated in the menu for APPLICATIONS.

			If ISS system specifications have been set to bypass application disk selection, skip to step 3. Otherwise, go on to the next step. (See Section 3.5 for information about how to change ISS system specifications.)
2.	ENTER THE DISK DEVICE ADDRESS OF APPLICATION ?_/ 1. 310 4. B10 2. 320 5. B20 3. 330 6. B30	2.	Enter the selection number for the address from which the application program will be processed.
3.	MOUNT APPLICATION PROGRAM DISK IN UNIT KEY RETURN(EXEC) TO RESUME	3.	If, after mounting the disk and keying RETURN(EXEC), the mount message reappears, then the application disk does not contain a module named START. See Section 4.2 for requirements for integrating application programs into ISS.

3.7 TRANSFERRING TO ANOTHER ISS DISKETTE

To transfer to another ISS diskette from START, depress Special Function Key 15. The message MOUNT DESIRED ISS VOLUME KEY RETURN(EXEC) TO RESUME appears. After mounting the desired ISS diskette at the standard loading address, the system automatically transfers the first three fields of the ISS data file "MEMORY", and passes control to the newly mounted START module.

3.8 COPYING THE ISS SYSTEM TO A MODEL 2230/2260 DISK DRIVE

The ISS system is designed so that it may be copied to, and operate from, a single disk platter mounted on Model 2230 or 2260 disk drives. The procedure outlined below for copying the four ISS diskettes to a single rigid disk automatically activates an additional master menu. This master menu is displayed whenever the system executes the START module, and offers access to the four sets of routines previously stored on four separate diskettes.

1. Mount Application Support Diskette #1 at the ISS Standard Loading Address. Access the COPY/VERIFY utility.

2. Set the COPY/VERIFY parameters as follows:

   a) FUNCTION = COPY/VERIFY
   b) EXTRA SECTORS = 2
   c) MODE = INDIRECT PART
   d) INPUT ADDRESS = (any diskette drive)
   e) OUTPUT ADDRESS = (the rigid disk drive)

3. Successively copy each diskette to the rigid disk. The Copy/Verify reference files provided on each disk must be used to specify the files to be copied.

   The reference file names are:

   #1FDF010 for Application Support Diskette #1
   #2FDF010 for Application Support Diskette #2
   #3FDF010 for Application Support Diskette #3
   PAFDF010 for Programming Aids

> CAUTION:
>
> These reference files are only for the purpose of copying the complete ISS system to a rigid disk. Backup or other single copies of diskettes should be made using the COPY statement or the ALL mode of the COPY/VERIFY utility.

4. From any ISS diskette menu, access the CHANGE SYSTEM SPECS routine. Change the ISS Standard Loading Address to the address of the rigid disk. Mount the rigid disk containing the ISS system at the new ISS address and initialize it to the new Standard Loading Address. Leave the ISS diskette mounted, and return to the diskette master menu.

5. With the diskette master menu displayed, key SELECT DISK xyy, where xyy is the new ISS Standard Loading Address.

6. Depress Special Function Key 15 to transfer to the new rigid disk copy of ISS.

Copying ISS from a Rigid Disk to Diskette

By switching the input and output addresses in the above proceedings and using the reference file names given below, the ISS system may be copied from a rigid disk back to four diskettes. The copy/verify reference file names that must be used for copying in this direction are:

#1DFF010	Application Support #1
#2DFF010	Application Support #2
#3DFF010	Application Support #3
PADFF010	Programming Aids

CHAPTER 4
SYSTEM REQUIREMENTS FOR ISS OPERATION

4.1 HARDWARE REQUIREMENTS

1. ISS requires dual disk handling capability with at least one diskette drive.

2. ISS requires a Wang 2200C processor which is equipped with Options 2 and 5.

3. All the ISS Utilities require 8K of memory, except LIST/CROSS REFERENCE and COMPRESSION, which require 12K. The KFAM-3 stand-alone utilities require 12K; the KFAM-4 stand-alone utilities require 16K. The SORT-3 system requires 8K, except if a KFAM file is being sorted in which case 12K is required.

4. The following programs require a printer (address 215).

   DISK DUMP
   LIST/CROSS REFERENCE
   LIST PROGRAM
   KFAM-3 and KFAM-4 Stand-alone Utilities
   (With minor programming changes described in Chapter 29, the printer may be omitted for KFAM.)

4.2 SOFTWARE REQUIREMENTS FOR INTEGRATING APPLICATION PROGRAMS INTO ISS

1. The application disk catalog index must contain a program file named START.

2. If an ISS diskette has been removed from the ISS standard loading address as a result of application processing, the application program must SELECT DISK XYY, where XYY is the ISS standard loading address, and then provide for remounting an ISS diskette. It must offer a means of loading START from the ISS standard loading address.

3. The START module on the application disk should provide return to ISS diskette START modules via Special Function key 15 (or 31, if 15 must be used otherwise). Though this is not strictly required, it is a recommended system convention.

PART II
THE ISS UTILITIES

CHAPTER 5
INTRODUCTION TO THE ISS UTILITIES

The ISS Utilities are a group of programs designed to perform standard tasks which are frequently required in a disk-based data processing system. They are referred to as stand-alone routines.

Disk operations in these utilities are performed only on a cataloged portion of a disk. A catalog must be established prior to execution. All printed (hardcopy) output is via address 215.

When the system is awaiting an operator entry during execution of ISS Utilities, Special Function Key 15 may be depressed to interrupt utility execution and return to the ISS diskette's master menu. Keying HALT/STEP followed by Special Function Key 15 produces the same effect at any time during ISS utility execution.

If the message ENTER INPUT ADDRESS or ENTER OUTPUT ADDRESS appears immediately after a utility is accessed from the master menu, the default input or output disk addresses are invalid for the system. (In a multiplexed disk environment this could be caused by the multiplexed disk being hogged by another CPU.) Enter the disk address requested (it can be changed, if necessary, after the default parameters have been displayed). If the message "RE-ENTER" appears after entering a device address, the entered address is invalid, either because the device is not present in the system or the address is the same as the other default address (input and output addresses may not be the same).

In providing step by step operating instructions in this section, the following conventions have been adhered to.

The verb "enter", used in the INSTRUCTIONS column, as in the sentence "Enter 1 to list only the active items", means that the keys specified are to be depressed followed by the standard terminator key RETURN(EXEC).

In contrast to "enter" the verb "key", used in the INSTRUCTIONS column, as in the sentence "Key H to interrupt execution at the end of the current file", means that only the keys specified are to be depressed. If any terminator is required when the verb "key" is used, the terminator is explicit.

The verb "mount", as in the sentences "mount paper on printer", "mount output disk at the specified address", refers to the group of actions required to fully ready the device for its impending function in the system. If the operator has any questions about how this is done or what the general requirements are, the appropriate device manual should be consulted.

In general, it is presumed that the operator is familiar with the operation of the hardware components of the system and has access to the manuals which describe them.

CHAPTER 6
THE COPY/VERIFY AND CREATE REFERENCE FILE

6.1 INTRODUCTION

The COPY/VERIFY utility copies files from one disk to another, and verifies the copy. Files are copied up to and including the trailer record. Unused sectors are not copied. Additional sectors may be added to the copied files. Copied files may be renamed and may replace existing files on the output disk. Selected files or all files may be processed. Selected files may be specified directly, during the parameter entry phase, or indirectly, by means of a COPY/VERIFY reference file. If files are specified directly, up to 100 files may be processed. If files are specified indirectly, in a reference file, 999 files may be processed. The copy and verify operations may be executed independently, or sequentially under program control.

Copying is accomplished by read/write operations rather than COPY or MOVE statements.

The utility can only process files with "hardware" header and trailer records, i.e., program files, or cataloged data files that have a DATA SAVE DC END trailer.

The index of the output disk is checked to ensure the uniqueness of each incoming file name. Files with names that already appear in the output disk catalog index are not copied.

The CREATE REFERENCE FILE utility is used to create, edit, and list a reference file for the COPY/VERIFY utility. A reference file is a listing of the names of files to be copied, and the names to be given to the output files. If a reference file is used, it must reside on the input disk.

6.2 OPERATING INSTRUCTIONS: COPY/VERIFY

1.		1.	From ISS Utilities menu load COPY/VERIFY via the indicated Special Function Key.
2.	ARE THE PARAMETERS OK? (Y/N) 1. FUNCTION = 2. EXTRA SECTORS = 3. MODE = 4. INPUT ADDRESS = 5. OUTPUT ADDRESS =	2.	Display shows current default parameters for the COPY/VERIFY utility. If all parameters are correct, enter Y; then, if MODE is ALL, go to step 15; if mode is not ALL, go to step 10.

To change any parameters enter N, and go to the next step.

3. ENTER ITEM TO BE CHANGED. (0=END, 6=SWITCH INPUT AND OUTPUT)?

3. To change a parameter, enter its selection number, and go to the step listed below.
To exchange the input and output address, enter 6.

TO CHANGE	ENTER	GO TO STEP
FUNCTION	1	4
EXTRA SECTORS	2	5
MODE	3	6
INPUT ADDRESS	4	7
OUTPUT ADDRESS	5	8

Enter zero when all parameters are correct; then go to step 9.

4. ENTER THE NUMBER OF THE FUNCTION DESIRED?

1 - COPY
2 - VERIFY
3 - COPY AND VERIFY

4. If copy only is desired, enter 1.

If verify only is desired, enter 2.

If copy and verify are desired, enter 3.

Go to step 3.

5. ENTER THE NUMBER OF ADDITIONAL SECTORS/FILE.
(-1 = LEAVE AS IS)?

5. Enter the number of sectors, to be included in each output file, in addition to the number which are listed as USED in the Catalog Index.
If -1 is entered, all sectors listed as USED are copied, and the total size of each output file is the same as the input file. However, sectors beyond the trailer record are not copied.
If VERIFY only is being performed, this parameter is not significant.
Go to step 3.

6. ENTER THE NUMBER OF THE DESIRED MODE?

MODES AVAILABLE

6. Enter 1 to copy/verify all of the files on the input platter.
Enter 2 to copy selected files whose names are to be entered from the keyboard.

1 - ALL
2 - PART
3 - REPLACE
4 - RENAME
5 - INDIRECT PART
6 - INDIRECT REPLACE

Enter 3 to replace existing files on the output disk with selected files from the input disk.
Enter 4 to copy selected files from the input disk, and rename them on the output disk. The old names and new names are entered from the keyboard.
Enter 5 to copy/verify files specified in a reference file.
Enter 6 to replace existing files on the output disk with files from the input disk, if the files to be replaced are specified in a reference file on the input disk.
Go to step 3.

7. ENTER THE INPUT ADDRESS?

7. Enter the device address at which the input disk is to be mounted.
If the input disk and the output disk are to be mounted at a multiplexed disk drive, then the hog mode address for the input disk location should be used.
Go to step 3.

> NOTE:
>
> If the input disk is currently hogged by another CPU, its address is not acceptable to COPY/VERIFY.

8. ENTER THE OUTPUT ADDRESS?

8. Enter the device address at which the output disk is to be mounted. If the output disk is to be mounted at a multiplexed disk drive, then the hog mode address for the disk location should be used.
Go to step 3.

9. DO YOU WISH TO SAVE THESE VALUES AS THE SYSTEM DEFAULTS? (Y/N)

9. To save the currently displayed parameters as the default parameters for the COPY/VERIFY utility, enter Y. Otherwise, enter N.
If mode is ALL go to step 15.
Otherwise, go to the next step.

10.	MOUNT INPUT PLATTER KEY RETURN(EXEC) TO RESUME?	10.	Mount the input disk at the displayed input address. If MODE parameter is PART, go to step 11. REPLACE, go to step 11. RENAME, go to step 11. INDIRECT PART, go to step 13. INDIRECT REPLACE, go to step 13.
11.	ENTER THE NAME OF FILE #X (0=END)	11.	Enter the name of the Xth input file to be copied/verified. If mode is PART, repeat step 11 until all desired file names have been entered. If mode is REPLACE or RENAME, go to step 12. If all desired file names have been entered, enter 0 and go to step 14.
12.	INPUT FILE = FILE NAME ENTER THE NAME OF THE OUTPUT FILE (EXEC = SAME AS INPUT)	12.	For the specified input file enter the name of the new output file (RENAME mode), or the name of the output file to be replaced (REPLACE mode). Go to step 11. Key (EXEC) to enter the input file name as the output file name.
13.	ENTER THE NAME OF THE REFERENCE FILE.	13.	Enter the name of the COPY/VERIFY reference file to be used to specify the files to be copied/verified. NOTE: The reference file must reside on the input disk.
14.	REMOUNT ISS PLATTER IF REMOVED. KEY RETURN(EXEC) TO RESUME?	14.	If the ISS disk containing COPY/VERIFY was removed to mount the input disk, then remount the ISS disk at the ISS Standard Loading Address.
15.	MOUNT PLATTERS AT THE INDICATED ADDRESSES. KEY RETURN(EXEC) TO RESUME?	15.	Mount the input and output disks at the displayed addresses. (The ISS disk may be removed if necessary.) Key (EXEC) to resume processing.

16. VERIFYING
    COPYING   FILE NUMBER XXX
    INPUT FILE NAME = file name
    OUTPUT FILE NAME = file name

16. The processing display indicates the file currently being copied/verified. After all files have been copied/verified, the message appears "MOUNT ISS PLATTER KEY RETURN EXEC TO RESUME". If the ISS platter has been removed, remount it at the ISS Standard Loading Address. Key (EXEC) to return to ISS Utilities menu.

    If the message appears:

    "FILE - file name - CANNOT BE COPIED KEY RETURN(EXEC) TO RESUME

    then the named file cannot be copied for any of the following reasons.

    1. The file name already exists on the output platter.

    2. The file does not exist on the output platter (REPLACE mode).

    3. There are insufficient sectors in the output file or disk to accommodate the input file and the requested extra sectors.

    To continue copying the other files, key (EXEC).

6.3 CREATE REFERENCE FILE

The CREATE REFERENCE FILE utility is used to create, modify, and list a reference file for the COPY/VERIFY utility. The reference file is stored as a data file on the disk containing the files to be copied. It specifies the names of the files to be copied and the name that is to be given to each copied file on the output disk. Files are copied in the sequence specified in the reference file. A single input file name may be specified twice if different output file names are used.

The "create" function creates a new reference file. It catalogs a new file or reuses a previously cataloged, scratch program or data file. The operator enters the number of files to appear in the reference file and the utility calculates the required file size. No extra sectors are included, but the operator may enter a value that anticipates future expansion.

The "modify" function allows file references to be changed, added, and deleted from an already existent reference file. It is recommended that the operator have a printed listing of the reference file to be modified.

The "list" function prints a list of all input and output file names in a reference file. Output is at device address 215.

A reference file can accommodate 999 file references.

		1.	From ISS Utilities menu load CREATE REFERENCE FILE via the indicated Special Function Key.
2.	ENTER THE DESIRED OPTION 0 - CREATE 1 - MODIFY 2 - LIST ?-/	2.	Enter 0 to create a new reference file. Enter 1 to modify an existing reference file. Enter 2 to list the contents of a reference file.
3.	ENTER THE INPUT ADDRESS ?---	3.	Enter the device address of the the disk which contains the files to be copied.
4.	ENTER THE NAME OF THE REFERENCE FILE. ?--------	4.	If OPTION = MODIFY or LIST, enter the name of the reference file to be modified or listed. If OPTION = CREATE, enter the name for the new reference file. NOTE: If OPTION = CREATE and the entered name is the name of an already cataloged scratched file, that file space is used for the new reference file. If OPTION = CREATE go to the next step; otherwise go to step 10.
5.	ENTER THE NUMBER OF FILE NAMES TO BE ENTERED. ?--/	5.	Enter the number of input file names to be in the reference file. This value can be approximate but must not be less than the actual number needed. You may enter a value that allows for future expansion.

NOTE:

The reference file saves 14 file references per sector. Since actual file length varies in one sector increments, the actual file size may be as much as 13 file references larger than that requested.

6. MOUNT INPUT DISK
KEY RETURN(EXEC) TO RESUME.

6. If the message appears "KEY RETURN(EXEC) TO RESUME? ACTIVE FILE SPECIFIED FOR REFERENCE FILE," the file name entered is the name of a currently active file and cannot be used. Key (EXEC) and go to step 17. If the message "KEY RETURN(EXEC) TO RESUME? INSUFFICIENT SPACE FOR FILE" appears, the file is a scratched file that is too small to accommodate the requested reference file size. Go to Step 17.

7. WILL ALL OUTPUT NAMES BE THE SAME AS THE INPUT NAMES. (Y/N)
?-

7. If you do not wish to change the names of any of the copied files, enter Y. Otherwise, enter N.

8. ENTER INPUT FILE NAME.
(0 = END)

8. Enter the name of the file to be copied.

If "Y" was answered at step 7, repeat this step; otherwise go to step 9.

When the names of all the files to be copied have been included in the refernece file, enter zero to end entry and go to step 15.

9. ENTER OUTPUT FILE NAME
(EXEC = SAME AS INPUT)
?--------

9. Enter the name to be given to the copied file.
Key (EXEC) to use the input name as the output name. Go to step 8.

If the message "DUPLICATE OUTPUT FILE NAME" appears, the output file name entered already appears as an output file name in the reference file. A different output file name must be specified. If the message, "DUPLICATE INPUT FILE

	NAME KEY RETURN(EXEC) TO RESUME?" appears, the input file name already appears in the reference file. Key (EXEC) to reenter the file name; key X (EXEC) to accept the duplicate input file name.
10. MOUNT INPUT DISK KEY RETURN(EXEC) TO RESUME?	10. Mount the disk containing the reference file to be modified. If OPTION = LIST, go to step 16; otherwise go on to the next step. If the message "FILE NOT ON DISK" appears, an active file with the entered name does not exist on the mounted disk. Either mount the correct disk, or key Special Function Key 15 to return to ISS utilities menu.
11. ENTER THE FILE NUMBER TO BE MODIFIED (0 = END)	11. Enter the number of the reference file entry that is to be modified. Go to the next step. If the reference file is now correct, enter zero and go to step 15.
12. ENTER THE NUMBER OF THE DESIRED OPTION ?_ OPTIONS AVAILABLE 0 - NO CHANGE 1 - MODIFY 2 - DELETE	12. The reference file entry is displayed. For the displayed entry choose from the available options. Enter 0 to accept the displayed reference file entry. Go to step 11. Enter 1 to modify the displayed reference file entry. Go to step 13. Enter 2 to delete the displayed reference file entry. Go to step 11.
13. ENTER THE INPUT FILE NAME ?--------	13. Enter the name of the file to be copied.
14. INPUT FILE NAME = FILE NAME ENTER OUTPUT FILE NAME. (EXEC = SAME AS INPUT) ?--------	14. Enter the name to be given to the copied file. Key (EXEC) to use the input

			name as the output name. Go to Step 11. If the message "DUPLICATE OUTPUT FILE NAME" appears, the output file name entered already appears as an output file name in the reference file. A different output file name must be specified. If the message "DUPLICATE INPUT FILE NAME/KEY RETURN EXEC TO RESUME" appears, the input file name already appears in the reference file. Key (EXEC) to reenter the input file name, or key X (EXEC) to accept the duplicate input file name.
15.	DO YOU WANT TO PRINT THE REFERENCE FILE? (Y/N) ?_	15.	To obtain a printed listing of the reference file, enter Y and go to the next step. Otherwise, enter N and go to step 17.
16.	PRINTING REFERENCE FILE	16.	Temporary display. If the processing light is on but the file is not being printed, then the printer is not ready. Ready the printer. 8 1/2" by 11" paper is required.
17.	REMOUNT ISS PLATTER IF REMOVED	17.	If the ISS disk was removed, remount it at the ISS standard loading address.

CHAPTER 7
THE SORT DISK CATALOG UTILITY

7.1 INTRODUCTION

The SORT DISK CATALOG utility prints a sorted list of the contents of a disk catalog index. The list may be sorted alphabetically by file name or numerically by starting sector address; it may be output to the display or to the printer. Active files, scratched files, or both, may be included in the list. The size of the sort array limits a single list to 255 items. During processing, if the array is filled before exhausting the selected index items, a partial list is produced.

7.2 OPERATING INSTRUCTIONS

1. Access the SORT DISK CATALOG utility from ISS Utilities menu by depressing the specified Special Function key.

2.
```
ARE THE PARAMETERS OK (Y/N)
?

1.    SORT OPTION =
2.    FILE TYPE =
3.    INPUT ADDRESS =
4.    OUTPUT DEVICE =
```

2. Display shows current default parameters for the SORT DISK CATALOG utility. If all parameters are correct, enter Y and go to step 9. To change any or all parameters, enter N, and go to the next step.

3.
```
ENTER THE NUMBER OF THE ITEM TO
CHANGE.  (0=END)
?
```

3. To change a parameter enter its selection number, and go to the step listed below. Enter zero when all parameters are correct; then go to step 8.

TO CHANGE	ENTER	GO TO STEP
SORT OPTION	1	4
FILE TYPE	2	5
INPUT ADDRESS	3	6
OUTPUT DEVICE	4	7

4.
```
ENTER THE SORTING OPTION.
0 - NAME  2 - STARTING SECTOR
?
```

4. To sort the index into ascendind order on the file name, enter zero. To sort the index into ascending order on the

	starting sector addresses of the files, enter 2. Go to step 3.
5. ENTER TYPE OF FILE TO LIST. ?	5. Enter zero to include all cataloged files in the output list. Enter 1 to include only active files in the output list. Enter 2 to include only scratched files in the output list. Go to step 3.
6. ENTER THE INPUT ADDRESS ?	6. Enter the device address at which the input disk is to be mounted. If the input disk is to be mounted at a multiplexed disk drive, then the hog mode address for the input disk device should be used. Go to step 3.

> NOTE:
>
> If the input disk is currently hogged by another CPU, its address is not acceptable to SORT DISK CATALOG.

7. ENTER THE OUTPUT DEVICE OPTION 0-CRT 1-PRINTER ?	7. Enter zero to output the sorted catalog index on the CRT (address 005). Enter 1 to output the sorted catalog index on a printer (address 215). Go to step 3.
8. DO YOU WISH TO SAVE THESE VALUES AS THE SYSTEM DEFAULTS?	8. To save the currently displayed parameters as the default parameters for the SORT DISK CATALOG utility, enter Y. Otherwise enter N.
9. ENTER TITLE FOR LIST. ?	9. Enter a title for the output sorted catalog index. The title appears on each page of output, if a printer is used.
10. MOUNT DISK TO BE LISTED - UNIT device address. KEY RETURN(EXEC) TO RESUME?	10. Mount the disk to be listed at the indicated address. Key (EXEC) to resume. The ISS system disk may be removed, if necessary.

11. SORTING DISK INDEX	11. Temporary display appears while the index is being sorted.
12.	12. The sorted catalog index is displayed or printed. If it is displayed on the CRT, keying (EXEC) when the screen is full displays the next 8 index entries. After all index entries have been displayed, key (EXEC) to continue.
13. MOUNT ISS PLATTER KEY RETURN(EXEC) TO RESUME?	13. If the ISS system disk was removed during utility processing, remount it at the standard loading address. Key (EXEC) to return to ISS Utilities menu.

NOTE:

To re-list the sorted catalog index on a CRT, key Special Function Key 0. To re-list the sorted catalog index on a printer, key Special Function Key 1.

CHAPTER 8
THE DISK DUMP UTILITY

8.1 INTRODUCTION

The DISK DUMP utility prints the contents of a disk file. Three kinds of dump can be obtained.

The Vertical and Horizontal dumps print the hexadecimal and alphanumeric character equivalents of the contents of the file. They differ only in output format. In the Horizontal dump the alphanumeric values are given on the same line as the hexadecimal values. In the Vertical dump the alphanumeric characters are on one line, with the hexadecimal values given on the two lines immediately below them. Hexadecimal values below 20 cause "." to be printed in place of an alphanumeric character; values above 7F print "@".

The third kind of dump is the Data File Structure dump. It prints the contents of a data file, field by field, giving the type of field (numeric or alphanumeric), the length, and the value represented relative to the type of field.

8.2 OPERATING INSTRUCTIONS

DISPLAY	INSTRUCTIONS
	1. Access the DISK DUMP utility via the specified Special Function key from ISS Utilities menu.
2. ENTER DISK UNIT NUMBER, SEE TABLE BELOW ?_	2. Enter the selection number for the desired input disk device address.

```
1 - 310   5 - B10
2 - 320   6 - B20
3 - 330   7 - B30
4 - 350   8 - 360
```

NOTE:

As parameters are entered for this utility, they are cumulatively displayed. Prior to report processing a procedure is provided for correcting erroneous entries.

3.	MOUNT DISK IN UNIT XXX KEY RETURN(EXEC) TO RESUME?	3.	Mount the input disk in the specified unit. Key RETURN (EXEC) to resume.
4.	ENTER THE NAME OF THE FILE TO BE DUMPED ?-------	4.	Enter the alphanumeric name of the file to be dumped. If the message REENTER appears, no file with that name is listed in the catalog index.
5.	READY PRINTER KEY RETURN(EXEC) TO RESUME?	5.	Ready the Printer. The vertical dump requires 8-1/2" x 11" paper; the others require 11" x 14".
6.	DUMP PHYSICAL RECORDS WITHIN FILE (FFFF*LLLL OR ALL) ?---------	6.	To dump the entire file enter ALL. To dump a portion of the file enter FFFF*LLLL where FFFF is equal to the first sector to be dumped (expressed as the decimal value of its displacement from the start of the file) and LLLL is the last sector to be dumped. For example, to dump the first 9 sectors of a file, enter 0*8; to dump the 402nd to the 1009th, enter 401*1008. Enter no spaces in this expression.
7.	ENTER TYPE OF DUMP, SEE TABLE BELOW ?_ **** DUMPS AVAILABLE **** 1 - HORIZONTAL 2 - VERTICAL 3 - DATA FILE STRUCTURE	7.	Enter 1, 2, or 3 to select the type of dump.
8.	PARAMETERS OKAY? (Y OR N) ?_	8.	Check the displayed parameters. If they are correct, enter Y and go to the next step. If they are not, enter N and go to step 2.
9.		9.	The dump is printed. To halt execution, key H. At the end of the sector being processed, execution ceases and the processing interrupt message appears.

10. PROCESSING INTERRUPT KEY 1 - CONTINUE 2 - MODIFY 3 - REUSE 4 - END	10. At the end of the dump, or after an H has been keyed, the processing interrupt selection is offered. Key 2 to re-enter program parameters beginning at step 7. Key 3 to reenter program parameters beginning at step 2. Key 4 to return to ISS Utilities menu. Key 1 is valid only if processing was interrupted by depression of H.

CHAPTER 9
THE DECOMPRESS UTILITY

9.1 INTRODUCTION

The DECOMPRESS utility breaks up all the multistatement lines in a program so that each statement appears on a numbered line by itself. As input it accepts any cataloged program file or series of files. It outputs the decompressed version on another disk as a cataloged program file (or files).

The utility breaks up multistatement lines by assigning to each BASIC statement, after the first in a line, a line number one greater than that of the previous statement in the line.

If there are not enough line numbers available between two lines in the input program, the utility decompresses until it runs out of line numbers. A multistatement line is left at the highest numbered line in such a group. When encountered, this condition is brought to the operator's attention.

In producing the output file, the utility creates a uniform system of indentation:

a) All REM statements are indented one space from the line number.

b) All other statements, except those within a FOR...NEXT loop, are indented 5 spaces.

c) Non-REM statements that are within a FOR...NEXT loop are indented 2 spaces in addition to any indentation they would have otherwise received.

If selected files are processed, the maximum number of files is 40. Any number of files may be processed under the ALL option.

9.2 OPERATING INSTRUCTIONS

1.	1. From ISS Utilities menu load DECOMPRESS via the indicated Special Function Key.
2. ARE THE PARAMETERS OK? (Y/N) 1. OPTION, = 2. EXTRA SECTORS = 3. INPUT ADDRESS =	2. Display shows current default parameters for the DECOMPRESS utility. To change any or all parameters enter N, and go to the next step. If all parameters are correct and OPTION=

4. OUTPUT ADDRESS =

PART, go to step 8. If all parameters are correct and OPTION=ALL, go to step 11.

3. ENTER THE ITEM TO BE CHANGED. (0=END, 5=SWITCH INPUT AND OUTPUT)?

3. To change a parameter enter its selection number, and go to the step listed below. Enter zero when all parameters are correct; then go to step 8. To exchange the input and output address, enter 5.

TO CHANGE	ENTER	GO TO STEP
OPTION	1	4
EXTRA SECTORS	2	7
INPUT ADDRESS	3	5
OUTPUT ADDRESS	4	6

4. ENTER THE DESIRED OPTION (ALL OR PART)
?

4. Enter ALL to decompress all files on the input disk. Enter PART to decompress selected files only.
Go to step 3.

5. ENTER THE INPUT ADDRESS?

5. Enter the device address at which the input disk is to be mounted. If the input disk and the output disk are to be mounted at a multiplexed disk drive, then the hog mode address for the input disk should be used.
Go to step 3.

NOTE:

If the input disk is currently hogged by another CPU, its address is not acceptable to DECOMPRESS.

6. ENTER THE OUTPUT ADDRESS?

6. Enter the device address at which the output disk is to be mounted. If the output disk is to be mounted at a multiplexed disk drive, then the hog mode address for the disk location should be used.
Go to step 3.

7.	ENTER THE NUMBER OF EXTRA SECTORS FOR EACH FILE (2-50) ?-	7.	Enter the number of sectors to be included in each file in addition to those required to save the file. Go to step 3.
8.	DO YOU WISH TO SAVE THESE VALUES AS THE SYSTEM DEFAULTS? (Y/N) ?	8.	To save the currently displayed parameters as the default parameters for the DECOMPRESS utility, enter Y. Otherwise, enter N. If OPTION=PART go to the next step. Otherwise go to step 12.
9.	MOUNT INPUT PLATTER KEY RETURN(EXEC) TO RESUME?	9.	Mount the input disk at the displayed input address.
10.	ENTER THE NAME OF FILE NUMBER X (0=END)	10.	Enter the name of the Xth input file to be decompressed. Repeat step 10 until all file names have been entered, then enter zero to end entry.
11.	REMOUNT ISS PLATTER IF REMOVED. KEY RETURN(EXEC) TO RESUME?	11.	If the ISS disk containing DECOMPRESS was removed, then remount the ISS disk at its system standard address.
12.	MOUNT DISK PLATTERS AT THE INDICATED ADDRESS. KEY RETURN(EXEC) TO RESUME?	12.	Mount the input and output disks at the displayed addresses. (ISS disk may be removed if necessary.) Key (EXEC) to resume processing.
13.	DECOMPRESSING FILE NUMBER=XXX FILE NAME = file name	13.	The processing display indicates the file currently being DECOMPRESSED. After all files have been DECOMPRESSED the message "MOUNT ISS PLATTER KEY RETURN(EXEC) TO RESUME" appears. If the ISS platter has been removed, remount it at its system standard address. Key (EXEC) to return to ISS Utilities menu. If the message appears "FILE CANNOT BE PROCESSED," the file is either a protected file or there is insufficient space on the output disk. If the message "DECOMPRESSION WAS INCOMPLETE FOR FILE"

appears, there were insufficient line numbers between two lines in the program to assign a unique line number to each statement. (If desired, the program can be RENUMBERED, saved, and decompressed again to complete the decompression.)

CHAPTER 10
THE LIST/CROSS-REFERENCE UTILITY

10.1 INTRODUCTION

The LIST/CROSS REFERENCE utility consists of two components which may be executed independently or sequentially under program control.

The LIST component breaks up all the multi-statement lines of a program, printing each BASIC statement on a separate line.

For example, the statement line

```
410COM F$1,T$1,N$(64)8,F,F1,C,O:COM W4$1,Q6$64,D$1,D$(2)3:COM L,
E,E1:DIM N$8,B$(16),L$1,H$2,W1$8:GOTO 660
```

is listed as:

```
410 COM F$1,T$1,N$(64)8,F,F1,C,O
      :COM W4$1,Q6$64,D$1,D$(2)3
      :COM L,E,E1
      :DIM N$8,B$(16),L$1,H$2,W1$8
      :GOTO 660
```

The CROSS REFERENCE component assembles and prints four cross reference tables: a line number cross reference, a variable cross reference, a DEFFN' cross reference, and a GOSUB' cross reference.

In the line number cross reference table each line number referenced in the program is printed, together with the numbers of the lines that contain the references. The variable cross reference lists each variable that appears in the program, and identifies the lines in which it appears.

NOTE:

In the variable cross reference there are certain BASIC statement forms which cause a non-variable to be referenced as a variable; there are others which cause array variables to be referenced as scalar variables. The BASIC statements in which a non-variable can appear that is referenced as a variable are ADD, ADD C, AND, XOR, BOOL, INIT, $TRAN, $GIO, PLOT, DATASAVE BT, and DATALOAD BT. Array variables in the matrix statements are referenced as scalar variables. For a complete specification of the conditions for these occurrences, see Appendix A.

The DEFFN' cross reference table lists the locations of all the DEFFN' statements. The GOSUB' cross reference table lists all the GOSUB' references to DEFFN' marked subroutines.

The date, file name, and page number appear atop each page of output.

During the program inspection stage of the CROSS REFERENCE utility, an internal table is built up as variables, subroutines, and line references are encountered. Should this internal table be filled before the entire program has been inspected, the utility prints the three cross reference tables, clears the internal table, and resumes program inspection from the point at which it left off. The final result is two sets of partial cross-reference tables with each set complete for the program section inspected.

Input programs for the LIST/CROSS REFERENCE utility are assumed to be free of syntax errors.

If only the LIST component is to be executed, the LIST utility generally offers improved performance over LIST/CROSS REFERENCE. See Chapter 12 for information about the LIST utility.

All the program files on the input disk may be processed, or selected files may be processed up to a maximum of 40 selected files.

10.2 <u>OPERATING INSTRUCTIONS</u>

1.		1.	From ISS Utilities menu load LIST/CROSS REFERENCE via the indicated Special Function Key.
2.	ARE THE PARAMETERS OK? (Y/N) 1. FUNCTION = 2. INPUT ADDRESS = 3. MODE = 4. MARGIN = 5. LINE LENGTH = 6. LINES/PAGE =	2.	Display shows current default parameters for the LIST/CROSS REFERENCE utility. To change any or all parameters enter N and go to the next step. If all parameters are correct, and MODE=ALL, go to step 14. If all parameters are correct and MODE=PART, go to step 11.
3.	ENTER ITEM TO BE CHANGED. (O=END)?	3.	To change a parameter enter its selection number and go to the step listed below.

TO CHANGE	ENTER	GO TO STEP
FUNCTION	1	4
INPUT ADDRESS	2	5
MODE	3	6
MARGIN	4	7
LINE LENGTH	5	8
LINES/PAGE	6	9

Enter zero and go to step 10 after all desired changes have been made.

4. ENTER THE NUMBER OF THE FUNCTION DESIRED? 1. LIST 2. CROSS REFERENCE 3. LIST/CROSS REFERENCE	4. If only a listing is desired, enter 1. If only a cross reference is desired, enter 2. If both a listing and a cross reference are desired, enter 3. Go to step 3.
5. ENTER THE INPUT ADDRESS?	5. Enter the device address at which the input disk is to be mounted. If the input disk is to be mounted at a multiplexed disk drive, then the hog mode address may be used. Go to step 3. NOTE: If the input disk is currently hogged by another CPU, its address is not acceptable to LIST/CROSS REFERENCE.
6. ENTER THE DESIRED MODE. (ALL OR PART)?	6. Enter ALL to perform the selected function on all files on the input disk. Enter PART to perform the selected function on selected files from the input disk. Go to step 3.
7. ENTER THE NUMBER OF SPACES FOR THE MARGIN. (1-10)?	7. Enter a number 1-10 for the number of spaces to appear in the left margin of the printed output. Go to step 3.
8. ENTER THE LINE LENGTH, INCLUDING MARGIN (70-130)?	8. Enter the maximum total line length, including spaces allocated to the margin. The entry must be in the range 70-130. Go to step 3.
9. ENTER THE NUMBER OF LINES PER PAGE. (10-55)?	9. Enter the number of print lines to be output on each page. The entry must be in the range 10-55. Go to step 3.

10.	DO YOU WISH TO SAVE THESE VALUES AS THE SYSTEM DEFAULTS? (Y/N)	10.	To save the currently displayed parameters as the default parameters for the LIST/CROSS REFERENCE utility, enter Y. Otherwise, enter N. If MODE=ALL,go to step 14. Otherwise, go on to the next step.
11.	MOUNT INPUT PLATTER KEY RETURN(EXEC) TO RESUME?	11.	Mount the input disk at the displayed input address.
12.	ENTER THE NAME OF FILE #X (0=END)	12.	Enter the name of the Xth program file to be listed/ cross referenced. Enter zero when all file names have been entered.
13.	REMOUNT ISS PLATTER IF REMOVED. KEY RETURN(EXEC) TO RESUME?	13.	If the ISS disk was removed, then remount it at the ISS Standard Loading Address.
14.	INITIALIZING TABLES FOR NEXT MODULE.	14.	Temporary display. No operator response required.
15.	MOUNT INPUT PLATTER KEY RETURN(EXEC) TO RESUME?	15.	Mount the input disk at the displayed input address.
16.	PROCESSING FILE X NAME - file name ERROR RECOVERY FOR PROCESSING MODULE S.F.0 - RE-START EXECUTION AT BEGINNING OF CURRENT FILE. (S.F.1 - RE-PRINT CROSS REFERENCE TABLES) KEY "H" TO INTERRUPT EXECUTION AT THE END OF THE CURRENT FILE. KEY "P" TO INTERRUPT EXECUTION AT THE END OF THE CURRENT PAGE.	16.	This display remains throughout processing. If H is keyed, processing stops at the end of the current file. If P is keyed, processing stops at the end of the current page of output. With processing stopped, keying Special Function Key 0 causes the program to re-execute the selected functions for the current file. Keying RETURN (EXEC) causes processing to resume. If a cross reference file was selected for output, the additional message "S.F. 1- RE-PRINT CROSS REFERENCE FILE" appears. If special function key 1 is depressed with processing halted, the cross reference tables for the current file are printed a second time.

CHAPTER 11
THE COMPRESSION UTILITY

11.1 INTRODUCTION

The COMPRESSION utility reduces the amount of memory occupied by an application program. In addition to permitting the execution of more powerful programs, COMPRESSION increases the speed of program execution, and reduces storage requirements.

The COMPRESSION utility does three things to an input program file:

a) It eliminates all REM statements, except those in the first statement line.

b) It eliminates all space characters not enclosed by quotation marks.

c) It eliminates as many unnecessary line numbers as possible by assigning to each line number the maximum number of BASIC statements consistent with program operation.

CAUTION:

A program to be compressed cannot contain branches to statement lines beginning with a REM statement, since all such lines (except the first line in the program) are deleted.

To preserve EDIT mode capability, the maximum compressed line length may be restricted to 180 bytes. Though this is less efficient than compressing to the absolute maximum of 256 bytes per line, it is recommended if maximum compression is not required. It is impossible to use the EDIT mode on a statement line containing 193 bytes or more.

The compression utility works in two stages. In the first stage the input program is examined, and a table is built of all line numbers referred to by statements in the program. This table is called the "protect table", since, if the program is to execute properly, these referenced line numbers must be preserved.

A convention is observed that allows the programmer to explicitly exempt any statement line from being compressed into another line. A blank REM statement appearing within an input program causes the next non-REM line to be protected. Blank REM statements which immediately surround a single line therefore, have the effect of exempting that line from compression. (A blank REM statement is REM followed immediately by RETURN(EXEC).) If a compressed program is compressed a second time, lines previously protected by blank REM's are no longer protected, since the protecting REM's have been eliminated.

The first statement line in a program is unaltered, regardless of its content.

During the utility's second stage, called "compression", the compressed version of the program is produced and written to the output disk.

The output disk:

a) Must have a catalog established on it.

b) Must not have a file with the same name as the input program file.

c) Must have sufficient space to store the input program file in its pre-compressed state.

The utility compresses selected files or all files from the input disk. However, the maximum number of selected files for a single execution is 40. The input disk files must be cataloged.

11.2 OPERATING INSTRUCTIONS

1.

1. From ISS Utilities menu load COMPRESSION via the indicated Special Function Key.

2. ARE THE PARAMETERS OK? (Y/N)

```
1.   MODE =
2.   OPTION =
3.   INPUT ADDRESS =
4.   OUTPUT ADDRESS =
```

2. Display shows current default parameters for the COMPRESSION utility. If all parameters are correct, enter Y; then, if MODE=ALL,go to step 12 or if MODE=PART, go to step 9. To change any or all parameters enter N, and go to the next step.

3. ENTER ITEM TO BE CHANGED. (0=END, 5=SWITCH INPUT AND OUTPUT)?

3. To change a parameter enter its selection number and go to the step listed below. To exchange the input and output address enter 5. Enter zero when all parameters are correct; then go to step 8.

TO CHANGE	ENTER	GO TO STEP
MODE	1	4
OPTION	2	5
INPUT ADDRESS	3	6
OUTPUT ADDRESS	4	7

4. ENTER THE DESIRED MODE. (ALL OR PART). ?	4. Enter ALL to compress all program files on the input disk. Enter PART to compress selected files on the input disk. Go to step 3.
5. ENTER THE DESIRED COMPRESSION OPTION. (1-180 or 2-256). ?	5. Enter 1 or 2 to select a maximum compressed line length of 180 or 256 bytes. Selection 1 restricts the maximum compressed line length to 180 bytes, thereby allowing use of the EDIT function on the compressed program. Selection 2 sets the maximum to 256 bytes. Statement lines of 193 bytes or more cannot be EDITed. Go to step 3.
6. ENTER THE INPUT ADDRESS?	6. Enter the device address at which the input disk is to be mounted. If the input disk and the output disk are to be mounted at a multiplexed disk drive, then the hog mode address for the input disk should be used. Go to step 3.

NOTE:

If the input disk is currently hogged by another CPU, its address is not acceptable to COMPRESSION.

7. ENTER THE OUTPUT ADDRESS?	7. Enter the device address at which the output disk is to be mounted. If the output disk is to be mounted at a multiplexed disk drive, then the hog mode address for the disk device should be used. Go to step 3.
8. DO YOU WISH TO SAVE THESE VALUES AS THE SYSTEM DEFAULTS? (Y/N)	8. To save the currently displayed parameters as the default parameters for the COMPRESSION utility, enter Y. Otherwise, enter N.

9. MOUNT INPUT PLATTER KEY RETURN(EXEC) TO RESUME?	9. Mount the input disk at the displayed input address.
10. ENTER THE NAME OF FILE NUMBER X (0=END)	10. Enter the name of the Xth input file to be compressed. Repeat this step until all desired file names have been entered. Enter zero to end entry.
11. REMOUNT ISS PLATTER IF REMOVED. KEY RETURN(EXEC) TO RESUME?	11. If the ISS disk containing COMPRESSION was removed to mount the input disk, then remount the ISS disk at the ISS Standard Loading Address.
12. MOUNT DISK PLATTERS. KEY RETURN(EXEC) TO RESUME?	12. Mount the input and output disks at the displayed addresses. (ISS disk may be removed if necessary.) Key (EXEC) to resume processing.
13. OPERATION FOR FILE X FILE NAME = file name BLOCKS READ = TOTAL BLOCKS =	13. The processing display shows the file currently being processed, its total size, and the operation currently being performed on it. If the message appears: "FILE CANNOT BE PROCESSED", either the file is a protected file or there is insufficient space on the output disk.
14. MOUNT ISS PLATTER KEY RETURN(EXEC) TO RESUME?	14. Processing is complete. Mount the ISS diskette at the ISS standard loading address. Key (EXEC) to return to ISS Utilities menu.

CHAPTER 12
THE LIST UTILITY

12.1 INTRODUCTION

The LIST utility lists a program, with each BASIC statement printed on a separate line.

For example, the statement line

```
410COM F$1,T$1,N$(64)8,F,F1,C,O:COM W4$1,Q6$64,D$1,D$(2)3:COM L,
E,E1:DIM N$8,B$(16),L$1,H$2,W1$8:GOTO 660
```

is listed as:

```
410COM F$1,T$1,N$(64)8,F,F1,C,O
  :COM W4$1,Q6$64,D$1,D$(2)3
  :COM L,E,E1
  :DIM N$8,B$(16),L$1,H$2,W1$8
  :GOTO 660
```

All the program files on a disk may be processed, or up to 40 selected files, in a single execution of this utility.

12.2 OPERATING INSTRUCTIONS

1.		1.	From ISS Utilities menu load LISTING PROGRAM via the indicated Special Function Key.
2.	ARE THE PARAMETERS OK? (Y/N) 1. MODE = 2. LINE LENGTH = 3. INPUT ADDRESS = 4. LINES/PAGE =	2.	Display shows current default parameters for the LISTING utility. To change any or all parameters enter N and go to the next step. If all parameters are correct and MODE=PART,go to step 9. If all parameters are correct and MODE=ALL,go to step 12.
3.	ENTER THE NUMBER OF THE ITEM TO BE CHANGED. (0=END)	3.	To change a parameter enter its selection number and go to the step listed below. Enter zero when all parameters are correct and go to step 8.

TO CHANGE	ENTER	GO TO STEP
MODE	1	4
LINE LENGTH	2	5
INPUT ADDRESS	3	6
LINES/PAGE	4	7

4. ENTER THE DESIRED MODE. (ALL OR PART).

4. Enter ALL to list all programs on the input disk. Enter PART to list selected programs on the input disk.
Go to step 3.

5. ENTER THE LINE LENGTH (64-130)?

5. Enter the desired maximum line length of the printed output. The entry must be in the range 64-130.
Go to step 3.

6. ENTER THE INPUT ADDRESS?

6. Enter the device address at which the input disk is to be mounted.

NOTE:

If a disk is currently hogged by another CPU, its address is not acceptable to the LISTING PROGRAM.

Go to step 3.

7. ENTER THE NUMBER OF LINES PER PAGE (30-55)
?

7. Enter the maximum number of lines per page. The entry must be in the range 30-55.
Go to step 3.

8. DO YOU WISH TO SAVE THESE VALUES AS THE SYSTEM DEFAULTS? (Y/N)

8. To save the currently displayed parameters as the default parameters for the LISTING utility, enter Y. Otherwise, enter N. If MODE=ALL,go to step 12.
Otherwise, go on to the next step.

9. MOUNT INPUT PLATTER
KEY RETURN(EXEC) TO RESUME?

9. Mount the input disk at the displayed input address.

10. ENTER THE NAME OF FILE NUMBER X (0=END)

10. Enter the name of the Xth input file to be processed. Repeat this step until all

file names have been entered. Then, enter 0 to end.

11. REMOUNT ISS PLATTER IF REMOVED. KEY RETURN(EXEC) TO RESUME?	11. If the ISS disk has been removed, remount it at the ISS standard loading address.
12. MOUNT INPUT DISK KEY RETURN(EXEC) TO RESUME?	12. Mount the disk containing the files to be listed. The ISS disk may be removed if necessary. Ready the printer.
13. LISTING FILE NUMBER X CURRENT BLOCK = FILE NAME = file name TOTAL BLOCKS =	13. The processing display shows the file name, the current block being listed, the total number of blocks in the file, and the sequence number of the file currently being listed.
	NOTE: The LISTING PROGRAM executes internal processing, then prints in bursts. Intermittent inactivity is normal.
14. MOUNT ISS PLATTER KEY RETURN(EXEC) TO RESUME.	14. Remount the ISS disk at the ISS Standard Loading Address. Key (EXEC) to return to ISS Utilities menu.

CHAPTER 13
THE RECONSTRUCT INDEX UTILITY

13.1 INTRODUCTION

The RECONSTRUCT INDEX utility is an aid to the recovery of disk files in the event of accidental destruction of the disk catalog index. The utility searches the disk, looking for file control sectors established during catalog operations. Based on the control sectors found, it attempts to reconstruct a catalog index for the files on the disk.

CAUTION:

Before executing this utility a backup copy of the disk should be made.

The utility constructs names for all data files and for duplicate program file names. The constructed names have the following form:

/*XXXX*/

where: XXXX is a four-digit number. Numbers are assigned consecutively to files that require constructed names.

13.2 OPERATING INSTRUCTIONS

1.	1. From ISS Utilities menu access the RECONSTRUCT INDEX utility via the indicated special function key.
2. ENTER THE DISK ADDRESS ?	2. Enter the device address of the disk whose catalog index is to be reconstructed.
3. ENTER THE HIGHEST SECTOR ADDRESS OF THE DISK.	3. Enter the highest sector address at which files are known to exist. If this information is not available, enter the highest sector address of the disk.
4. ENTER THE NUMBER OF INDEX SECTORS (0=UNKNOWN)	4. Enter the number of sectors in the original index and go to step 8. Enter zero if

			this is unknown, and go to the next step.
5.	MOUNT INPUT DISK KEY RETURN(EXEC) TO RESUME?	5.	Mount the disk whose index is to be reconstructed. If number of index sectors is unknown, go to the next step. Otherwise go to step 9.
6.	FINDING INDEX SECTORS	6.	Temporary display; no action required.
7.	REMOUNT ISS PLATTER IF REMOVED. KEY RETURN(EXEC) TO RESUME	7.	Remount the ISS disk if it has been removed.
8.	MOUNT INPUT DISK KEY RETURN(EXEC) TO RESUME?	8.	Mount the disk whose index is to be reconstructed.
9.	RECONSTRUCTING DISK INDEX	9.	Processing display.
10.	REMOUNT ISS PLATTER KEY RETURN(EXEC) TO RESUME	10.	Execution is complete. Remount the ISS disk at the ISS Standard Loading Address. Key (EXEC) to return to ISS Utilities menu.

PART III
THE ISS DISK SORT UTILITY

CHAPTER 14
OVERVIEW

14.1 SYSTEM SUMMARY

The ISS DISK SORT UTILITY is a stand-alone system designed to rapidly sort the records in a cataloged disk data file. An output file is used to receive the records written in their new sequence. The new sorted sequence is determined by the values of designated key fields in each record. Collectively, these key fields are referred to as the sort key.

The records to be sorted must have identical formats, that is, the order, length, type and number of fields in each record must be the same. Additional file requirements are given in Section 14.2.

The sort key can contain up to ten fields. Its total length must not exceed 64 bytes not counting the control bytes. Sort order can be specified for each field of the sort key, either ascending or descending for each field. Numeric and alphanumeric fields may be included in the sort key.

In addition to input and output files, the utility requires a work file. A stand-alone routine, DETERMINE WORK FILE SIZE, is provided to calculate the number of sectors which must be allocated to this file.

Though there is no limit to the number of records which can be sorted, the input, output, and work file are each limited to the capacity of a single disk.

Part of a file may be sorted by specifying starting and ending record numbers for the sort.

The ISS DISK SORT utility uses the BASIC statements described in SORT STATEMENTS (Publication #700-3559A).

Depending upon the size of the records to be sorted and the available disk space, either a full-record or a key sort is performed. The choice of which type of sort to perform is made by the program. In general, with short records, a full-record sort is faster than a key sort. In a full-record sort the input record is reformatted so that the record can be efficiently moved. When the sort and merge operations take place, the reformatted input record is used. After all records have been sorted, on the final merge pass, the records are restored to the input format before being written to the output file.

In a key sort, only the sort key is extracted from each input record and carried, with a pointer to the input record, as the sort record. The sort records are sorted into sorted strings which are merged until all records are in the proper sequence. At the end of program execution, the sort records are read, the appropriate input records are found, and the output file is created. This method is generally used with long records, or when the work file space on disk is inadequate to hold the entire input file.

The Disk Sort Utility runs in three or four phases, depending on whether it is performing a full-record sort or a key sort:

1. Accept input specifications, generate code.
2. Perform internal sort (pass 1).
3. Perform merge (if full-record sort, write output file) (pass 2).
4. (If key sort, write output file) (pass 3).

14.2 FILE REQUIREMENTS

The Input File

The input file must conform to the following specifications:

1) It must be a cataloged file with all records written with DATA SAVE or DATA SAVE DA. The DATASAVE DC END trailer must be present.

2) It must have all records in the same format (no special header or trailer records, no variable length records).

3) It may have either blocked or unblocked records; however,

   a. If unblocked, it must have not more than 55 fields per record.

   b. If blocked, it must be written in array form, for example

```
DIM A$(4)20, B(4), C$(4)2
DATASAVE DC A$(), B(), C$()
```

      i. It must not have more than 38 fields per record.

      ii. It must not have more than 255 records per block.

      iii. It must have all blocks filled (unused records in the last block must be filled with padding records that will sort high if used in an ascending sort, or sort low, if used in a descending sort).

4) It must have all records on a single disk.

5) A record or block of records may occupy more than one sector.

The Output File

The output file is a cataloged file. If the file is not established prior to execution, the program can establish the output file. It allocates to the file the exact amount of space needed. If a need for additional space in the output file is anticipated, the file must be established prior to execution.

If the input file is blocked, the blocking of the output file is identical to the input file.

The Work File

The work file must be a cataloged file. It must be established on one of the available disks prior to execution. In most cases the length of the work file required does not exceed the length of the file being sorted. The ISS DISK SORT menu offers a utility, DETERMINE WORK FILE SIZE, which calculates the exact amount of space needed for the file and indicates whether a key sort or a full-record sort will be performed. (If the work file size is inadequate at the time of execution, the operator is informed of the fact and actual sorting does not begin.)

CHAPTER 15
DETERMINE WORK FILE SIZE

15.1 INTRODUCTION

The DETERMINE WORK FILE SIZE utility is used to calculate the size of the work file needed by the sort utility. It also determines whether a key sort or a full-record sort will be performed. If it indicates a full-record sort, then the full-record sort is faster under the given conditions than a key sort. However, a full-record sort requires a larger work file.

If the required work file size for a full-record sort is larger than can be provided, it may still be possible to execute a key sort with the available work file space. Instructions are provided for calculating the key sort work file size under these circumstances. If, during execution of the sort utility a full-record sort is indicated, and the work file size is inadequate to perform it, the utility automatically evaluates the work file needed for a key sort, and executes it, if sufficient space is available.

To use the DETERMINE WORK FILE SIZE utility an input file containing at least one record, or block of records, is required. The format of this record must be the exact format of the actual records to be sorted. If the actual input file exists, it may be used; otherwise a dummy file can be created by executing the following 5 line program.

```
10 DIM (sizes and dimensions of fields)
20 DATASAVE DC OPEN, platter, disk sectors, "name"
30 DATASAVE DC (write record or block of records)
40 DATASAVE DC END
50 DATASAVE DC CLOSE
```

The content of the records is irrelevant. Only the format is used in the calculations.

A minimal work file of 25 sectors is also needed for the calculations. A statement such as DATASAVE DC OPEN R 25, "WORK" can be executed to establish this file.

15.2 OPERATING INSTRUCTIONS

DISPLAY	INSTRUCTIONS
1.	1. From ISS DISK SORT menu, access the DETERMINE WORK FILE SIZE UTILITY via the appropropriate Special Function Key.

2.		2.	Mount the disks containing the input and work files.
3.	INPUT FILE NAMES _ _ _ _ _ _ _ _ _	3.	Enter the name of the input file. If a dummy file is used instead, enter its name.
4.	INPUT FILE DEVICE ADDRESS	4.	Enter the device address of the input (or dummy) file. Valid entries are 310, 320, B20, B10, 350.
5.	RECORDS PER BLOCK	5.	Enter the number of records per block. Enter 1 if the records are not blocked.
6.	STARTING RECORD # TO BE SORTED?	6.	Enter 1.
7.	NUMBER OF RECORDS TO BE SORTED (OR ALL)?	7.	Enter 1.

8.	NUMBER OF KEY FIELDS (1 to 10)?	8.	Enter the number of fields to be included in the sort key.
			
			For example, if the record contains:
			A$ = account
			N$ = name
			S$ = address
			Z$ = zip code
			To sort by zip code and name, enter 2 for number of key fields.

9.	ENTER SEQUENCE NUMBER OF KEY FIELD xx IN RECORD	9.	The sequence number is the number of the field within the record. In the above example, zip code is sequence number 4. Since zip code is the first key field, 4 would be entered here the first time.
10.	KEY FIELD xx ASCENDING OR DESCENDING? (A OR D)?	10.	Enter A or D to indicate for each key field whether it is to be sorted into ascending or descending sequence. Repeat steps 9 and 10 until the last key field is done.
11.	WORK FILE NAME	11.	Enter the name of the work file.
12.	WORK FILE DEVICE ADDRESS	12.	Enter the device address of the work file. Valid entries are 310, 320, B20, B10, 350.

13.	ENTER THE NUMBER OF RECORDS TO BE SORTED ?_ _ _ _ _/	13.	Enter the best available estimate of the number of records to be sorted.
14.	KEY RETURN(EXEC) TO RESUME KEY SORT (OR) FULL-RECORD SORT NUMBER OF SECTORS FOR THE SORT-WORK FILE = XXX	14.	The type of sort to be performed as well as the work-file size in sectors is given. If a full-record sort is indicated and the work file size is larger than is available, the work file size for a key sort may be calculated by keying RESET RUN 2820 RETURN(EXEC).

CHAPTER 16
EXECUTING THE DISK SORT UTILITY

16.1 INTRODUCTION

Before executing the Sort Utility, the input and work files should be cataloged on a disk and available. The output file can be established by the utility, but if it is desired that it contain extra sectors, then it too must be established prior to execution.

NOTE:

Each file (input, output, and work) must fit on a single disk. No overlapping of a file from one disk to another is permitted. All three files must be mounted throughout the sorting procedure. The Disk Sort Utility modules must also be mounted. Normally, this is accomplished by simply mounting the ISS diskette. However, if disk space is scarce, the utility modules may be copied from the ISS diskette. The module names are:

SORT
DSM200BA
DSM200CA
DSM201AA
DSM202AA
DSM202BA
DSM203AA

The following information is required during the parameter entry stage:

1. the input file name
2. the input file disk device address
3. the number of records per block
4. the number of the first record to be sorted
5. the number of records to be sorted (or all)
6. the number of key fields on which to sort (no more than 10)
7. the sequence number of each key field
8. the sort order on each key field (ascending or descending)
9. the work file name
10. the work file device address
11. the output file name
12. the output file device address
13. the status of the output file, cataloged or not cataloged

Usually the starting record number is 1, and the number of records to be sorted is entered as "all". The principal exception to this occurs if the input file is too large to be sorted with the available disk storage. The input file can then be sorted in sections, producing separate sections of sorted output. Merging of these sections must be accomplished in a separate program (not provided).

Assuming the records contain the four fields:

A$ = account no.
N$ = name
S$ = address
Z$ = zip code

The sequence number of each field is:

Field Name	Sequence no.
A$	1
N$	2
S$	3
Z$	4

To sort on zip code and name, the number of key fields is 2, and the sequence numbers of these two fields are 4 (keyfield #1, the zip code) and 2 (keyfield #2, name).

16.2 OPERATING INSTRUCTIONS

DISPLAY		INSTRUCTIONS
1.	1.	From ISS DISK SORT menu, access the Disk Sort via the specified Special Function Key.
2.	2.	Mount the disks containing the input and work files. Mount the disk containing, or to contain, the output file.
3. INPUT FILE NAME ?_ _ _ _ _ _ _ _	3.	Enter the name of the input file. If RE-ENTER appears, too many characters were entered.
4. INPUT FILE DEVICE ADDRESS	4.	Enter the device address of the input file. Valid entries are 310, 320, B10, B20, 350. If INPUT INVALID is displayed, input file is not a data file.

5.	RECORDS PER BLOCK	5.	Enter the number of records per block. Enter 1 if the records are not blocked. If STOP TOO MANY FIELDS is displayed, there are more than 38 fields/record (blocked) or 55 fields/record (unblocked) in the input/file; the program cannot sort this file. If STOP INVALID RECORD FORMAT is displayed, the program cannot sort this file. If STOP NOT BLOCKED AS SPECIFIED is displayed, the value input is inconsistent with the actual file format; either rerun or stop. (If blocking was not done in array form, the program cannot sort the file.)
6.	STARTING RECORD # TO BE SORTED?	6.	To sort the entire file, enter 1. To sort part of the file, enter the number of the first record to be sorted.
7.	NUMBER OF RECORDS TO BE SORTED (OR ALL)?	7.	To sort the entire file, or all the remaining records starting with the record specified in Step 6, above, enter ALL. To sort a specific number of records, enter the number of records to be sorted. If STOP INVALID RECORD FORMAT is displayed, the program cannot sort the input file; you must use another file and return to Step 1. If a number is specified which is greater than the number of records in the file, starting with the record specified in Step 6, above, then all the remaining records will be sorted.
8.	NUMBER OF KEY FIELDS (1 to 10)?	8.	Enter the number of fields to be included in the sort key. For example, if the record contains: A$ = account

			N$ = name
			S$ = address
			Z$ = zip code
			To sort by zip code and name, enter 2 for number of key fields.

9.	ENTER SEQUENCE NUMBER OF KEY FIELD xx IN RECORD	9.	The sequence number is the number of the field within the record. In the above example, zip code is sequence number 4. Since zip code is the first key field, 4 would be entered here for the example case; 2 would be entered the second time for name. If STOP SORT KEY TOO LONG is displayed, the sort key fields exceed 64 bytes total; this is outside program limits.
10.	KEY FIELD xx ASCENDING OR DESCENDING? (A OR D)?	10.	Enter A or D to indicate for each key field whether it is to be sorted in its ascending or descending sequence. Repeat steps 9 and 10 until the last key field is done.
11.	WORK FILE NAME	11.	Enter the name of the work file.
12.	WORK FILE DEVICE ADDRESS	12.	Enter the device address of the work file. Valid entries are 310, 320, B10, B20, 350.
13.	OUTPUT FILE NAME	13.	Enter the name of the output file.
14.	OUTPUT FILE DEVICE ADDRESS	14.	Enter the device address of the output file. Valid entries are 310, 320, B10, B20, 350.
15.	IS OUTPUT FILE CATALOGED? (Y OR N)	15.	If the named output file is already cataloged, enter Y. Otherwise, enter N to let the utility establish the file. If the message OUTPUT SPACE TOO SMALL appears, the size of the specified file is inadequate. Repeat step 15 with a new output name and let the utility establish the file. The utility executes without further operator attention.

	Messages indicate the "pass" in operation.
16. PASS 1 - INTERNAL SORT	16. If ERR 43 occurs, the input file is not all the same format and cannot be sorted.
17. PASS 2 - MERGE	17. If STOP SEQUENCE ERROR is displayed, the program must be rerun from the beginning.
18. PASS 3 - OUTPUT	18. This message appears only if a key sort is performed. If STOP SEQUENCE ERROR is displayed, the program must be rerun from Step 1. If this error persists even when rerunning, call your Wang Service Representative.

```
INPUT RECORDS _ _ _ _ _
OUTPUT RECORDS _ _ _ _ _
STOP END OF SORT
```

CHAPTER 17
ISS DISK SORT UTILITY TIMINGS

Sort times depend on the number of records to be sorted, the size of each record, the length and position of the sort key in the record and the amount of available RAM (random access memory) in the CPU. In the table below, approximate empirically determined times for sorting sample data files using the Disk Sort Utility are given; the Model 2230-3 disk was used. Time to enter sort parameters is not included.

K = key sort, R = full-record sort. A = alphanumeric only, N = numeric only, A/N = mixed alphanumeric and numeric.

Input Records	Record Length	Blocked	Key Length	CPU Memory Size	Type of Sort	Time (min)
2000	24	10	24A	8K	R	8.22
20000	24	10	24A	8K	R	92.00
2000	24	10	24A	32K	R	7.02
8000	24	10	24A	32K	R	27.27
1000	120	2	8N	8K	K	7.83
1000	120	2	8N	32K	K	7.50
4000	120	2	8N	32K	K	29.42
1000	120	2	64A/N	8K	K	12.35
1000	120	2	64A/N	32K	R	9.22
4000	120	2	64A/N	32K	R	41.62

PART IV
KFAM

CHAPTER 18
OVERVIEW OF THE KFAM SYSTEMS

18.1 WHAT IS KFAM?

The 2200 BASIC language includes a group of statements used for disk operations that are known as the Catalog Mode statements. They are given this name because they create and maintain on a disk, a catalog, or index, of the files which are stored on the disk. This catalog includes, among other things, the name given to the file and the file's starting and ending sector addresses. The catalog system allows a file to be found by simply supplying its name (a service performed for data files by the statement DATA LOAD DC OPEN).

Though the catalog system keeps track of where each file is located on a disk, and thereby allows files to be easily found, it does not keep track of the individual records within a file. For example, a given disk may have an employee file called "PAY", an accounts receivable file called "A/R", and an inventory file called "INVT". The disk catalog system keeps track of where each of these files is located. However, the "PAY" file may consist of 250 employee records, the "A/R" file of 400 customer records, and the "INVT" file of 5000 product records. KFAM is a system for keeping track of and locating these individual records within a file.

For each file of records, KFAM creates and maintains an index of the individual records and their locations in the file. For the purpose of this index, each record is identified by some key field that can serve to mark it off from all other records. For example, for a payroll file, the employee name or number might be designated as the key field; for an inventory file a product number might be the key field. A record's key field is called its "key". The index constructed and maintained by KFAM can be thought of as a list of all the keys for a given file. Associated with each key in the index is the location of the record that the key identifies.

The index that KFAM constructs and maintains is itself kept as a cataloged file on a disk. It is called the Key File to distinguish it from the file of records that it indexes. The latter is called the User File.

When a file is indexed by KFAM, you can say in a program, "Find me the record for product AB-4975-1." KFAM subroutines, incorporated into the program, then search the Key File index and put the sector address of record AB-4975-1 into the User File's Current Sector address parameter in the Device Table. The program can then simply execute a DATA LOAD DC statement to read the desired record.

KFAM subroutines, incorporated into the user's programs, do all the work of searching and updating the Key File. There are KFAM subroutines to find records in a random sequence, and in ascending key sequence; there are subroutines to delete records, and to find a location for a new record and add the new key to the Key File. Thus, the programmer who uses KFAM need never know how the Key File is constructed. KFAM subroutines carry out all the necessary operations on the Key File.

The Key File that KFAM constructs is a sophisticated tree structure, designed so that keys can be found quickly in a random sequence, and even more quickly in ascending key sequence. It allows keys to be added and deleted easily, without disturbing the organization of the Key File.

Whenever a KFAM subroutine is to find a record, or add a new key to the key file and find a location for the record in the User File, the KFAM subroutine puts the User File record location into the Current Sector address parameter of the Device Table, opposite the file number (#0-#6) being used for the User File. Thus, on return from the subroutine, an ordinary Catalog Mode DATA LOAD DC or DATA SAVE DC can be executed, and will take place at the desired sector location.

There are two versions of KFAM included in ISS. KFAM-3 is the general purpose KFAM system for use when a file is to be accessed by only one CPU at a time. KFAM-4 is a modification of the KFAM-3 system designed for a multiplexed disk environment, in which more than one CPU may wish to access a file simultaneously. The key file structures built by KFAM-3 and KFAM-4 are identical, and operations performed by the utilities and subroutines are very similar. The chief difference is that KFAM-4 includes special protective procedures to prevent destructive conflict by different CPU's. Though the main functions performed by KFAM-4 software are very similar to those of KFAM-3, once a file is organized under one version, only the software associated with that version may be used on it. A conversion program is provided to convert a KFAM-3 Key File to a KFAM-4 Key File. KFAM-3 offers better performance than KFAM-4, and should be used whenever it is certain that a file is used by only one CPU at a time.

There are two versions of KFAM not included in ISS. These are the original KFAM, (referred to as KFAM-1 in this document) and KFAM-2. Unlike these versions of KFAM, KFAM-3 and KFAM-4 use the BASIC statements described in SORT STATEMENTS (Publication #700-3559A). This permits improvements in execution speed, memory requirements, program simplicity, and system flexibility which could not otherwise be achieved. Utility programs are provided to convert to KFAM-3 from KFAM-1 and KFAM-2.

18.2 THE FUNCTIONAL COMPONENTS OF KFAM-3 AND KFAM-4

KFAM-3 and KFAM-4 can each be broken down into the following functional types of software.

1. Set-up Utilities: Stand-alone programs used to initialize a new Key File, and to create a Key File for an already existent User File.

2. KFAM Subroutines: DEFFN' subroutines which are incorporated into a User program. These are used to locate records in the user file and to add and delete keys from the Key File. These are the operational heart of KFAM.

3. Supplementary Maintenance Utilities: Stand-alone programs which perform a variety of tasks related to the maintenance of a Key File and User File.

18.3 <u>HOW TO GET STARTED WITH KFAM</u>

KFAM provides a means for accessing records that are saved in a disk file. However, it does not process these records in any way. After it has found a record, the processing of the record, (loading it, updating it, saving it, etcetera) is left to the user-written program. Thus, to use KFAM one must have a working knowledge of elementary BASIC and of the fundamentals of Catalog Mode disk operations.

It is strongly recommended that the first-time user of KFAM begin by setting up a dummy KFAM-3 file, and experiment with the subroutines and utilities on this file before attempting to operate on valuable files.

The following is a step-by-step outline of how to begin setting up KFAM files.

1. Decide whether to use KFAM-3 or KFAM-4. If your system includes just one CPU then you should use KFAM-3. If your system includes more than one CPU attached to a multiplexed disk drive, and the file to be accessed may be accessed by several CPU's simultaneously, then you must use KFAM-4. If you are a first time user and wish to learn to use KFAM-4, it may be advisable for you to begin by experimenting with a dummy KFAM-3 file and then graduate to KFAM-4 as you gain confidence.

2. Read Chapter 19, KFAM Requirements and Conventions. This chapter describes the four types of User File records which are acceptable to KFAM, limitations on the size and characteristics of the key field, and certain KFAM conventions which must be adhered to.

3. A Key File is stored as a cataloged file on a disk. It may reside on the same disk as the User File, or on another disk (which must be mounted whenever the User File is accessed). A set-up utility called INITIALIZE KFAM FILE must be run whenever a new KFAM file (Key File and User File) is to be established.

   INITIALIZE KFAM FILE calculates the required size of the Key File, given an estimate of the maximum number of records to be saved in the User File, and will catalog a Key File with the required number of sectors. It saves in the Key File some vital information about the User File, based on information supplied by the operator. Optionally it will also catalog a User File with the proper number of sectors, if the User File does not already exist.

   A guide to the information required by INITIALIZE KFAM FILE and detailed operating instructions are provided in Chapter 20.

4. If your User File already exists, a second set-up utility program can be run after INITIALIZE KFAM FILE. It reads your User File and creates an entry in the Key File for each record in the User File.

This utility is called KEY FILE CREATION. Operating instructions are provided in Chapter 20.

5. If your User File does not already exist, then after running INITIALIZE KFAM FILE you will use the KFAM subroutines in a program you write to build your User File and Key File simultaneously.

6. The KFAM subroutines are DEFFN' subroutines which are incorporated into an application program (written by the KFAM user) to perform standard tasks for files indexed by KFAM. Before writing a program, or program module, you should determine the KFAM subroutines that will be needed. You should then run the Utility BUILD SUBROUTINE MODULE. This lets you choose what subroutines, and optional subroutine features, you wish to have. It builds a program file on disk which contains the selected subroutines. (BUILD SUBROUTINE MODULE for KFAM-3 is described in Chapter 21. The KFAM-4 version appears in Chapter 22.)

Subroutines are available to perform the following tasks.

TYPE AND NAME	FUNCTION
<u>General Purpose</u>	
OPEN	Open specified User File and companion Key File.
CLOSE	Close User File and companion Key File.
<u>Random Access</u>	
FINDOLD	Locate specified key in the Key File; set User File Current Sector Address to record in User File with that key.
<u>Key Sequence Access</u>	
FINDFIRST	Locate record with lowest key in User File; set User File Current Sector address to that sector.
FINDNEXT	Locate next record in User File in logical key sequence; set User File Current Sector Address to that sector.
FINDLAST	Locate record with highest key in User File; set the User File Current Sector Address to that sector.
<u>Add and Delete</u>	
FINDNEW	Add specified key to Key File; allocate space for a new record in the User File, and set the User File Current Sector Address to that sector.

FINDNEW (HERE)	Add specified key to Key File; set the User File Current Sector Address to the sector where the new record is to be written.
DELETE	Remove specified key from Key File; set the User File Current Sector Address to the record that has the deleted key.

Special Purpose (KFAM-4 ONLY)

RELEASE	Allow a User File record, previously protected by one CPU, to be accessed by any CPU.

Since the KFAM subroutines allow records to be added and deleted, as well as accessed, all routine file maintenance takes place in application programs which use KFAM subroutines. As a general rule all operations on the Key File are accomplished by the KFAM subroutines, while all operations on the User File are accomplished by user written statements in the application program.

Detailed descriptions of the GOSUB' statements needed to call each of the KFAM subroutines are given in Chapter 21 for KFAM-3 and 22 for KFAM-4.

7. Though the KFAM subroutines are the heart of the KFAM system, and perform most of the file maintenance, a group of Supplementary Maintenance Utilities are included to carry out certain maintenance tasks that will occasionally be required.

   a) The REORGANIZE Utilities: When a record is "deleted" by using the DELETE subroutine, its key and location are simply removed from the Key File. It then cannot be accessed by KFAM. The record itself in the user file is not removed. It is possible to reuse the spaces occupied by deleted records in the User File, but if this is not done, the User File gradually becomes bloated with DELETED records. The reorganize utilities reorganize the User File putting its records into key sequence and eliminating DELETED records. They then automatically construct a new Key File for accessing the reorganized User File. KFAM-3 and KFAM-4 each have two versions of REORGANIZE utilities.

      THE REORGANIZE SUB-SYSTEM: Is a three-module utility program which reorganizes a file by outputting a new reorganized User File and Key File. The old Key File and User File are left intact. It is called by a user written set-up module which provides parameters for the reorganization.

      REORGANIZE KFAM FILE: Is a utility program which reorganizes the User File and Key File in place. It should be used only for a file so large that adequate output files could not be mounted at the same time as the file to be reorganized.

Detailed instructions for KFAM-3 and KFAM-4 Reorganize utilities are given in Chapter 23.

b) The Adjust Files Utilities include two utilities which can be used together to copy a KFAM file and increase or decrease the amount of disk space allocated to the file. These utilities are called REALLOCATE KFAM FILE SPACE and DISK COPY AND REORGANIZE. The latter can be used alone to copy any cataloged file to another disk.

c) PRINT KEY FILE: This utility prints the complete contents of the Key File with appropriate labeling of data. It can be useful as a diagnostic tool, and helpful to advanced programmers who may wish to examine the Key File structure.

d) Recovery Utilities: A KEY FILE RECOVERY utility is provided to reconstruct a Key File in the event of its accidental destruction. The User File must be intact for this program to operate successfully.

For KFAM-4 only, there is a second kind of recovery utility called RESET ACCESS TABLE. KFAM-4 maintains in the Key File information about which CPU's are operating on the file. This information is kept in a part of the Key File called the "access table". This access table will contain erroneous information if a CPU fails to CLOSE a file it has opened. due to power failure or program error. The RESET ACCESS TABLE utility is provided to clear this erroneous information from the access table.

e) The KFAM Conversion Utilities. Utility programs are provided with KFAM-3 to convert from KFAM-1 to KFAM-3, and from KFAM-2 to KFAM-3. A utility program is provided with KFAM-4 to convert from KFAM-3 to KFAM-4.

18.4 OVERVIEW OF KFAM-4

KFAM-4 is a modification of the KFAM-3 system, designed for a disk multiplexed environment. It allows up to four CPU's to access a KFAM disk file, and includes protective procedures designed to prevent destructive intrusions of one CPU into the file operation of another CPU. These procedures are designed to offer the minimal protection consistent with the type of operation being performed, so that other CPU's can have the safe maximum availability of the disk and the file.

To use KFAM-4 in a multiplex environment one must have some understanding of the types of problems presented by this environment. To illustrate these problems, assume that several CPU's attempt to use KFAM-3 to access a User file, via a single Key File. Serious problems can occur at several different levels:

1. The key file can be accidentally destroyed simply by two CPU's executing KFAM subroutines contemporaneously. Recall that the disk multiplexer (Model 2224 or 2230MXA/B) allows a CPU to execute a single disk instruction, and then polls the other CPU's to see if they are waiting to execute a disk instruction. Since a single KFAM-3 subroutine (FINDNEW, FINDOLD, DELETE, etc.) may execute many disk instructions, its disk instructions could be interspersed with those of another CPU. Several of the KFAM-3 subroutines can alter the Key File structure, and require that that structure be stable from the time the key file is read until it is successfully restructured. Two such subroutines operating on the key file contemporaneously could result in an illegitimate Key File structure, which would destroy its effectiveness as an index to the User File.

2. If one CPU is attempting to access records sequentially using FINDNEXT, and, during record processing, another CPU accesses the file, then a subsequent execution of FINDNEXT by the first CPU will not access the next sequential record.

3. If one CPU is attempting to perform an operation on an entire User File, (Print Closing Balances, etc.) and another is updating records in the file, then the overall file status reported may, in fact, have never existed at any one point in time.

4. If two or more CPU's contemporaneously access the same User Record for updating, so that, from the point of view of the disk, the accessing sequence looks like this:

```
DATALOAD DC X          (CPU #1)
DATALOAD DC X          (CPU #2)
.
.
.
DATASAVE DC X+4        (CPU #1)
DATASAVE DC X+7        (CPU #2)
```

   The record is now erroneous. It should contain X+11 but instead contains X+7.

There are, of course, other problems which can occur, but these problems may be taken as characteristic of the types of problems involved. KFAM-4 offers solutions to these types of problems by several different means:

1. KFAM-4 subroutines hog the disk during their execution. Before passing control back to the User program, the KFAM subroutine returns the disk to normal, non-hog, mode. This solves the problem of key file destruction caused by simultaneous execution of KFAM subroutines.

2. Information which pertains to the program accessing the file rather than to the Key File itself is maintained in the CPU's by KFAM-4, rather than in the Key File (KDR) as in KFAM-3. This information includes last key accessed, limb path, file numbers, etcetera. By keeping the information for a CPU, in the respective CPU, one CPU

can be accessing the file sequentially with FINDNEXT while another is accessing it, and the FINDNEXT from the first CPU always accesses the next sequential record.

3. In KFAM-4 an additional GOSUB' parameter is required to call the OPEN subroutine. With this additional parameter a CPU initiating operations on a KFAM-4 file can request exclusive access to the entire file for the duration of its operation. When exclusive access is requested, the OPEN subroutine checks to see if any other CPU currently has the file open; if not, it grants exclusive access. Once exclusive access is granted, no other CPU can open the file until the CPU with exclusive access has executed the CLOSE subroutine. This permits operations on an entire file to be completed with the file protected from outside alteration. Files may also be opened in a non-exclusive mode which permits other CPU's to open the file.

   The information as to how many of the four possible CPU's have opened the file, and whether it is open in exclusive or non-exclusive mode is saved in an access table. The access table is part of the KDR record in the Key File. In order that this information be accurate, it is essential that each CPU execute the CLOSE subroutine when its program is finished with the file. If the file is not CLOSED, the entry will remain in the KDR. Four non-exclusive entries fill the access table and bar any CPU from opening it; one exclusive entry, left in file, has the same effect.

   KFAM-4 includes a utility called RESET ACCESS TABLE. The purpose of this utility is to clear the access table in the event of a power failure or other disaster. This utility clears the entire access table, and therefore should be run only when no CPU has the file open.

4. In KFAM-4 an additional GOSUB' parameter is required to call each record-access subroutine. With this parameter the CPU accessing a record can set a protect flag for the accessed record. If a CPU has set a protect flag for a record, then that record may be accessed only by that CPU, until the protect flag is turned off. If records are blocked, setting a protect flag for one record in the block prevents other CPU's from accessing any of the records in the block.

   When a CPU performs an update of a record, it must set a protect flag for that record. This ensures that it can complete its update operation before another CPU can access the record, and thereby eliminates the problem of simultaneous updates described as problem 4, above.

   The protect flag on a record (or block of records) is automatically turned off when the CPU that set the protect flag executes any other KFAM subroutine on the same file. Thus, for example, if a series of updates are being performed by a CPU, each access of a record turns off the protect flag for the record previously accessed by that CPU, and, optionally, sets the protect flag for the new record.

KFAM-4 includes a subroutine called RELEASE. When executed by a particular CPU, this subroutine simply turns off a protect flag previously set by that CPU. This subroutine should be used if there may be a substantial delay before the next KFAM subroutine call.

It is not possible for a single CPU to have open a KFAM-4 file and a KFAM-3 file at the same time.

CHAPTER 19
KFAM REQUIREMENTS AND CONVENTIONS

19.1 USER FILE

The User File must be a cataloged disk file. It must be wholly contained on one disk platter. (See KFAM Advanced Programming Techniques for files too large to fit on one platter.) All records must be of a fixed length. Four record types are supported. All records in a file must be the same type. The record types are:

Type "N" - No Blocking

Each record occupies exactly one sector.

The key must be located in the same position within each record.

Records may be written in the "DC" mode, with control bytes, or in the "BA" mode, without control bytes.

(This corresponds to record type "F" in KFAM-1.)

Type "A" - Array Type Blocked Records

Records must be written in array form:

```
     DIM A$(4)3, B(4), C$(4)20
     DATASAVE DC n, A$(), B(), C$()
```

indicating 4 records per block, each containing an A$, B, and C$. The block of records must be written with control bytes; DATASAVE BA may not be used.

All records must have the same format.

The key must be located in the same position within each record. The key may be a part of a field, i.e., STR(C$, 11, 10), but may not span fields, may not include control bytes, and may not be a numeric field or any part of a numeric field.

The block of records may not exceed one sector in length.

There may not be more than 38 fields per record.

Type "C" - Contiguous Blocked Records

All records must be the same length.

All the fields of a given record are stored contiguously on the disk, for example:

```
DIM A1$3, C1$20, A2$3, C2$20, A3$3, C3$20, A4$3, C4$20
DATASAVE DC#n, A1$, B1, C1$, A2$, B2, C2$, A3$, B3, C3$,
    A4$, B4, C4$
```

indicating 4 records per block, each containing an Aj$, Bj, and Cj$. (This corresponds to record type "FB" in KFAM-1.)

The key must be located in the same position within each record.

The block of records may not exceed one sector in length.

Records may be written in the "DC" mode, with control bytes, or in the "BA" mode without control bytes. However, if the file must be reorganized in place using the REORGANIZE KFAM FILE utility, it must be written with control bytes.

Type "M" - Multiple Sector Records

Each record occupies more than one sector.

Each record occupies the same number of sectors.

The key must be located in the same position within each record. The key may be located in any sector of the record, but may not span sectors.

Records may be written in the "DC" mode, with control bytes, or in the "BA" mode, without control bytes.

Records may be up to 255 sectors in length. However, the following restrictions apply in REORGANIZE KFAM FILE:

a. Records may not exceed 40 sectors in length.

b. Reorganization cannot be executed in 12K of memory if the record length exceeds 8 sectors.

(This corresponds to record type "FM" in KFAM-1.)

User File Name

The User file name, as recorded in the disk catalog, must conform to the following conventions:

The 5th character must be the letter "F".

The 6th character must be a digit 0-9.

19.2 KEY

The record key as it appears in the User File may be from 1 to 30 bytes of alphanumeric data (including hexadecimal data or packed numbers). The key may not be a numeric field.

The first byte of an active key may not contain the value HEX(FF). The value HEX(FF) in the first byte of a key in the User file indicates that the record has been deleted from the Key File.

The key may not contain a value of all bytes HEX(00). (This corresponds to the packed number 0, or the binary number 0, as a key value.) This lowest possible value is reserved for the system.

Duplicate keys are not allowed.

19.3 KEY FILE

The Key File name is constructed by INITIALIZE KFAM FILE from the User File name, as follows:

The 5th character in the User File Name is changed from "F" to "K".

The 6th character is assigned the Key File number. This is always 1 unless multiple Key Files are maintained for the one User File, in which case it may be any digit 1-9.

Size of the Key File

The first sector of the Key File contains the Key Descriptor Record (KDR). The KDR contains control information necessary for KFAM.

The remaining sectors of the Key File are available for Key Index Records (KIR's). Each KIR occupies one sector and contains Key Index Entries (KIE's). The KIE is a field containing a key and a 3-byte pointer. The key is the same as one of the keys in the User File. The pointer points to a User record on the disk, either directly or indirectly. The maximum number of KIE's per KIR is given by:

```
N = INT (240/(K+3))

where:  K = Key length
        3 = pointer length
        N = maximum KIE's per KIR
```

The average number, A, of KIE's per KIR is calculated (conservatively) as follows:

```
A = INT(N*.6)
```

The number of sectors required for the Key File, for a given number of records, R, is as follows:

```
S = INT(R/(A-1))+5
```

19.4 DEVICE ADDRESSES

Device addresses, 310, 320, 330, 350, B20, B30 and B10, are recognized by KFAM as valid disk device addresses. KFAM-4 also uses the hog mode versions of these addresses.

CHAPTER 20
THE KFAM SET-UP UTILITIES

20.1 OVERVIEW OF INITIALIZE KFAM FILE (KFAM-3 and KFAM-4)

INITIALIZE KFAM FILE must be run, as the first step in setting up a KFAM file.

INITIALIZE KFAM FILE optionally catalogs an area on disk for the User File, or the Key File, or both, or operates with an existing User File, or Key File, or both. It sets up the KDR record (the first record of the Key File, containing vital information about the User File and the Key File), based on information supplied by the operator. It then creates a "null" (empty) Key File.

The DATASAVE DC END trailer is set to the next to last sector in the User File, regardless of whether or not the User File was cataloged prior to running the utility.

Information Required by The Utility

The utility requires that the following information be supplied by the operator:

User File Name
Device Address for User File: 310, 320, 330, 350, B10, B20, B30
Is User File Cataloged?: (Y OR N)
Key File Number: 1-9
Device Address for Key File: 310, 320, 330, 350, B10, B20, or B30
Is Key File Catalogued?: Y or N
Record Type: A, C, M, or N
Logical Record Length (Type A or C): nnn
Blocking Factor (Type A or C): nn
Sectors per Record (Type M): nnn
Key Length: 1-30
Starting Position of Key: nnnnn
Estimated Number of Records: nnnnn
Are File Specifications OK?: Y or N
Hard Copy Printout?: Y or N
Do Another File: Y or N

This is a formidable set of questions for an operator. It is suggested that this utility be run by the application programmer, or that the programmer write a set of specific answers for a specific KFAM file, as a supplement to the general operating instructions below.

Some of the answers to the above questions are not obvious, and require some discussion:

User File Name

The User File Name must conform to the KFAM naming convention. The 5th byte must be "F". The 6th byte must be a digit 0-9. The remaining bytes may be any alphanumeric characters.

An existing User File may be renamed to conform to KFAM's requirements by executing the following two commands in immediate mode:

```
SCRATCH d "old name"
DATASAVE DC OPEN d "old-name", "new-name"
```

where d = the disk platter, F or R
old-name = the original file name
new-name = new name, conforming to KFAM convention

The data trailer record, or "END" record, is lost following this procedure. This should do no harm if the file is to be used strictly as a KFAM file, because INITIALIZE KFAM FILE always sets the "END" record at the next-to-last sector of the cataloged space, regardless of where it was before, and the Key File Creation Utility uses the last key, not the "END" record, to determine end-of-file.

Key File Number

Normally, 1 should be entered. However, if multiple Key Files are to be used to index the same User File, they must be uniquely identified by the Key File Number, which can be any digit from 1 to 9.

The Key File name is derived from the User File name, by replacing the "F" in position 5 with "K", and the digit in position 6 with the Key File Number.

Record Type

KFAM supports four different record types:

Type A: Array type blocked records.

More than one data record is contained in a sector. The block of records is written as an array, i.e.:

```
DIM A$(4)12, B(4), C$(4)36
DATASAVE DC #n, A$(), B(), C$()
```

indicating 4 records per sector, each record containing an A$, B, and C$.

Type C: Contiguous blocked records.

More than one data record is contained in a sector. Each record occupies a contiguous amount of space on the disk. For example, there are three records per sector, each containing a key (K$) and data (D$):

```
DIM K1$12, D1$64, K2$12, D2$64, K3$12, D3$64
DATASAVE DC #n, K1$, D1$, K2$, D2$, K3$, D3$
```

This corresponds to record type "FB" in the original KFAM.

Type N: No blocking

Each data record occupies one sector. This corresponds to record type "F" in the original KFAM, except that in KFAM-3 data may begin at byte 0 and extend to byte 255.

Type M: Multiple Sectors per record.

Each data record occupies two or more sectors. This corresponds to record type "FM" in the original KFAM.

Logical Record Length

The logical record length for record types A and C is calculated as follows:

a. Add up the lengths of the fields contained in a single record (numeric fields are 8 bytes long).

b. Add 1 per field of the record, for control bytes.

For example, in the above example for type A records, the record length is 59. In the above example for type C records, the record length is 78.

All records of the file must have the same length. For type A, all records must also have the same format, for example, a 12-byte alpha field, followed by a numeric field, followed by a 36-byte alpha field, each field contained in an array of 4 elements.

Blocking Factor

The blocking factor is the number of records per sector (Type A and C).

Starting Position of Key

This is the absolute starting position of the key within the sector or sectors, except for type A records, where it is the position within the record plus two for sector control bytes. This requires some explanation.

When a record is written on disk, in the normal mode (DATASAVE DC or DATASAVE DA), two control bytes are written at the start of the sector. Following these two control bytes, the Start-of-Value (SOV) control byte for the first data value is written, followed by the data itself. Then the SOV control byte for the second value, and the second value itself, are written, and so on. An end-of-block (EOB) control byte is written following the last data value written.

The layout of the sector on disk looks like this:

CONTROL	CONTROL	SOV	Field 1	SOV	Field 2	SOV	Field 3	SOV	Field 4	SOV	Field 5	EOB	Not Used
0	1	2	3										

For the purposes of determining the starting position of the key, the bytes of the sector are numbered from 0 to 255. The starting control bytes are bytes 0 and 1. The SOV control byte for the first field is byte 2. The first byte of data is byte 3. The second field starts in byte 3 + L1 + 1, where L1 is the length of the first field, and so on.

The starting position of the key is the number of the first byte of the key. For blocked records, the blocking is ignored when calculating the starting position of the key. It is calculated as if there were only one record per block. The KFAM utilities make the necessary adjustments to calculate the position of the key in subsequent records within the sector.

In particular, with type A records, the blocking should be ignored when calculating the starting position of the key. Given the record in the example:

```
DIM A$(4)12, B(4), C$(4)36
DATASAVE DC #n, A$(), B(), C$()
```

the actual layout of the sector is as follows:

Bytes	Contents
0,1	Control bytes
2	SOV
3-14	A$(1)
15	SOV
16-27	A$(2)
28	SOV
29-40	A$(3)
41	SOV
42-53	A$(4)
54	SOV
55-62	B(1)
63	SOV
64-71	B(2)
72	SOV
73-80	B(3)
81	SOV
82-89	B(4)
90	SOV
91-126	C$(1)
127	SOV
128-163	C$(2)
164	SOV
165-200	C$(3)
201	SOV
202-237	C$(4)
238	EOB

The fact that there are four records per sector should be ignored in determining the starting position of the key. The sector should be seen as if it contained only one record, as follows:

Byte	Contents
0,1	Control bytes
2	SOV
3-14	A$(1)
15	SOV
16-23	B(1)
24	SOV
25-60	C$(1)
61	EOB

If the key starts in the first byte of C$(), the starting key position is 25, and not 91, as would be indicated by the actual blocking.

For record types C, M and N, it is possible to have records written in the DATASAVE BA mode. In that case, no control bytes are inserted, and the starting position of the key is exactly where it is located in the array defining the record (starting byte 0).

For record type M, it is possible to have the key begin in the second, or higher, sector of the record. In that case, add 256 for each sector preceding the one containing the key, and then add the starting position of the key within the sector (first byte of the sector = 0).

When writing multiple sectors in the normal mode (DATASAVE DC or DATASAVE DA), it is necessary to determine which field will begin the second sector, etc. The 2200 System does not write partial fields in a sector. Where there is not room to write the next field in the current sector, the 2200 System writes an EOB control byte, leaves the rest of the space unused, and starts another sector. For example, if the record is defined:

```
DIM D$(6)64
DATASAVE DC #n, D$()
```

the record occupies 2 sectors, as follows:

First sector:

Bytes	Contents
0,1	Control bytes
2	SOV
3-66	D$(1)
67	SOV
68-131	D$(2)
132	SOV
133-196	D$(3)
197	EOB
198-255	Not used (58 bytes, not room to write next complete field)

Second sector:

Bytes	Contents
0,1	Control bytes
2	SOV
3-66	D$(4)
67	SOV
68-131	D$(5)
132	SOV
133-196	D$(6)
197	EOB
198-255	Not used (end of data)

Once the actual record layout is determined, then the starting position of the key can be calculated. If the key occupies, for example, the first 8 bytes of D$(4), then the starting position of the key is 259, and not 198 as might be calculated by ignoring the actual way that the system writes records.

Estimated Number of Records

This is the maximum number of records that the User File will contain. If the User File is not yet cataloged, the program will catalog enough sectors to hold this many records. If the User File is already cataloged, the program checks that enough space is cataloged to contain this many records.

The program also calculates the size of the Key File, based on the estimated number of records. If the Key File is not yet cataloged, the program catalogs the required number of sectors. If the Key File is already cataloged, the program checks to see that enough space is cataloged. If there is not enough space cataloged, the program issues a warning message.

The estimated number of records should be calculated in advance, as the maximum number of records which the User File will contain, plus an estimate of the number of deleted records which will be in the file when it is at its maximum size.

This estimate is not critical. It can be revised later, using the REALLOCATE KFAM SPACE and DISK COPY/REORGANIZE utilities.

Hard COPY Printout

If the configuration of the available 2200 system does not include a printer, the answer to this question should always be "N".

NOTE:

In the KFAM-4 version of this utility, hog mode is selected for the disks containing the Key File and User File. To operate the KFAM-4 utility in non-hog mode or to execute it at a non-multiplexed disk drive, key

M$ = "X" (EXEC)

at KFAM-4 utilities menu, prior to loading the utility.

20.2 INITIALIZE KFAM FILE OPERATING INSTRUCTIONS (KFAM-3 and KFAM-4)

DISPLAY	INSTRUCTIONS
1.	1. From KFAM-3 or KFAM-4 menu, access the INITIALIZE KFAM FILE utility via the specified Special Function Key.
2.	2. Mount the disk platter(s) containing, or to contain, the User File and Key File.
3. ENTER USER FILE NAME (SSSSFJNN)	3. Enter the name of the User File. MESSAGE: 2,4,5

NOTE:

Error messages and recovery procedures follow the operating instructions.

4. ENTER THE NUMBER FOR THE DATA FILE DEVICE ADDRESS

1.	310	5.	B10
2.	320	6.	B20
3.	330	7.	B30
4.	350		

4. Enter the selection number for the user file disk device address.

MESSAGE: 2, 6, 7

5. IS DATA FILE CATALOGED? (Y OR N)

5. Enter "Y" if the User File already exists. Enter "N" if the User File does not exist.

MESSAGE: 2, 9, 10

6. ENTER KEY FILE NUMBER

6. Normally enter 1.

If there is more than one Key File for a single user file, the Key File Number is used to distinguish the Key Files. The Key File Number can be any digit from 1 to 9.

MESSAGE: 2, 3, 11

7. ENTER NUMBER OF THE KEY FILE DEVICE ADDRESS

1.	310	5.	B10
2.	320	6.	B20
3.	330	7.	B30
4.	350		

7. Enter the selection number for the key file device address.

MESSAGE: 2, 6, 7

8. IS KEY FILE CATALOGED? (Y OR N)

8. Enter "Y" if space has already been cataloged for a Key File. Enter "N" if the Key File has not been cataloged.

9. ENTER RECORD TYPE (A,C,N,M)

9. Enter A, C, N or M.
A=array type blocking
C=contiguous blocking
N=no blocking
M=multiple sector records

If record type A or C, proceed with Step 10, below.
If record type M, proceed with

	Step 12, below. If record type N, proceed with Step 13, below. MESSAGE: 2. 14
10. ENTER LOGICAL RECORD LENGTH	10. Enter logical record length. See above for calculation of logical record length. MESSAGE: 2, 3
11. ENTER BLOCKING FACTOR	11. Enter number of records per sector. Proceed with Step 13, below. MESSAGE: 2, 3, 15
12. ENTER NUMBER OF SECTORS PER RECORD	12. Enter the number of Sectors per record. Record Type M only. MESSAGE: 2, 3, 16
13. ENTER KEY LENGTH	13. Enter key length (1 to 30). All record types. MESSAGE: 2, 3, 17
14. ENTER STARTING POSITION OF KEY	14. Enter starting position of key field within sector. See above for calculation of starting position of key. MESSAGE: 2, 3, 7, 8 18, 19
15. ENTER ESTIMATED NUMBER OF RECORDS	15. Enter estimated maximum number of records in User File. See above for calculation of number of records. MESSAGE: 2, 3, 20
16.	16. The system calculates disk space required for User File and Key File. MESSAGE: 21, 22

17.	17. The system displays file specifications on the screen. MESSAGE: 23
18. ARE FILE SPECIFICATIONS OK (Y OR N)	18. Check file specifications displayed on the screen. Enter Y to continue. Enter N to start again at Step 3. MESSAGE: 2
19. DO YOU WANT A HARD COPY PRINTOUT OF FILE DESCRIPTION? (Y OR N)?	19. For a hard copy printout, mount paper on printer and enter Y. For no hardcopy, enter N. MESSAGE: 2, 24
20.	20. The system prints file specifications on the printer. This is an exact duplicate of the screen display, Step 17.
21.	21. The system initializes the User File and Key File. MESSAGE: 7, 8
22. DO YOU WISH TO DO ANOTHER FILE (Y OR N)	22. Enter Y to do another file. Enter N to stop. If Y is entered, the program repeats from Step 3, above. MESSAGE: 2
23.	23. The system returns to the KFAM menu.

Error Messages

ERROR MESSAGE	EXPLANATION/RECOVERY
2. RE-ENTER	Too many characters were entered. RECOVERY: Repeat the step, entering not more than the number of characters indicated on the screen.

		Not "Y" or "N", in response to a "yes" or "no" question. RECOVERY: Repeat the step, entering "Y" or "N".
3.	ERR29	A non-numeric quantity was entered when a numeric quantity was requested. RECOVERY: Reenter numeric quantity.
4.	FILE NAME MUST HAVE F IN POSITION 5	User File name must have an "F" in the 5th position. RECOVERY: Repeat Step 3 with correct User File name.
5.	FILE NAME MUST HAVE NUMBER IN POSITION 6	User File name must have a digit, 0-9, in the 6th position. RECOVERY: Repeat step 3 with correct user file name.
6.	INVALID DEVICE ADDRESS	RECOVERY: Number entered not an integer 1-7. Repeat the step, entering a valid selection number.
7.	ERR72	Disk read error. RECOVERY: Rerun the program. If the error persists, recreate the platter from a backup copy (#1 = Key File, #2 = User File).
8.	ERR85	Disk write error. RECOVERY: Rerun the program. If the error persists, the platter has a bad sector and must be replaced.
9.	FILE NOT FOUND	The User File is not cataloged on the specified device. RECOVERY: Repeat from Step 3.
10.	FILE ALREADY CATALOGED	The User File, specified as not cataloged, is cataloged on the specified device. RECOVERY: Repeat from Step 3.
11.	ZERO INVALID	The Key File Number may not be 0.

	RECOVERY: Repeat Step 6, entering a Key File Number from 1 to 9.
12. FILE NOT FOUND	The Key File is not cataloged on the specified device. RECOVERY: Repeat from Step 6.
13. FILE ALREADY CATALOGED	The Key File, specified as not cataloged, is cataloged on the specified device. RECOVERY: Repeat from Step 6.
14. INVALID RECORD TYPE	Record type must be A, C, M, or N. RECOVERY: Repeat Step 9. Enter A, C, M or N.
15. BLOCKING FACTOR OR RECORD LENGTH INCORRECT	Record Length times blocking factor, for record type A, may not be greater than 253; for type C may not be greater than 256. RECOVERY: Repeat from Step 10. Recalculate record length or blocking factor if the product is too large.
16. INVALID-MUST BE 2 TO 255	Type M: Number of sectors per record must be 2-255. RECOVERY: Repeat Step 12 with correct value.
17. INVALID-KEY MUST BE 1 TO 30	The key length must be 1-30. RECOVERY: Repeat Step 13 with correct value.
18. KEY OVERLAPS END OF RECORD	The key goes beyond the boundaries of the record, as determined by the record type and record length. RECOVERY: Recalculate starting position of key and reenter (Step 14).
19. KEY MAY NOT SPAN SECTORS	Type M: The key goes over a boundary between sectors. RECOVERY: Recalculate starting position of key and reenter (Step 14).

20. USER FILE TOO SMALL	The User File, which is already cataloged, does not have room for the estimated number of records. RECOVERY: Repeat Step 15 with a smaller estimate. If necessary, reallocate KFAM space and DISK/COPY REORGANIZE can be run later to increase the size of the User File and Key File.
21. STOP NO ROOM FOR KEY FILE	The User File is already cataloged. There is not sufficient space on the designated platter to catalog the Key File. RECOVERY: The Key File must be cataloged on another platter, or the User File must be shortened.
22. See error display SECTORS AVAILABLE SECTORS REQUESTED	There is not enough available space on the designated platter(s) to catalog User File and/or Key File. RECOVERY: Repeat Step 15 with a smaller estimate. If necessary, rerun the program, using disk platters with more available space, or split the User File into 2 parts (see "KFAM Programming Techniques").
23. WARNING--KEY FILE TOO SMALL	An existing Key File is too small to accommodate the estimated number of records. The program continues with a warning message. RECOVERY: Run REALLOCATE KFAM SPACE and DISK/COPY REORGANIZE to increase the size of the Key File.
24. System hangs up - no message.	Printer not turned on, or no device 215. RECOVERY: Turn on printer.

20.3 THE FILE CREATION UTILITY (KFAM-3 AND KFAM-4)

Program Description

The KEY FILE CREATION utility creates a Key File for the records in an existing User File. INITIALIZE KFAM FILE must have been run first, to initialize both the User File and the Key File.

KEY FILE CREATION ignores any records which have HEX(FF) in the first byte of the key, under the assumption that they are deleted records. It also ignores records which have duplicate keys, but lists the relative sector number and record number where such duplicate keys are encountered.

The utility requires the operator to enter the key for the last record in the User File in physical sequence. The program uses this to detect the end-of-file condition. The value of the last key should be made available to the operator before runninq this program.

For KFAM-4 only, the file is opened in the exclusive mode.

KEY FILE CREATION Operating Instructions

DISPLAY	INSTRUCTIONS
1.	1. From KFAM-3 or KFAM-4 menu, access KEY FILE CREATION utility via the specified Special Function Key.
2. ENTER USER FILE NAME (SSSSFJNN)	2. Enter the name of the User File. MESSAGE: 2,4
	NOTE: Error messages and recovery procedures follow the operating instructions.
3. ENTER THE NUMBER OF THE USER FILE DEVICE ADDRESS 1. 310 5. B10 2. 320 6. B20 3. 330 7. B30 4. 350	3. Enter the selection number for the user file disk device address. MESSAGE: 2, 3, 5
4. ENTER KEY FILE NUMBER (NORMAL=1)	4. Enter the Key File Number. The Key File Number should always be 1, unless there are multiple key files for a single User File, in which case the Key File Number may be any digit from 1 to 9. In any case it must have been initialized. MESSAGE: 2, 3, 6

5.	ENTER THE NUMBER OF THE KEY FILE DEVICE ADDRESS 1. 310 5. B10 2. 320 6. B20 3. 330 7. B30 4. 350	5.	Enter the selection number for the Key File Device address. MESSAGE: 2, 3, 5
6.	ENTER LAST KEY	6.	Enter the key of the last record in the User File. MESSAGE: 2
7.	TURN ON PRINTER KEY RETURN(EXEC) TO RESUME	7.	Mount paper on printer; key RETURN(EXEC) to resume. Duplicate keys, if any exist, are printed. If the system configuration does not include a printer, ignore this instruction, key RETURN(EXEC). MESSAGE: 2
8.	Record locations and keys are displayed on the screen so that the operator can check the progress of the program.	8.	The system opens the files, sets up the screen display, and creates the Key File. MESSAGE: 7, 8, 9, 10 11, 12, 13, 14 15, 16, 17, 18
9.		9.	The system returns to KFAM subsidiary menu.

<u>Error Messages</u>

	ERROR MESSAGE	EXPLANATION/RECOVERY
2.	RE-ENTER	Too many characters were entered. RECOVERY: Repeat the step, entering not more than the number of characters indicated on the screen.
3.	ERR29	A non-numeric quantity was entered when a numeric quantity was requested. RECOVERY: Reenter numeric quantity.

4. NOT KFAM FILE NAME	The User File name must have an "F" in position 5 and a digit 0-9 in position 6. RECOVERY: Repeat Step 2. Enter correct User File name.
5. INVALID DEVICE ADDRESS	The device address is invalid. RECOVERY: Repeat the step. Enter correct device address selection number.
6. INVALID	The Key File Number may not be 0. RECOVERY: Repeat step 4. Enter a Key File Number 1-9.
7. ERR80	File not found. Either the User File or the Key File specified is not on the device specified. RECOVERY: Rerun the program, making sure the platters containing the User File and Key File are mounted, and that User File name, Key File number, and device addresses are specified correctly.
8. STOP ERROR OPENING FILES	File could not be opened. Possible cause: The program was stopped and restarted after processing had begun. RECOVERY: Rerun INITIALIZE KFAM FILE. Rerun this utility. If the error persists, notify Wang Laboratories, Inc.
9. KEY FILE NOT INITIALIZED (STOP)	Either INITIALIZE KFAM FILE was not run, or an attempt was made to rerun this utility without rerunning INITIALIZE KFAM FILE. RECOVERY: Run INITIALIZE KFAM FILE for this User File and Key File. Rerun this utility.
10. INVALID RECORD FORMAT (STOP)	Record type A, array-type blocking: more than one sector per block, more than 38 fields per record, or not written with correct control bytes. RECOVERY: See "KFAM Specifications." Recreate User File according to specifications for A-type records.

11. NOT BLOCKED AS SPECIFIED	Record type A, array-type blocking: records per block specified incorrectly, or records not written in array format. RECOVERY: See "KFAM Specifications." Recreate User File according to specifications or correct blocking factor in INITIALIZE KFAM FILE. Rerun INITIALIZE KFAM FILE. Rerun this utility.
12. RECORD LENGTH NOT SPECIFIED CORRECTLY (STOP)	Record type A, array-type blocking: record length specified in INITIALIZE KFAM FILE does not equal record length of sample record. RECOVERY: See instructions for INITIALIZE KFAM FILE. Recalculate record length. Rerun INITIALIZE KFAM FILE. Rerun this utility.
13. KEY FIELD OUT OF BOUNDS (STOP)	Record type A, array-type blocking: the key must be wholly contained within one field of the record. RECOVERY: See instructions for INITIALIZE KFAM FILE. Recalculate starting position of key. Rerun INITIALIZE KFAM FILE. Rerun this utility.
14. NUMERIC KEY INVALID (STOP)	Record type A, array-type blocking: the key falls within a numeric field. RECOVERY: See "KFAM Specifications." See instructions for INITIALIZE KFAM FILE. Recalculate starting position of key so that it falls within an alphanumeric field. Rerun INITIALIZE KFAM file. Rerun this utility.
15. PROGRAM ERROR	Should not occur. RECOVERY: Notify Wang Laboratories, Inc.
16. NO SPACE	Not sufficient space for Key File. Possibly last key was entered incorrectly.

	RECOVERY: Run DISK/COPY/REORGANIZE to increase Key File space. INITIALIZE KFAM FILE. Rerun this utility.
17. SYSTEM HANGS	Printer not turned on or not selected manually, or no device 215 in system. RECOVERY: Turn printer on and press "SELECT". If no device 215, this program will not run without modification. (See "KFAM Programming Techniques"- Eliminating the Printer.)
18. LAST KEY NOT FOUND	The program has reached the physical end of the User File without finding the "last key." Last key was entered incorrectly. RECOVERY: Rerun INITIALIZE KFAM FILE. Rerun this utility with correct "last key" entered.

CHAPTER 21
THE KFAM-3 SUBROUTINES

21.1 OVERVIEW OF KFAM-3 SUBROUTINES

The KFAM-3 subroutines are designed to simplify the file access and maintenance operations most frequently performed on files organized by KFAM-3. Included are GOSUB' subroutines to add new records to a file, delete old records, and locate existing records.

A single KFAM-3 file consists of two cataloged disk files, a User File and its Key File. KFAM subroutines never alter the data in the User File. They operate upon the data in the Key File, locate a record, update the Key File whenever a record is to be added or deleted. Their function in relation to the User File is only to set the User File's Current Sector address to the location of the desired record in the User File, and, for blocked records, to pass back to the application program the record location within the sector. This process is initiated when the application program passes a key to the KFAM subroutine in one of the GOSUB' arguments. Upon completion of the KFAM subroutine, the application program must perform the proper operation on the User File.

Just as KFAM-3 does not operate upon the User File, so the application program should never operate directly upon the Key File. All operations involving the Key File, including OPEN and CLOSE, should be accomplished via the KFAM-3 subroutines.

The functions performed by the KFAM-3 subroutines are:

Name	Function
OPEN	Open specified User File and companion Key File.
DELETE	Remove specified key from Key File; set User File Current Sector address to record in User File whose key has been deleted from the Key File.
FINDOLD	Locate specified key in the Key File; set Current Sector address to record in User File with that key.
FINDNEW	Add specified key to Key File; allocate space for a new record in the User File, and set User File Current

	Sector address to the location for the new record.
FINDNEW (HERE)	Add specified key to Key File; set Current Sector address to User File sector where the new record is to be written.
FINDFIRST	Locate record with lowest key in User File; set User File Current Sector address to that sector.
FINDLAST	Locate record with highest key in User File; set User File Current Sector address to that sector.
FINDNEXT	Locate next record in User File in logical key sequence; set User File Current Sector address to that sector.
CLOSE	Close User File and companion Key File.

Programming Procedure

The first step for the programmer is to decide what subroutines will be required for a given program or program module. The utility program BUILD SUBROUTINE MODULE is used to build a cataloged program file containing the desired subroutines.

The subroutines occupy statement lines 200-3075. It is recommended that the user-written application program follows them, beginning at a line number greater than 3075. If necessary, KFAM-3 subroutines can be renumbered taking care that the rules of BASIC be observed for COM and DIM statement location.

KFAM-3 subroutines use Q, T, and V variables and arrays (alpha and numeric) for storage of critical internal pointers. The user-written program should, therefore, strictly avoid the use of a Q, T, or V variable.

Identification of KFAM Files

A User File and Key File can be thought of collectively as a KFAM file. To use a KFAM file, the User File and the Key File must be open simultaneously. Thus, each KFAM file requires two "slots" or "rows" in the processor's Device Table. The 2200 system offers a total of seven Device Table slots, each identified by a "file number" symbol #0-#6. Since each KFAM file uses two file numbers, and there are seven file numbers available, a total of three KFAM files can be open concurrently.

The OPEN subroutine must be used to open a Key File and User File, before any KFAM file access operations can take place. The user-written application program must pass to the OPEN subroutine the file numbers (#0-#6) to be used for the Key File and User File. The application program also passes to OPEN a digit, 1-3, which will be used to identify this KFAM file (User File/Key File) for all other KFAM subroutines. This digit, 1-3, is called the "KFAM ID number." When passed to OPEN, OPEN establishes this number as the single identifier for the pair of cataloged files being opened. In subsequent operations, while these files are open, they are identified collectively simply by passing the KFAM I.D. Number to the desired KFAM subroutine.

The file number for the User File is employed only when a record is to be written to or read from a User File. Then, the file number of the User File (#0 - #6) must be specified in the DATASAVE DC or DATALOAD DC statement. The KFAM I.D. Number is not used. In general, the file number of the Key File is never used, since any reference to the Key File should be via a KFAM subroutine. Note that accidental use of the Key File's file number in place of the User File's file number in a DATASAVE DC statement causes the user data to be written over data in the Key File. This destroys the Key File.

The same User File may be open concurrently with more than one Key File, but each such pair (User File/Key File) must have a different KFAM I.D. Number, and the User File must be referenced by a different file number in each case.

Key File Recovery Information

The KFAM maintenance utilities include a program called KEY FILE RECOVERY. The purpose of this program is to reconstruct a Key File from an existing User File, in the event that the Key File is accidentally destroyed. Unlike the program KEY FILE CREATION, it does not require that the key of the last physical record in the User File be known. However, it does require that all user-written application programs operating on the file adhere to two conventions:

1. All DELETE'ed records in the User File must have hex FF as the first byte of the key. This means that after a program calls the DELETE subroutine, it must then save hex FF into the first byte of the record's key.

2. If FINDNEW is to be executed on a file, the RECOVERY option must be included in the OPEN, FINDNEW, and CLOSE subroutines, and the file must be closed with the CLOSE subroutine at the conclusion of operations on the file. (The RECOVERY OPTION is offered in BUILD SUBROUTINE MODULE as one of several optional additional capabilities for the selected subroutine.)

21.2 BUILD SUBROUTINE MODULE

BUILD SUBROUTINE MODULE builds a module of selected KFAM subroutines for use in an application program. It allows the programmer to include in an application program module only those subroutines and subroutine capabilities which actually are used in the application program module, and, thereby, keeps to a minimum the amount of memory occupied by KFAM subroutines.

Each KFAM subroutine may be included or excluded independently of the others. After selecting the desired subroutines, the capability to operate on multiple files (two or three KFAM files open at the same time) may be included or excluded for the chosen subroutines. If only one KFAM file is to be open at any one time, then excluding the multiple file capability further reduces memory requirements.

Finally, the RECOVERY OPTION is offered. Whenever FINDNEW is executed, the RECOVERY OPTION must be included in the OPEN, FINDNEW, and CLOSE subroutines, if the ability to execute KEY FILE RECOVERY is desired. If the ability to use KEY FILE RECOVERY is not desired, the RECOVERY OPTION need not be included.

The OPEN subroutine, when chosen for a module, includes all the common variables needed for subroutine operation, except those for FINDNEW and FINDNEW(HERE). The utility separately asks whether the variables for FINDNEW and FINDNEW(HERE) are to be included in the OPEN subroutine. These variables must be included in the OPEN subroutine, if it is to be used to OPEN a file on which, subsequently, FINDNEW or FINDNEW(HERE) is executed.

> NOTE:
>
> All subroutine modules operating on a given open KFAM file must include the same options. (Options are OPEN FOR FINDNEW, MULTIPLE FILES, RECOVERY.) For example, if RECOVERY is chosen, it must be chosen for the module that contains OPEN, any processing modules, and the module that contains CLOSE.

To illustrate how BUILD SUBROUTINE MODULE might be used to maximize available memory, suppose that a file is to be "purged" in key sequence, deleting all obsolete records. The subroutines which are needed are:

```
OPEN
FINDFIRST
FINDNEXT
DELETE
```

These subroutines require 3370 bytes, approximately, if loaded at the same time. Suppose, further, that there is not enough memory available to accomplish the processing and include all of these subroutines. Two separate modules could be built, the first to contain OPEN and FINDFIRST, the second to contain the rest. The first two of these subroutine modules would then become part of a start-up module in the application program which would open the KFAM file, perform other preliminary processing, and then FINDFIRST. This start-up module would then overlay the processing module, clearing the OPEN and FINDFIRST subroutines as it does so. The processing module would contain FINDNEXT and DELETE. Memory overhead at any one time, due to subroutines, is thereby reduced from 3370 bytes to:

$$\left.\begin{array}{l}\text{OPEN}\\\text{FINDFIRST}\end{array}\right\}\ \text{2975 bytes}$$

$$\left.\begin{array}{l}\text{FINDNEXT}\\\text{DELETE}\end{array}\right\}\ \text{2380 bytes}$$

NOTE:

There are two modules of KFAM-3 subroutines included with the KFAM-3 system. If desired, the programmer may simply use one of these modules rather than building a custom module with the BUILD SUBROUTINE MODULE utility. The modules are as follows:

MODULE NAME	INCLUDES
KFAM0003	All subroutines and subroutine options.
KFAM0103	All subroutines and options except: DELETE, FINDNEW, FINDNEW(HERE), RECOVERY, OPEN FOR FINDNEW.

Operating Instructions - BUILD SUBROUTINE MODULE

	DISPLAY		INSTRUCTIONS
1.		1.	From KFAM subsidiary menu access its "BUILD SUBROUTINE MODULE" utility via the specified Special Function key.
2.	ENTER THE NAME OF PROGRAM TO BE GENERATED?	2.	Enter the name of the program file which is to contain the selected subroutines, maximum of 8 characters. If a file of the same name is not already cataloged, the utility allocates just enough space for the selected subroutines. If a file of the same name is already cataloged, that file is used for the output program and overwritten. If the file is a data file, it is changed to a program file.

CAUTION:

Before entering a name, ensure that the name entered is not the name of a valuable data file or program file. If a file already exists with the same name, its contents are destroyed by this utility.

			If extra space is desired in the output file, it should be cataloged in advance as a data file. For example, DATASAVE DC OPEN T/B10, 50, "FILE"
3.	ENTER THE NO. OF THE OUTPUT PROGRAM DEVICE? 1. 310 5. B10 2. 320 6. B20 3. 330 7. B30 4. 350	3.	Enter the selection number for the device address at which the subroutines are to be saved. ERROR MESSAGE: 1,2,3
4.	230 OPEN (Y OR N)? OPEN FOR FINDNEW (Y OR N)? 231 DELETE (Y OR N)? 232 FINDOLD (Y OR N)? 233 FINDNEW (Y OR N)? 234 FINDNEW (HERE) (Y OR N)? 235 FINDFIRST (Y OR N)? 236 FINDLAST (Y OR N)? 237 FINDNEXT (Y OR N)? 239 CLOSE (Y OR N)? MULTIPLE FILES (Y OR N)? RECOVERY OPTION (Y OR N)?	4.	Each of the listed prompts is displayed sequentially. For each subroutine or subroutine capability enter Y to include it in the output module; to exclude it, enter N. ERROR MESSAGE: 1
5.	OK TO PROCEED?	5.	Enter Y to accept selected subroutines, or N to return to step 4. ERROR MESSAGE: 1
6.	PHASE 2 - BUILDING PROGRAM NAME	6.	The output module is generated. ERROR MESSAGE: 4, 5, 6, 7
		7.	The system returns to the KFAM-3 subsidiary menu.

Error Messages

ERROR MESSAGE	EXPLANATION/RECOVERY
1. RE-ENTER	1. Too many characters were entered, or an invalid character was entered in response to a "yes" (Y) or "no" (N) question. RECOVERY: Repeat the step. Re-enter the data.

2. INVALID DEVICE ADDRESS	2. The numbers 1-7 may be used to specify a device address, according to the table of device addresses displayed. RECOVERY: Repeat the step. Re-enter the data.
3. ERR29	3. A non-numeric quantity was entered when a numeric quantity was requested. RECOVERY: Repeat the step. Enter a number.
4. INVALID DELIMITER (STOP)	4. Errors 4,5, and 6 are hardware or software errors. They should not occur. RECOVERY: Rerun the program. If the error persists, notify Wang Laboratories.
5. OUTPUT PROGRAM SPACE EXCEEDED (STOP)	
6. SYSTEM ERROR (STOP)	
7. NO ROOM ON DISK FOR OUTPUT PROGRAM (STOP)	7. There is not room enough on the disk for the output program to be cataloged. RECOVERY: Rerun the program, with an output disk with more free space (25 sectors maximum requirement).

21.3 CALLING THE SUBROUTINES

Dummy Variable Names

In defining the argument lists for the subroutines, certain standard dummy variable names are used. These dummy names are used only to describe the general forms of the respective GOSUB' statements. In the actual program, the programmer may use any value or expression valid for use in a GOSUB' statement. Zeros in the general statement represent parameters which are not used by KFAM-3. They should be included, as zeros in the GOSUB' statement.

For example, the general statement:

```
GOSUB'232 (I, 0, A$)
```

may be written as:

```
GOSUB'232(I,0,K$)
GOSUB'232(2,0,"A48-3029")
GOSUB'232(F1+1,0,STR(P1$,7,8))
etc.
```

The dummy variable names for KFAM-3, and their meanings, are as follows:

Dummy Variable	Meaning
I	KFAM I.D. Number (1, 2, or 3).
K	File number assigned to the Key File (#0-#6).
U	File number assigned to the User File (#0-#6).
F	Key File number (1-9), specified as the 6th character in the Key File name, as assigned in INITIALIZE KFAM FILE.
A$	The record key (alphanumeric).
N$	User File name.

Return Codes

Upon returning to the main line program from the subroutines, the variables Q and Q$ contain the following information:

Q returns the record position indicator for blocked files (i.e., files with more than one record per sector). The record position indicator is a numeric value which specifies the position of a desired record within a block. For example, if Q=2, the key passed to the subroutine specifies the second record in the block. For unblocked records Q is returned as 1, and may be ignored.

Q is not defined following the OPEN or CLOSE subroutines.

Q$ contains the completion return code. It indicates the result of the particular operation. The possible values of Q$, and their meanings, are as follows:

Q$ Value	Meaning
blank	The subroutine execution was OK.
D	Duplicate key (attempting to add a duplicate key to the file). The Key File is unchanged.
E	End of file (FINDNEXT only).
N	Key not found.
S	No more space, either for the User File or the Key File, or 8 levels of index have been exhausted attempting to add a record to the file. The Key File is unchanged. (FINDNEW and FINDNEW(HERE) only.)

X	Improper call to a KFAM subroutine (argument values erroneous, etc.).

If Q$ is anything other than blank, the User File Current Sector address parameter is undefined, and the value of Q is undefined.

Immediately upon return from any of the subroutines, the main line program should check Q$ for possible error indications.

The system assumes there are no programming errors in the main line program. The KFAM Subroutines can perform improperly, and can destroy a file, if the parameters supplied by the main line program are erroneous. Therefore, during the testing stage, it is recommended that the user keep a backup file so that test data can be recovered in the event that it is destroyed.

The subroutines check data errors, and the kind of errors likely to occur during normal operation, such as duplicate key, key not found, or no more space. The following errors, which are programming errors, may or may not be caught by the subroutines:

Error	Q$ Value, or ERR Code
KFAM I.D. Number not an integer between 1 and 3.	X ERR 18
KFAM I.D. Number is the same as I.D. Number for a file already open.	X
File to be opened is already open.	X
Individual file numbers not integers between 0 and 6.	ERR 18 ERR 41
Individual file number is duplicate of another file number.	X
File name not in proper format, with 5th byte="F" and 6th byte a 0 (zero).	ERR 78 ERR 80
Key File number not an integer from 1 to 9.	ERR 56
File to be accessed has not been opened.	X
SELECT statements and file numbers do not actually correspond.	none

File names are not correct, or do not exist on the disk platters specified.	ERR 78

21.4 OPEN

The OPEN subroutine is used to open a User File and its companion Key File. OPEN must be executed prior to execution of any other KFAM-3 subroutine. In the OPEN subroutine, a pair of DATALOAD DC OPEN statements are executed to open the named User File and its companion Key File. Specified file numbers are assigned to each file. OPEN also assigns a specified KFAM I.D. Number to the pair of files. To call the OPEN subroutine you must write two statements of the following general form:

```
SELECT #U XXX, #K YYY
GOSUB' 230 (I,K.U,F,N$)
```

For the SELECT Statement

"#U" is the file number to be associated with the User File; "U" can be a number from 1 to 6. "#U" must be used in all subsequent DATASAVE DC or DATALOAD DC statements to reference the User File.

"XXX" is the device address of the platter on which the User File is stored.

"#K" is the file number to be associated with the Key File; "K" can be a number from 1 to 6.

"YYY" is the device address of the platter on which the Key File is stored.

For The GOSUB' Statement

"I" is the KFAM I.D. Number to be associated with the newly opened pair of files and must be used to reference the KFAM file in subsequent KFAM subroutines. "I" can be a number from 1 to 3.

"K" is the file number to be assigned to the Key File (see "#K" above).

"U" is the file number to be assigned to the User File (see "#U" above).

"F" is the Key File number (the sixth character in the Key File name). It may be an integer from 1 to 9, but normally is 1.

"N$" is the name of the User File to be opened. The Key File name need not be specified; it is built from the User File name and the Key File number by KFAM itself.

Return Codes for OPEN

Q$ = " " (space) if the subroutine execution was O.K.

Q$ = "X" for an improper call (i.e., one of the arguments in the GOSUB' 230 argument list was incorrect, or the file is already open). Note, if a file is already open, or the KFAM I.D. number is already in use, OPEN returns Q$ = "X".

21.5 DELETE

The DELETE subroutine deletes from the Key File a specified key and its associated record location pointer. The Current Sector address for the User File is set to the location of the record whose key has been deleted, and for blocked records the variable Q is set to the record position within the sector. The record itself, in the User File, is not altered or removed. Thus, although the record is not physically removed from the User File, its key entry is removed from the Key File, and the record can no longer be accessed through KFAM.

The calling sequence for DELETE is:

```
GOSUB' 231 (I, 0, A$)
```

"I" is the KFAM I.D. Number, assigned to the file in an OPEN subroutine.

"A$" is the key of the record that is to be deleted from the file.

DELETE Return Codes

Q$ = "N" if the key passed cannot be found in the Key File.
Q$ = "X" for an improper call.
Q$ = " " ("space") if the subroutine executed properly.

After calling a DELETE subroutine and checking for its successful completion, the application program should flag the DELETED record in the User File by changing the first character of the deleted record's key to hex FF. For unblocked files this can be done as follows:

Suppose:

```
    DIM A$15, H(4,4), J(6)
```
and
```
    DATA SAVE DC #1, A$, H(), J()
```

define a type "N" record where A$ is the key field.

The DELETE-and-flag operation might look like this:

```
4060 GOSUB' 231 (1, 0, A$): REM DELETE
4070 IF Q$<>" " THEN 6000:REM ERROR?
4080 DATA LOAD DC #1, A$, H(), J()
4090 STR(A$,1,1)=HEX(FF):REM HEX(FF) IN 1ST BYTE OF KEY
4100 DBACKSPACE #1,1S:REM RECORDS ARE 1 SECTOR LONG
4110 DATA SAVE DC #1,A$,H(),J()
.
.
.
6000 STOP "DELETE UNSUCCESSFUL"
```

Instead of flagging deleted records, the space in the User File can be reused; however, this normally requires special techniques together with the use of FINDNEW(HERE). For information on these techniques see Chapter 29.

21.6 FINDOLD

The FINDOLD subroutine is used to locate a desired record in the User File. Following subroutine execution, the Current Sector address for the User File is set to the address of the record whose key was passed to the subroutine. For blocked records, variable Q is set to the record position within the sector. The record can then be read with a DATALOAD DC statement. The calling sequence is:

```
GOSUB' 232 (I, 0, A$)
```

"I" is the KFAM I.D. Number assigned to the file in the OPEN subroutine.

"A$" is the key of the record being sought.

FINDOLD Return Codes

Q$ = "N" if the specified key is not located in the Key File.
Q$ = "X" for an improper call.
Q$ = " " ("space") if the key was located without difficulty.

21.7 FINDNEW

The FINDNEW subroutine is used to enter a new key in the Key File and to find a location for the new record in the User File. FINDNEW enters the key passed to it in the Key File, then sets the Current Sector address for the User File to the next sequential sector available for writing a new record. For blocked records, variable Q is set to the record position within the sector. The calling sequence is:

```
GOSUB' 233 (I,0,A$,0)
```

"I" is the KFAM I.D. Number, assigned to the file in an OPEN subroutine.

"A$" is the new key to be entered in the Key File.

FINDNEW Return Codes

Q$ = "D" if the key specified is a duplicate of one already in the Key File.
Q$ = "S" if there is no space in the User File for another record, or in the Key File for another key entry, or 8 index levels have been exhausted.
Q$ = "X" for an improper call.
Q$ = " " ("space") if the key was entered without difficulty.

The following example illustrates the procedure for adding a record to a type A blocked file following FINDNEW. Note the test on Q before the DATA SAVE.

```
4100       INPUT "KEY FIELD", A$          :REM OPERATOR ENTERS KEY
4120       GOSUB '233 (1,0,A$,0)          :REM FINDNEW
4130    REM TEST COMPLETION CODE
4140       IF Q$ = "D" THEN 5010          :REM DUPLICATE KEY?
4150       IF Q$ = "S" THEN 5050          :REM FILE FULL?
4160       IF Q$ <> " " THEN 5060         :REM ERROR?
4170    REM NEW BLOCK OR OLD?
4180       IF Q = 1 THEN 4220  :REM FIRST RECORD IN NEW BLOCK?
4185    REM READ EXISTING RECORDS IN BLOCK
4190       DATA LOAD DC #2, A$(),B$(),C(),D()
4200       DBACKSPACE #2, 1 S  :REM BACKSPACE AFTER DATALOAD
4210    REM ASSIGN RECORD VALUES TO PROPER ARRAY ELEMENTS
4220       A$(Q) = A$
4230       INPUT "SECOND FIELD", B$(Q)
4240       INPUT "THIRD FIELD", C(Q)
4250       INPUT "FOURTH FIELD", D(Q)
4260    REM SAVE BLOCK IN USER FILE
4270       DATA SAVE DC #2, A$(),B$(),C(),D()
  .
  .
  .
5000    REM ERROR MESSAGES
5010       STOP "KEY ALREADY EXISTS"
5050       STOP "KEY FILE OR USER FILE IS FULL"
5060       STOP "FINDNEW ERROR"
```

21.8 FINDNEW(HERE)

The FINDNEW(HERE) subroutine is a specialized routine whose primary use is in changing the keys of existing keyed records. It can only be used following the DELETE subroutine. (To get around this rule, see Chapter 29.) Once DELETE has removed the old key from the Key File, FINDNEW(HERE) enters the new key, along with the location of the deleted record, in the Key File. The Current Sector address is unchanged. For blocked records, the variable Q is set to the record position within the block.

The difference between FINDNEW and FINDNEW(HERE) is that FINDNEW makes available the next available free space for the new record, whereas FINDNEW (HERE) enables the user to use the space occupied by a DELETED record.

The calling sequence is:

```
GOSUB' 234 (I,0,A$,0)
```

The FINDNEW(HERE) argument list is identical to the argument list for FINDNEW (see FINDNEW).

FINDNEW(HERE) Return Codes

Q$ =	"X" for an improper call.
Q$ =	"D" if the key specified is a duplicate of a key already in the Key File.
Q$ =	"S" if there is no space in the Key File for another entry, or if 8 index levels have been exhausted.
Q$ =	" " (space) if the subroutine executed properly.

The following example illustrates the use of FINDNEW(HERE) following DELETE:

```
5000 GOSUB '231 (1,0,"ABCD") :REM DELETE "ABCD" FROM KEY FILE
5010 IF Q$ = "X" THEN 5130
5040 IF Q$ = "N" THEN 5150
5050 GOSUB '234 (1,0,"EFGH",0) :REM INSERT "EFGH" IN KEY FILE
5060 IF Q$ = "X" THEN 5140
5070 IF Q$ = "D" THEN 5160
5075 IF Q$="S" THEN 5170
5080 DATALOAD DC #2,A$,B$,C$,N
5090 A$ = "EFGH" :REM CHANGE KEY TO "EFGH"
5100 DBACKSPACE #2, 1S
5110 DATASAVE DC #2,A$,B$,C$,N
5115 GOSUB'239(1) :REM CLOSE FILES
5120 END
5130 STOP "ERROR IN 'DELETE' CALLING SEQUENCE"
5140 STOP "ERROR IN 'FINDNEW(HERE)' CALLING SEQUENCE"
5150 STOP "KEY NOT FOUND"
5160 STOP "DUPLICATE KEY"
5170 STOP "NO SPACE"
```

21.9 FINDFIRST

The FINDFIRST subroutine sets the Current Sector address for the User File to the first record in logical key sequence. For blocked records, variable Q is set to the record position within the sector. A DATALOAD DC statement can be used after FINDFIRST to read the record. The calling sequence is:

```
GOSUB' 235 (I)
```

"I" is the KFAM I.D. Number, assigned to the file in an OPEN subroutine.

FINDFIRST Return Codes

Q$ = "N" if the User File contains no records.
Q$ = "X" for an improper call.
Q$ = " " (space) if the subroutine executed properly.

21.10 FINDLAST

The FINDLAST subroutine sets the Current Sector address for the User File to the last record in logical key sequence. For blocked records, the variable Q is set to the record position within the sector. A DATALOAD DC statement can be executed following FINDLAST to read the record. The calling sequence is:

```
GOSUB' 236 (I)
```

"I" is the KFAM I.D. Number assigned to the file in an OPEN subroutine.

FINDLAST Return Codes

Q$ = "N" for a null file.
Q$ = "X" for an improper call.
Q$ = " " (space) if the subroutine executes normally.

21.11 FINDNEXT

The FINDNEXT subroutine sets the Current Sector address for the User File to the record immediately following (in logical key sequence) the last record accessed by KFAM. For blocked records, the variable Q is set to the position of the record within the sector. A DATALOAD DC statement can be executed following FINDNEXT to read the record. FINDNEXT is useful for processing files in key sequence. The calling statement is:

```
GOSUB' 237 (I)
```

"I" is the KFAM I.D. Number assigned to the file in an OPEN subroutine.

FINDNEXT Return Codes

Q$ = "X" for an improper call.
Q$ = "E" if the previous reference was to the last record in logical key sequence.

Otherwise, Q$ = " " (space).

NOTE:

FINDNEXT cannot be executed as the first subroutine following an OPEN routine. Also, FINDNEXT cannot normally be executed immediately following any subroutine which returned an error code (Q$ other than blank). Otherwise FINDNEXT will locate the next sequential key, following any subroutine.

If FINDNEXT is executed following a FINDOLD that returned Q$ = "N" (not found), then FINDNEXT finds the next sequential record which would follow the record sought in the FINDOLD, were that record actually in the file.

21.12 CLOSE

The CLOSE subroutine is used to close a currently open User File and its companion Key File. The KFAM I.D. Number assigned to a closed file can then be reassigned to another file in an OPEN routine. Similarly, the file numbers assigned to a User File and Key File can be reassigned in an OPEN routine once the files have been closed. The CLOSE subroutine also saves certain critical information for the KEY FILE RECOVERY utility, provided that the RECOVERY OPTION was included during BUILD SUBROUTINE MODULE execution. The calling sequence is:

```
GOSUB' 239 (I)
```

"I" is the KFAM I.D. Number assigned to the file in an OPEN routine. Following execution of the CLOSE routine, this number can no longer be used to access the User File and its associated Key File.

CLOSE Return Codes

Q$ = "X" for an improper call.
Otherwise, Q$ = " " (space).

CHAPTER 22
THE KFAM-4 SUBROUTINES

22.1 PROGRAMMING WITH THE KFAM-4 SUBROUTINES

22.1.1 Differences Between KFAM-4 Subroutines and KFAM-3 Subroutines

The KFAM-4 subroutines perform the same functions as those of KFAM-3. Before using KFAM-4, the programmer should become familiar with the use of KFAM-3. Except as noted in this chapter, the elementary KFAM subroutine programming conventions and procedures described in Section 21.1, apply to KFAM-4 as well as KFAM-3.

The subroutine RELEASE is added in KFAM-4 to turn off a protect flag on a record. The principal differences between the other KFAM-3 and KFAM-4 subroutines fall into four categories:

1. Two additional arguments to be supplied in the GOSUB' statements which call the subroutines.

2. Two additional return codes which indicate that the subroutine operation could not be carried out, due to a protective procedure invoked by another CPU.

3. A completely different procedure for SELECTing the Key File device address.

4. The CLOSE subroutine must be executed at the conclusion of file operations.

Additional Arguments

In the OPEN subroutine a class-of-access argument is required. Symbolized by the dummy variable C$, an argument value of "A" means that any other CPU's may open the file while this CPU has the file open. An argument value of "X" means that this CPU seeks exclusive access to the file.

In general, exclusive access should not be sought, except for those operations which must be carried out on an entire file with the entire file protected from alteration by another CPU. For example, printing an end of period status report might be an appropriate use of exclusive access, when it is important that the report reflect the file status at one particular time. However, in addition to this use, exclusive access may be sought whenever maximum access speed is required.

A protect-flag argument is required for the subroutines.

DELETE
FINDOLD
FINDNEW
FINDNEW(HERE)
FINDFIRST
FINDLAST
FINDNEXT

The dummy variable P is used to represent this argument. If an argument value of 1 is passed to the subroutine, then the protect flag is turned on for the record or block or records accessed by this subroutine call. As long as the protect flag is on for a record or block of records, that record or block cannot be accessed by any CPU other than the CPU which turned on the protect flag. All other records in the file may, however, be accessed. If an argument value of 0 is passed to the subroutine, then the protect flag is not turned on.

Once turned on, a protect flag is automatically turned off as soon as the CPU that turned on the flag executes another KFAM-4 subroutine on the same file. The subroutine RELEASE can be used if there may be a long delay before another subroutine is executed. RELEASE simply turns off the protect flag.

Additional Return Codes

As a result of the OPEN subroutine, Q$ may be returned with the value "C". This indicates access-class conflict. Either this CPU is seeking exclusive access when another CPU has the file open, or this CPU seeks access when another CPU has exclusive access.

After executing any of the subroutines, except OPEN, CLOSE, and RELEASE, Q$ may be returned with the value "B". This is the busy signal. It means that the User File record which was sought has had its protect flag turned on by another CPU. The User File's Current Sector address is unchanged.

SELECT Procedure

In KFAM-3 the user program executes a SELECT statement for the key file device address as well as the user file device address, immediately prior to calling the OPEN subroutine. The KFAM-3 subroutines themselves never have to SELECT a device address. However, this situation is changed for KFAM-4. A KFAM-4 subroutine must SELECT a hog mode address for the key file, hog the disk during its execution, and then SELECT the non-hog mode address and leave hog mode before returning control to the user program.

The KFAM-4 subroutines select hog mode and non-hog mode by calling short subroutines that reside in the user program. These subroutines must be written by the application programmer, and included in every module which accesses a KFAM-4 subroutine.

For example, suppose that a program accesses just one KFAM file, that the Key File is at device address 320, and that file number #2 is used for the Key File. The following two subroutines must be included somewhere in the user program.

```
4000  REM SELECT HOG MODE
4010  DEFFN' 210 (T6)
4020  SELECT #2 3A0
4030  RETURN
4040  REM SELECT NON-HOG MODE
4050  DEFFN' 211 (T6)
4060  SELECT #2 320
4070  RETURN
```

Shortly after it is called, the KFAM-4 subroutine calls DEFFN'210 to select hog mode for its operations on the Key File. When the KFAM-4 subroutine is nearly complete, it calls DEFFN'211 to select non-hog mode. After selecting non-hog mode, the subroutine executes a disk statement so that the hog mode is actually released before control is passed back to the user program.

Thus, the SELECT subroutines are the reverse of the normal KFAM subroutines. In general, the user program calls a KFAM subroutine; however, these particular subroutines are written by the user, and called by the KFAM-4 subroutines.

Notice in the SELECT subroutines shown above, that the variable T6 is assigned a value by the GOSUB' statement which calls the subroutine. This variable must always appear in the DEFFN' statements of the SELECT subroutines; however, the value of this variable becomes significant only if several KFAM files are to be accessed by the user program. The variable T6 is assigned the KFAM I.D. Number when the SELECT subroutines are called by a KFAM subroutine. It is used as follows.

Suppose that there are three KFAM files (Key File/User File) to be accessed by the user program. The pertinent information is

KFAM I.D. NO.	Key File File Number	Key File Device Address
1	#2	B20
2	#4	B20
3	#6	320

The SELECT subroutines should be written as follows:

```
4000 REM SELECT HOG MODE
4010 DEFFN'210 (T6)
4020 ON T6 GOTO 4030, 4040, 4050
4030 SELECT #2 BA0: RETURN
4040 SELECT #4 BA0: RETURN
4050 SELECT #6 3A0: RETURN

4060 REM SELECT NON-HOG MODE
4070 DEFFN'211 (T6)
4080 ON T6 GOTO 4090, 4100, 4110
4090 SELECT #2 B20: RETURN
4100 SELECT #4 B20: RETURN
4110 SELECT #6 320: RETURN
```

Notice that T6, which is the KFAM I.D. number, is used to control which of the Key Files is SELECTed for hog (or non-hog) mode.

The KFAM-4 subroutines cannot operate successfully without the SELECT subroutines included in the application program.

The CLOSE Subroutine

When the KFAM-4 OPEN subroutine is executed, the CPU opening the file is assigned to a slot in the Access Table of the Key File's KDR record. It "occupies" this slot until it executes the CLOSE subroutine. If a CPU opens a file and fails to execute the CLOSE subroutine at the conclusion of its file operations, the Access Table retains the open file notation for that slot. This false notation in the Access Table prevents the file from being opened in the exclusive mode. Four such false notations, or one false "exclusive access" notation, prevent the file from being opened at all, by any CPU. It is therefore imperative that the CLOSE subroutine be executed at the conclusion of operations on a KFAM-4 file.

22.1.2 Procedural Notes For Programming With KFAM-4 Subroutines

In general the programming procedures used with KFAM-4 subroutines are not unlike those used with KFAM-3. In addition to the differences in calling sequences, return codes, and SELECT procedures, the following difference should be noted.

1. The protect flag should be set for a record if a DATA SAVE is to be executed on the record. (The protect flag is set by specifying a 1 for the dummy variable "P" in the GOSUB' argument lists of the KFAM-4 subroutines.) Updating records, adding new records, and flagging deleted records all require that a DATA SAVE DC be executed; therefore, the protect flag should be set for all these operations. Operations which only execute DATA LOAD DC on the record should not set the protect flag.

2. Application programs should not attempt to hog the disk continuously. The user file address should be a normal disk address, not a hog mode address.

3. The application program must check for a return code of Q$ = "B", indicating that the record (or block of records) sought is protected. On a Q$ = "B" condition the application program can simply reexecute the subroutine. For example,

```
4250    GOSUB' 237 (2,1):REM FINDNEXT
4260    IF Q$ = "B" THEN 4250: REM KEEP TRYING
```

4. In general a file should not be opened in exclusive mode, except in either of the following circumstances:

   a) The operation on the file must take place with the file status fixed as of the beginning of the operation. For example, printing a report as of the end of an accounting period.

   b) Maximum file access speed is needed. (When a file is open in exclusive mode, the KFAM-4 subroutines can search the key file without first reading and writing the KDR record. This allows subroutine execution speed to approximate that of KFAM-3.)

5. Application programs must never write trailer records of any kind into the user file. The RECOVERY OPTION should be used to provide recovery capability for the possibility of accidental Key File destruction. In general, application programs must never make any assumptions about the status of user file sectors other than those specifically returned by a subroutine. For example under KFAM-4, it is possible for the next sequential record location, after that returned by a FINDNEW, to be already occupied by a live record, written by another CPU.

6. Application programs should execute the RELEASE subroutine if the previous call set the protect flag, and there may be a long delay before the next KFAM-4 subroutine call on that file.

   You may wish to consider any keyboard entry operation as involving a long delay, and execute RELEASE prior to the keyboard entry. Alternatively, a Special Function Key subroutine that executes RELEASE may be made available during all keyboard entry operations. The operator would then be instructed to depress the specified Special Function Key if there is any delay prior to responding.

7. The CLOSE subroutine must be executed at the conclusion of operations on a KFAM-4 file. The operator should always have available a procedure for CLOSING the file in the event of program malfunctions, or other disaster. (If the CPU power is turned off without CLOSING the file, the Access Table retains a notation for a "phantom" CPU; the RESET ACCESS TABLE utility must be run.) A Special Function key subroutine such as DEFFN' 31 might be made available to CLOSE a file at any time.

22.2 BUILD SUBROUTINE MODULE (KFAM-4)

BUILD SUBROUTINE MODULE builds a module of selected KFAM subroutines for use in an application program. It allows the programmer to include in an application program module only those subroutines and subroutine capabilities which actually are used in the application program module, and, thereby, keeps to a minimum the amount of memory occupied by KFAM-4 subroutines.

Each KFAM subroutine may be included or excluded independently of the others. After selecting the desired subroutines, the capability to operate on multiple files (two or three KFAM-4 files open at the same time) may be included or excluded for the chosen subroutines. If only one KFAM-4 file is to be open at any one time, then excluding the multiple file capability further reduces memory requirements. Finally, the RECOVERY OPTION is offered. It must be included in order for the KEY FILE RECOVERY program to execute successfully in the event of accidental Key File destruction. If the ability to use KEY FILE RECOVERY is not desired, the RECOVERY OPTION need not be included.

The OPEN subroutine, when chosen for a module, includes all the common variables needed for subroutine operation except those for FINDNEW and FINDNEW(HERE). The utility separately asks whether the variables *for FINDNEW* and FINDNEW(HERE) are to be included in the OPEN subroutine. These variables must be included in the OPEN subroutine, if it is to be used to OPEN a file on which, subsequently, FINDNEW or FINDNEW(HERE) is executed.

NOTE:

All subroutine modules operating on a given open KFAM file must include the same options. (Options are OPEN FOR FINDNEW, MULTIPLE FILES, RECOVERY.) For example, if RECOVERY is chosen, it must be chosen for the module that contains OPEN, any processing modules, and the module that contains CLOSE.

To illustrate how BUILD SUBROUTINE MODULE might be used, to maximize available memory, suppose that a file is to be "purged" in key sequence, deleting all obsolete records. The subroutines which are needed are:

OPEN
FINDFIRST
FINDNEXT
DELETE
CLOSE

These subroutines require 4406 bytes, approximately, if loaded at the same time. Suppose, further, that there is not enough memory available to accomplish the processing, and include all of these subroutines. Two separate modules could be built, the first to contain OPEN and FINDFIRST only, the second to contain the rest. The first of these subroutine modules would then become part of a start-up module in the application program which would open the KFAM file and perform other preliminary processing. This start-up module would then overlay the processing module, clearing the OPEN and FINDFIRST subroutines as it does so. The processing module would contain the second group of subroutines: FINDNEXT, DELETE and CLOSE. Memory overhead at any one time, due to subroutines, is thereby reduced from 4406 bytes to

OPEN FINDFIRST	3830 bytes
FINDNEXT DELETE CLOSE	3255 bytes

Note 1:

There are two modules of KFAM-4 subroutines included with the KFAM-4 system. If desired, the programmer may simply use one of these modules rather than build a custom module with the BUILD SUBROUTINE MODULE utility. The modules are as follows:

MODULE NAME	INCLUDES
KFAM0004	All subroutines and subroutine options.
KFAM0104	All subroutines and options except: DELETE, FINDNEW, FINDNEW(HERE), RECOVERY, OPEN FOR FINDNEW.

NOTE 2:

BUILD SUBROUTINE MODULE selects hog mode for the output disk device. To execute it in non-hog mode, or to execute it at a non-multiplexed disk drive, key:

```
M$ = "X" (EXEC)
```

at KFAM-4 utilities menu, prior to accessing the utility.

Operating Instructions - BUILD SUBROUTINE MODULE KFAM-4

DISPLAY	INSTRUCTIONS
1.	1. From KFAM-4 subsidiary menu access its "BUILD SUBROUTINE MODULE" utility via the specified Special Function key.
2. ENTER THE NAME OF PROGRAM TO BE GENERATED? ?--------	2. Enter the name of the program file which is to contain the selected subroutines, maximum of 8 characters.

If a file of the same name is not already cataloged, the utility allocates just enough space for the selected subroutines.

If a file of the same name is already cataloged, that file is used for the output program and overwritten. If the file is a data file, it is changed to a program file.

> CAUTION:
>
> Before entering a name, ensure that the name entered is not the name of a valuable data file or program file. If a file already exists with the same name, its contents are destroyed by this utility.

If extra space is desired in the output file, it should be cataloged in advance as a data file. For example, DATASAVE DC OPEN T/B10, 50, "FILE"

3. ENTER THE NO. OF THE OUTPUT PROGRAM DEVICE?

```
1. 310      5. B10
2. 320      6. B20
3. 330      7. B30
4. 350
```

3. Enter the selection number for the device address at which the selected subroutines are to be saved.

> NOTE:
>
> The utility operates in hog mode on the output disk drive unless M$ is set to "X".

ERROR MESSAGE: 1,2,3

4.
```
230 OPEN (Y OR N)?
OPEN FOR FINDNEW (Y OR N)?
231 DELETE (Y OR N)?
232 FINDOLD (Y OR N)?
233 FINDNEW (Y OR N)?
234 FINDNEW (HERE) (Y OR N)?
235 FINDFIRST (Y OR N)?
236 FINDLAST (Y OR N)?
239 CLOSE (Y OR N)?
```

4. Each of the listed prompts is displayed sequentially. For each subroutine or subroutine capability enter Y to include it in the output module; otherwise, enter N.

237 FINDNEXT (Y OR N)? 238 RELEASE (Y OR N)? MULTIPLE FILES (Y OR N)? RECOVERY OPTION (Y OR N)?	ERROR MESSAGE: 1
5. OK TO PROCEED? (Y OR N)	5. Enter Y to accept selected subroutines, or N to return to step 4. ERROR MESSAGE: 1
6. PHASE 2 - BUILDING PROGRAM NAME	6. The output module is generated. ERROR MESSAGE: 4, 5, 6, 7
	7. The system returns to the KFAM subsidiary menu.

Error Messages

ERROR MESSAGE	EXPLANATION/RECOVERY
1. RE-ENTER	1. Too many characters were entered, or an invalid character was entered in response to a "yes" (Y) or "no" (N) question. RECOVERY: Repeat the step. Re-enter the data.
2. INVALID DEVICE ADDRESS	2. The numbers 1-7 may be used to specify a device address, according to the table of device addresses displayed. RECOVERY: Repeat the step. Re-enter the data.
3. ERR29	3. A non-numeric quantity was entered when a numeric quantity was requested. RECOVERY: Repeat the step. Enter a number.
4. INVALID DELIMITER (STOP)	4. Errors 4,5, and 6 are hardware or software errors. They should not occur. RECOVERY: Rerun the program. If the error persists, notify Wang Laboratories.
5. OUTPUT PROGRAM SPACE EXCEEDED (STOP)	

6. SYSTEM ERROR (STOP)	
7. NO ROOM ON DISK FOR OUTPUT PROGRAM (STOP)	7. There is not room enough on the disk for the output program to be cataloged. RECOVERY: Rerun the program, with an output disk with more free space (25 sectors maximum requirement).

22.3 CALLING THE KFAM-4 SUBROUTINES

Dummy Variable Names

In defining the argument lists for the subroutines below, certain standard dummy variable names are used. These dummy names are used only to describe the general forms of the respective GOSUB' statements. In the actual program, the programmer may use any value or expression valid for use in a GOSUB' statement. Zeros in the general statement represent parameters which are not used by KFAM-4. They should be included, as zeros in the GOSUB' statement.

For example, the general statement:

```
GOSUB' 233 (I,P,A$,0)
```

may be written as:

```
GOSUB'233(I,P,K$,0)
GOSUB'233 (2,1,"A48-3029",0)
GOSUB'233(F1+1,0,STR(P1$,7,8),0)
etc.
```

The dummy variable names for KFAM-4, and their meanings, are as follows:

Dummy Variable	Meaning
I	KFAM I.D. Number (1, 2, or 3).
K	File number assigned to the Key File (#0-#6).
U	File number assigned to the User File (#0-#6).
F	Key File number (1-9), specified as the 6th character in the Key File name, as assigned in INITIALIZE KFAM FILE.
A$	The record key (alphanumeric).
N$	User File name.

P	Protect flag. If P=0, then other CPU's may access this record or block of records. If P=1, then only this CPU may access this record or block of records.
C$	Class of access desired in opening the file. "A" means any CPU may access the file. "X" means this CPU seeks exclusive access to the file.

Return Codes

Upon returning to the main line program from the subroutines, the variables Q and Q$ contain the following information:

Q returns the record position indicator for blocked files (i.e., files with more than one record per sector). The record position indicator is a numeric value which specifies the position of a desired record within a block. For example, if Q=2, the key passed to the subroutine specifies the second record in the block. For unblocked records Q is returned as 1, and may be ignored.

Q is not defined following the OPEN or CLOSE subroutines.

Q$ contains the completion return code. It indicates the result of the particular operation. The possible values of Q$, and their meanings, are as follows:

Q$ Value	Meaning
blank	The subroutine execution was successful.
D	Duplicate key (attempting to add a duplicate key to the file). The Key File is unchanged.
E	End of file (FINDNEXT only).
N	Key not found.
S	No more space, either for the User File or the Key File, or 8 levels of index have been exhausted attempting to add a record to the file. The Key File is unchanged. (FINDNEW and FINDNEW(HERE) only.)
B	Busy Signal. The user file record or block of records being accessed has been "protected" by another CPU.
C	Access Class conflict (OPEN only). Either this CPU is asking for

	exclusive access when another CPU has the file open, or this CPU is asking for access when another CPU has exclusive access.
X	Improper call to a KFAM subroutine, (argument values erroneous, etc.).

If Q$ is anything other than blank, the User File Current Sector address parameter is undefined, and the value of Q is undefined.

Immediately upon return from any of the subroutines, the main line program should check Q$ for possible error indications.

The system assumes there are no programming errors in the main line program. The KFAM Subroutines can perform improperly, and can destroy a file, if the parameters supplied by the main line program are erroneous. Therefore, during the testing stage, it is recommended that the user keep a backup file so that test data can be recovered in the event that it is destroyed.

The subroutines check data errors, and the kind of errors likely to occur during normal operation, such as duplicate key, key not found, or no more space. The following errors, which are programming errors, may or may not be caught by the subroutines:

Error	Q$ Value, or ERR Code
KFAM I.D. Number not an integer between 1 and 3.	X ERR 18
KFAM I.D. Number is the same as I.D. Number for a file already open.	X
File to be opened is already open.	X
Individual file numbers not integers between 0 and 6.	ERR 18 ERR 41
Individual file number is duplicate of another file number.	X
File name not in proper format, with 5th byte="F" and 6th byte a 0 (zero).	ERR 78 ERR 80
Key File number not an integer from 1 to 9.	ERR 56
File to be accessed has not been opened.	X

SELECT statements and file numbers do not actually correspond.	none
File names are not correct, or do not exist on the disk platters specified.	ERR 78 ERR 80

22.4 OPEN

The OPEN subroutine is used to open a User File and its companion Key File. OPEN must be executed prior to execution of any other KFAM subroutine. In the OPEN subroutine, a pair of DATALOAD DC OPEN statements are executed to open the named User File and its companion Key File. Specified file numbers are assigned to each file. OPEN also assigns a specified KFAM I.D. Number to the pair of files. To call the OPEN subroutine you must write two statements of the following general form:

```
SELECT #U XXX
GOSUB' 230 (I,K,U,F,N$,C$)
```

For The SELECT Statement

"#U" is the file number to be associated with the User File; "U" can be a number from 1 to 6. "#U" must be used in all subsequent DATASAVE DC or DATALOAD DC statements to reference the User File.

"XXX" is the device address of the platter on which the User File is stored.

For The GOSUB' Statement

"I" is the KFAM I.D. Number which is to be associated with the newly opened file, and must be used to reference the file in subsequent KFAM subroutines. "I" can be a number from 1 to 3.

"K" is the file number to be assigned to the Key File (see NOTE below).

"U" is the file number to be assigned to the User File (see "#U" above).

"F" is the Key File number (the sixth character in the Key File name, it may be an integer from 1 to 9, but normally it is 1).

"N$" is the name of the User File to be opened. The Key File name need not be specified; it is built from the User File name and the Key File number by KFAM itself.

"C$" is the class of access desired. If C$ = "A", then any CPU may open the file. If C$ = "X", then only this CPU may access the file.

Return Codes for OPEN

Q$ = " " (space) if the subroutine execution was O.K.

Q$ = "C" if there is an access class conflict. Either this CPU seeks exclusive access (C$ = "X") when another CPU has the file open, or another CPU has exclusive access.

Q$ = "X" for an improper call (i.e., one of the arguments in the GOSUB' 230 argument list is incorrect, or the file is already open). Note, if a file is already open or the KFAM I.D. number is already in use, OPEN returns Q$ = "X".

NOTE:

The application program must include the SELECT subroutines DEFFN'210 and DEFFN'211 to select hog mode and non-hog mode for the Key File device address. All KFAM-4 subroutines require that these subroutines be included in the application program. See Section 22.1 for information about how to write these subroutines.

22.5 DELETE

The DELETE subroutine deletes from the Key File a specified key and its associated record location pointer. The Current Sector address for the User File is set to the location of the record whose key has been deleted, and for blocked records the variable Q is set to the record position within the sector. The record itself, in the User File, is not altered or removed. Thus, although the record is not physically removed from the User File, its key entry is removed from the Key File, and the record can no longer be accessed through KFAM.

The calling sequence for DELETE is:

GOSUB' 231 (I, P, A$)

"I" is the KFAM I.D. Number, assigned to the file in an OPEN subroutine.

"P" is the protect flag option. If P=0, other CPU's may access this record of block of records. If P=1,only this CPU may access this record or block of records.

"A$" is the key of the record that is to be deleted from the file.

DELETE Return Codes

Q$ = "B" Busy Signal. The record sought is protected by another CPU.
Q$ = "N" if the key passed cannot be found in the Key File.
Q$ = "X" for an improper call.
Q$ = " " ("space") if the subroutine executed properly.

After calling a DELETE subroutine and checking for its successful completion, the application program should flag the DELETED record in the User File by changing the first character of the deleted record's key to hex FF. For unblocked files this can be done as follows:

Suppose:

```
    DIM A$15, H(4,4), J(6)
```
and
```
    DATA SAVE DC #1, A$, H(), J()
```

define a type "N" record where A$ is the key field.

The DELETE and flag operation might look like this:

```
4060 GOSUB' 231 (1, 1, A$): REM DELETE
4065 IF Q$ = "B" THEN 4060:REM BUSY TRY AGAIN
4070 IF Q$<>" " THEN 6000:REM UNSUCCESSFUL
4080 DATA LOAD DC #1, A$, H(), J()
4090 STR(A$,1,1)=HEX(FF):REM HEX(FF) IN 1ST BYTE OF KEY
4100 DBACKSPACE #1,1S:REM RECORDS ARE 1 SECTOR LONG
4110 DATA SAVE DC #1,A$,H(),J()
.
.
.
6000 STOP "DELETE UNSUCCESSFUL"
```

The space occupied by DELETED records in the User File can be reused; this normally requires special techniques together with the use of FINDNEW(HERE). For information on these techniques see Chapter 29.

22.6 FINDOLD

The FINDOLD subroutine is used to locate a desired record in the User File. Following subroutine execution, the Current Sector address for the User File is set to the sector address of the record whose key was passed. For blocked records, variable Q is set to the record position within the sector. The record can then be read with a DATALOAD DC statement. The calling sequence is:

```
GOSUB' 232 (I, P, A$)
```

"I" is the KFAM I.D. Number assigned to the file in the OPEN subroutine.

"P" is the protect flag option. If P=0,then other CPU's may access this record or block of records. If P=1,only this CPU may access this record or block of records.

"A$" is the key of the record being sought.

FINDOLD Return Codes

Q$="B" Busy Signal. The record sought is protected by another CPU.
Q$ = "N" if the specified key is not located in the Key File.
Q$ = "X" for an improper call.
Q$ = " " ("space") if the key was located without difficulty.

22.7 FINDNEW

The FINDNEW subroutine is used to enter a new key in the Key File and to find a location for the new record in the User File. FINDNEW enters the key passed to it in the Key File, then sets the Current Sector address for the User File to an available User File location for writing a new record. For blocked records, variable Q is set to the record position within the sector.

```
GOSUB' 233 (I,P,A$,0)
```

"I" is the KFAM I.D. Number, assigned to the file in an OPEN subroutine.

"P" is the protect flag option. If P=0, other CPU's may access this record or block of records. If P=1, only this CPU may access this record or block of records.

"A$" is the new key to be entered in the Key File.

FINDNEW Return Codes

Q$ =	"B" Busy Signal. The record (or block) sought is protected by another CPU.
Q$ =	"D" if the key specified is a duplicate of one already in the Key File.
Q$ =	"S" if there is no space in the User File for another record, or in the Key File for another key entry, or 8 index levels have been exhausted.
Q$ =	"X" for an improper call.
Q$ =	" " ("space") if the key was entered without difficulty.

NOTE:

The User File location returned by FINDNEW is unoccupied by live data, but is not necessarily at the end of all live data in the User File.

The following example illustrates the procedure for adding a record to type A blocked files following FINDNEW. Note the test on Q before the DATASAVE, and that the protect flag is set by FINDNEW.

```
4100        INPUT "KEY FIELD", A$     :REM OPERATOR ENTERS KEY
4120        GOSUB '233 (1,1,A$,0)     :REM FINDNEW
4130    REM TEST COMPLETION CODE
4135        IF Q$ = "B" THEN 4120     :REM BUSY TRY AGAIN
4140        IF Q$ = "D" THEN 5010     :REM DUPLICATE KEY?
4150        IF Q$ = "S" THEN 5050     :REM FILE FULL?
4160        IF Q$ <> " " THEN 5060    :REM ERROR?
4170    REM NEW BLOCK OR OLD?
4180        IF Q = 1 THEN 4220  :REM FIRST RECORD IN NEW BLOCK?
4185    REM READ EXISTING RECORDS IN BLOCK
4190        DATA LOAD DC #2, 1 S  :REM BACKSPACE AFTER DATALOAD
4210    REM ASSIGN RECORD VALUES TO PROPER ARRAY ELEMENTS
4220        A$(Q) = A$
4230        INPUT "SECOND FIELD", B$(Q)
4240        INPUT "THIRD FIELD", C(Q)
```

```
4250        INPUT "FOURTH FIELD", D(Q)
4260     REM SAVE BLOCK IN USER FILE
4270        DATA SAVE DC $2, A$(),B$(),C(),D()

5000     REM ERROR ROUTINES
5010        STOP "KEY ALREADY IN KEY FILE"
5050        STOP "KEY FILE OR USER FILE IS FULL"
5060        STOP "FINDNEW ERROR"
```

A similar procedure must be used for type C files, for which Q represents a record location rather than a subscript.

22.8 FINDNEW(HERE)

The FINDNEW(HERE) subroutine is a special purpose subroutine which can be used to reuse the User File space occupied by DELETE'd records or to change the value of the key of an existing record. It adds a new key to the Key File, but, unlike FINDNEW, the User File location, which it associates with that key, is the User File location returned by the last KFAM subroutine call. To use FINDNEW(HERE) to reuse the User File space occupied by DELETE'd records, see Chapter 29. An illustration of the use of FINDNEW(HERE), to change the value of the key of an existing record, is shown below.

The calling sequence is:

```
GOSUB' 234 (I,P,A$,0)
```

The FINDNEW(HERE) argument list is identical to the argument list for FINDNEW (see FINDNEW).

FINDNEW (HERE) Return Codes

Q$ =	"B" Busy Signal. The record or block sought is protected by another CPU.
Q$ =	"X" for an improper call.
Q$ =	"D" if the key specified is a duplicate of a key already in the Key File.
Q$ =	"S" if there is no space in the Key File for another entry, or if 8 index levels have been exhausted.
Q$ =	" " (space) if the subroutine executed properly.

The following example illustrates the use of FINDNEW (HERE) following DELETE:

```
5000 GOSUB '231 (1,0,"ABCD") :REM DELETE "ABCD" FROM KEY FILE
5005 IF Q$ = "B" THEN 5000:REM BUSY
5010 IF Q$ = "X" THEN 5130
5040 IF Q$ = "N" THEN 5150
5050 GOSUB '234 (1,1,"EFGH",0) :REM SET "PROTECT", INSERT "EFGH" IN KEY FILE
5060 IF Q$ = "X" THEN 5140
5070 IF Q$ = "D" THEN 5160
5075 IF Q$="S" THEN 5170
5080 DATALOAD DC #2,A$,B$,C$,N
5090 A$ = "EFGH" :REM CHANGE KEY TO "EFGH"
5100 DBACKSPACE #2, 1S
5110 DATASAVE DC #2,A$,B$,C$,N
5115 GOSUB'239(1) :REM CLOSE FILES
5120 END
5130 STOP "ERROR IN 'DELETE' CALLING SEQUENCE"
5140 STOP "ERROR IN 'FINDNEW(HERE)' CALLING SEQUENCE"
5150 STOP "KEY NOT FOUND"
5160 STOP "DUPLICATE KEY"
5170 STOP "NO SPACE"
```

22.9 FINDFIRST

The FINDFIRST subroutine sets the Current Sector address for the User File to the first record in logical key sequence. For blocked records, variable Q is set to the record position within the sector. A DATALOAD DC statement can be used after FINDFIRST to read the record. The calling sequence is:

```
GOSUB' 235 (I,P)
```

"I" is the KFAM I.D. Number, assigned to the file in an OPEN subroutine.

"P" is the protect flag option. If P=0 other CPU's may access this record or block of records. If P=1 only this CPU may access this record or block of records.

FINDFIRST Return Codes

Q$ = "B" Busy Signal. The record sought is protected by another CPU.
Q$ = "N" if the User File contains no records.
Q$ = "X" for an improper call.
Q$ = " " (space) if the subroutine executed properly.

22.10 FINDLAST

The FINDLAST subroutine sets the Current Sector address for the User File to the last record in logical key sequence. For blocked records, the variable Q is set to the record position within the sector. A DATALOAD DC statement can be executed following FINDLAST to read the record. The calling sequence is:

```
GOSUB' 236 (I,P)
```

"I" is the KFAM I.D. Number assigned to the file in an OPEN subroutine.

"P" is the protect flag option. If P=0, other CPU's may access this record or block of records. If P=1, only this CPU may access this record or block of records.

FINDLAST Return Codes

Q$ = "B" Busy Signal. The record sought is protected by another CPU.
Q$ = "N" for a null file.
Q$ = "X" for an improper call.
Q$ = " " (space) if the subroutine executes normally.

22.11 FINDNEXT

The FINDNEXT subroutine sets the Current Sector address for the User File to the record immediately following (in logical key sequence) the last record accessed by KFAM. For blocked records, the variable Q is set to the position of the record within the sector. A DATALOAD DC statement can be executed following FINDNEXT to read the record. FINDNEXT is useful for processing files in key sequence. The calling statement is:

```
GOSUB' 237 (I,P)
```

"I" is the KFAM I.D. Number assigned to the file in an OPEN subroutine.

"P" is the protect flag option. If P=0, other CPU's may access this record or block of records. If P=1, only this CPU may access this record or block of records.

FINDNEXT Return Codes

Q$ = "B" Busy Signal. The record sought is protected by another CPU.
Q$ = "X" for an improper call.
Q$ = "E" if the previous reference was to the last record in logical key sequence.

Otherwise, Q$ = " " (space).

NOTE:

FINDNEXT cannot be executed as the first subroutine following an OPEN routine. Also, FINDNEXT cannot normally be executed immediately following any subroutine which returned Q$ = "X" or "E". Otherwise FINDNEXT will locate the next sequential key, following any subroutine.

If FINDNEXT is executed after a FINDNEXT which returned Q$ = "B" (the Busy Signal), it will attempt to access the same record that it previously found to be protected.

If FINDNEXT is executed following a FINDOLD that returned Q$ = "N" (not found), FINDNEXT locates the record whose key logically follows the key passed to FINDOLD.

22.12 RELEASE

The RELEASE subroutine turns off the protect flag previously set by the calling CPU.

Any call to a KFAM-4 subroutine for a particular file turns off any protect flag for that file. RELEASE should be used only if there may be a long delay before the next KFAM-4 subroutine is called.

The calling sequence is:

```
GOSUB'238 (I)
```

"I" is the KFAM I.D. Number assigned to the file in an OPEN subroutine.

RELEASE Return Codes

Q$ = "X" for an improper call.
Q$ = " " (space) after successful execution.

22.13 CLOSE

The CLOSE subroutine is used to close a currently open User File and its companion Key File. The KFAM I.D. Number assigned to a closed file can then be reassigned to another file in an OPEN routine. Also, the file numbers assigned to a User File and Key File can be reassigned. CLOSE alters the Access Table in the Key File's KDR record to indicate that the CPU no longer has the file open. The CLOSE subroutine also saves certain critical information for the KEY FILE RECOVERY utility, provided that the RECOVERY OPTION was included during BUILD SUBROUTINE MODULE execution. The calling sequence is:

```
GOSUB' 239 (I)
```

"I" is the KFAM I.D. Number assigned to the file in an OPEN routine. Following execution of the CLOSE routine, this number can no longer be used to access the User File and its associated Key File.

CLOSE Return Codes

Q$ = "X" for an improper call.
Otherwise, Q$ = " " (space).

NOTE:

CLOSE must be executed at the conclusion of operations on a file.

CHAPTER 23
THE KFAM REORGANIZE UTILITIES (KFAM-3 AND KFAM-4)

23.1 THE REORGANIZE SUB-SYSTEM

23.1.1 Overview

The REORGANIZE SUB-SYSTEM performs the following KFAM file maintenance operations:

1. Based on an input KFAM file (Key File and User File) it constructs a new output User File which contains active records only, written in ascending key sequence.

2. Creates a Key File based on the new output file. Optionally the new Key File may occupy the same physical space as the input Key File, overwriting the input Key File.

3. Optionally, the new output User File may be copied back to the disk area occupied by the input User File, overwriting the input file.

Unlike other KFAM maintenance utilities, the REORGANIZE SUB-SYSTEM is loaded by means of a user-written set-up program that specifies all the parameters for the reorganization. The REORGANIZE SUB-SYSTEM can, optionally, call another user program module after completing execution. Since it must be loaded via a user-written set-up module, the REORGANIZE SUB-SYSTEM does not appear on the KFAM-3 or KFAM-4 menus. However, the three modules of the utility are included on Application Support Diskette #2 for KFAM-3, and Application Support Diskette #3 for KFAM-4. Also included on each diskette is a module of KFAM subroutines used by the utility. Finally, on each diskette a COPY/VERIFY reference file is included to facilitate copying the complete utility. The names of the reference files and modules are as follows:

KFAM-3 VERSION

COPY/VERIFY Reference File Name	=	K-3RF010
REORGANIZE SUB-SYSTEM modules	=	KFAM3503
	=	KFAM3603
	=	KFAM3703
Module of Subroutines	=	KFAM0103

KFAM-4 VERSION

COPY/VERIFY Reference File Name	=	K-4RF010
REORGANIZE SUB-SYSTEM Modules	=	KFAM3504

	=	KFAM3604
	=	KFAM3704
Module of Subroutines	=	KFAM0104

23.1.2 Writing The Set-up Module

To use the REORGANIZATION SUB-SYSTEM you must write a brief set-up program which provides the operating parameters and loads the first module. The set-up program can be broken down into two parts, with a third part used only for KFAM-4.

1. Lines 1-3499 contain statements executed before the REORGANIZATION SUB-SYSTEM is loaded. These lines must clear the CRT screen, select disk file devices, and load KFAM3503 (or KFAM3504 for KFAM-4). These lines must be cleared by the LOAD DC statement. They can include additional preprocessing, if desired.

2. Lines 4200-4799 contain statements which assign reorganization parameters to specific variables. They remain as an overlay to the first reorganization module. They are executed after the first reorganization module defines its common variables and sets default values.

3. (KFAM-4 ONLY) Lines 3500-3699 must contain the subroutines DEFFN'210 (T6), to select hog mode, and DEFFN'211 (T6) to select non-hog mode. These lines are not cleared; they remain as an overlay to the first module.

A skeleton of the set-up program is shown below. A line may be omitted if the default value shown is the desired value. Read all comments before writing a set-up module.

Line	Contents	Default Value	See Comment
10	REM program identification		
20	PRINT HEX(03)	-	1
50	SELECT DISK (disk address for REORGANIZATION SUB-SYSTEM disk)	-	
60	SELECT #1 input User File device address	-	
70	SELECT #2 input Key File device address	-	
80	SELECT #3 output User File device address	-	2
90	SELECT #4 output Key File device address	-	3
100	SELECT #5 user program device address	-	4
110	LOAD DC T#0, "KFAM3503" 1,3499	-	5
4210	N1$ = input User File name	-	
4220	P1$ = input User File device address as "xyy"	-	6
4230	N2 = input Key File number	1	7
4240	P2$ = input Key File device address as "xyy"	-	
4250	N3$ = output User File name	-	8
4260	P3$ = output User File device address as "xyy"	-	
4270	O3$ = "C" catalog output User File if uncataloged.	C	9

		= "Y" output User File already cataloged, do not catalog it. = "N" output User File is not already cataloged, catalog it.		
4280	S3	= number of sectors to allocate to output User File. This statement not needed if 03$ = "Y" (above). *If the utility catalogs the file, the default value for S3 is the number of sectors in the input User File.	*	9
4290	06$	= "C" to copy back Output User File over Input User file when reorganization completed. = blank to leave input User File intact.	blank	10
4300	N4	= output Key File number	1	11
4310	P4$	= output Key File device address as "xyy"		
4320	04$	= "C" catalog output Key File, if uncataloged. = "Y" output Key File cataloged, do not catalog it. = "N" output Key File not cataloged, catalog it.	C	12
4330	S4	= number of sectors to allocate to the output Key File. Omit this if 04$ = "Y". *If the utility catalogs the file, the default value is calculated in proportion to the input Key File size times the increase or decrease in User File size.	*	12
4340	N5$	= name of program to be loaded following reorganization = blank - no program to be loaded	blank	13
4350	P5$	= device address of user program as "xyy"		

<u>Additional Lines For KFAM-4 Only</u>

Line	Contents	Default Value	See Comment
3510	DEFFN'210 (T6)	-	14
3520	SELECT #2 input Key File hog mode address		
3530	RETURN		
3540	DEFFN'211 (T6)		14
3550	SELECT #2 input Key File non-hog mode address		
3560	RETURN		

Comments

1. The CRT screen should be cleared prior to calling the REORGANIZATION SUB-SYSTEM. Lines 0-3 are used by the utility for messages. Lines 4-15 may be used for a user written display. For example, line 20 might be:

```
20 PRINT HEX(030A0A0A0A); "REORGANIZE INVENTORY FILE."
```

2. The output User File device address may be the same as the input User File device address, if the two files have different names. (For KFAM-4 see comment 15.)

3. If the output Key File device address is the same as the input Key File device address, and the output Key File name is the same as the input Key File name, then the output Key File replaces the input Key File. See comment 10.

4. If the REORGANIZATION SUB-SYSTEM is to call another program when it completes execution, the device address of this program file must be selected for file number #5.

5. The last statement to be executed in the range 1-3499 must be a LOAD DC that loads module 1 of the utility and clears lines 1-3499 as it does so. If the KFAM-4 utility is being used,the module name is "KFAM3504".

6. Example:

```
4220 P1$ = "B10"
```

7. This number is assigned during INITIALIZE KFAM FILE and appears as the 6th character in the input Key File name (normally it is 1).

8. The output user file name need not conform to the KFAM file naming conventions. This relaxation of normal KFAM requirements may be useful if the "copy back" option is chosen (line 4290), since in this case it may be desirable to use an established work file that may have any name.

9. If "C" is assigned to O3$, the output User File is cataloged by this utility, if the output file does not already exist. "C" is the default value of O3$.

   If "N" is assigned to O3$, the system ensures that the named output User File does not already exist on the disk, and catalogs the output User File.

   If the utility catalogs the output User File, it allocates to it the same number of sectors that are in the input User File, unless a different number is specified by assigning the desired number of sectors to S3.

   If "Y" is assigned to O3$, the system checks that the named output file already exists. S3 need not be assigned a value.

   The output User file must contain at least 10 sectors.

10. If O6$ is assigned the value "C", then the utility constructs the output Key File name from the input Key File name, and copies the output User File back into the input User File area, overwriting the input User File. If O6$ is assigned a blank, then the output Key File name is constructed from the output User File name, and the output file is not copied back.

11. This number is used in the construction of the output Key File name, in which it becomes the 6th character. If the constructed name is the same as the input Key File name, and the same address is specified for both input and output Key Files, then the output Key File replaces the input Key File.

12. The effect of these responses for O4$ and S4 is analogous to O3$ and S3 discussed in comment 9. However, if the utility catalogs the Key File, its size is proportional to the input Key File size times the increase or decrease in User File size.

13. If N5$ is assigned a program name, the program is loaded upon completion of the utility. The program must reside at the address SELECTed for file number #5 (line 100).

14. (KFAM-4 ONLY) Lines 3500-3699 must contain the KFAM-4 SELECT subroutines. These subroutines are discussed in detail in Chapter 22. These lines must not be cleared by the LOAD DC statement at line 110. If the programmer wants the entire utility to execute in hog mode, or non-hog mode, lines 3520 and 3550 may be omitted, leaving the subroutines to consist merely of a RETURN statement. However, the subroutines themselves must be present. (See Comment 15.)

15. ADDITIONAL PROGRAMMING NOTES FOR KFAM-4 - The REORGANIZATION SUB-SYSTEM restricts access to the input files (Key File and User File) by opening the input file in exclusive mode. However, access to the output files cannot be controlled in this manner. If the user's own conventions cannot assure the integrity of the output files during the reorganization, then hog mode addresses should be specified for file numbers #0-#4 (lines 50-90) and the subroutines (lines 3510-3560) should omit the SELECT statements at lines 3520 and 3550. When the reorganization is complete, hog mode can be deselected by loading a user program that turns off hog mode, or RESET can be keyed.

Shown below are two set-up programs, one for KFAM-3 and another for KFAM-4.

Example 23-1 A Set-Up Program To Call KFAM3503

```
10 REM EXAMPLE OF A REORGANIZATION SET-UP PROGRAM FOR KFAM-3
20     PRINT HEX(030A0A0A0A); "REORGANIZE INVENTORY FILE"
50     SELECT DISK 310 :REM REORG. SUB-SYSTEM
60     SELECT #1 B10   :REM INPUT USER FILE
70     SELECT #2 B10   :REM INPUT KEY FILE
80     SELECT #3 B10   :REM OUTPUT USER FILE
90     SELECT #4 B10   :REM OUTPUT KEY FILE
110    LOAD DC T#0, "KFAM3503" 1, 3499
4210   N1$ = "INVTF010"
4220   P1$ = "B10"
4240   P2$ = "B10"
4250   N3$ = "INVTF011"
4260   P3$ = "B10"
4310   P4$ = "B10"
```

Example 23-2 A Set-Up Program To Call KFAM3504

```
10 REM EXAMPLE OF A REORGANIZATION SET-UP PROGRAM FOR KFAM-4
20     PRINT HEX(030A0A0A0A); "REORGANIZE INVENTORY FILE"
50     SELECT DISK 310 :REM REORG. SUB-SYSTEM
60     SELECT #1 320   :REM INPUT USER FILE
70     SELECT #2 B20   :REM INPUT KEY FILE
80     SELECT #3 320   :REM OUTPUT USER FILE
90     SELECT #4 B20   :REM OUTPUT KEY FILE
100    SELECT #5 310    :REM USER PROGRAM
110    LOAD DC T#0, "KFAM3504" 1, 3499
3510   DEFFN' 210 (T6)
3520   SELECT #2 BA0
3530   RETURN
3540   DEFFN' 211 (T6)
3550   SELECT #2 B20
3560   RETURN
4210   N1$ = "INVTF040"
4220   P1$ = "320"
4240   P2$ = "B20"
4250   N3$ = "WORK"
4260   P3$ = "320"
4290   06$ = "C" :REM COPY BACK OUTPUT USER FILE
4310   P4$ = "B20"
4340   N5$ = "START"
4350   P5$ = "310"
```

23.1.3 Utility Operation and Error Messages

The operation of the utility may be divided into three parts:

1) The User File is read sequentially, using FINDFIRST/FINDNEXT, and copied to the output file so that the records are physically in sequential order, and DELETED records are eliminated.

2) A new Key File is built, based on the keys in the output User File, using a special procedure. The new Key File, optionally, may occupy the same physical space as the old Key File.

3) If indicated by the set-up program, the output user file is copied back to the input user file, overwriting the original.

The original Key File and User File are not altered until the output User File has been written, complete with the necessary information to restore the Key File. Therefore, it is not essential to have backup copies of the User File and Key File. If the system fails during Part 1 of the reorganization, the original Key File and User File are intact. If the system fails during Part 2, both the input User File and the output User File are intact, and a Key File may be built for either one, using the Key File Recovery Utility. During Part 3, the output User File remains intact, as well as the Key File. Although backup disks are not necessary for this operation, it is good practice to make backup copies regularly, especially of the User File.

There are no operating instructions for this program, because normally no operator intervention is required. However, there are recovery procedures, for certain error conditions. These are described below:

ERROR MESSAGE	EXPLANATION/RECOVERY
1. ERR 72	Disk read error. Part 1: Input User File or Key File has an unreadable sector. Part 2: Output User File or Key File has an unreadable sector. Part 3: Output User File has an unreadable sector. RECOVERY: Part 1: Run Key File Recovery Utility. Rerun. Part 2: Run Key File Recovery Utility on input User File, if input Key File is being overwritten. Rerun. Part 3: Run Key File Recovery Utility on output User File. Set up Reorganize Subroutine to reorganize output User File, giving input User File.
2. ERR 85	Disk write error. Part 1: Output User File contains a bad physical sector. Part 2: Output Key File contains a bad physical sector. Part 3: Input User File contains a bad physical sector. RECOVERY: Part 1: Replace the output disk, or recreate file to bypass the bad sector. Rerun. Part 2: Replace the disk containing the output Key File, or recreate the file to bypass the bad sector. If

		input and output Key File are the same, run Key File Recovery. Rerun. Part 3: Replace input disk or recreate input User File to bypass the bad sector. See recovery procedure for ERR 72, Part 3.
3.	ERR 80	Program not on disk. The four modules listed in 23.1.1 must reside on the device specified by "SELECT DISK". Correct and rerun.
4.	FILE ######## NOT FOUND ON DEVICE ###	Required KFAM file (User File or Key File) is not on designated disk. RECOVERY: Mount disk containing the designated file. Rerun.
5.	INPUT AND OUTPUT USER FILE MAY NOT BE THE SAME FILE	Both input and output User Files are designated by the same file name, on the same device. RECOVERY: Correct the file designations. Rerun.
6.	######## NOT KFAM FILE NAME	File name must have "F" in position 5, and a digit 0-9 in position 6. RECOVERY: Correct the file name. File name may be changed on disk using SCRATCH and DATASAVE DC OPEN. Rerun.
7.	INVALID KEY FILE NUMBER #	Key File number not 1-9, or not an integer. RECOVERY: Correct the Key File number. Rerun.
8.	INSUFFICIENT SPACE FOR FILE ######## ON DEVICE ###	There is not enough space on the designated disk device to catalog the file. RECOVERY: Mount an output disk with enough free space to accommodate the output User File and/or Key File. Rerun.
9.	FILE ######## ALREADY CATALOGED ON DEVICE ###	A file designated as "not cataloged" is already cataloged, or another file exists of the same name. RECOVERY: Mount a scratch disk or other disk which does not already

	have this file name cataloged. Rerun. If this is a rerun, change the values of O3$ and O4$ (user set-up module) to "C", and run.
10. ERROR OPENING FILES	KFAM "OPEN" subroutine returns an error indication. RECOVERY: If this is a rerun, set V9=0 and rerun.
11. INVALID RECORD FORMAT	Type A records: Invalid control byte or more than 38 fields per record. RECOVERY: The program will not reorganize this file.
12. NOT BLOCKED AS SPECIFIED	Type A records: Blocking of record not the same as blocking specified in the KDR. RECOVERY: Write a program to open the file, change V8$ in the KDR, and close the file. Run it. Then rerun the reorganization.
13. RECORD LENGTH NOT SPECIFIED CORRECTLY	Type A records: Record length not the same as specified in the KDR. RECOVERY: Change STR(V1$,2,1) in the KDR (see 12, above). Rerun.
14. KEY FIELD OUT OF BOUNDS	Type A records: The starting position of the key and/or key length are such that the key is not wholly included within a field of the record. RECOVERY: Change the starting position of the key, STR(V1$,4,1), or the key length, STR(V1$,5,1), in the KDR (see 12, above). Rerun.
15. NUMERIC KEY INVALID	Type A records: The key field is indicated as lying within a numeric variable. RECOVERY: See 14, above, to change key field position. The key may not be a numeric variable. Rerun.

16. NULL FILE	There are no active records in this file. RECOVERY: Run "INITIALIZE KFAM FILE" to reorganize this file.
17. FINDFIRST ERROR	Hardware or software error. RECOVERY: Rerun. Notify Wang Laboratories if the problem persists.
18. FINDNEXT ERROR	Hardware or software error. RECOVERY: Rerun. Notify Wang Laboratories if the problem persists.
19. OUTPUT USER FILE SPACE EXCEEDED	Output User File is too small to contain all the active records from the input User File. RECOVERY: Allocate more space for the output User File, and rerun.
20. OUTPUT KEY FILE SPACE EXCEEDED	Output Key File is too small. RECOVERY: If the output Key File is the same as the input Key File, run KEY FILE RECOVERY on the input User File. Allocate more space for the output Key File, and rerun.
21. 8 LEVELS OF INDEX EXCEEDED	More than 390,625 30-byte keys, or more than 429,981,696 12-byte keys, etc. This error should not occur. RECOVERY: Notify Wang Laboratories.
22. SEQUENCE ERROR, HEX KEY = (key value)	The key contained in the input User File is not the same as the key in the input Key File. RECOVERY: Check for errors in application programs that may cause this condition. Run KEY FILE RECOVERY on input User File. Rerun.
23. INVALID KEY, HEX VALUE = (key value) KEY RETURN(EXEC) TO SKIP RECORD	The record is included as active in the input Key File, but flagged as deleted in the input User File. RECOVERY: See 22, above. The option is also provided to skip the record and continue.

24. MOUNT DISK CONTAINING PROGRAM ######## ON DEVICE ### KEY RETURN(EXEC) TO RESUME	The reorganization is finished and ready to load the next program, which is not found on the designated device. RECOVERY: Mount the disk containing the next program on the designated device. KEY RETURN(EXEC).
25. ACCESS NOT EXCLUSIVE	The input KFAM File is currently open by another CPU. RECOVERY: Other CPU's must close the file.

23.2 REORGANIZE KFAM FILE

23.2.1 Overview

This program reorganizes a KFAM File in place. The record which belongs first, in ascending key sequence, is switched with the record that is physically first. This process is repeated for the second record, and so forth, until the entire User File has been placed in sequential order. In the process, all DELETED records are removed. Then, the Key File is reinitialized and a new Key File is created in the space formerly occupied by the old Key File. No additional disk space is required for the User File or the Key File; a 15 sector work file is required. This program is 4 to 5 times slower than the REORGANIZATION SUB-SYSTEM utility, and therefore should be used only if the file is too large to permit simultaneous mounting of an output file as required by the REORGANIZATION SUB-SYSTEM.

Because this program destroys both the User File and the Key File, as they formerly existed, there is no possible recovery in the event of hardware or software error. For that reason, backup copies of the User File and the Key File must be made before running this program.

This program reorganizes any KFAM file, with the exception of type M records with more than 40 sectors per record. However, it should be noted that multiple-sector records require extra storage space in memory, and that this program cannot operate in a 12K system if the record length exceeds 8 sectors.

NOTE:

For KFAM-4 only, hog mode is automatically selected for the disk drives containing the User File, Key File, and Work File. To operate the utility at a non-multiplexed disk drive key:

M$ = "X"(EXEC)

at KFAM-4 utilities menu, prior to loading the utility.

23.2.2 Operating Instructions

	DISPLAY		INSTRUCTIONS
1.		1.	Make backup copies of both the User File and the Key File. If anything goes wrong during the execution of the utility, both files are destroyed.
2.		2.	Mount disk platter(s) containing the User File and the Key File.
3.		3.	To access the REORGANIZE KFAM FILE utility, depress the specified Special Function key from KFAM-3 or KFAM-4 subsidiary menu.
4.	ARE THERE BACKUP COPIES OF USER FILE AND KEY FILE? (Y OR N)	4.	Enter Y if backup copies exist. Proceed with Step 6 below. Enter N for "no" or "don't know". Proceed with Step 5.
5.	The system displays: ANY ERROR DURING THE RUNNING OF PROGRAM FILE WILL DESTROY BOTH FILES. MAKE COPIES OF THE DISK PLATTER(S) CONTAINING THE USER FILE AND THE KEY FILE BEFORE RUNNING THIS PROGRAM. STOP	5.	Make backup copies of both the User File and the Key File. Key CONTINUE RETURN(EXEC) and go to step 4.
6.	ENTER USER FILE NAME (SSSSFJNN)	6.	Enter the name of the User File. MESSAGE: 2, 4
7.	ENTER THE NO. OF THE USER FILE DEVICE ADDRESS 1. 310 5. B10 2. 320 6. B20 3. 330 7. B30 4. 350	7.	Enter the selection number for the device address of the User File. NOTE: Error messages and recovery procedures follow the operator instructions. MESSAGE: 2, 5

8. The system displays:
ENTER KEY FILE NUMBER (NORMAL=1)

8. Enter the Key File Number.

The Key File Number should always be 1, unless there are multiple key files for a single User File, in which case, the Key File Number can be any digit from 1 to 9.

MESSAGE: 2, 3, 6

9. The system displays:
ENTER THE NUMBER OF THE KEY FILE DEVICE ADDRESS

1. 310	5. B10
2. 320	6. B20
3. 330	7. B30
4. 350	

9. Enter the selection number for the device address of the Key File.

MESSAGE: 2, 5

10. ENTER WORK FILE NAME

10. 15 sectors are required for a work file. This work file may be a cataloged file, either a scratch file that already exists or a new file created by this program, or it may reside in the temporary work file area (if any) beyond the cataloged area of one of the disk platters.

If the work file is a file already cataloged, or if it is to be cataloged, enter the name of the work file.

If the work file is to reside in the uncataloged area of one of the disk platters, key RETURN(EXEC) alone.

11. ENTER THE NUMBER OF THE WORK FILE DEVICE ADDRESS

1. 310	5. B10
2. 330	6. B20
3. 330	7. B30
4. 350	

11. Enter the selection number for the work file device address.

If no name was entered for the work file, the program proceeds to Step 13, below.

MESSAGE: 2, 5, 7

12. IS WORK FILE CATALOGED? (Y OR N)	12. Enter Y if the work file has been previously cataloged. Enter N if the work file has not been previously cataloged. MESSAGE: 2, 8, 9, 10
13.	13. The system opens the Key File and User File and begins processing. No operator intervention is required from this point on. MESSAGE: 8, 11, 12, 13, 14 15, 16, 17, 18, 19
14. REORGANIZE KFAM FILE	14. This file is being reorganized. Any error from this point on effectively destroys both the User File and the Key File. Both files should be re-created from backup copies before attempting to rerun. MESSAGE: 1, 12, 13, 20, 21, 32
15.	15. The number of records in the User File is displayed and the system returns to KFAM subsidiary menu.

<u>Error Messages</u>

ERROR MESSAGE	EXPLANATION/RECOVERY
2. RE-ENTER	Too many characters were entered, or not "Y" or "N" in response to a "yes" or "no" question. RECOVERY: Repeat the step, entering the correct value.
3. ERR29	A non-numeric quantity was entered when a numeric quantity was requested. RECOVERY: Reenter numeric quantity.
4. NOT KFAM FILE NAME	The User File name must have an "F" in position 5 and a digit (0-9) in position 6.

		RECOVERY: Repeat Step 6. Enter correct User File name.
5.	INVALID DEVICE ADDRESS	The device address for User File, Key File, or work file was invalid. RECOVERY: Repeat the step. Enter correct device address selection number.
6.	INVALID	The Key File number may not be 0. Repeat Step 8. Enter Key File number 1-9.
7.	ERR64	Sector not on disk. The temporary work file area on the specified platter is not large enough to hold the work file (15 sectors). RECOVERY: Rerun the program from Step 4. Specify a different platter or a cataloged file for the work file.
8.	ERR80	File not found. User File, Key File, or work file. RECOVERY: LISTDCF and/or LISTDCR. Check which file is not there. Correct and rerun the program.
9.	ERR79	File already cataloged. The work file is already cataloged. RECOVERY: Rerun the program from Step 4. Pick another name for the work file, or answer "Y" to "IS WORK FILE CATALOGED?".
10.	WORK FILE TOO SMALL	The cataloged file named as a work file contains less than 15 sectors. RECOVERY: Repeat from step 10. Enter a new name for the work file.
11.	STOP ERROR OPENING FILES	File could not be opened. Possible cause: The program was stopped and restarted after processing had begun. RECOVERY: Recreate User File and Key File from backup copies. Rerun this program. If the error persists, notify Wang Laboratories, Inc.

12. ERR72	Disk read error. See accompanying program statement for file being read: #1 = Key File #2 = User File #3 = Program File #4 = Work File #T1= Key File #T1(T9)=Key File If "REORGANIZE KFAM FILE (KFAM3203)" is displayed on the screen, the User File and Key File must be recreated from backup copies. Rerun the program. If the error persists, the file being read is permanently damaged, or there is a hardware malfunction.
13. ERR85	Disk write error. See accompanying program statement for file being written: #1=Key File #2=User File #4=Work File #T1=Key File RECOVERY: If "REORGANIZE KFAM FILE" is displayed on the screen, the User File and Key File must be recreated from backup copies. Rerun the program. If the error persists, either the disk platter is permanently damaged, or there is a hardware malfunction.
14. STOP MORE THAN 40 SECTORS PER RECORD	This program will not reorganize a file with a record length of more than 40 sectors. RECOVERY: This program may be modified by the user for the particular application, by dropping ISS module KFAM3103 (or KFAM3104 for KFAM-4) and coding the necessary statements in KFAM3203 (or KFAM3204 for KFAM-4) (see KFAM "Programming Techniques" - Generated Code).
15. INVALID RECORD FORMAT (STOP)	Record type A, array-type blocking: more than one sector per block, more than 38 fields per record, or not written with correct control bytes. RECOVERY: The program will not reorganize this file.

16. NOT BLOCKED AS SPECIFIED (STOP)	Record type A, array-type blocking: records per block specified incorrectly, in INITIALIZE KFAM FILE or records not written in array format. RECOVERY: The program will not reorganize this file.
17. RECORD LENGTH NOT SPECIFIED CORRECTLY (STOP)	Record type A, array-type blocking: record length specified in INITIALIZE KFAM FILE does not equal record length of sample record. RECOVERY: The program will not reorganize this file.
18. KEY FIELD OUT OF BOUNDS (STOP)	Record type A, array-type blocking: the key must be wholly contained within one field of the record. RECOVERY: The program will not reorganize this file.
19. NUMERIC KEY INVALID (STOP)	Record type A, array-type blocking: the key falls within a numeric field. RECOVERY: The program will not reorganize this file.
The following errors are preceded by the general message:	The status of the User File and Key File, being partially reorganized, is not defined at this point. Both files are effectively destroyed.
RESTORE BOTH USER FILE AND KEY FILE FROM BACKUP COPIES BEFORE ATTEMPTING TO RE-RUN THIS PROGRAM	RECOVERY: Recreate User File and Key File from backup copies.
20. PROGRAM ERROR or 21. SEQUENCE ERROR or 22. LAST KEY NOT FOUND	a) The keys in the Key File do not match the keys in the corresponding User File records. b) Machine error. RECOVERY: a) The applications programmer should write a small program to determine which keys do not match. Following FINDFIRST or FINDNEXT, T7$ contains the key value from the Key File. This can be compared to the corresponding key value in the User File. The non-matching keys should

	be corrected or deleted before the reorganization program is run. b) Rerun the program.
23. STOP ACCESS NOT EXCLUSIVE	KFAM-4 ONLY. The User File is in use by another CPU, or a "phantom" entry exists in the ACCESS TABLE. RECOVERY: Wait until other CPU's close the file, or if an erroneous entry is in the ACCESS TABLE,run RESET ACCESS TABLE utility.

CHAPTER 24
THE ADJUST KFAM FILES UTILITIES

24.1 REALLOCATE KFAM FILE SPACE (KFAM-3 AND KFAM-4)

24.1.1 Overview

The utility programs REALLOCATE KFAM FILE SPACE and DISK COPY AND REORGANIZE can be used in conjunction with one another to lengthen or shorten KFAM Key Files and User Files. The latter program can be used alone to copy any disk file.

Non-KFAM data files, maintained with the Catalog Mode statements, keep track of the end of live data with a special trailer record written by the statement DATA SAVE DC END. When saved, this record updates the USED parameter in the disk catalog. The value of this parameter appears in the USED column of a catalog listing. The absolute end of the file area is marked by a fixed control sector whose address appears as "END" in a catalog listing.

KFAM does not use this system for keeping track of the end of live data in the User File, but it does operate within the framework of Catalog Mode files. KFAM automatically puts the DATA SAVE DC END trailer in the next to last sector of the file (the sector immediately preceding the end-of-file control sector). This is done during INITIALIZE KFAM FILE, for both the Key File and the User File. During normal operations KFAM leaves the end-of-data trailer in this location.

KFAM keeps its own file boundary information in the Key File's first record, the KDR. It keeps two items of information for each file: the total number of available sectors and the total number of sectors presently occupied by data. The first item, the total number of available sectors, can have an absolute maximum value which is two less than the total number of sectors allocated to the cataloged file. This is because the DATA SAVE END trailer and the control sector at the end of the file are always present, and can never be used for file records.

Thus, for each file, Key File and User File, there exist four values:

Two Catalog system values:

T	=	total space allocated for the file. This is the total number of sectors occupied by the file, from the starting sector through the ending sector, as recorded in the disk catalog. T = "END" - "START" + 1. T is established by DATA SAVE DC OPEN, and can only be changed by copying the file.

U	=	sectors used by the file. This is the number recorded in the disk catalog for the number of sectors used. With KFAM, this value does not reflect the number of sectors occupied by live data. It is established by the location of the DATA SAVE DC END TRAILER. This is automatically put into the sector immediately preceding the control sector, at the end of the file, by INITIALIZE KFAM FILE.

Two KFAM-maintained values (kept in the KDR record):

K	=	total number of sectors available for file records, K=U-2, since the value of U includes the two sectors of "overhead" (the control sector, and the DATA SAVE DC END trailer sector).
L	=	number of sectors occupied by live data (index records, for the Key File) in a KFAM File. This value is regularly updated by the FINDNEW subroutine. The difference, K-L, is sectors which presently contain no data, but have been set aside for future expansion by KFAM, based on the user's specification of the maximum number of records the file may contain.

REALLOCATE KFAM FILE space changes the value of K, and rewrites the DATASAVE DC END trailer to adjust U, so that the relationship U-2=K is preserved. K may not be made less than L nor greater than T-2. Thus, the purpose of this program is to change the size of a file from the point of view of KFAM's internal control of file space. Of itself, this does not change the cataloged disk space, T, allocated to the file. It is one phase of a two-phase operation to change the actual size of a KFAM file. The other phase is performed by DISK COPY AND REORGANIZE which, by copying a file, can change the value of T.

REALLOCATE KFAM FILE SPACE can change the value of K and U for both a Key File and User File in a single execution of the utility. Generally, if one file size is changed, the other should be changed, proportionally. The files may be on the same disk or different disks. The program displays the "LENGTH", K, "LOW LIMIT", L, and "HIGH LIMIT", T-2, for both the user file and the key file. It then provides the option to change the length, K, of either or both.

To Shorten a KFAM File:

1. Run REALLOCATE KFAM FILE SPACE decreasing the values of K as desired.

2. Run DISK COPY AND REORGANIZE to copy the files into new files that have fewer total sectors, T.

NOTE:

To preserve RECOVERY capability when copying the User File, copy the exact number of sectors specified as "SECTORS USED".

To Lengthen a KFAM File:

1. Run DISK COPY AND REORGANIZE to copy the files into new files that have more sectors, T.

2. Run REALLOCATE KFAM FILE SPACE to increase K for each of the copied files.

> NOTE:
>
> To preserve RECOVERY capability set the User File size to the exact number of sectors specified as "HIGH LIMIT."

For KFAM-4 only, hog mode is selected for the disks containing the User File and the Key File. To execute the utility at a non-multiplexed disk drive key

M$ = "X" (EXEC)

at KFAM-4 utilities menu, prior to loading the utility.

24.1.2 Operating Instructions REALLOCATE KFAM FILE SPACE

DISPLAY	INSTRUCTIONS
1.	1. Mount the disks containing the user file and key file. From KFAM subsidiary menu, access REALLOCATE KFAM FILE SPACE via the indicated Special Function key.
2. ENTER USER FILE NAME (SSSSFJNN)	2. Enter the name of the file. MESSAGE: 2, 4, 5 NOTE: Error messages and recovery procedures follow the operator instructions.
3. ENTER THE NO. OF THE USER FILE DEVICE ADDRESS 1. 310 5. B10 2. 320 6. B20 3. 330 7. B30 4. 350	3. Enter the selection number for the user file device address. MESSAGE: 2, 6

4. ENTER THE NO. OF THE KEY FILE DEVICE ADDRESS.

```
1. 310      5. B10
2. 320      6. B20
3. 330      7. B30
4. 350
```

4. Enter the selection number for the device address of the key file.

MESSAGE: 2, 6

5. ENTER NUMBER OF KEY FILE

5. Normally, enter 1.

If there is more than one key file associated with the user file, enter the number of the key file to be accessed.

MESSAGE: 2, 3, 7

6.

6. The system reads the KDR record of the specified index and displays the current space allocation. It also displays the low and high limits to which it may be changed, for both the user file and the key file.

MESSAGE: 8, 9, 10

7. DO YOU WISH TO REALLOCATE UF SPACE? (Y OR N)

7. To change the user file space allocation,
enter Y or nothing.

To leave the user file space allocation unchanged,
enter N.

If N is entered, the program proceeds to Step 10 below.

8. INPUT NEW UF SECTOR ALLOCATION

8. Enter the number of sectors to be allocated to the user file for the use of KFAM.

NOTE:

To preserve RECOVERY capability when lengthening the file, enter the number of sectors specified as "HIGH LIMIT".

To increase the physical size of a disk file, run the "Disk Copy and Reorganize" program

	first; then run this program to make the increased number of sectors available to KFAM. This program does not increase the space allocated to the file on the disk. It only adjusts internal pointers so that KFAM is able to use more, or less, of that space. MESSAGE: 2, 3, 12
9.	9. The system displays the new sector allocation.
10. DO YOU WISH TO REALLOCATE SPACE FOR KF? (Y OR N)	10. To change the key file space allocation, enter Y or nothing. To leave the key file space allocation unchanged, enter N. If N is entered, the program proceeds to Step 14 below. MESSAGE: 2, 11
11. ENTER NEW KF SECTOR ALLOCATION	11. Enter the number of sectors to be allocated to the key file for the use of KFAM. This program only adjusts internal pointers in the KFAM file. To increase the physical size of a disk file, run "Disk Copy and Reorganize" first. MESSAGE: 2, 3, 12
12.	12. The system displays the new sector allocation.
13.	13. The program makes the internal adjustments to the user file and the key file. MESSAGE: 9, 13
14. DO YOU WISH TO DO ANOTHER FILE? (Y OR N)	14. To do another file, if both user file and key file are already mounted, enter Y or nothing.

To stop, or to do another file which is not already mounted, enter N.

If "Y" is entered, or RETURN(EXEC) alone is keyed, the program proceeds to Step 2 above.

If "N" is entered, the program continues with Step 15 below.

MESSAGE: 2, 11

15.	15. The system returns to KFAM subsidiary menu.

<u>Error Messages</u>

ERROR MESSAGE	EXPLANATION/RECOVERY
2. RE-ENTER	Too many characters were entered. RECOVERY: Repeat the step, entering no more than the number of characters indicated on the screen.
3. ERR29	A non-numeric quantity was entered when a numeric quantity was requested. RECOVERY: Reenter numeric quantity.
4. FILE NAME MUST HAVE F IN POSITION 5	The file name entered does not conform to KFAM naming convention. RECOVERY: Repeat Step 2, entering the name of a KFAM file.
5. FILE NAME MUST HAVE NUMBER IN POSITION 6	The file name entered does not conform to KFAM naming convention. RECOVERY: Repeat Step 2 entering the name of a KFAM file.
6. INVALID DEVICE ADDRESS	An invalid selection number was entered. RECOVERY: Repeat the step, entering a valid selection number.
7. INVALID	The number 0 is not valid for a key file number.

	RECOVERY: Repeat Step 5, entering a number from 1 through 9.
8. ERR80	Either the user file or the key file was not found on the designated platter. RECOVERY: PRINT U1$ for the name of the user file. PRINT K1$ for the name of the key file. Execute LIST DC to determine what files are present on the disk platters. Rerun the program.
9. ERR72	Disk read error, either a machine error or bad data on disk. RECOVERY: Recreate the KFAM file from a backup volume, and rerun the program.
10. STOP RUN KFAM2003 FIRST	Internal pointers indicate the KEY FILE CREATION UTILITY has not been run to build the key file. RECOVERY: Run the KEY FILE CREATION UTILITY; then rerun this program.
11. ANSWER Y OR N	The answer to a "yes" or "no" question must be Y or N, or RETURN(EXEC) alone, indicating "yes". RECOVERY: Repeat the step with the correct response.
12. INVALID-OUT OF BOUNDS	The new length specified for the file is greater than the high limit or less than the low limit. RECOVERY: Repeat the step, entering a number which is within the limits displayed on the screen, for the file (key file or user file).
13. ERR85	Disk write error. The key file and/or user file are destroyed. RECOVERY: Recreate the KFAM files from a backup volume, and rerun this program.

NOTE:

FOR KFAM-4 ONLY

If a file (User File or Key File) is not found, the message FILE NOT FOUND is generated, and not ERR80, as with KFAM-3.

If the file is flagged as being in use, the error message "FILE BUSY" is generated.

"HARDWARE" error messages of the type "ERR XX" are intercepted by the program and the message "ERR XX LINE XXXX" is generated.

Any of the above errors cause the program to stop. To return to the menu following such a stop, key CONTINUE, (EXEC).

24.2 DISK COPY AND REORGANIZE (KFAM-3 AND KFAM-4)

24.2.1 Overview

This program copies a file from one disk to another. It can be used in conjunction with REALLOCATE KFAM FILE SPACE to copy a KFAM file and change its length. It can be used alone to copy any cataloged file to another disk, and in this way can be used to rearrange disk files.

The input disk contains the files to be copied. Files are copied one at a time with intervening operator parameter entries. Any cataloged file may be copied including program files. The utility provides for operator specification of the number of sectors in the output file. However, if the input file is a program file, or a data file with a DATASAVE DC END trailer, the output sectors may not be less than the number of Used sectors in the input file. A data file without a DATASAVE DC END trailer may be copied without this restriction.

The output disk receives the copied files. It must have been initialized before running this program, using the SCRATCH DISK statement. It may contain other files written on it in Catalog Mode prior to execution of this program.

The program uses the cataloging mechanism provided by the system. Copied files begin at the next free sector. File names are entered in the index of the output disk.

This program is an addition to, but not a replacement of, existing hardware functions such as MOVE, COPY, and VERIFY.

To copy complete KFAM files, the key file and the user file must both be copied. However, since Key File and User File can reside on separate disks, this program permits rearranging these files into the most advantageous combinations.

NOTE:

For KFAM-4 only, hog mode is selected for the input disk and output disk. To execute the utility at a non-multiplexed disk drive key

M$ = "X"

at KFAM-4 utilities menu, prior to loading the utility.

24.2.2 Operating Instructions

DISPLAY	INSTRUCTIONS
1.	1. Mount the input and output disks.
2.	2. From KFAM-3 or KFAM-4 menu access DISK COPY/REORGANIZE via the specified Special Function key.
3. ENTER THE NO. OF THE INPUT PLATTER DEVICE ADDRESS 1. 310 5. B10 2. 320 6. B20 3. 330 7. B30 4. 350	3. Enter the selection number of the input disk device address.
4. ENTER THE NO. OF THE OUTPUT PLATTER DEVICE ADDRESS 1. 310 5. B10 2. 320 6. B20 3. 330 7. B30 4. 350	4. Enter the selection number of the output disk device address. MESSAGE: 2, 4, 5 NOTE: Error messages and recovery procedures follow the operating instructions.
5. ENTER FILE NAME	5. Enter the name of the file to be copied. To end the program, key RETURN(EXEC) and go to step 11.

			NOTE: A new input or output platter may be mounted at this point.
			MESSAGE: 2, 6, 7, 14
6.		6.	The system displays the input platter designation, the output platter designation, and the number of sectors available on the output platter.
7.		7.	The system displays the name of the file to be copied, the number of sectors it occupies on the input platter, and the number of sectors used for program or data storage.
8.	ENTER NUMBER OF SECTORS TO BE COPIED	8.	Enter the number of sectors to be allocated to the file on the output platter. MESSAGE: 2, 3, 8, 9, 10
			NOTE: To preserve KEY FILE RECOVERY capability when decreasing the size of User Files, specify the exact number of sectors shown as "SECTORS USED".
9.		9.	The program copies the file from the input platter to the output platter. MESSAGE: 11, 12, 13
10.		10.	The program displays the number of sectors now available on the output platter. Go to step 5.
11.	MOUNT SYSTEM DISK KEY RETURN(EXEC) TO RESUME	11.	Mount the KFAM ISS disk. Key (EXEC).
12.		12.	The system returns to the KFAM menu.

Error Messages

	ERROR MESSAGE	EXPLANATION/RECOVERY
2.	RE-ENTER	Too many characters were entered. RECOVERY: Repeat the step, entering not more than the number of characters indicated on the screen.
3.	ERR29	A non-numeric quantity was entered when a numeric quantity was requested. RECOVERY: Reenter numeric quantity.
4.	INVALID	Platter address designation invalid. RECOVERY: Repeat the step entering a valid selection number.
5.	INPUT AND OUTPUT PLATTERS MUST BE DIFFERENT	The same platter designation was entered for both input and output. RECOVERY: Repeat from Step 3 entering different platter designations for input and output.
6.	FILE NOT FOUND	Input file not found. RECOVERY: Repeat from Step 5. Enter correct input file name.
7.	STOP NO ROOM TO COPY	The number of sectors used by the input file is greater than the number of cataloged sectors available on the output platter. Therefore there is not enough room on the output platter to copy the file. For KFAM-4 only, key (EXEC) to return to the KFAM-4 menu. RECOVERY: 1. Replace the output platter with another platter, with more space available.

2. Or, use MOVE END to increase the cataloged area on the output platter.
3. LIST DC may be used at this point to determine the contents of the output platter.
4. SCRATCH may be used to remove unwanted files from the output platter. This will not free any space, however. To free the space occupied by scratched files, remove the input platter and replace it with a scratch disk, use MOVE to copy the output disk to the scratch disk, and use COPY to copy the new contents of the scratch disk back to the output disk.

   To resume, begin at step 1.

8. LESS THAN SECTORS USED

This program will not copy less than the number of sectors used.

RECOVERY: Repeat Step 13. Enter a number at least as large as sectors used.

To shorten sectors used in a KFAM file, see REALLOCATE KFAM FILE SPACE.

9. GREATER THAN AVAILABLE SPACE

The number of sectors to be copied is greater than the number of sectors available on the output platter.

RECOVERY: Repeat Step 8. Enter a number not greater than available space, as shown in the screen display.

To increase the size of the output cataloged area, see Error Message 7, above.

10. ERR79

A file of the same name is already cataloged on the output platter. Two files of the same name may not be cataloged on the same platter.

RECOVERY: To replace the existing file on the output platter with a new file of the same name,

enter: SCRATCH T#2, N$
key: RETURN(EXEC)
enter: DATASAVE DC OPEN T$#2,N$, "dummy name"
key: RETURN(EXEC)
repeat from Step 1 reentering the name and length of the new file to be copied.

To leave the existing file on the output platter and proceed on to the next file, repeat from Step 1.

11. ERR72

Disk read error.

12. ERR85

RECOVERY: For either ERR 72 or ERR85, the procedure is as follows: To retry the disk read or write operation, enter RUN XXXX, where XXXX is the line number shown with the error message.

If the error persists, either:
a. There is a hardware malfunction,
b. If ERR 72, the input sector is recorded in error, or
c. If ERR 85, the output sector is physically no good.

The presence of a bad sector on either the input or output platter can be detected by using the VERIFY command.

13. STOP DISASTER

Program error. The copy of the file is not completed.

RECOVERY: Rerun from Step 1. Recopy the same file again. If ERR79 occurs, follow the Scratch and Rename Recovery procedure.

14. PROTECTED PROGRAM

An attempt is being made to copy a protected program.

RECOVERY: This program will not copy a protected program.

CHAPTER 25
THE PRINT KEY FILE UTILITIES

25.1 PRINT KEY FILE KFAM-3

This program prints the current contents of the Key Descriptor Record (KDR) and the Key Index Records (KIR) for any KFAM-3 Key File.

Operating Instructions

DISPLAY	INSTRUCTIONS
1.	1. Mount the disk platter containing the KFAM-3 Key File to be printed; mount paper on the printer.
2.	2. Access PRINT KEY FILE via the appropriate Special Function key from the KFAM-3 menu.
3. ENTER FILE NAME	3. Enter the name of the User File with which this Key File is associated, or the name of the Key File itself.
4. ENTER KEY FILE #	4. Enter the number of the Key File to be printed. The Key File Number is normally 1, but may be any digit from 1 to 9 if there are multiple Key Files for one User File. MESSAGE: 3
	NOTE: Error messages and recovery procedures follow the operating instructions.
5. ENTER THE NO. FOR THE KEY FILE DEVICE ADDRESS	5. Enter the selection number for the device address of the Key File.

	1. 310 2. 320 3. 330 4. 350	5. B10 6. B20 7. B30	MESSAGE: 2, 3, 4
6.		6.	The program prints the contents of the Key File. MESSAGE: 5, 6, 7
7.		7.	The system returns to KFAM-3 menu.

<u>Error Messages</u>

	ERROR MESSAGE	EXPLANATION/RECOVERY
2.	RE-ENTER	Too many characters were entered. RECOVERY: Repeat the step entering the correct information.
3.	ERR 29	A non-numeric quantity was entered when a numeric quantity was requested. RECOVERY: Reenter numeric quantity.
4.	INVALID DEVICE ADDRESS	Device address entered was invalid. RECOVERY: Repeat the step. Enter correct device address selection number.
5.	ERR80	File not found. The specified file does not exist on the specified platter. RECOVERY: Make sure the correct platter is mounted. Rerun from Step 2, entering the correct information.
6.	ERR72	Disk read error. RECOVERY: Rerun from Step 2.
7.	ERR43	Wrong file format read. RECOVERY: This program will only print a Key File created under KFAM-3.

25.2 PRINT KEY FILE KFAM-4

This program prints the current contents of any KFAM-4 Key File.

> NOTE:
>
> Hog mode is selected for the device containing the Key File. To execute the utility at a non-multiplexed disk drive key
>
> M$ = "X" (EXEC)
>
> at KFAM-4 utilities menu, prior to loading the utility.

DISPLAY	INSTRUCTIONS
1.	1. From KFAM-4 menu access the PRINT KEY FILE utility via the indicated special function key.
2. ENTER USER FILE NAME (SSSFJNN)	2. Enter the name of the User File with which the Key File is associated. MESSAGE: 4,5
3. ENTER THE KEY FILE NUMBER (NORMAL=1)	3. Enter the Key File Number of the Key File to be printed. MESSAGE: 4,6,7
4. ENTER THE NO. OF THE KEY FILE DEVICE ADDRESS. 1. 310 5. B10 2. 320 6. B20 3. 330 7. B30 4. 350	4. Mount the disk containing the Key File. Enter the selection number (1-7) to choose the device address of the Key File. MESSAGE: 4,6,8
5.	5. The program prints the contents of the Key File. MESSAGE: 1,2,3
6.	6. The system returns to KFAM-4 menu. MESSAGE: 2

Error Messages

MESSAGE	EXPLANATION/RECOVERY
1. ERR80	File not found. RECOVERY: Mount disk containing Key File. Rerun.

2.	ERR72	Disk read error. RECOVERY: Rerun the program. If error persists, program or Key File will have to be recreated from backup.
3.	ERR85	Disk write error. RECOVERY: Rerun the program. If error persists, the disk containing the Key File is bad.
4.	RE-ENTER	Too many characters were entered, or an invalid character was entered in response to a "yes" (Y) or "no" (N) question. RECOVERY: Repeat the step. Re-enter the data.
5.	NOT KFAM FILE NAME	The 5th character of the file name must be "F", and the 6th character must be a zero. RECOVERY: Repeat the step. Enter a valid KFAM file name.
6.	ERR29	Non-numeric data was entered when a numeric quantity was requested. RECOVERY: Repeat the step. Enter a number.
7.	INVALID	Key File number may not be zero. RECOVERY: Repeat the step. Enter a number 1-9.
8.	INVALID DEVICE ADDRESS	The numbers 1-7 may be used to specify a device address, according to the table displayed. RECOVERY: Repeat the step. Enter a number 1-7.

CHAPTER 26
THE RECOVERY UTILITIES

26.1 KEY FILE RECOVERY (KFAM-3 AND KFAM-4)

26.1.1 Overview

If a Key File is destroyed, the Key File Recovery utility permits it to be reconstructed from the data in the User File, provided that application programs that operate on the file adhere to the following conventions:

1) All DELETED records are flagged in the User File with HEX(FF) in the first byte of the key.

2) Programs that execute FINDNEW on the file include the RECOVERY OPTION in all subroutines, and subsequently close the file with the CLOSE subroutine. (Note that under KFAM-4, CLOSE is a system requirement, apart from the RECOVERY requirements.)

The information required to operate the utility is:

1) User File name
2) User File device address
3) Key File number (normally = 1)
4) Key File device address
5) Is the Key File already cataloged?

If the Key File already exists on the designated disk, this utility reuses that file; otherwise, it catalogs a new file with sufficient space to index the maximum number of records in the User File.

If more than one Key File exists for this one User File, it may be impossible to use this utility.

The utility uses a printer (address 215) to list duplicate keys. If no printer is available, or a different printer device address is desired, see Chapter 29.

For KFAM-4 only, the utility selects hog mode for the disk containing the Key File, initializes the Key File, then deselects and leaves hog mode. (To execute the utility at a non-multiplexed disk drive, key

```
M$ = "X" (EXEC)
```

at KFAM-4 utilities menu, prior to loading the utility.) The file is then opened in exclusive mode and the disk is not hogged.

26.1.2 Operating Instructions

DISPLAY	INSTRUCTIONS
1.	1. Mount the disk containing the User File, and the disk to contain the reconstructed Key File.
2.	2. From KFAM-3 or KFAM-4 menu load KEY FILE RECOVERY via the indicated Special Function Key.
3. ENTER USER FILE NAME (SSSSFJNN)	3. Enter the name of the User File for which the Key File is to be reconstructed. MESSAGE: 2,4
	NOTE: Error Messages and recovery procedures follow the operator instructions.
4. ENTER THE NO. OF THE USER FILE DEVICE ADDRESS 1. 310 5. B10 2. 320 6. B20 3. 330 7. B30 4. 350	4. Enter the selection number (1-7) to choose the device address of the User File. MESSAGE: 2,5
5. ENTER THE KEY FILE NUMBER	5. The Key File Number should always be 1, unless there are multiple key files for a single User File, in which case the Key File Number may be any digit from 1 to 9. MESSAGE: 2,6
6. ENTER THE NO. OF THE KEY FILE DEVICE ADDRESS. 1. 310 5. B10 2. 320 6. B20 3. 330 7. B30 4. 350	6. Enter the selection number (1-7) to choose the device address of the Key File. MESSAGE: 2,3,5
7. IS KEY FILE CATALOGED (Y OR N)	7. If the Key File is cataloged at the address selected in step 6, enter Y; otherwise enter N. MESSAGE: 1,2,7,8,9

8. TURN ON PRINTER KEY RETURN(EXEC) TO RESUME	8. Ready the printer. The printer is used to list duplicate keys or unreadable sectors, if any. (If no printer is available, this program must be slightly modified; see Chapter 29.) MESSAGE: 2
9.	9. The system recreates the Key File. MESSAGE: 1,10,11,12,13,14, 15,16,17,18
10.	10. The system returns to the KFAM menu.

<u>Error Messages</u>

ERROR MESSAGE	EXPLANATION/RECOVERY
1. ERROR ## LINE ####	This is the same as ERR ## in BASIC. RECOVERY: Appropriate to error type.
2. RE-ENTER	Too many characters were entered, or the entry was invalid. RECOVERY: Repeat the step, correcting the entry.
3. ERROR 80 LINE 6190	The User File is not mounted on the device specified. RECOVERY: Mount the disk containing the User File and rerun the program.
4. NOT KFAM FILE NAME	The User File name must have an "F" in position 5 and a 0 in position 6. RECOVERY: Repeat step 3. Enter correct User File name.
5. INVALID DEVICE ADDRESS	The device address for a KFAM file must be 310, B10, 320, B20, 330, B30, or 350. RECOVERY: Repeat the step. Enter correct device address.
6. INVALID	The Key File Number may not be 0. RECOVERY: Repeat step 5. Enter a Key File Number 1-9.

7. FILE ALREADY CATALOGED	Key File already exists. RECOVERY: Repeat from step 1. Mount new disk if necessary.
8. FILE NOT FOUND	Key file does not exist on the specified device. RECOVERY: Repeat from step 1. Mount new disk if necessary.
9. NO SPACE ON DISK FOR KEY FILE (STOP)	There is not sufficient space on disk to catalog the Key File. RECOVERY: Mount new disk and rerun.
10. STOP ERROR OPENING FILES	Return code "X" from "OPEN" subroutine. Possible cause: The program was stopped and restarted after processing had begun. RECOVERY: Load and rerun from menu. If the error persists, notify Wang Laboratories.
11. INVALID RECORD FORMAT (STOP)	Record type A, array-type blocking: more than one sector per block, more than 38 fields per record, or not written with correct control bytes. RECOVERY: None. "END" record invalid.
12. NOT BLOCKED AS SPECIFIED	Recovery type A, array-type blocking: records per block specified incorrectly, or records not written in array format. RECOVERY: None. "END" record invalid.
13. RECORD LENGTH NOT SPECIFIED (STOP)	Record type A, array-type blocking: record length specified in INITIALIZE KFAM FILE does not equal record length of sample record. RECOVERY: None. "END" record invalid.
14. KEY FIELD OUT OF BOUNDS (STOP)	Record type A, array-type blocking: the key must be wholly contained within one field of the record. RECOVERY: None. "END" record invalid.

15. NUMERIC KEY INVALID (STOP)	Record type A, array-type blocking: the key falls within a numeric field. RECOVERY: None. "END" record invalid.
16. NO SPACE (STOP)	Return code "S" from FINDNEW(HERE). Not sufficient space for Key File. RECOVERY: Allocate more space for the Key File. Rerun.
17. INVALID POINTER (STOP)	Sector accessed is not in the User File. Probably the "END" record does not contain the necessary information to rebuild the Key File. RECOVERY: Rerun the program. If the error persists, no recovery is possible.
18. SYSTEM HANGS	Printer not turned on or not selected manually, or no device 215 in system. RECOVERY: Turn printer on and press "SELECT". If no device 215, this program will not run without modification. (See "Eliminating the Printer", Chapter 29.)

26.2 RESET ACCESS TABLE (KFAM-4 ONLY)

For KFAM-4 only there is an Access Table included in the Key File (in the KDR). This table is 4 bytes long, one byte for each possible CPU accessing the file. If no CPU is accessing the file, all 4 bytes should be blank. If one or more CPU's are accessing the file, then one byte is set in the Access Table for each CPU currently accessing the file. It is set to "A" or "X" depending on whether the access is shared or exclusive.

When the file is closed, by a particular CPU, the corresponding byte in the Access Table is set back to blank. If, for any reason, such as system failure or power failure, the program is terminated without closing the file, there will remain, in the Access Table, bytes which are not set to blank. If this happens, the file cannot subsequently be opened in the exclusive mode. If a byte happens to remain set to "X" in the Access Table, the file cannot subsequently be opened in either exclusive or shared mode. Also the number of files which can be opened in the shared mode is cut down by the number of bytes in the Access Table which remain set to "A". (The programs assume that these slots represent CPU's which are currently accessing the file.)

This utility is provided to reset the Access Table to blanks, in the event of a program failure or system failure that has left the access table with erroneous non-blank characters. The utility also turns off any "protect flags" that may be left on. PRINT KEY FILE (KFAM-4) shows the current settings of the Access Table.

This utility should not be run if any other CPU is currently accessing the file. The utility has no way of knowing whether entries in the Access Table are "live" or "dead", and resets all access table bytes to blanks indiscriminately. Before running this program, the user should check to make sure that no other CPU is currently accessing the file. Otherwise there could be an unpredictable scrambling of results.

Hog mode is selected for the disk containing the Key File, while it is being read and re-written. To execute the utility at a non-multiplexed disk key

M$ = "X" (EXEC)

at KFAM-4 utilities menu, prior to loading the utility.

The User File is not accessed by this program. Only the Access Table in the KDR is altered.

Operating Instructions

DISPLAY	INSTRUCTION
1.	1. From KFAM-4 subsidiary menu access RESET ACCESS TABLE via the indicated specified Special Function Key. ERROR MESSAGE: 2
2. ENTER USER FILE NAME (SSSFJNN) RESET ACCESS TABLE	2. Enter the User File name. ERROR MESSAGE: 4,5
3. ENTER KEY FILE NUMBER (NORMAL=1)	3. Enter the Key File number. Normally this is 1, unless there are multiple Key Files indexing the same User File. ERROR MESSAGE: 4,6,7
4. ENTER THE NUMBER OF THE KEY FILE DEVICE ADDRESS. 1. 310 5. B10 2. 320 6. B20 3. 330 7. B30 4. 350	4. Enter the selection number for the address at which the Key File is mounted. ERROR MESSAGE: 4,6,8
5.	5. The Access Table is reset. ERROR MESSAGE: 1,2,3

6. DO YOU WISH TO DO ANOTHER FILE? (Y OR N)	6. If the Access Table of another Key File must be reset, enter Y and go to step 2. Otherwise, enter N to return to KFAM-4 subsidiary menu. ERROR MESSAGE: 2

<u>Error Messages</u>

ERROR MESSAGE	EXPLANATION/RECOVERY
1. ERR 80	1. File not found. RECOVERY: Mount disk containing Key File. Rerun.
2. ERR 72	2. Disk read error. RECOVERY: Rerun the program. If error persists, program or Key File will have to be recreated from backup.
3. ERR 85	3. Disk write error. RECOVERY: Rerun the program. If error persists, the disk containing the Key File is bad.
4. RE-ENTER	4. Too many characters were entered, or an invalid character was entered in response to a "yes" (Y) or "no" (N) question. RECOVERY: Repeat the step. Re-enter the data.
5. NOT KFAM FILE NAME	5. The 5th character of the file name must be "F", and the 6th character must be a number 0-9. RECOVERY: Repeat the step. Enter a valid KFAM file name.
6. ERR 29	6. Non-numeric data was entered when a numeric quantity was requested. RECOVERY: Repeat the step. Enter a number.
7. INVALID	7. Key File number may not be zero.

RECOVERY: Repeat the step. Enter a number 1-9.

8. INVALID DEVICE ADDRESS	8. The numbers 1-7 may be used to specify a device address, according to the table displayed. RECOVERY: Repeat the step. Enter a number 1-7.

CHAPTER 27
THE KFAM CONVERSION UTILITIES

27.1 The KFAM-3 CONVERSION UTILITIES

27.1.1 Overview

KFAM-3 includes two KFAM conversion utilities. These are CONVERT KFAM-1 to KFAM-3 and CONVERT KFAM-2 to KFAM-3. They are provided to convert a file created under one of the earlier KFAM's to the format of KFAM-3. The procedures and operating instructions for these programs are the same regardless of which one is being used.

The structure and format of the Key File is not the same for KFAM-3 as for KFAM-1 or KFAM-2. Therefore, the conversion process consists of dropping the old Key File and creating a new Key File, in the KFAM-3 format.

Certain information must be preserved before the old Key File is dropped. Deleted records must be identified, so that they will not be included in the new Key File. The location of the last physical record in the User File must be determined, and the value of the last key must be displayed to define the end of the file. The utility performs these two functions. It does this by using the KFAM subroutines of the KFAM version originally used to organize the file. It extracts the key from each record in the User File. It then executes "FINDOLD" for that key. If the key is not found, or if the pointer in the Key File points to a different location in the User File, it assumes that record is deleted. It flags each deleted record with HEX(FF) in the first byte of the key. In a printed report, it lists the location and key of each deleted record.

If the record is not deleted, the utility checks the key to see whether it violates the restrictions of KFAM-3 (first byte HEX(FF) or the entire key binary zero). It lists on the printed report, as "invalid", any key which violates these restrictions, together with the record location, It also lists on the report the last valid key, so that it may be used to set up the new Key File.

Once the User File has been conditioned in this manner by the conversion utility, INITIALIZE KFAM FILE (KFAM-3 version) and KEY FILE CREATION (KFAM-3 version) utilities are run, to create the new Key File. Any key that is flagged HEX(FF) in the first byte is ignored by KEY FILE CREATION, thereby creating the new Key File only from the active records in the User File.

27.1.2 Conversion Procedure

To convert a file to KFAM-3, three programs must be run, the appropriate "CONVERT" program, INITIALIZE KFAM FILE and KEY FILE CREATION. The overall procedure is as follows:

1. Backup copies should be made of the User File and the old Key File.

2. Mount the User File and the old Key File.

   Execute the appropriate "CONVERT" program.

3. Initialize the new (KFAM-3) Key File:

   Load and execute INITIALIZE KFAM FILE.

4. Build the new Key File:

   Load and execute KEY FILE CREATION UTILITY.

The User File and Key File are now converted and can be accessed via KFAM-3.

If converting from KFAM-1, modifications may be required to user programs to run under KFAM-3. The return codes from the KFAM-1 subroutines have been changed as follows:

Subroutine	Condition	KFAM-1 Subroutines Return	KFAM-3 Subroutines Return
231 DELETE	Key not found	L, H, or N	N
232 FINDOLD	Key not found	L, H, or N	N
235 FINDFIRST	Normal	L	blank
236 FINDLAST	Normal	H	blank

All other return codes are the same as before.

KFAM-2 and KFAM-3 have identical subroutine return codes.

Operating Instructions

DISPLAY	INSTRUCTIONS
1.	1. Make backup copies of the User File and Key File to be converted.
2.	2. Mount the disk platters containing the User File and Key File to be converted.
3.	3. From KFAM-3 menu, depress the appropriate Special

			Function key to access CONVERT KFAM-1 to KFAM-3 or CONVERT KFAM-2 to KFAM-3.
4.	ENTER USER FILE NAME (SSSSFJNN)	4.	Enter the User File name. MESSAGE: 2, 4 NOTE: Error messages and recovery procedures follow the operating instructions.
5.	ENTER THE NO. OF THE USER FILE DEVICE ADDRESS 1. 310 5. B10 2. 320 6. B20 3. 330 7. B30 4. 350	5.	Enter the selection number for the user file device address. MESSAGE: 2, 3, 6
6.	ENTER KEY FILE NUMBER (NORMAL=1)	6.	Enter the Key File Number. The key file number should always be 1, unless there are multiple key files for a single User File, in which case, the Key File Number may be any digit from 1 to 9. MESSAGE: 2, 3, 5
7.	ENTER THE NO. OF THE KEY FILE DEVICE ADDRESS 1. 310 5. B10 2. 320 6. B20 3. 330 7. B30 4. 350	7.	Enter the selection number of the key file device address. MESSAGE: 2, 3, 6
8.	"KFAM FILE CONVERSION (KFAM5000)" etc.	8.	The files are opened. MESSAGE: 7, 8, 9, 10, 15
9.	TURN ON PRINTER KEY RETURN(EXEC) TO RESUME	9.	Mount paper in the printer. Key RETURN(EXEC) to resume. MESSAGE: 2

10.	10. The system proceeds with the conversion. MESSAGE: 12, 13
11. DO YOU WISH TO DO ANOTHER FILE? (Y OR N)	11. To convert another file, enter Y and go to step 4. Otherwise, enter N and go to the next step. MESSAGE: 2
12.	12. The system returns to the KFAM-3 menu.

<u>Error Messages</u>

ERROR MESSAGE	EXPLANATION/RECOVERY
2. RE-ENTER	Too many characters were entered, or an invalid character was entered in response to a "yes" (Y) or "no" (NO) question. RECOVERY: Repeat the step, entering the correct value.
3. ERR29	A non-numeric quantity was entered when a numeric quantity was requested. RECOVERY: Reenter numeric quantity.
4. NOT KFAM FILE NAME	The User File name must have an "F" in position 5 and a zero in position 6. RECOVERY: Repeat Step 4. Enter correct User File name.
5. INVALID	The Key File number may not be 0. RECOVERY: Repeat Step 6. Enter Key File number 1-9.
6. INVALID DEVICE ADDRESS	The device address is invalid. RECOVERY: Reenter correct selection number.
7. STOP ERROR OPENING FILES	This should not occur. Either the program is being rerun without closing the file, or there is an error in the Key File. RECOVERY: Recreate the Key File from a backup. Rerun the program from Step 1.

8. NOT EVEN MULTIPLE	This should not occur. The total sectors occupied by the User File is not an even multiple of the number of sectors per record. RECOVERY: Recreate the Key File from a backup copy. Rerun the program from Step 1.
9. ERR80	Files not found. Either the User File or the Key File does not exist on the specified platter. RECOVERY: Rerun the program from Step 4. Enter correct information.
10. ERR72	Disk read error. If the error persists, either the User File (#1) or the Key File (#2) is permanently damaged. RECOVERY: Rerun the program from Step 1.
11. ERR85	Disk write error. If the error persists, the platter is physically damaged. RECOVERY: Recreate both the User File and the Key File from backup copies. Rerun the program from Step 1.
12. System hangs	Either the printer is not turned on, or there is no device 215 in the system. RECOVERY: Press both "ON/OFF" and "SELECT" to turn on the printer. Both should be lighted. If no printer, the program must be modified in order to run (see Chapter 29, "Eliminating the Printer").
13. ERROR X	This should not occur. The FINDOLD subroutine reports an improper call. RECOVERY: Rerun from Step 1. If the error persists, notify Wang Laboratories, Inc.
14. ERR 43	Wrong record format read. The Key File is not in the original KFAM format.

	RECOVERY: Check the system which created the Key File. Rerun from Step 3.
15. INVALID RECORD FORMAT or NOT BLOCKED AS SPECIFIED or RECORD LENGTH NOT SPECIFIED CORRECTLY or KEY FIELD OUT OF BOUNDS or NUMERIC KEY (STOP)	Record type A array type blocking: record format or key position is invalid. RECOVERY: File cannot be converted.

27.2 THE KFAM-4 CONVERSION UTILITY

The KFAM-4 system includes a utility program to convert from KFAM-3 to KFAM-4. The structure of the Key File's key index entries is identical from KFAM-3 to KFAM-4. Only the Key File's special header record, the KDR, differs, and, therefore, only this needs to be altered to convert a KFAM-3 file to a KFAM-4 file. The utility CONVERT KFAM-3 TO KFAM-4 performs the necessary alteration of the KDR.

To convert a file from KFAM-1 or KFAM-2 to KFAM-4, it must first be converted to KFAM-3 using the KFAM-3 conversion utilities.

Application programs written for KFAM-3 will require modification to operate on KFAM-4 files. See Chapters 18, 22 and 29 for KFAM-4 programming procedures.

NOTE:

The utility selects hog mode for the disk device containing the key file to be converted. To execute the utility at a non-multiplexed disk drive key

M$ = "X" (EXEC)

at KFAM-4 utility menu, prior to loading the utility.

Operating Instructions

1.	1. From the KFAM-4 menu access CONVERT KFAM-3 TO KFAM-4 via the specified Special Function Key.
2. ENTER USER FILE NAME (SSSSFJNN)	2. Enter the name of the User File to be converted. MESSAGE: 4,5

3.	ENTER KEY FILE NUMBER (NORMAL=1)	3.	Enter the Key File Number. Normally this is 1, except if there is more than one Key File for the User File. MESSAGE: 4,6,7
4.	ENTER THE NO. OF THE KEY FILE DEVICE ADDRESS 1. 310 5. B10 2. 320 6. B20 3. 330 7. B30 4. 350	4.	Enter the selection number (1-7) to choose the Key File device address. MESSAGE: 4,6,8
5.		5.	The utility converts the Key File from KFAM-3 to KFAM-4 format. MESSAGE: 1,2,3
6.	DO YOU WISH TO DO ANOTHER FILE (Y OR N)	6.	Enter Y to repeat program for another file; go to step 2. Enter N to return to KFAM-4 menu. MESSAGE: 4

Error Messages

ERROR MESSAGE		EXPLANATION/RECOVERY
1.	ERR 80	File not found. RECOVERY: Mount disk containing Key File. Rerun.
2.	ERR72	Disk read error. RECOVERY: Rerun the program. If error persists, program or Key File will have to be recreated.
3.	ERR85	Disk write error. RECOVERY: Rerun the program. If error persists, the disk containing the Key File is bad.
4.	RE-ENTER	Too many characters were entered, or an invalid character was entered in response to a "yes" (Y) or "no" (N) question. RECOVERY: Repeat the step. Re-enter the data.

5.	NOT KFAM FILE NAME	The 5th character of the file name must be "F", and the 6th character must be a number 0-9. RECOVERY: Repeat the step. Enter a valid KFAM file name.
6.	ERR29	Non-numeric data was entered when a numeric quantity was requested. RECOVERY: Repeat the step. Enter a number.
7.	INVALID	Key File number may not be zero. RECOVERY: Repeat the step. Enter a number 1-9.
8.	INVALID DEVICE ADDRESS	The numbers 1-7 may be used to specify a device address, according to the table displayed. RECOVERY: Repeat the step. Enter a number 1-7.

CHAPTER 28
GENERAL TECHNICAL INFORMATION

28.1 KEY FILE RECORD LAYOUTS

The first sector of the key file contains the Key Descriptor Record (KDR). The remaining sectors contain Key Index Records (KIR's), as many as are necessary to index the User File. The layouts of the KDR's of KFAM-3 and KFAM-4 are different; both are given below. The layouts of the KIR's are identical for KFAM-3 and KFAM-4.

Key Descriptor Record (KDR)--KFAM-3

Variable Name	Bytes On Disk	Contents
Q2$2	3	Last data sector (last sector used for data in User File, relative to starting sector = 0, hex number).
Q3$2	3	Data sector limit (last sector available for data in User File, relative to starting sector = 0, hex number).
V5$1	2	Record number within sector, last slot used for data in User File, hex number. (First record within sector = 1.)
V8$1	2	Records per block, hex number. (Set to 1 for type M or N records.)
V0$2	3	Key File, absolute address (hex) of starting sector.
V1$8	9	Byte 1: Record type, A, C, M, or N. Byte 2: Record length (hex) if type A or C. Bytes 3, 4: Starting position of key (hex) relative to first byte of first sector of record = 0. Byte 5: Key length (hex). Byte 6: The number of KIE's per KIR (hex). Bytes 7, 8: Not used.
V2$2	3	Key File, last sector used (hex), relative to starting sector = 0.
V3$2	3	Key File, last sector available (hex), relative to starting sector = 0.
V6$1	2	Sectors per logical record (hex). (Set to 1 for type A or C records.)
T2$2	3	Sector address of highest level index sector (hex), relative to starting sector of Key File = 0.

T0	9	Number of index levels.
T1	9	Current Key File # (file number assigned in SELECT statement).
T2	9	Current User File # (file number assigned in SELECT statement).
V8	9	Bias for splitting KIR expressed as a percentage of the number of KIE's which can be contained in the KIR, range .2 to .8.
T4$3	4	Last record accessed, pointer: Bytes 1, 2: Relative sector within User File. Byte 3: Record number (hex) within sector.
T5$30	31	Last key added to file.
T7$30	31	Last key accessed.
T2$(8)2	24	Path to last record accessed: Sector address, of KIR's, from level T0 down to level 1.
T(8)	72	Path to last record accessed. Number of KIE within KIR, from level T0 down to level 1.
T8$1	2	Internal completion code: Same as Q$, except: Following DELETE, O.K., T8$="1". Following FINDOLD, not found, T8$ = "2". Following FINDNEW or FINDNEW (HERE), O.K., T8$ = "3". Following OPEN, O.K., T8$=letter "O". Following CLOSE, O.K. T8$ not defined.
TOTAL	233 bytes	

Key Descriptor Record (KDR) (KFAM-4)

Variable Name	Bytes on Disk	Contents
Q2$2	3	Last live data sector (last sector used for data in User File, relative to starting sector = 0, hex number).
Q3$2	3	Data sector limit (last sector available for data in User File, relative to starting sector = 0, hex number).
V5$(4)1	8	Per CPU, record number within sector, last slot used for data in User File, hex number. (First record within sector = 1.) Initialized to V8$.
V8$1	2	Records per block, hex number. (Set to 1 for type M or N records.)

V1$8	9	Byte 1: Record type, A, C, M, or N. Byte 2: Record length (hex) if type A or C. Bytes 3, 4: Starting position of key (hex) relative to first byte of first sector of record = 0. Byte 5: Key length (hex). Byte 6: Number of KIE's per KIR. Bytes 7-8: Not used.
V2$2	3	Key File, last sector used (hex), relative to starting sector = 0.
V3$2	3	Key File, last sector available (hex), relative to starting sector=0.
V6$1	2	Sectors per logical record (hex). (Set to 1 for type A or C records.)
T2$2	3	Sector address of highest level index sector (hex), relative to starting sector of Key File = 0.
T0	9	Number of index levels.
T8$(4)1	8	Per CPU, internal completion code: Same as Q$, except: Initialized to "Z" by KFAM1004, KFAM1005, or KFAM7004. Following FINDOLD, not found, = "2". Following FINDNEW, FINDNEW(HERE), or DELETE, all slots with a value of "9" or less are set to "3". Following OPEN, successful, set to letter "O". Following CLOSE, successful, set to "Z". If Q$="B", set to "9".
Q0$4	5	Access table: one byte slot per CPU. Byte is blank if this slot not used. "A" if access is shared. "X" if access is exclusive. The slot assigned to a particular CPU varies. It is determined by the first blank character of Q0$ at the time the particular CPU opens the particular file. This slot number also becomes the subscript for V5$(), T8$(), V4$(), and V2$().
V4$(4)2	12	Per CPU, protected sector. Hex (FFFF) if no protected sector.
V2$(4)2	12	Per CPU, data sector for FINDNEW. V5$() defines the record within the sector. The combination of V2$() and V5$() defines the last physical location assigned to a new record by this CPU via FINDNEW. The value of V2$(), per CPU, is taken from Q2$, which is the last sector address used by any CPU.
Total	82 bytes	

<u>Key Index Record (KIR) (KFAM-3 AND KFAM-4)</u>

Variable Name	Bytes On Disk	Contents
T9$2	3	Sector address (hex), this sector, relative to first sector of Key File = 0.
T0$(4)60	244	A 240 byte array containing KIE's. Number of KIE's per KIR can vary from 7 to 60. Unused KIE's are

filled with all bytes HEX(FF). Active KIE's are packed as follows:

K bytes: key
3 bytes: pointer

Pointer points to next lower index level KIR or User File record if lowest level. The first two bytes of the pointer contain the sector address (hex) relative to the start of the file. The last byte contains the record number (hex) within the sector if the pointer is to a data record, and is not defined if the pointer is to a lower level KIR.

TOTAL 247 bytes

Internal Storage Under KFAM-4

Certain information that is stored in the KDR under KFAM-3 is stored internally in CPU memory under KFAM-4. It is stored in 3-element arrays for which the KFAM ID number serves as a subscript, (T9=KFAM ID Number), and in some cases is stored in scalar variables for the most recently active file. The internal storage fields are as follows:

Most Recently Active File	Per KFAM I.D. Number	Contents
V0	V0(3)	CPU number assigned when the file was opened (see KDR, Q0$).
V0$2	V0$(3)2	Absolute starting sector (hex) of the Key File, or HEX(FFFF) if file not open.
T1	T1(3)	Key File # (file number assigned in SELECT statement).
T2	T2(3)	User File # (file number assigned in SELECT statement).
T4$3	T4$(3)3	Last record accessed, pointer: Bytes 1,2: Relative sector within User File. Byte 3: Record number (hex).
T7$30	T7$(3)30	Last record accessed, key.
T2$(8)2	T6$(3)16	Path to last record accessed: Sector addresses of KIR's from highest level down to level 1. Subscript of T2$() corresponds to index level. Subscript of T6$() is file I.D.
T$8	T$(3)8	Path to last record accessed: Starting byte of KIE within KIR, from highest level down to level 1 (hex). Byte within T$ corresponds to index level.
--	V3$(3)1	Access mode, "A" = shared, "X" = exclusive.

--	V8(3)	Bias for splitting KIR. Reset to .5 when the file is opened.
--	T5$(3)30	Last key added to file.

28.2 KEY FILE STRUCTURE

The structure of the Key File in KFAM-3 and KFAM-4 is the same. It is similar to the structure called a B-tree, which is discussed on pages 473-479 of THE ART OF COMPUTER PROGRAMMING: Volume 3/Sorting and Searching, by Donald E. Knuth.

The problem is to create a Key File that permits rapid access to any particular User File record, and may also be updated at any time without a major reorganization of the file. The B-tree structure, as modified, satisfies this double requirement.

The structure of the Key File is best described by showing how the file is constructed. The first step, in INITIALIZE KFAM FILE, is to create one KIR record, which contains one dummy KIE with a key value of binary zero (all bytes HEX(00)). This dummy KIE serves to "prime" the system so that the same program logic can be applied to a null, or empty, file as is applied to a file containing active records. Being the lowest possible key, it also serves to mark the lower limit of the Key File. For example, FINDFIRST (see KFAM subroutines in Programming Aids Section) is done by searching for the binary zero key and then doing FINDNEXT. This dummy key can be thought of as the 0th entry in the Key File. Of itself, it represents nothing, except as the marker of the lower boundary.

In the examples below, this dummy key is designated as "000". Please note that the actual value is binary zero, HEX(000000), and not the characters "000", or HEX(303030). The characters "000" may be used as an active key, and will not conflict with the dummy key.

The unused KIE's in any KIR always have all bytes set to HEX(FF). Thus the original KIR record has the first key set to all HEX(00) and the remaining keys set to all HEX(FF).

In the examples below, these unused keys are designated as "FFF." Please note that the actual value is HEX(FFFFFF...) and not the characters "FFF," or HEX(464646).

Two items in the KDR record are essential to searching the Key File. One is the number of index levels, T0. To start with, T0 = 1, because there is only one level of index. The other item is the relative sector address of the highest level index, T2$. At the starting point, there is only the one index sector, the KIR record described above, and its sector address is always HEX(0001). (The KDR record always occupies sector HEX(0000), or the first sector of the Key File, and the initial KIR follows it, in the second sector, at relative address HEX(0001).)

The Key File is now set up to begin entering active KIE's. As new keys are added to the file, the respective KIE's are inserted in the KIR in their proper sequential order. Higher keys are moved up one position, and one HEX(FF) key is dropped off the end.

For example, if the first three keys to be inserted are 276, 913, and 198, the KIE's would be arranged as follows:

```
Start:      000, FFF, FFF, etc.
First Key:  000, 276, FFF, etc.
Second Key: 000, 276, 913, FFF, etc.
Third Key:  000, 198, 276, 913, FFF, etc.
```

Keys are inserted in the first KIR in this manner until it is filled. The number of keys per KIR depends upon the size of the key. Let us assume for this example that the first KIR has been completely filled by one dummy key plus 14 active keys:

```
000, 009, 147, 198, 276, 292, 589, 591, 671, 710, 730,
809, 851, 903, 913
```

At this point the key 796 is to be added. Since there is no room in the one KIR to add another key, the KIR is split in two. A new KIR is created, and the KIE's are divided between the old KIR and the new KIR:

```
Old KIR:  000, 009, 147, 198, 276, 292, 589, 591, FFF, etc.
New KIR:  671, 710, 730, 796, 809, 851, 903, 913, FFF, etc.
```

The new KIR occupies relative sector HEX(0002). Note that the key added, 796, is inserted in its proper sequential order, which in this case just happens to fall in the new KIR.

With more than one KIR now in the file, the concept of "level" enters in. Both KIR's so far created are on level 1, the lowest level. The lowest level is defined as the level which contains the pointers to the data records in the User File. Whenever a KIR is split, the new KIR is on the same level as the old KIR.

Rather than search the KIR's sequentially for a given key, the system searches via a tree structure. There is one and only one KIR at the highest level. Its sector address is recorded in the KDR. The search is started by reading this sector. Up to this point, the search has been complete by locating the position of the key within the one sector. But at this point, there are two KIR's on level 1, and a higher level index must be created to reference them.

Therefore a third KIR is created. It is a level 2 index. It contains two keys, 000 and 671, which are the first keys of each of the two level 1 KIR's. The pointers associated with these two keys are the relative sector addresses of the two level 1 KIR's, which happen to be, by coincidence, HEX(0001) and HEX(0002). This 2nd-level KIR is stored in relative sector HEX(0003) of the Key File, and its contents are:

Keys:	000, 671, FFF, etc.
Pointers:	1, 2, FFF, etc.

The KDR is now updated: T0 = 2, to show that the index now has 2 levels; T2$ = HEX(0003), to show that the highest level index is located at relative sector HEX(0003).

Assuming that the next key to be added is 562, the search now proceeds as follows. 562 is compared to the entries in the level 2 index, to see where it falls. It is greater than or equal to 000, but less than 671. Therefore it falls in the range 000 to 670. The pointer associated with 000 in the level 2 index is HEX(0001), and therefore the level 1 index stored in relative sector HEX(0001) is read. Then 572 is inserted in its proper place in the level 1 index, as before. The system knows when it has reached level 1, because it is counting down from T0 to 1 as each level is read and searched.

When the key 562 has been added, the key file structure looks like this:

Sector	Level	Keys
1	1	000, 009, 147, 198, 276, 292, 562, 589, 591, FFF, etc.
2	1	671, 710, 730, 796, 809, 851, 903, 913, FFF, etc.
3	2	000, 671, FFF, etc.

As further keys are added, the KIR's on level 1 will again become full, and again the KIR must be split to provide room for all the keys. Let us assume that keys 401, 402, 403, 404, 405, 406, and 407 are added. The first six keys will cause sector 1 to be full, and the addition of 407 will make a split necessary. Relative sector HEX(0004) will be assigned to the new KIR, and the resulting structure will look like this:

Sector	Level	Keys
1	1	000, 009, 147, 198, 276, 292, 401, 402, FFF, etc.
2	1	671, 710, 730, 796, 809, 851, 903, 913 FFF, etc.
3	2	000, 403, 671, FFF, etc.
4	1	403, 404, 405, 406, 407, 562, 589, 591, FFF, etc.

Note that no new level has been added this time. In this example, there is room in the level 2 index to reference up to 15 level-1 KIE's. Therefore at least 15 x 8, or 120 records (and probably more, up to 225) can be accessed by a two-level index search.

Once the second level index is full, it is split, the same way the original KIR was split, and a third level is created, pointing to two 2nd-level KIR's, which in turn point to the first-level KIR's. The first level KIR's always contain the pointers to the actual data records. As new levels are added, more superstructure is added, but the bulk of the Key File remains the same.

If for a given key file there is an average of 10 KIE's per KIR, the number of records which can be accessed by a given number of levels of index is as follows:

INDEX LEVELS	NUMBER OF RECORDS
1	14
2	150
3	1500
4	15,000
5	150,000
6	1,500,000
7	15,000,000
8	150,000,000

For the largest possible key (30 bytes), each KIR holds a maximum of 7 KIE's and a guaranteed average minimum of 4 KIE's. For such a file the maximum 8 levels of index access at least 114,687 records.

Perhaps the best illustration of the Key File structure for a large file could be obtained by running PRINT KEY FILE with an actual KFAM file. The structure can then be traced from the highest level index sector (T2$, in KDR) down to the level 1 pointers to the actual data record.

The general procedure for locating a key in KFAM is as follows:

1) The number of index levels (T0) and the relative sector address of the highest level index (T2$) are taken from the KDR.

2) The index sector (KIR) is read from disk.

3) A search of the KIR is made to locate the key. The search returns a pointer (T) to the key in the KIR which is equal to, or next lower than, the key being searched.

4) The relative sector address of the KIR and the pointer to the KIE found (T) are stored in tables, T2$(T3) and T(T3), defining the path taken to locate the particular key, where T3 is the current index level.

5) If the current index level is greater than 1, the sector address for the next lower level index is taken from the KIE found (T), and the process is repeated from Step 2, above, for the next lower level.

6) If the current index level is 1, then the search is finished. T points to a KIE on level 1, and V indicates whether the key found is equal to or lower than the key being searched. Control is returned to the particular subroutine (FINDOLD, FINDNEW, DELETE, etc.).

The general procedure for inserting a key is as follows:

1) The proper position for the key is determined by the search procedure, above.

2) If the KIR is not full, the key and its associated record pointer are inserted at location T+1 in the KIR. All KIE's from location T+1 and up are moved up one position.

3) If the KIR is full, a new KIR is created, on the same level as the old KIR. The KIE's are divided between the old KIR and the new KIR. The new key and its associated record pointer are inserted in proper sequential order in either the old KIR or the new KIR, depending on where the new key happens to fall. The next available sector address in the Key File is assigned to the new KIR.

4) If the split is not at the highest index level, the first key and the sector address of the new KIR are inserted in proper key sequence in the next highest level KIR (as determined by tables T2$() and T()). If the next highest level KIR is full, Step 3 is repeated at that level.

5) If the split is at the highest index level, a new level is created. A new KIR is created, with two KIE's. The first KIE contains the binary zero key and the relative sector address of the old KIR (formerly the highest level KIR). The second KIE contains the first key and sector address of the new KIR (created by the split). The next available sector in the Key File is assigned to this new highest level index. The KDR is updated (T0 and T2$) to reflect the new level.

When the KIR is split, it is not always divided equally. There is a reason for this. Consider keys which are being added sequentially. Again assume the first index sector is filled by 14 active KIE's and one dummy KIE.

```
000, 001, 002, 003, 004, 005, 006, 007, 008, 009, 010
011, 012, 013, 014
```

The next key added, 015, causes a split:

```
Old KIR:  000, 001, 002, 003, 004, 005, 006, 007, FFF, etc.
New KIR:  008, 009, 010, 011, 012, 013. 014, 015, FFF, etc.
Level 2:  000, 008, FFF, etc.
```

The next keys added, 016, 017. etc. are all added to the new KIR eventually causing it to be split:

Sector	Level	Keys
1	1	000, 001, 002, 003, 004, 005, 006, 007, FFF, etc.
2	1	008, 009, 010, 011, 012, 013, 014, 015 FFF, etc.
3	2	000, 008, 016, FFF, etc.
4	1	016, 017, 018, 019, 020, 021, 022, 023, FFF, etc.

The process continues, always adding to the latest KIR and splitting it, leaving behind a residue of KIR's which are only half full. It should be clear in this case that if the split were 12/4 instead of 8/8, the process of indexing a sequential file would leave behind a residue of KIR's each containing 12 KIE's or 80% full. This would result in better utilization of Key File space and also tend to reduce the number of index levels required to access a given file.

But a 12/4 split would be disastrous if the keys were being added at random. There would be a greater probability of new keys being added to the KIR's already containing 12 entries, because of the greater range of values represented. So the Key File could actually fall below 8 keys per sector, and a very inefficient skew distribution would be the result.

Therefore there is no particular split that is best in all cases. Because of this, a moving bias has been included in the system. As each new key is added, the program checks whether it is higher or lower than the previous key added. If it is higher, the bias is adjusted downward. If it is lower, the bias is adjusted upward. The bias is a percentage of the maximum number of KIE's which, for a particular key size, can be contained in a KIR. When a KIR must be split the current bias percentage is multiplied by the maximum number of KIE's per KIR to give the split, i.e., the number of KIE's which go into the new KIR. The range of the bias is .2 to .8.

On the basis of past experience, the system determines the best possible split, based on the order in which keys are added sequentially, mostly sequentially, random or backwards. The bias tends to approach .2 as keys are added sequentially, and tends to stay at .5 if keys are added in random order.

The bias, V8, is initially set to .5. It is adjusted upwards or downwards by 2% of the distance to .8 or .2 as each new key is added, depending on whether the key is lower or higher than the previous key added.

In REORGANIZE KFAM FILE where it is known that keys will be added sequentially, the bias is set to .2 at the begining. It is reset to .5 following the reorganization.

In KEY FILE CREATION the bias is set to .5 initially, and reset to .5 when the program is finished, because the order of keys added when initially creating the Key File could very well be different than the order of keys added at some later time (for example, sequential vs. random). The random hypothesis is always the "safest" to start with, unless experience proves differently.

Between the creation of the Key File and the reorganization, if any, the bias is allowed to fluctuate on the basis of how keys are added. It is stored in the KDR, and preserved as a permanent record, i.e., not reset every time the program is reloaded.

In summary, for KFAM, there are two minor departures from the B-tree structure as described in Knuth: First, keys are duplicated in higher level indexes, and second, a bias is introduced for the splitting of KIR's.

28.3 KEY FILE RECOVERY INFORMATION

KEY FILE RECOVERY utilities are provided for KFAM-3 and KFAM-4 to reconstruct a Key File in the event of its accidental destruction. In reconstructing the Key File these utilities use information saved by the CLOSE subroutine in the next-to-last sector of the User File.

At the end of a User File are two sectors of "overhead". The last sector is a control sector written by the DATA SAVE DC OPEN statement. In the next-to-last sector is a "trailer" record written by DATA SAVE END during the INITIALIZE KFAM FILE utility. Two control bytes in this trailer record mark it as a trailer record for the 2200 system; however the remaining bytes are ignored by the 2200 system logic. Some of these remaining bytes are used by KFAM to store recovery information. The information is stored each time the CLOSE subroutine is executed (provided that the RECOVERY OPTION has been chosen during BUILD SUBROUTINE MODULE).

The data saved by CLOSE in the next-to-last sector of the User File is taken from the Key File's KDR record, and is as follows:

KFAM-3

Bytes	Contents
1-2	HEX(AOFD)
3-4	Q2$2 = last sector used, User File
5	V5$1 = last record within last sector
6	V8$1 = records per block
7-14	V1$8 = record type, record length, starting position of key, key length, number of KIE entries per sector
15	V6$1 = sectors per record

KFAM-4

Bytes	Contents
1-2	HEX(AOFD)
3-4	Q2$2 = last sector used, User File
5-8	V5$(4)1 = per CPU, last record used in sector assigned for FINDNEW
9	V8$1 = records per block
10-17	V1$8 = record type, record length, starting position of key, key length, number of KIE entries per sector
18	V6$1 = sectors per record
19-26	V2$(4)2 = per CPU, sectors assigned for FINDNEW

This data saved by CLOSE is written in the next-to-last sector of the User File. This should always be the DATA SAVE DC END trailer. However, if the rules for shortening and lengthening files given in Chapter 24 are not carefully adhered to, the "trailer" may end up in some other sector. This could cause the RECOVERY utility to fail in some cases.

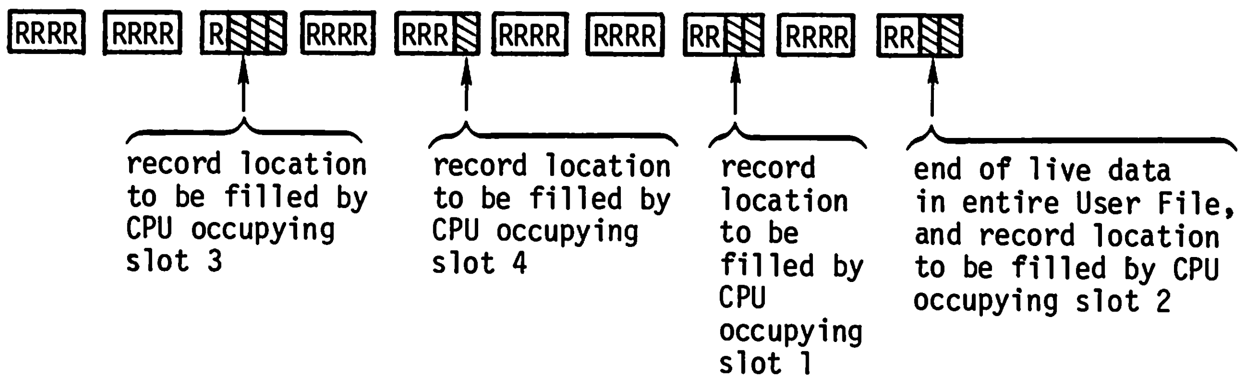
28.4 FINDNEW WITH BLOCKED FILES UNDER KFAM-4

Under KFAM-3 FINDNEW always sets the Current Sector address for the User File to the next available sector at the end of the live data in the User File. If records are blocked (type A or C), it passes back the next record location as well. With blocked files under KFAM-4, the operation of FINDNEW is more complex.

Under KFAM-4 up to four CPU's can have a KFAM file open simultaneously. When a CPU executes OPEN for a file,it is assigned one of the four slots in the KDR's Access Table. Associated with each slot in the Access Table is a relative sector location, and, for blocked files, a record number within that sector. This sector location and record number always point to the last location in the User File assigned when a CPU, occupying that Access Table slot, executed a FINDNEW. If, after OPENing a blocked file, a CPU executes FINDNEW, the location passed to it (sector and record location within the sector) will be the next available location after the last location given to a CPU occupying the same Access Table slot. This new location will be the sector following the last sector of live data in the file only if a new block must be started.

In summary, when using blocked files with KFAM-4, whenever a new block must be used, FINDNEW assigns an entire block to a particular Access Table slot. That block then becomes the exclusive property of that slot in the Access Table for the purpose of FINDNEW. It can only be filled by FINDNEW's executed by a CPU occupying that slot. The result is that all record locations up to the end of live data in the User File may not be filled at any one time.

For blocked files under KFAM-4, the User File might look like this, for example,


RRRR
RRRR
R▨▨▨
RRRR
RRR▨
RRRR
RRRR
RR▨▨
RRRR
RR▨▨
record location to be filled by CPU occupying slot 3
record location to be filled by CPU occupying slot 4
record location to be filled by CPU occupying slot 1
end of live data in entire User File, and record location to be filled by CPU occupying slot 2

where: R = record
▨ = unoccupied record locations

28.5 COMPATIBILITY BETWEEN KFAM-1 AND KFAM-3

KFAM-3 is upwards - compatible with the original KFAM-1 with the following exceptions:

1. The structure of the Key File has been completely revised. There is a conversion program to facilitate the conversion of files organized under KFAM-1 to KFAM-3.

2. Restrictions are placed on the key. The first byte of the key may not be HEX(FF). The entire key may not have a value of binary-zero (the lowest possible value).

3. The following alterations have been made to the Return Codes in the KFAM-3 incorporable subroutines.

   a. KFAM-3 returns Q$=blank in all cases if the subroutine was executed properly. Formerly, FINDFIRST returned "L" and FINDLAST returned "H" for normal execution. This will require a program change in any user program using FINDFIRST or FINDLAST.

   b. Error codes "H" and "L" have been dropped in KFAM-3. Code "N" means "Not Found" in DELETE and FINDOLD. This will require a change to any user program testing Q$= "H" or Q$= "L" following a DELETE or FINDOLD.

   c. All other Return Codes remain as before.

CHAPTER 29
KFAM ADVANCED PROGRAMMING TECHNIQUES

29.1 ELIMINATING THE PRINTER IN KFAM-3 AND KFAM-4

Certain utilities in KFAM use a printer (device 215). The printer is used only for marginal functions, and can be eliminated with some slight modifications if the particular System does not include a printer.

The printer is used in the following modules, for the following purposes:

INITIALIZE KFAM FILE: Hard copy printout of file description.

KEY FILE CREATION: Print duplicate keys and record locations.

CONVERT KFAM-1 TO KFAM-3, CONVERT KFAM-2 TO KFAM-3: Prints deleted keys and invalid keys and their respective record locations. Prints last key. Prints record counts.

KEY FILE RECOVERY: Prints duplicate keys and unreadable sectors.

PRINT KEY FILE: Print the contents of the Key File.

PRINT KEY FILE must be eliminated entirely if there is no printer, since its sole function is to print the contents of the Key File. This function is useful when programs are being tested, but is not necessary for the running of KFAM.

INITIALIZE KFAM FILE may be run as written, but when the system asks "DO YOU WANT A HARD COPY PRINTOUT OF FILE DESCRIPTION? (Y OR N)", the operator must always enter "N" for "no".

The other utilities must be modified, though the changes, as described below, are minor.

KEY FILE CREATION

The KEY FILE CREATION uses a printer to log duplicate keys, because duplicate keys are otherwise ignored in the construction of the Key File. Without some indication that a duplicate key was encountered, and some record of where it was encountered, the data identified by the duplicate key would be lost. Because duplicate keys are not allowed in KFAM, the recovery procedure is left to the user. This utility only assumes the responsibility to log them, if they occur.

The suggested modification to it is to display a message on the screen, and stop, if a duplicate key occurs. The operator then has the option of keying CONTINUE (EXEC) to continue, once the duplicate key and its location is manually recorded. The program changes to module KFAM2003 (or KFAM2004 for KFAM-4) to accomplish this are as follows:

```
CLEARP  5396, 5400
6194  GOSUB'248(12,0,0)
6960 GOSUB'248(7,0,4)
CLEARP 7100, 7120
7100 STOP
7105 GOTO 5770
```

Key File Recovery

The printer is used to list duplicate keys and unreadable sectors. The prompt and operator entry for "TURN ON PRINTER" can be eliminated by deleting line 7010. To display on the CRT, and stop, following a duplicate key or unreadable sector, make the following changes to KFAM9003 (or KFAM9004 for KFAM-4):

```
8285 GOSUB' 248 (7,0,4)
8345 STOP
```

This displays the duplicate key in the center of the screen (lines 7-10) erasing the fixed information there. Following the STOP, the operator can key CONTINUE (EXEC) to resume processing.

The KFAM-3 Convert Utilities

The KFAM-3 "CONVERT" utilities use the printer to log deleted records, invalid keys, and the last key, and to print record counts. It is not really necessary to log deleted records, nor is it necessary to print record counts. Since this information fits on the screen, it can be displayed.

Invalid keys are keys which violate the KFAM-3 restrictions. (The first byte of the key may not be HEX(FF). The entire key may not be binary zero, or all bytes HEX(00).) If such keys exist under KFAM-1, their values should be changed before conversion to KFAM-3. Invalid keys should not occur, but if they do, some provision should be made to log their occurrence, because otherwise data could be lost. The suggested procedure to handle invalid keys is the same as with duplicate keys in KEY FILE CREATION: an error message is displayed on the screen, and the program stops. The operator should record, manually, the key value (hex) and the record location. Then, optionally, the operator may continue by keying CONTINUE (EXEC).

The last key is necessary in order to run KEY FILE CREATION later. Therefore, the last key must be displayed on the screen and the operator must record its value before clearing the screen.

The suggested changes to the KFAM-3 "CONVERT" utilities to eliminate the printer are:

1. No display of deleted records.

2. Display invalid key and record location and stop so that the information can be written down. Optionally continue.

3. Display record counts on the screen.

4. Display the last key. The last key should be written down before the screen is cleared.

Program changes to CONVERT KFAM-1 TO KFAM-3 (KFAM5000) to accomplish the above are as follows:

```
CLEARP  7545, 7574
CLEARP 7765, 7775
7765 D=D+1: GOTO 7795
CLEARP 7815, 7825
7815 GOSUB'248(4,0,11)
CLEARP  7935, 7940
CLEARP  7955, 7970
7935  GOSUB'248(4,0,11)
8004  STOP
```

Program changes to CONVERT KFAM-2 TO KFAM-3 (KFAM5002) to accomplish the above are as follows:

```
CLEARP 5510, 5544
CLEARP 6030, 6040
6030 D=D+1:  GOTO 6060
CLEARP 6080, 6090
6080 GOSUB' 248 (4,0,11)
CLEARP 6200, 6205
CLEARP 6220, 6235
6200 GOSUB' 248 (4, 0, 11)
6314 STOP
```

<u>The Model 2201 Output Writer</u>

The Model 2201 Output Writer can perform any of the functions assigned to a line printer in KFAM. The "SELECT PRINT 215" statements need merely be changed to "SELECT PRINT 211". For the listed programs, these statements occur on the following lines.

PROGRAM	LOCATION
PRINT KEY FILE KFAM-3	240
PRINT KEY FILE KFAM-4	810
KEY FILE CREATION (KFAM-3 and KFAM-4)	6194, 6960
CONVERT KFAM-1 TO KFAM-3	7545
CONVERT KFAM-2 TO KFAM-3	5510
INITIALIZE KFAM FILE KFAM-3	2710
INITIALIZE KFAM FILE KFAM-4	2860
KEY FILE RECOVERY (KFAM-3 AND KFAM-4)	8285

29.2 <u>FILES TOO LARGE FOR ONE PLATTER IN KFAM-3 AND KFAM-4</u>

A cataloged disk file must be wholly contained on one disk. If the User File is too large for one disk, it must be broken into two separate files. (Both files may have the same name, since they are on different disks.)

Separate Key Files must be created, one for each User File. (If both Key Files are on the same disk, they may not have the same name.)

Perhaps the simplest scheme for splitting the User File is to determine a "cutoff" point. A key value is picked, somewhere in the middle, which will be the highest key in User File #1. Records with lower keys are stored in User File #1, and records with higher keys are stored in User File #2.

If each User File and its companion Key File are stored on the same platter, both User Files may have the same name, as may both Key Files. In that case, the same routines can be used to access both files, simply by changing the platter designation. For example, suppose that the User Files "FILEF1" are open on the 'F' and 'R' platter, as are the Key Files "FILEK1". Assume that these files are parts of a single inventory file, and that the part number of the last record in UF #1 (on the 'F' platter) is "9006AS-B4". Under KFAM-3 the following routine could then be used to open both files, accept an input key, determine which file the record associated with this key is in, and read the record from the appropriate file for processing.

```
   1    GOTO 4000
4000    SELECT #1 320    :REM  KF #1
4010    SELECT #2 B20    :REM  KF #2
4020    SELECT #3 320    :REM  UF #1
4030    SELECT #4 B20    :REM  UF #2
4040    REM OPEN BOTH FILES
4050    GOSUB'230(1,1,3,1,"FILEF1")
4060    GOSUB'230(2,2,4,1,"FILEF1")
4070    REM NOW PROCESS
4080    INPUT "PART NUMBER", P$
4090    X=2:  REM ASSUME PART # BELONGS IN FILE #2
4100    IF P$   "9006AS-B4" THEN 4130
4110    X=1:  REM IF PART # SMALLER THAN 9006AS-B4 IT GOES
           IN FILE #1
4120    REM FINDOLD
4130    GOSUB'232(X,0,P$):  REM SEARCH FOR KEY IN APPROPRIATE KF
4140    IF Q$   " " THEN 6020
4150    REM GET DATA RECORD
4160    X=X+2:  REM COMPUTE UF FILE NO. BY ADDING 2 TO KFAM ID
           NUMBER
4170    DATALOAD DC#X, Data Record


4180
  .
  .     Process Data
  .
  .
5990

6000    GOTO 4080:  REM LOOP BACK TO INPUT NEXT PART NO.

6010    REM ERROR PROCEDURE
6020    PRINT "PART # NOT ON FILE"
6030    GOTO 4080
```

Notes:

1) Line numbers less than 4000 should not be used in the program (since KFAM subroutines end at line 3075), with the exception of Line 1 (GOTO 4000), which makes it possible to execute the program simply by keying RUN, EXEC.

2) Line 4130: Alpha variables can be used as valid parameters for alphanumeric arguments, and numeric expressions can be used as numeric arguments.

3) Line 4170: A numeric variable is valid as a file number in the DATALOAD DC (or DATASAVE DC) statement. A variable cannot be used for the file number in a SELECT statement, however.

4) For KFAM-4, the SELECT subroutines DEFFN' 210 and DEFFN' 211 must be included in the program, and the GOSUB' arguments must be changed to conform to KFAM-4 specifications.

29.3 <u>REUSING DELETED SPACE WITH FINDNEW(HERE)</u>

Immediately following a DELETE, FINDNEW(HERE) may be used to insert a new record in the space just vacated by the deleted record. This function is useful for changing a key, but is not generally useful to reuse the deleted space because a new record is not generally available immediately following a DELETE.

The user may, however, store the pointer to the deleted record in a separate file for later use. The procedures, for KFAM-3 and KFAM-4, are given below.

<u>KFAM-3 Procedure</u>

1. DELETE a record.

2. Test T8$. If T8$="1", then T4$ contains a valid pointer to a deleted record, and may be saved.

3. If T8$="1", save the contents of T4$ in some file or list external to KFAM. (T4$ is a 3-byte pointer containing the relative sector address (2 bytes, hex) and record number (1 byte, hex) of the deleted record.)

To reuse the space at some later time:

1. Set T8$ = "1".

2. Move the saved record pointer to T4$.

3. Use FINDNEW(HERE) with the new record key.

4. FINDNEW(HERE) will return with the Current Sector address of the correct sector and Q = the record number within the sector.

Notes:

1. T8$ is an internal return code. T8$ = "1" indicates that the last operation was a DELETE, which was successfully executed. FINDNEW(HERE) will not be executed unless T8$ = "1".

2. T4$ always points to the record being accessed, following normal execution of any subroutine except OPEN and CLOSE. T4$ is the source field from which the disk read/write head is positioned and Q is extracted. The first two bytes of T4$ can be used as an absolute sector address by adding the starting sector (hex) of the User File.

3. T8$ and T4$ are both contained in the KDR. When more than one file is open concurrently, the KDR residing in memory is the KDR of the last file accessed and not necessarily the next file to be accessed. Therefore, when more than one file is open, care should be taken in modifying variables in the KDR. The current KDR in memory may belong to another file.

   The I.D. Number of the currently active file is in T9. If there is any question, T9 can be tested. To ensure that the proper KDR is in memory, the following statements should be executed:

```
T6 = (I.D. Number)
GOSUB 920
IF Q$ = "X" THEN (file not open)
```

The subroutine at line 920 checks T6 = T9. If they are not the same, it writes the current KDR and reads the KDR belonging to file T6, and then sets T9 = T6. If they are the same, it does nothing.

Once the proper KDR is in memory, then the values of T8$ and T4$ can be modified for the correct file.

KFAM-4 Procedure

Unlike KFAM-3, KFAM-4 does not check that FINDNEW(HERE) follows DELETE.

Under KFAM-4, the pointer to a deleted record may be saved as follows:

1. DELETE a record.

2. Test to make sure that Q$ = blank.

3. Save the contents of T4$ in some file or list external to KFAM. (See Section 28.1, "Internal Storage KFAM-4" for definition of T4$.)

To re-use the space at some later time:

1. Move the saved record pointer to T4$. (See Note 1 below.)

2. Use FINDNEW(HERE) with the new record key.

3. FINDNEW(HERE) will return with the Current Sector address set to read the correct sector and Q = the record number within the sector.

> NOTE:
>
> If the file to be accessed is not the same as the file last accessed by a KFAM subroutine, move the saved record pointer to T4$ (i), where i = this file's KFAM I.D. Number. If not sure which file was last accessed, test T9 = KFAM I.D. Number last accessed.

29.4 MULTIPLE KEY FILES PER USER FILE

KFAM does not support multiple Key Files for a single User File. Though the Key File number provides a means of identifying different Key Files for a single User File, the subroutines and utility programs are designed for operations in which there is only one Key File per User File. The Programming Department of Wang Laboratories, Inc. does not support KFAM based file access systems that attempt to maintain multiple Key Files for a single User File.

29.5 STATUS OF THE KEY DESCRIPTOR RECORD (KDR) IN KFAM-3

The Key Descriptor Record (KDR) contains vital information pertaining to a KFAM file. The contents of the KFAM-3 KDR have been described in Section 28.1, but the dynamics of the KDR, namely when it is read, when it is written, and when certain fields are updated, may be of interest to the applications programmer.

The contents of the KFAM-3 KDR can be split into the following categories:

a. Fields which are set up in INITIALIZE KFAM FILE and remain unchanged:

Q3$	Ending (relative) sector address, upper bound, User File (may be changed with REALLOCATE KFAM FILE SPACE).
V8$	Records per block, user file.
V1$	File type, Record Length, Starting position on key, Key length, Number of entries in KIR.
V3$	Ending (relative) sector address, upper bound, Key File (may be changed with REALLOCATE KFAM FILE SPACE).
V6$	Sectors per logical record or block.

b. Fields which are set up by the OPEN subroutine and remain set until the file is opened again:

V0$	Starting (absolute) sector address, Key File. This field is set at OPEN time to allow for moving the Key File to a different location on disk.
T1	Current Key File #.
T2	Current User File #.

c. Fields which are initialized in INITIALIZE KFAM FILE and changed (or subject to change) whenever a record is added to the file:

Q2$	Last (relative) sector address assigned, User File.
V5$	Record number within sector, last record of User File.

When a record is added via the FINDNEW subroutine, V5$ is incremented by 1. If V5$ exceeds V8$ (records per block), Q2$ is incremented by V6$ (sectors per block) and V5$ is set to 1. When a record is added via the FINDNEW(HERE) subroutine, Q2$ and V5$ are not changed.

V2$	Last (relative) sector address assigned, Key File. This will be updated only when the KIR is split, requiring a new sector (or sectors) for the Key File.
T2$	Relative sector address of highest level index, Key File.
T0	Number of index levels in Key File.

T2$ and T0 are only updated when a new level is added to the key index.

V8	Bias percentage for KIR split.
T5$	Last key added to the file.

V8 and T5$ are updated every time a record is added to file.

d. Fields which are changed every time a record is accessed:

T4$	Pointer to record found in User File. Corresponds to positioning of disk read/write head and Q value.
T7$	Last key accessed. In the case of "key not found", this is the key value specified by the user. In the case of FINDFIRST, FINDLAST, or FINDNEXT, it is the key found in the Key File.
T2$()	Path to locate key, in terms of sector addresses of KIR's searched.
T()	Path to locate key, in terms of KIE within KIR per level searched.

T2$() and T() are not defined following FINDNEW, FINDNEW(HERE), or DELETE.

T8$ Internal completion code.

The KFAM-3 KDR is resident in memory once the file is opened. It remains resident in memory, as long as the file is "current", and is written back onto disk when the file ceases to be "current".

The distinction between "open" and "current" is necessary because more than one file may be open concurrently. The "current" file is defined as the last file accessed by a KFAM-3 subroutine. Following a CLOSE of any file, no file is "current".

Regardless of how many files are open, there is only one space in memory for the KDR, that space being defined by the variables Q2$, Q3$, ..., T8$, that define the KDR. The KDR of the current file occupies these variables. When another file is specified by the user, the KDR for the current file is written onto the disk, the KDR for the new file is read into memory, and the new file becomes "current".

Therefore care should be taken, in the multi-file situation, in modifying any of the variables in the KDR. The KDR residing in memory may belong to another file. If there is any doubt, the following instructions will invoke the subroutine which checks which file is current and switches KDR's if necessary.

```
T6 = (I.D. number of file to be accessed)
GOSUB 920
IF Q$ = "X" THEN (file not open)
```

In addition to being written every time the file ceases to be current, the KDR is also written every time a record is added to the file, either by FINDNEW or FINDNEW(HERE). In other words, the KDR is written every time that critical information is updated. This is a safety factor, so that the file will not be destroyed in the event of system failure. (However, this is not an absolute guarantee that the file cannot be destroyed. A system failure during the critical rewrite operations of FINDNEW, FINDNEW(HERE), or DELETE can cause the key index itself to become invalid. Also a hardware malfunction can make the disk unreadable. It is a good practice to make a back-up copy of the disk at regular intervals.)

The fields of the KDR which are changed with every record accessed are not critical, and therefore the KDR is not rewritten every time a record is accessed. When the contents of the KDR is printed, these fields may or may not reflect the latest status of the KDR. (In the event of system failure, they will not generally reflect the latest status of the KDR.) They are printed only because they are there, and they may or may not be meaningful. The PRINT KEY FILE program prints the latest version of the KDR written on disk, which means the status of the KDR either the last time a record was added or the last time the file ceased to be current.

There are legitimate reasons why a user may wish to change information in the KDR. One problem which is likely to occur is that the starting position of the key or the record length is wrong, causing the Reorganization program to fail. These fields, which are critical in reorganizing, cannot really be checked prior to reorganizing. And at the point of reorganizing, it is not generally feasible to recreate the Key File from the beginning. If such problems, or similar problems, occur, the contents of the KDR can be changed by the user, via a very simple procedure:

```
SELECT (Key File#, User File#)
OPEN the file
Modify the appropriate variable in the KDR
CLOSE the file
```

This will read in the KDR, change it, and write it back on the disk.

29.6 STATUS OF THE KEY DESCRIPTOR RECORD (KDR) KFAM-4

In KFAM-4, the KDR is not so easy to modify, via user program, as in KFAM-3, nor is it necessary or recommended to modify the KDR.

The fields which are of most interest to the user, T4$ (current pointer) and T7$ (current key), are stored internally. (See the Section 28.1 on "File Layouts" and "Internal Storage.")

In KFAM-4, the KDR serves as the communications link between multiple CPU's accessing the file. In shared mode ("A"), it is read at the start of each KFAM subroutine, updated, and rewritten at the end. In exclusive mode ("X") it is read and written the same as in KFAM-3, except that it is written in the OPEN subroutine to indicate to other CPU's that the file is held exclusively.

There are legitimate reasons why a user may wish to change information in the KDR. One problem which is likely to occur is that the starting position of the key or the record length is wrong, causing the Reorganization program to fail. These fields, which are critical in re-organizing, cannot really be checked prior to re-organizing. At the point of re-organizing, it is not generally feasible to re-create the Key File from the beginning. If such problems, or similar problems, occur, the contents of the KDR can be changed by the user, via a very simple procedure:

```
SELECT (User File #)
OPEN the file, exclusive mode
Modify the appropriate KDR variable
CLOSE the file
(DEFFN'210, 211 to SELECT Key File #)
```

This will read in the KDR, change it, and write it back on the disk.

29.7 FILE NAMES FOR THE KFAM UTILITIES

File names for the KFAM-3 and KFAM-4 Utilities are as follows:

UTILITY		KFAM-3	KFAM-4
INITIALIZE KFAM FILE	-	KFAM1003	KFAM1004
KEY FILE CREATION	-	KFAM2003	KFAM2004
REORGANIZE KFAM FILE			
DIALOG	-	KFAM3003	KFAM3004
GENERATE CODE	-	KFAM3103	KFAM3104
MAIN PROGRAM	-	KFAM3203	KFAM3204
REALLOCATE KFAM FILE SPACE	-	KFAM4003	KFAM4004
DISK COPY/REORGANIZE	-	KFAM4103	KFAM4104
CONVERT KFAM-1 TO KFAM-3	-	KFAM5000	-
CONVERT KFAM-2 TO KFAM-3	-	KFAM5002	-
PRINT KEY FILE	-	KFAM6003	KFAM6004
BUILD SUBROUTINE MODULE	-	KFAM8003	KFAM8004
REORGANIZE SUB-SYSTEM			
MODULE 1		KFAM3503	KFAM3504
MODULE 2		KFAM3603	KFAM3604
MODULE 3		KFAM3703	KFAM3704
KEY FILE RECOVERY		KFAM4003	KFAM9004
RESET ACCESS TABLE		-	KFAM7004
CONVERT KFAM-3 TO KFAM-4		-	KFAM5004

PART V
THE PROGRAMMING AIDS

CHAPTER 30
OVERVIEW

30.1 SUMMARY

The Programming Aids Diskette contains a library of DEFFN' subroutines designed to reduce the time required to develop application programs. It also contains SORT-3, a disk sort subsystem that is called by a user written set-up program. Since it must be called by a user written program, SORT-3 does not appear in a Programming Aids menu.

There are two groups of subroutines: the SCREEN/DISK subroutines and the TRANSLATION TABLES subroutines. The SCREEN/DISK subroutines perform standard tasks related to operator to CPU, and CPU to disk interaction. The TRANSLATION TABLES subroutines initialize 256-byte arrays with the proper hex codes for four standard code translations. The arrays are designed for use with the BASIC statement $TRAN.

All of the subroutines may be loaded at once. They are numbered within the range 3000-9899. The program lines associated with the menus are outside this range.

All scalar and array, alpha and numeric variables used by the subroutines have their initial symbol within the range Q - W. All DEFFN' routines are identified by numbers 200-255. While individual items within these ranges may not be used by any given release of ISS, in supporting ISS it is assumed that no variables or DEFFN' subroutines in these ranges are used for application purposes unrelated to the subroutines.

All the subroutines are compatible with one another in regard to usage of variables. However, all the translation table subroutines load the same array variable.

In the descriptions of the various programming aids, a familiarity with the BASIC language is assumed, in particular with the DEFFN' and GOSUB' statements.

30.2 HOW TO LOAD THE SUBROUTINES

Subroutines should be loaded before you begin to key in the application program.

To load subroutines the recommended procedure is:

1. From the Programming Aids Master Menu access the appropriate menu for the desired subroutines.

2. Key the specified Special Function keys for all the desired subroutines on the menu. For the translation tables menu, after each key depression be sure the processing light goes out before depressing the next key. For the SCREEN/DISK menu be sure to key Special Function Key 16 after selecting the desired subroutines. If subroutines from only one menu are required, go to step 6; otherwise, go to the next step.

3. Save the gathered subroutines with a SAVE DC command, restricting the line numbers to be saved to 3000-9899. For example,

   ```
   SAVE DC R "AID1" 3000, 9899.
   ```

   This preserves only the lines associated with the programming aids themselves, since the "menu" lines lie outside this range.

   If another menu is needed, depress Special Function key 15 to return to the START module, and go to step 1.

   Otherwise, go on to the next step.

> NOTE:
>
> Loading a subsidiary menu clears from memory all program text and non-common variables.

4. Depress CLEAR, RETURN(EXEC).

5. In the immediate mode, load all the files that have been created. For example, execute

   ```
   LOAD DC R "AID1"
   LOAD DC R "AID2"
   ```

6. If translation table subroutines have been loaded, a DIM statement appears at line 9700. It must be moved to a line number lower than that on which the $TRAN statement is to appear. All DIM statements for the SCREEN/DISK subroutines lie in the range 3000-3050. They should be moved to the beginning of the application program. If more than one translation table is used, it may be desirable to change one of the designating array variables.

7. Save the adjusted programming aids collectively. For example,

   ```
   SAVE DC R "AIDS"
   ```

   Though not strictly required, this step will ensure that steps 5 and 6 will not have to be repeated, should the memory be accidentally altered or destroyed.

8. Key in the application program.

CHAPTER 31
THE SCREEN/DISK SUBROUTINES

The Screen/Disk subroutines are DEFFN' subroutines which can be incorporated into application programs. They each accomplish a standard task relating to system to disk, or system to operator, interaction.

None of the Screen/Disk subroutines destructively overlap one another; therefore, all may be used in a single program if that is desired. All load within the line numbers 3000-9599, though only DIM statements appear below line 7000.

The subroutine arguments are shown as variables; however, any valid GOSUB' argument form may be used by the application programs. The variables shown as subroutine arguments have been chosen for mnemonic reasons; they are not the variables actually used by the subroutines.

If a subroutine argument specifies a disk file number, the file number must be associated with a device address prior to calling the subroutine. For file numbers other than 0, this must be accomplished by executing a SELECT statement such as 50 SELECT #3 310.

31.1 DATA ENTRY

This subroutine accepts a keyboard entry and checks that its value, length, and number places before and after the decimal fall within specified limits. It uses the KEYIN statement. Therefore, the possibility of a hardware-signaled data input error is eliminated. It can be used for alphanumeric or numeric input. It displays a prompt and creates an appropriate entry mask with decimal location indicated by a slash (/), and all other entry positions indicated by a hyphen (-). Note that the significance of the subroutine arguments, as given below, depends upon whether the field to be entered is numeric or alphanumeric. T is the argument that specifies which type of field is to be entered.

Transfer to the subroutine is via the statement:

```
GOSUB' 200 (L$, H$, L1, R1, P$, T)
```

where: L$ - If the field to be entered is alphanumeric, L$ is the lowest acceptable alphanumeric string value of the entry. If the field to be entered is numeric, L$ is the lowest acceptable numeric value of the entry. Since the argument L$ is itself alphanumeric, it must always be expressed as an alphanumeric string, even if the field to be entered is numeric, e.g., L$ = "-99.99".

H$ - If the field to be entered is alphanumeric, H$ is the highest acceptable alphanumeric string value of the entry. If the field to be entered is numeric, H$ is the highest acceptable numeric value of the entry, expressed as an alphanumeric string.

L1 - If the field to be entered is alphanumeric, L1 is the maximum number of characters it may contain. If the field to be entered is numeric, L1 is the maximum number of digits to the left of the decimal.

R1 - If the field to be entered is alphanumeric, R1 should be entered as 0. If the field to be entered is numeric, R1 is the maximum number of digits to the right of the decimal.

P$ - is the prompt. Maximum of 64 characters.

T - is the operation type.

If the field to be entered is alphanumeric, T is 2. If the field to be entered is numeric, T is 1.

The prompt is displayed on line 1, and the mask on line 2.

Two types of checks on the entry are performed: checks on each character as it is entered, and checks on the entire response after the depression of RETURN(EXEC).

The character checks are:

1. Is the character a RETURN(EXEC)? A RETURN(EXEC) signals the end of the response.

2. Is the character a BACKSPACE? A BACKSPACE erases the previous character from memory, and replaces it on the display with the proper mask character.

3. Is the character a LINE ERASE? A LINE ERASE wipes out all entered characters and reconstructs the mask.

4. Is the hexadecimal value of the entry less than 20 (space) or greater than 7F? If so, the error message is displayed.

After RETURN(EXEC) is keyed, the following checks are performed on the entire response:

1. Does the response conform to the minimum and maximum value specifications?

2. If the entry is alphanumeric, is its length within the maximum length? If the entry is numeric, is the number of digits before the decimal within the maximum?

3. If the entry is numeric, is the number of digits to the right of the decimal within the maximum?

If these tests reveal that the response does not meet the specifications detailed in the GOSUB' arguments, then the error message is displayed.

The error message is "INVALID. RE-ENTER", displayed on line 3, accompanied by the audio alarm. When the error message is displayed, the mask is reconstructed and the subroutine is readied to reaccept data.

A valid response is returned to the program in

Q9 - for numerics
Q6$ - for alphanumerics

31.2 <u>FREE UNUSED SECTORS</u>

This subroutine examines the last file in a disk catalog area. It de-allocates those sectors between the end of the file and the DATASAVE DC END trailer. It repositions the end of file control sector. The de-allocation may be restricted by specifying that a minimum number of extra sectors be maintained in the file.

The file must have been ended with a DATASAVE DC END statement. If this subroutine is executed on a file which lacks a DATASAVE DC END trailer, the file is destroyed.

This subroutine is designed as a counterpart to Allocate Data File Space.

The subroutine is called by

```
SELECT #F xyy
GOSUB' 227 (F, N$, S1)
```

where:

xyy is the disk device address.
F is the File Number.
N$ is the name of the file to be examined.
S1 is the number of extra sectors to be maintained in the file.

There are three independent conditions under which the file will not be altered. In the sequence of their evaluation, they, and their return codes, are:

If the file does not exist, the return code R2$ is set to 3.

If the specified file is not the last file in the catalog area, the subroutine returns 2 in R2$.

If the specified number of extra sectors to be maintained in the file is less than or equal to the number found in the file, the subroutine returns 1 in R2$.

If none of the above conditions occurs, the file is altered and the subroutine returns 0 in R2$.

31.3 ALLOCATE DATA FILE SPACE

This subroutine opens a data file on any selected disk and allocates to it the available sectors between the current end of cataloged files and the end of the cataloged area. It checks the catalog index to ensure the uniqueness of the file name; it allows a minimum acceptable file size to be specified.

This subroutine is designed to be a counterpart to Free Unused Sectors.

The subroutine is called by

```
SELECT #F xyy
GOSUB' 228 (F,N$,S)
```

where:

xyy is the disk device address.
F is the File Number.
N$ is the name of the new file.
S is the minimum acceptable number of sectors for the file.

There are three conditions sufficient to prevent the file from being opened. In the sequence of their evaluation they, and their return codes, are:

If the file name is the same as an indexed scratched file, the return code R2$ is set to 3.

If the file name is the same as an indexed active file, the return code R2$ is set to 2.

If there are insufficient sectors in the catalog area, beyond the current end, to open the specified minimum file, the return code R2$ is set to 1.

If none of these conditions occurs, the file is opened and the return code R2$ is set to 0.

31.4 SEARCH INDEX

The Search Index subroutine searches a disk catalog index for a specified file name. It returns the status of the file as active, scratched or nonexistent.

The subroutine is called by:

```
SELECT #F xyy
GOSUB' 229 (F, N$)
```

where:

xyy is the disk device address.
F is the File Number.
N$ is the file name.

R2$ returns the file status code.

```
R2$ = HEX(10) the file is active.
R2$ = HEX(11) the file is scratched.
R2$ = HEX(00) the named file does not exist.
```

31.5 OPEN/CLOSE OUTPUT

These subroutines open for output, and subsequently close, disk data files which utilize special header and trailer information. In addition to satisfying the file open and close requirements for disk catalog operation, they produce single sector software header and trailer records with the following fields:

FIELD	TYPE	ELEM LENGTH	DISK LENGTH	CONTENTS
1	Alphanumeric	3	4	HDR-indicates header EOF-indicates end of file EOR-indicates end of volume
2	Alphanumeric	8	9	file name
3	numeric	8	9	creation date (Julian format)
4	numeric	8	9	number of days to retain file (the "retention period")
5	numeric	8	9	volume number

Based on the data in the header and trailer records, these subroutines enforce certain system standards. For example, when a file is opened for output, a life span in days is specified for it. The file, then, cannot be opened for output again until this life span has expired.

Open Output

The subroutine is called by

```
SELECT #F xyy
GOSUB' 240 (F,N$,D,V)
```

where:

xyy is the disk device address.
F is the file number.
N$ is the name of the file to be opened.
D is the number of days the file is to be preserved (the "retention cycle").
V is the volume number of the file.

The subroutine displays the message MOUNT DISK TO CONTAIN VOL. XX OF FILE (FILE NAME) UNIT X. After the specified disk is mounted, the catalog index is searched for the file name.

If the file is not listed in the disk index, it is opened using the technique of the Allocate Data File Space subroutine.

If the file is indexed but scratched, the scratched file is reopened as an active file.

If the file is indexed and active and the retention period has expired, the file is reopened.

Regardless of which one of the above conditions is found, the subroutine writes the software header record in the first available file sector. The Julian date is obtained from Q1. Control returns to the application program with the read/write head at the first available file sector after the software header.

If the file name is indexed and active but the volume number is different from argument V, then, the mount message is redisplayed. If the file name is in the index, active, but the retention period has not expired, the message RETENTION CYCLE NOT EXPIRED appears together with the mount message. If there is insufficient space to open a file, the message INSUFFICIENT SPACE appears together with the mount message.

NOTE:

Keying X RETURN(EXEC) in response to the mount message causes any file with the same name to be reopened.

Close Output

The subroutine is called by

```
SELECT #F xyy
GOSUB' 241 (F, T$)
```

where:

xyy is the disk device address.
F is the file number.
T$ is the software trailer indicator
"EOF" for end of file, or
"EOR" for end of volume.

The subroutine writes the software trailer followed by the hardware (DATASAVE DC END) trailer.

If the file is the last file in the catalog area, the techniques of the Free Unused Sectors subroutine are employed to return the unused sectors to the available disk catalog area.

The file is closed and a message requests removal of the disk.

If T$ is set to "EOF", control is returned to the application program. If T$ is set to "EOR", the volume counter is incremented for the next software header, and the Open Output subroutine is called again.

31.6 OPEN/CLOSE INPUT

These subroutines open for input and subsequently close disk data files which utilize special header and trailer information. They are designed to work in conjunction with the Open/Close Output subroutines and depend upon properly structured software headers and trailers. (See Section 9-5 for this structure.)

Open Input

The subroutine is called by

```
SELECT #F xyy
GOSUB' 250 (F, N$, V)
```

where:

xyy is the disk device address.
F is the file number.
N$ is the file name.
V is the volume number.

The subroutine displays the prompt MOUNT VOL. XX OF FILE _ _ _ _ _ _ _ _ - UNIT X. After the proper disk is mounted, the catalog index is searched for the file name. If the file name is found, the software header is read to determine if the volume number is correct. A correct volume number causes the subroutine to return control to the application program with the file open.

If the file is scratched, or cannot be found, or the volume number of the file is not the specified volume number, an error message is displayed together with the mount prompt.

Close Input

The subroutine is called by

```
GOSUB' 251(F)
```

where: F is the file number.

The subroutine reads the software trailer and checks whether it specifies an end of file or end of volume. An end of file trailer causes the subroutine to close the file and return control to the application program. An end of volume trailer causes the subroutine to increment the volume counter by one, and initiate the Open Input subroutine with the same file name and the new volume number specified.

31.7 POSITION CURSOR

This subroutine moves the cursor to any location on the display and, optionally, erases the characters to the right of the new cursor position, and the lines below it.

Transfer to the subroutine is via the statement:

```
GOSUB' 248 (R, C, E)
```

where:

R = row (0-15).
C = column (0-63).
E = number of lines to erase.

The cursor is moved to the specified position. If E is zero, no characters are erased. If E is one, characters to the right of the cursor on the specified row are erased. If E is greater than one, an additional number of lines equal to the value E-1 are erased.

31.8 ALPHANUMERIC INPUT

These two subroutines allow keyboard entry of alphanumeric data. One displays a message on line 1 as well as prompting dashes; the other displays only the prompting dashes. The prompting dashes appear on line 2 and indicate the maximum field size which can be entered. They are replaced by the entered information. The routines use the INPUT statement of the BASIC language. Line 3 is cleared on exiting the subroutines. If the entry exceeds the maximum field size, "RE-ENTER" appears on line 3; the prompting dashes are reconstructed on line 2.

With message the subroutine is entered via

```
GOSUB' 243 (P$, L1)
```

where:

P$ is the prompt up to 63 characters.

L1 is the maximum number of characters to be entered. If it is 0, no prompting dashes appear and the field size check is omitted. The maximum value of L1 is 62.

Without message the subroutine is entered via

```
GOSUB' 244 (L1)
```

where:

L1 is the maximum number of characters to be entered. If it is 0, no prompting dashes appear and the field size check is omitted. The maximum value of L1 is 62.

Both subroutines return the entered data in Q6$.

31.9 NUMERIC INPUT

These two subroutines allow keyboard entry of numeric data. One displays a message on line 1 as well as prompting dashes; the other displays only the prompting dashes. The prompting dashes appear on line 2 and indicate the maximum number of digits to be entered before and after a decimal point. The decimal point position is indicated by a slash (/). The entered numeric data replace the prompting dashes. Line 3 is cleared on exiting the subroutine.

With message the subroutine is entered via

```
GOSUB' 245 (P$, L1, R1)
```

Where: P$ is the prompt up to 64 characters.

L1 is the maximum number of digits left of the decimal point.

R1 is the maximum number of digits right of the decimal point.

Without message the subroutine is entered via

```
GOSUB' 246 (L1, R1)
```

where L1 and R1 are defined as above.

The following applies to both subroutines. If L1 is positive, only a positive number may be entered. If it is negative, a negative or a positive number may be entered, and the absolute value of L1 specifies the number of digits left of the decimal.

If L1 and R1 are zero, no prompting dashes are displayed, and any numeric entry is accepted.

Three error conditions cause "RE-ENTER" to be displayed on line 3, and the prompting dashes to be recreated on line 2. These error conditions are:

a) The number of digits entered exceeds that specified by L1 or R1.

b) No digits are entered.

c) The entry is negative and L1 is positive.

Numeric input is returned by the subroutines in Q9.

31.10 DATE ROUTINES

The Special Function Key listed for "DATE" in the SCREEN/DISK menu brings in a group of independently accessible subroutines which facilitate the entry and use of dates.

Dates may assume two forms. These are known as the "Gregorian" and "Julian" forms, respectively. Gregorian form is alphanumeric MM/DD/YY

Where: YY is the 2 low order digits of the year.
MM is the number of the month such that $1 \leq MM \leq 12$.
DD is the day of the month $1 \leq DD \leq 31$.

Julian form is numeric YYDDD.

Where: YY is the 2 low order digits of the year.
DDD is number days since the beginning of YY counting January 1 as 1.

A Julian date is in proper form if

YY > 0 and
$1 \leq DDD \leq 365$ whenever YY specifies a non-leap year,

or $1 \leq DDD \leq 366$ whenever YY specifies a leap year.

A Julian date must be in proper form to be correctly converted to Gregorian form by any of the subroutines.

All the routines are designed to automatically account for leap years.

Enter Date - Gregorian Form

This subroutine provides for keyboard entry of a Gregorian date. It returns the entered date in Gregorian and Julian form. A prompt must be specified. The entered date is displayed in Gregorian and Julian form for operator verification before the subroutine is exited.

The subroutine is entered via

```
GOSUB' 220 (P$)
```

Where: P$ is the prompt, 64 characters maximum.

The prompt is displayed on line 1. On line 2 "?_ _ _ _ _ _ _ _ _" appears indicating the maximum number of characters to be entered.

Entered characters replace the prompting dashes. The slashes (/) in the date must be entered, though leading zeroes need not be. If MM or DD assume values outside their valid ranges, the prompting dashes will reappear after depression of RETURN(EXEC). Otherwise, the message is DATE OK (Y/N) appears on line 2 with the entered date in its Gregorian and Julian forms. If N is entered, the prompting dashes reappear. If Y is entered, the Gregorian date is returned in U9$ and the Julian in U9; the subroutine is exited.

Convert Date - Gregorian to Julian

This subroutine converts a date from Gregorian to Julian format. It is entered via

```
GOSUB' 221 (G$)
```

G$ is the Gregorian date to be converted.

The routine returns U9$ with the Gregorian date and U9 with the Julian equivalent of G$. If G$ could not be converted because the values of MM or DD were outside the valid range, Q6$ is returned as "E".

Enter Date - Julian Form

This subroutine provides for keyboard entry of a Julian date. A prompt must be specified. The entered date is displayed in Gregorian and Julian form for operator verification.

The subroutine is entered via

```
GOSUB' 222 (P$)
```

where:

P$ is the prompt, 64 characters maximum.

The prompt is displayed on line 1. On line 2 "?_ _ _ _ _/" appears indicating the maximum number of characters to be entered. Entered digits replace the prompting dashes. No check is performed to ensure the proper form of the entered Julian date. The message IS DATE OK (Y/N) appears on line 2 with the entered date in its Gregorian and Julian forms. If a Julian date was entered which was not in proper form, the Gregorian date is incorrect. If N is entered, the prompting dashes reappear. If Y is entered, the Gregorian date is returned in U9$ and the Julian in U9; the subroutine is exited.

Convert date - Julian to Gregorian

This subroutine converts a date from Julian to Gregorian form. It is entered via

```
GOSUB' 223 (J)
```

Where: J is the Julian date to be converted.

The routine returns U9$ with the Gregorian equivalent of J and U9 with the entered Julian date. No check is performed on J. A Julian date not in proper form will produce a Gregorian date with MM or DD outside the valid range.

Convert Julian Date to Proper Form

This subroutine converts any 5 digit Julian date to a Julian date in proper form. It is entered via

```
GOSUB' 224 (J)
```

Where: J is a Julian date.

The subroutine returns the entered date in Q9 in proper form.

For example:

72367 is returned as 73001
71733 is returned as 73002

Calculate Days Between Two Dates

This subroutine calculates the number of days between two Julian dates. It is entered via

```
GOSUB' 225 (J1, J2)
```

Where:

J1 is the earlier date.

J2 is the later date.

U3 is returned equal to the number of days between J1 and J2.

For example:

If J1 = 75004 and J2 = 75009,

then U3 is returned as 5.

If J1 = 71360 and J2 = 72060, then U3 is returned as 65.

31.11 OPERATOR WAIT

This subroutine displays the message "KEY RETURN(EXEC) TO RESUME?" on line 2. Execution is halted on an INPUT instruction until RETURN(EXEC) is depressed. Up to one entered character is returned in variable Q6$.

Transfer to the subroutine is via the statement:

```
GOSUB' 254
```

CHAPTER 32
TRANSLATION TABLE SUBROUTINES

The translation table subroutines assign specific sets of hex codes to an array so that it may be used as a translation table with the BASIC statement $TRAN. The subroutines do not actually accomplish the translation; they merely initialize the array. The array is TO$(). It may be initialized for any of the following translations by means of the indicated GOSUB' subroutine call.

TABLE	SUBROUTINE
EBCDIC TO ASCII	GOSUB'201
ASCII TO EBCDIC	GOSUB'202
2200 TO 1200	GOSUB'203
1200 TO 2200	GOSUB'204

The subroutines load into program lines 9700-9900 without overlap; they may all be loaded at once.

All the subroutines initialize the same array variable, dimensioned as TO$(8)32. If more than one table is to be used in an application, either the array variable must be changed by modifying the subroutines, or the application program must execute the appropriate subroutine each time a different translation is to be effected.

When translating 1200 to 2200, all non-translatable codes are translated into hexadecimal FF.

Assuming that D$ contains data to be translated from ASCII to EBCDIC and the program contains the ASCII to EBCDIC translation table subroutine, the following statement sequence could be used to translate D$:

```
  20 DIM TO$(8)32    :REM MOVED FROM LINE 9700
 110 GOSUB' 202      :REM INITIALIZE TABLE
 120 $TRAN (D$,TO$()):REM TRANSLATE
 130 STOP "D$ TRANSLATED"

9748 DEFFN'202
.
. (translation table subroutine ASCII to EBCDIC)
.
9780                                 ...:RETURN
```

CHAPTER 33
SORT-3

33.1 INTRODUCTION

SORT-3 is a subsystem for sorting the records in a disk data file. It is loaded from disk by a user written set-up program. The set-up program provides the parameters for the sort, and thereby eliminates a lengthy screen dialog that would otherwise be required for operator entry of the sort parameters. When sorting is complete, SORT-3 can load a specified application program module. SORT-3, therefore, can be used as a subsystem to an application program. It requires very little operator attention.

SORT-3 offers the following operational features.

1. For maximum efficiency it uses the extended BASIC statements described in SORT STATEMENTS (Publication #700-3559A).

2. The programmer may specify whether a key sort or a full-record sort is to be performed, or let SORT-3 decide.

3. Four input file formats are accepted.

   a) an ordinary cataloged data file,
   b) a BAS-1 data file,
   c) a KFAM-3 file,
   d) A data file opened and closed with ISS OPEN/CLOSE subroutines.

4. The sort key can contain up to 10 fields. They may be alphanumeric or numeric, but their total length must not exceed 64 bytes, not counting control bytes. Sort order may be specified as ascending or descending for each field.

5. If a key sort is specified, the sort keys may be partial fields, that is, a STR() function of an alphanumeric variable.

6. If a full-record sort is specified, the mounting of the output platter may be deferred until the last pass, at which time the input platter may be removed. This permits the sorting of a full disk platter in a dual platter system. (A full record sort can only be performed if the record length is less than 128 bytes.)

7. The programmer may write a special input procedure, to be overlaid in Pass 1, to process or screen individual records before input to the sort.

8. If a key sort is specified, the programmer may write a special output procedure to be used instead of the normal Pass 3 program. Such an output procedure can be used to screen sorted records, print them, or output them to other media.

In addition to the cataloged input file, SORT-3 requires a cataloged work file. Section 33.4 describes a procedure for calculating the exact number of sectors required for the work file.

SORT-3 may be loaded directly from the Programming Aids Diskette at the time of execution. In this case the Programming Aids Diskette must remain mounted throughout the sorting procedure. However, it is not recommended that the unused sectors of the Programming Aids Diskette be used for the work file. Therefore, to maximize available disk space, it is recommended that SORT-3 be copied from the Programming Aids diskette prior to use. A Copy/Verify reference file has been included on the Programming Aids diskette to facilitate copying SORT-3. The name of the reference file is SRT3F010.

The SORT-3 system consists of the following modules:

SORT3	Check sort parameters, get record layout from sample record, calculate sort and merge array sizes, calculate work file space.
SORT300B	Calculate output file space, generate code for internal sort (SORT301A).
SORT300C	Generate code for merge (SORT302B, full-record sort).
SORT300D	Generate code for output pass (SORT303A, key sort only).
SORT301A	Pass 1, Internal Sort.
SORT302A	Pass 2, Merge (key sort only).
SORT302B	Pass 2, Merge (full-record sort only).
SORT303A	Pass 3, locate original input record, using pointer attached to sort key, and write output records (key sort only).
KFAM0103	KFAM-3 subroutines.
SRT3F010	Copy/Verify Reference File to copy all the above SORT-3 modules.

33.2 INPUT FILE REQUIREMENTS

Four kinds of input files may be sorted. These are known as file formats 0, 1, 2 and 3.

File Format 0 - A normal cataloged file conforming to the following specifications:

1) It must be a cataloged file with all records written using DATASAVE DC or DATASAVE DA. The DATASAVE DC END trailer must be present.

2) It must have all records in the same format (no special header or trailer records, no variable length records).

3) It may have either blocked or unblocked records; however,

   a. If unblocked, it must have not more than 55 fields per record.

   b. If blocked,

      i. It must be written in array form, for example

```
DIM A$(4)20, B(4), C$(4)2
DATASAVE DC A$(), B(), C$()
```

      ii. It must not have more than 38 fields per record.

      iii. It must not have more than 255 records per block.

      iv. It must have all blocks filled (unused records in the last block must be filled with padding records that will sort high if used in an ascending sort, or low, if used in a descending sort).

4) It must have all records on a single disk.

5) The record or block of records may occupy more than one sector.

File Format 1 - A BAS-1 File:

1) The file name in the disk catalog is "SCRATCH".

2) The first sector of the file contains a BAS-1 format header record which contains the file name.

3) With the exception of the header record and the trailer record, all records must have the same format.

4) The first field in each record must be a 2 byte alphanumeric field, the "RECORD ID". An end-of-file condition is recognized when the second byte of the RECORD ID is either 2 or 3. The record with a RECORD ID of 2 or 3 is not included in the sort; it merely signals the end of the file. For multi-volume files, Sort-3 treats each volume as a separate file. It does not sort more than one volume.

5) Sorted output is always written in Format 0. The blocking of output records is identical to the input blocking.

File Format 2 - A KFAM-3 File

1) Files organized by KFAM-3 may be sorted, provided a minimum of 12K of memory is available.

2) Records may be KFAM type N (one record per sector), type A (array-type blocking), or type M (multiple sectors).

3) Records must be of the same format and be written in normal (DC or DA) mode.

4) Unblocked records may not contain more than 55 fields, blocked.

5) Sorted output is always written in Format 0.

6) The KFAM user file is read using "FINDFIRST" and "FINDNEXT". This adds about 300 ms per record or 5 minutes per 1000 records to the sort time.

7) Deleted records are not included in the sort, regardless of whether or not they are flaqged in the user file as deleted.

File Format 3 - An ISS OPEN/CLOSE Data File, With a Special End of File Indicator

1) The first sector of the file contains a header record as defined by ISS subroutine DEFFN'240 (see Section 31.5 for a description of the contents of this sector).

2) A one sector software trailer record precedes the DATA SAVE DC END trailer. The content of this software trailer is defined by ISS subroutine DEFFN'241 (see Section 31.5 for a description of the contents of this trailer).

3) It conforms to specifications 1, 4 and 5 of format 0.

4) Sorted output is written in format 0.

5) Sort-3 does not use the software trailer to determine the end-of-file condition. Instead end-of-file is determined as follows: If the first field of the record is numeric, a value of 9E99 in the first field signals end-of-file. If the first field of the record is alphanumeric, a value of HEX(FF) in the first byte of the first field signals end-of-file. The record containing the end-of-file indicator is not included in the sort.

33.3 WRITING THE SET-UP MODULE

In order to use SORT-3, you must write a set-up program which provides the operating parameters for the sort, and loads the first module.

This set-up program has two parts. Statements in the first part must be assigned line numbers 10 to 179. The statements in this part are executed when the set-up program is executed, and cleared by it as it loads SORT-3. Included in this part of the set-up program are REM statements, SELECT statements, and a DIM statement. The second part of the set-up program must use line numbers between 3400 and 3699. The statements in this part of the set-up program are not cleared when SORT-3 is loaded. They become a part of the first module of the SORT-3 system, and are not executed until that module is executed. They include, in assignment statement form, most of the parameters for the sort.

The following is a skeleton of the set-up program used to call SORT-3. A line may be omitted if the default value is the desired value. Be sure to read all the comments the first time you write a set-up program.

Line	Contents	Default Value	See Comment
10	REM program identification	-	
50	SELECT #1 input file device address	-	
60	SELECT #2 work file device address	-	1
70	SELECT #3 output file device address	-	2
80	SELECT DISK system program device address for SORT-3 programs	-	3
90	DIM K(10), B(10), N(10)	-	4
178	LOAD DC T#0, "SORT3" 10, 178	-	5
179	LOAD DC T#0, "KFAM0103" 10, 179	-	6
3400	M = memory size, K bytes	S	7
3410	F = input file format 0,1,2,3 0 - ordinary file 1 - BAS-1 file 2 - KFAM-3 file 3 - ISS OPEN/CLOSE file	0	8
3420	I$ = "input file name"	-	
3430	J = key file number (if KFAM-3 input file)	-	
3440	SELECT #5 key file device (if KFAM-3 input file)	-	9
3450	R0 = records per block	1	10
3460	L0 = starting record # to be sorted	1	11
3470	L$ = "number of records to be sorted" (always in quotes)	"ALL"	11
3480	K0 = number of sort key fields	-	12
3490	Per key field, n:		
	K(n) = sequence number of key field	-	13
	B(n) = starting byte (if partial field)	-	13
	N(n) = number of consecutive bytes (if partial field)	-	13
	STR(X9$,n,1) = HEX (01) (if descending sort on this field) = HEX(00) if ascending	HEX(00)	14
3550	F$ = work file name	-	1
3560	C$ = "Y" output file cataloged "N" output file not cataloged "X" no output file	"Y"	15
3570	O$ = output file name	-	15
3580	P8$ = "K" to force key sort "R" to force full-record sort blank - program decides	blank	16
3590	D$ = "D" deferred mounting of output (full-record sort only) blank - normal mounting	blank	17
3600	G$ = "name of special input procedure" (if blank, none)	blank	18

3610	K5 = number of bytes occupied by special input procedure	0	18
3620	H$ = "name of special output procedure" (if blank, none)	blank	19
3630	M$ = "name of program to return to following the sort" (if blank, none)	blank	20
3640	SELECT #6 user program device (device address for G$, H$, and M$ modules)	-	21

Comments

1. WORK FILE: The work file must be a cataloged disk file, and must be cataloged in advance of running the program. To catalog a disk file, the following can be executed in immediate mode:

   ```
   DATASAVE DC OPEN platter sectors, "name"
   ```

   The work file must remain mounted throughout the sort and any output procedure. The number of sectors required for the work file can be calculated approximately as follows:

   K = number of bytes in the sort key
   I = number of bytes in the input record (not including control bytes)
   N = number of numeric fields in the sort key
   R = number of input records
   W = work file space in sectors

   If key sort:

   ```
   W = 20 + R/INT(250/(K+5) )
   ```

   If full-record sort:

   ```
   W = 20 + R/INT(250/(K+1) ) + R/INT (253/(I-K+1+N*8) )
   ```

   This figure should be accurate to within 10% of the actual work space required. For a more exact calculation, the SORT-3 program may be used to calculate the work file space. See Section 33.4.

2. SELECTING OUTPUT DEVICE ADDRESS: This SELECT statement may be omitted if a special output procedure eliminates the need for an output file. Otherwise, select the device at which the output file resides, or is to reside. (See comment 15 for more information on the output file.)

3. SELECTING THE SYSTEM DEVICE ADDRESS: This SELECT DISK statement selects the device address at which the SORT-3 system resides. The SORT-3 system includes KFAM module KFAM 0103 which must be present at this device address, if a KFAM-3 file is being sorted.

4. DIMENSIONING THE KEY SPECIFICATION ARRAYS: Arrays B() and N() are required only if the key is formed from partial fields. Array K() must always be dimensioned here as shown.

5. LOADING SORT-3: In this statement, 10 must be the starting line number. The ending line number must be the line number of this statement.

6. LOADING THE KFAM SUBROUTINES: This statement is used only if the input file is a KFAM-3 file. The starting line number must be 10; the ending line number must be the line number of this statement. Module SORT3 must be loaded before KFAM0103 is loaded.

7. MEMORY SIZE: SORT-3 calculates the size of work arrays and the sort blocking based on the memory size. The program can run in 8K with a non-KFAM input file, but runs faster if more memory is available.

   If this statement is omitted, SORT-3 obtains the system memory size from ISS COM variable S. If this statement is omitted and S=0, a default memory size is obtained from module SORT3. This final default is set to 8K initially but may be changed as follows in the immediate mode:

   ```
   CLEAR
   LOAD DCF "SORT3"
   3150 M = memory size, k bytes
   SCRATCH F "SORT3"
   SAVE DC F ("SORT3") "SORT3"
   ```

   This change need not be made for 8K systems, or for operations under ISS, or if a value for M is specified in each *set-up module*.

   SORT-3 uses all the memory space available to it. If there are common variables from the user program resident in memory while SORT-3 is being run, a value of M should be specified which accounts for these common variables. For example, on a 16K system, if 500 bytes of common variables are resident while the sort is being performed, set M=15.5. The SORT-3 variable check-off list (Appendix B) should be consulted to ensure the compatibility of user COM variables with SORT-3 variables. Variable S should only be used for system memory size.

8. FILE FORMAT: Set F equal to the input file format 0, 1, 2 or 3. See Section 33.2 to determine the proper format. If the format is 0, this statement may be omitted.

9. KEY FILE DEVICE ADDRESS: Select the device address at which the key file resides. This statement may be omitted for a non-KFAM input file.

10. RECORDS PER BLOCK: Assign to R0 the number of records per block in the input file. Output file blocking is always the same as input blocking (normal output procedure). There may not be more than 255 records per block. If records are unblocked, no assignment need be made.

11. STARTING RECORD and NUMBER OF RECORDS TO BE SORTED: In the case of particularly large files, for file space reasons it may be

necessary to split the file in half in order to sort it. For example, if the file contains 9000 records, the first half would be sorted by specifying LO = 1 and L$ = "4500". The second half would be sorted by specifying LO = 4501 and L$ = "ALL". A program to merge the two halves must then be written.

In the normal case, these parameters need not be specified. The starting record is always 1, and the number of records is always 'ALL'. There is no limit to the number of records that may be sorted.

12. NUMBER OF SORT KEY FIELDS: The maximum number of sort key fields is 10. The sort key fields collectively make up the sort key. The maximum number of bytes in the sort key is 64. Numeric key fields count as 8 bytes.

13. SPECIFYING THE SORT KEY FIELDS: Key field 1 assumes the most significant position in the sort key. Key field 2 assumes the next most significant position, and so on to key field 10 which assumes the least significant position in the sort key. In specifying the key fields, K(1), B(1), N(1), and STR(X9$,1,1) define key field 1. Similarly K(2), B(2), N(2), and STR(X9$,2,1) define key field 2; and so on.

The sequence number of the sort key field is the position of the sort key field in the record. For example, if the record contains:

```
A$ = account
C$ = customer name
S$ = address
Z$ = zip code
```

To sort by zip code (Z$) and customer name (C$), ascending on both,

```
3490 K(1) = 4
3492 K(2) = 2
```

defining the high order sort key as the 4th field in the record, and the next highest key as the 2nd field in the record.

B(n) and N(n) should only be used for sort key fields which are partial fields.

To use a partial field for a sort key B(n) is assigned the position of the starting byte in the field and N(n) is assigned the number of consecutive bytes to be included, beginning at the starting byte. For example, assume that bytes 47 and 48 of the address (S$) contain the postal abbreviation for state. To sort in ascending order by state and customer name, the following assignments would be made.

```
3490 K(1) = 3
3492 B(1) = 47
3494 N(1) = 2
3496 K(2) = 2
```

Partial fields may not be specified for numeric fields.

14. SPECIFYING ASCENDING OR DESCENDING ORDER FOR EACH SORT KEY FIELD: If a sort key field is to be sorted in ascending order, this assignment may be omitted. For each sort key field, n, to be sorted in descending order, an assignment of the form

```
STR(X9$,n,1) = HEX(01)
```

must be made. Modifying the last example, to sort descending by state and ascending by customer name:

```
3490 K(1) = 3
3492 B(1) = 47
3494 N(1) = 2
3496 STR(X9$,1,1) = HEX(01)
3498 K(2) = 2
```

15. THE OUTPUT FILE: If the output file is not cataloged prior to executing the sort, SORT-3 catalogs it. In this case write

```
3560 C$ = "N"
```

If the input file is format 0, SORT-3 catalogs the file with exactly the number of sectors required for output.

If the input file is format 1 or 3, SORT-3 catalogs the file with 4 extra sectors beyond those required for output. Note, however, that normal output procedure does not write header or trailer records. If the input file is format 2, a KFAM file, SORT-3 catalogs the file with the number of sectors actually occupied by data in the input file. This size is sufficient to accommodate all the "live" as well as DELETE'ed input file records, even though only live records are included in the sort. This size may be less than the number of sectors shown as "used" in the input file disk catalog since KFAM maintains its own "end of data" information.

If SORT-3 catalogs the output file, it gives it the name assigned to O$, (maximum of 8 characters). For example

```
3570 O$ = "NEWOUT"
```

If the output file is cataloged prior to executing the sort, indicate this by assigning "Y" to C$, and the name of the file to O$

```
3560 C$ = "Y"
3570 O$ = "OLDOUT"
```

Normally, SORT-3 checks the specified output file to determine if it contains enough sectors for the sorted output (that is, it checks if it contains as many sectors as SORT-3 would assign were SORT-3 cataloging the file). If a special input or output procedure is specified (see comments 18 and 19), SORT-3 does not check the number of sectors in the output file; the programmer must ensure that the output file size is adequate.

If a special output procedure is specified, the output file may not be necessary (printing only, etc.) in which case C$ should be assigned the value "X".

```
3560 C$ = "X"
```

The output file name should be omitted.

16. KEY SORT or FULL-RECORD SORT: Normally the program is allowed to decide which sort is more efficient. In this case, the default value of P8$ may be used; no assignment need be made. However, in some cases it may be desirable to specify either a key sort or a full-record sort. For example, if a special output procedure is used, a key sort must be specified. If deferred mounting of the output file is desired, a full record sort must be chosen. Finally, under some circumstances a particular type of sort may happen to run faster than the one the program would normally pick. In any of these cases, a statement such as

```
3580 P8$ = "K"
```

is used to force a key sort, or

```
3580 P8$ = "R"
```

to force a full record sort.

A key sort can be performed for any file. A full-record sort can be performed only when these conditions are met:

a. Total length of fields not included in the sort key, plus numeric sort key fields does not exceed 64 bytes.

b. No partial sort key field specified.

c. Sufficient work file space exists.

d. No special output procedure is specified.

Generally a key sort is more efficient with large records, and a full-record sort is more efficient with short records, where the sort key is a significant portion of the total record. If P8$ does not specify a sort type, SORT-3 performs a full record sort when the sort key is at least 40% of the total record length, and the bytes not included in the sort key do not exceed 64 (provided that partial fields are not specified and provided that there is sufficient space in the sort work file). In calculating the total record length of the sort record, numeric sort key fields must be counted twice.

To maximize efficiency in a full-record sort, it is useful to know how the sort record is constructed. The input record is packed into one or two "buckets" to form the sort record. Bucket "S" contains the sort key. Bucket "R" contains the remainder of the record. Numeric fields are converted to alphanumeric via "MAT

CONVERT" if they are sort key fields, or "PACK" if they are not sort key fields. Numeric sort key fields are stored both ways, MAT CONVERT'ed in bucket S and PACK'ed in bucket R. (This is because there is no "MAT UN-CONVERT" to restore them at the end.)

However, the full-record sort does not always require both buckets. If every field in the input record is specified as a sort key field, and there are no numeric fields, and the total record length is less than 64 bytes, then only bucket "S" is used. If the input record is less than 64 bytes, and contains all alphanumeric fields, it may produce a faster sort to specify all fields as part of the sort key (the previously non-key fields becoming low-order sort key fields).

If two buckets must be used for the full-record sort, there are certain bucket lengths which are more efficient than others, in terms of sort record blocking. This is because certain size fields fit more efficiently in a sector than others.

The first sector of the sort block contains a 2-byte chain pointer, which leaves 250 bytes for data. The remaining sectors have 253 bytes for data. The number of sectors can be from 2 to 17, depending on memory size. The "S" bucket array is written first, which means that the sort key should be tailored to fit into 250 bytes, and the "R" bucket tailored to fit into 253 bytes. Certain bucket lengths fit more efficiently than others, for example:

BUCKET LENGTH		CONTROL BYTE		NUMBER PER SECTOR		TOTAL BYTES
62	+	1	x	4	=	252
61		1		4		248
49		1		5		250
48		1		5		245
41		1		6		252
40		1		6		246
35		1		7		252
34		1		7		245
30		1		8		248
29		1		8		240
27		1		9		252
26		1		9		243
24		1		10		250

Certain bucket lengths are very inefficient, for example:

BUCKET LENGTH		CONTROL BYTE		NUMBER PER SECTOR	TOTAL BYTES
64	+	1	x	3	195
63	+	1	x	3	192
50	+	1	x	4	204
42	+	1	x	5	215

If the record can be split between key and non-key fields to produce efficient bucket sizes, some sort time will be saved. If the contents of a full disk platter are to be sorted, it may be necessary to have efficient bucket sizes, so that the work file space required does not exceed one platter.

17. DEFERRED MOUNTING OF OUTPUT: Deferred mounting of the output file can be a useful technique when the records to be sorted occupy a large portion of the available disk space. Deferred mounting of the output file is possible only if a full record sort is performed.

If deferred mounting is used with a fixed/removable disk system, the input file occupies the removable disk while the work file is cataloged on the fixed disk. The SORT-3 system must also be available, on a third flexible disk drive, or on one of the rigid disks. When Pass 1 is finished, the input disk is removed and a scratch disk is mounted for the output.

A key sort, by contrast, requires that all three files, input, output and work be present during the last pass.

To choose deferred mounting you must first specify a full record sort, (see comment 16 for the conditions under which a full record sort may be performed). This is specified by

```
3580 P8$ = "R"
```

Deferred mounting is then specified by

```
3590 D$ = "D"
```

If deferred mounting is not desired, a value for D$ need not be specified.

18. SPECIAL INPUT PROCEDURE: A special input procedure may be specified. A special input procedure is a user-written module which is overlaid into Pass 1 (module SORT301A). This procedure is executed after the input record has been read, but before the record is entered into the sort. It can be used to sort records selectively, or for any other input processing that does not interfere with the functioning of the sort. See Section 33.6 for information about how to write a special input procedure, and how to calculate the number of bytes it uses.

If no special input procedure is used, values for G$ and K5 need not be specified.

19. SPECIAL OUTPUT PROCEDURE: A special output procedure may be specified if a key sort is performed. A special output procedure is either a user-written module, or a user-modified version of module SORT303A, the normal output procedure. This special output procedure is used instead of the normal SORT303A. For information on how to write a special output procedure see Section 33.7. To specify a special output procedure, assign its name (8 characters maximum) to H$. For example,

```
3620 H$ = "SP.OUT"
```

If the normal output procedure is used, no assignment need be made.

20. RETURN AFTER SORT: When sorting is complete, SORT-3 can load, and initiate execution of, a user program module. It does so if M$ is assigned the name of the program. Thus, to return to an application program following the sort one might write

```
3630 M$ = "APPLPROG"
```

where "APPLPROG" is the name of the application program, 8 characters maximum. When transfer is made, the only remaining COM variables (if any) are those which were present when the set-up module called SORT 3. All other variables are cleared. Execution of the application program begins at the lowest numbered line.

If no value is assigned to M$, the message

```
STOP END OF SORT
:_
```

is displayed, when the sort is complete.

21. SELECTING A USER PROGRAM DEVICE: If a special input or output procedure is used, or return to an application program is desired, the disk device address at which these modules reside must be selected for file number #6. If none of these are used, this select statement may be omitted.

33.4 CALCULATING THE EXACT REQUIRED WORK FILE SIZE

In Section 33.3 Comment 1, formulas for estimating the size of the work file are given. The exact calculation of the number of sectors required for the work file is a long and complicated one, and the formulas given above may yield results which are off by about 10%. For exceptionally large files, or when disk space is scarce, a more exact figure may be required.

Since the first module of the program calculates the work file space required, this module can be used, by itself, to calculate the work file space required for a given number of records of a particular file. The procedure to do this is as follows:

1) Create a cataloged file on disk containing at least one record or block of records in the exact format of the records to be sorted. This can be done by using an existing file, or writing and executing a program of only 5 statements:

```
DIM (sizes and dimensions of fields)
DATASAVE DC OPEN (space, file name)
DATASAVE DC (write record or block of records)
DATASAVE DC END
DATASAVE DC CLOSE
```

The content of the records is unimportant. Only the format is important, because the sort program analyzes the format of the first data record or block of records, to determine field lengths, code to be generated, etc.

2) Set up a sort work file of at least 25 sectors.

3) Write a set-up module containing the specifications for the sort. Add the following two statement lines to the set-up module:

```
7750 STOP "7750"
7830 STOP "7830"
```

4) Run the program.

5) When the system displays "STOP 7750", change the number of records to be sorted to any desired value, by executing in the immediate mode.

```
R2 = number of records to be sorted
```

6) Key: CONTINUE
RETURN(EXEC)

7) When the program displays "STOP 7830", key:

```
PRINT 15 + Z + Q8    RETURN(EXEC)
```

Where:

```
15 = sectors required for generated code
 Z = sectors required for String Index
Q8 = sectors required for Sort Records
```

The number displayed will be the size of the sort work file, in sectors. Given a significant file size (R2 = 1000 or more), the value Z + Q8 will increase almost exactly in proportion to R2.

8) To calculate exact work file space required for other file sizes, key:

```
RUN 7750 RETURN(EXEC)
```

The program will resume at Step 5, above.

Other values which might be of interest at either of these stopping points:

```
   R1$ = "K" key sort
         "R" full-record sort
     K = sort key length
    L3 = sort record length
    P2 = sort blocking, records per block
     B = sort blocking, sectors per block
    P6 = power of the merge, Pass 2
 P4*P9 = records per sorted string, Pass 1
    S9 = work file sectors available
```

9) If a full-record sort is indicated (R1$ = "R"), it may be of interest to know how much work file space is necessary to do a key sort, especially in the case of exceptionally large files, where disk space is limited. To force the program to recalculate work file space for a key sort, key:

```
RUN 6780 RETURN(EXEC)
```

Resume at Step 5, above.

10) To calculate work space for another file, repeat from Step 3, above.

> CAUTION:
>
> Do not continue sorting beyond this point. Once program variables have been modified, results are unpredictable.

33.5 NORMAL OPERATING PROCEDURE

Prior to executing the set-up module, the disks containing the SORT-3 system, the input file, the work file and the output file must be mounted. (For deferred mounting of the output file see below.)

The set-up module must be executed to load module SORT3.

Module SORT3 displays "SORT-3 SPECIFICATIONS" followed by the sort parameters specified in the set-up module. These remain on the screen for about one minute, for a visual check.

The screen is cleared, and the system displays "PASS 1 - INTERNAL SORT" at the start of the internal sort phase.

If deferred mounting of the output file has been specified, the system displays the following, at the end of the internal sort phase:

```
REMOVE INPUT VOLUME AND MOUNT OUTPUT VOLUME
ENTER 'GO' TO RESUME
```

At this time the operator should remove the disk containing the input file and mount the disk to contain the output file. The SORT-3 system may also be removed at this time. When this is complete, the operator should key "GO"(EXEC).

The system displays "PASS 2 - MERGE" at the start of the merge phase.

If a key sort is being performed, the system displays "PASS 3 - OUTPUT" at the start of the final (output) phase.

After the final pass is complete,the system displays

```
STOP END OF SORT
:_
```

or, if the set-up module specified that an application program is to be called, the system clears all program text and non-common variables, loads the specified application program and initiates execution of it at its lowest line number.

A list of possible error messages and recovery procedures is included in Section 33.8.

33.6 WRITING A SPECIAL INPUT PROCEDURE

The set-up module may specify that a special input procedure is to be used. (See Comment 18, Section 33.3.) A special input procedure is a user-written program module which is overlaid into a specific location within the pass 1 module (SORT301A). The special input module is overlaid into SORT301A, located so that its principal portion is executed for each input record, before the record is entered into the sort.

A special input procedure can be used to selectively delete records from the sort, modify record values prior to sorting, or for any processing which does not interfere with the functioning of the sort.

When writing a special input procedure the following rules must be observed:

1. The first line number which can be used is 3200. There must be a statement at this line. It may be a REM, DIM or initialization procedure. Lines 3200-3249 may be used for these purposes; however, at this point in the program the records are not yet available.

2. Lines 3460-3499 are available for record processing. If more space is needed lines 8000-9999 may also be used. At line 3460 the input record has been read and is available, but has not yet been entered into the sort. To include a record in the sort, let the normal execution sequence prevail, or GOTO 3500. To delete the record from the sort, GOTO 3430.

   The fields in the input record are given variable names by SORT-3 according to the following convention. Variable names are A0-A9, B0-B9, ..., F0-F4. The first field in the record is A0, the second field is A1, ..., the tenth field is A9, the eleventh field is B0, and so on, in the order that the fields appear in the record. If the field is alphanumeric, $ is added to the field name. If records are blocked, the subscript (Q) is added to the field name.

   For example, if the block of records was originally written as:

   ```
   DIM R1$(4)8, N(4), C$(4)20

   DATASAVE DC R1$(), N(), C$()
   ```

   then at successive executions of line 3460, each record will be known as:

   ```
   A0$(Q), A1(Q), and A2$(Q)
   ```

Numeric sort keys are a special case. They are always designated as an array, whether or not the record is blocked. For example, if one record per sector was originally written,

```
DATASAVE DC A$, B$, C$, D, E, F, G, H, I$
```

and the sort key fields are 1 and 5 (A$ and E), then the fields will be identified by the sort as:

```
A0$, A1$, A2$, A3, A4(Q), A5, A6, A7, A8$
```

At line 3460, numeric sort keys have already been converted to sortable form via MAT CONVERT. The sortable form resides in an alpha array of the same name. (IF A4(Q) is the numeric sort key then A4$(Q) is the key in converted form.) If, for any reason, a numeric key is altered at this point, the MAT CONVERT should be repeated, for example:

```
MAT CONVERT A4() TO A4$()
```

It should be noted that, if a key sort is performed, changing a value in the input record will not change the value in the output file, since the output file is created by accessing the original input file in sorted key sequence.

If working variables are needed, other than the fields in the input record, the variable names G0-G9, H0-H9,...L0-L9 may be used. They may not be COM designated. These variable names are reserved for the output record, which is not referenced in Pass 1.

3. If a special input procedure is to be added to Pass 1, a value must be assigned to K5 in the set-up module. That value must be equal to or greater than the number of bytes occupied by the special input procedure. The value of K5 is necessary so that SORT-3 can calculate the memory space available in Pass 1.

   To calculate the value of K5:

   ```
   CLEAR
   END (total space)
   LOAD DC (name of special input procedure)
   END (space left)
   subtract space left from total space.
   Add number of bytes in working variables, plus 5 for each
   scalar variable and 7 for each array.
   ```

   K5 may be approximate, but should not be less than the space occupied by the special input procedure.

Example 33-1 shows a special input procedure module. This module was designed to operate on a file with blocked records written:

```
DIM I$(4)8, D$(4)30, B(4), C(4)
.
.
.
DATA SAVE DC I$(), D$(), B(), C()
```

Its purpose is to delete from the sort all input records for which the value of the second field is greater than "N".

Example 33-1 A Special Input Procedure Module

```
3200 REM "SPCLINPT"
3205 REM ** SP. INPUT PROC'S MUST HAVE A STMT. AT LINE 3200 **
3210 REM THIS SPECIAL INPUT PROCEDURE DROPS ALL RECORDS
IN WHICH FIRST CHARACTER OF SECOND FIELD IS GREATER THAN "N"
3460 IF A1$(Q) > "N" THEN 3430 :REM DELETE FROM SORT?
3470 REM DROP THROUGH TO INCLUDE IN SORT
```

33.7 WRITING A SPECIAL OUTPUT PROCEDURE

If a key sort is performed, a special output procedure may be substituted for SORT303A, the normal output procedure. If such a special output procedure is to be used, it must be specified in the set-up module (see Section 33.3, Comment 19).

There are two possible approaches to writing a special output procedure. The existing output procedure (module SORT303A) may be modified, or an entirely new procedure written. If a disk output file is not going to be created (for example, only printing the sorted records), then an entirely new module should be written. If a disk output file will be created, then it is generally easier to modify SORT303A.

Modifying SORT303A

During the normal execution of the SORT-3 system, module SORT300D generates certain program statements and saves them in the work file. SORT303A loads these statements during its execution. The statements generated by SORT300D depend upon the format of the file being sorted. Without these statements generated by SORT300D, SORT303A is incomplete, and cannot function. Therefore, if SORT303A is to be modified into a special output procedure, SORT300D must first be allowed to generate the program statements required by SORT303A, and SORT303A must load these generated statements.

The recommended procedure is to execute SORT-3 on a file with the same format as that to be used with the special output procedure. Be sure that a key sort is performed, and do not specify an application program to be loaded at the end of the sort. When the sort is complete, the SORT303A module, is in memory and contains the statements generated by SORT300D. It may then be used as a basis for modification; however, the statements it contains, that were generated by SORT300D, are valid only for the specific format of the input records it operated upon.
YR.

When a special output procedure is specified in the set-up module, SORT300D is not executed. It is assumed that the special output procedure contains all the program statements it needs for successful execution. Since a special output procedure may result in there being fewer output records than input records, the normal check on the output file size is omitted, if the output file was cataloged prior to calling SORT3.

In module SORT303A lines 1000-5999 are available for a special output procedure. The program structure is such that a statement appearing at line 1000 is executed once for each of the potential output records. The variable names G0-G9, H0-H9,...,L0-L4 are used for the output record. The first field in the record is G0, the second field is G1,..., the tenth field is G9, the eleventh is H0, and so on, in the order that the fields appear in the record. If the field is alphanumeric, $ is added to the field name. If records are blocked, the subscript "(O)" (numeric variable O) is added to the field name. There is no special treatment of numeric sort keys.

SORT303A must also read input records. The variable names assigned to the input file are A0-A9, B0-B9, ..., F0-F4. $ is added for alphanumeric fields. The subscript "(X)" is added if records are blocked.

Thus, at line 1000 each data record becomes available, in its sorted sequence, in both the output variables and input variables, according to the assignment scheme presented above. To delete a particular output record from the output file, GOTO 6100. To include the record, let the normal sequence of execution prevail, or GOTO 6010.

Example 33-2 shows a special output procedure which consists of SORT303A, with lines 1000-1050 added to perform the special processing. It is important to note that the SORT303A statements which appear here include those generated by SORT300D, and are valid only for the specific record shown. For this reason the file should not be saved under the name SORT303A.

Example 33-2 A Special Output Procedure Written as a Modification to SORT303A

```
10 REM SPEC.OUT (LINES 1000-1050 ADDED TO SORT303A -
                          -    AFTER OPERATION OF SORT300D)
400DIM Y2$3,X1$2,R$2,Y3$3
500DIM A0$(004)08,A1$(004)30,A2(004),A3(004),G0$(004)08,G1$(004)
30,G2(004),G3(004),S0$08
600IF C$= "X"THEN 6370:SCRATCH T#3,O$:DATA SAVE DC OPEN T$#3,O$,
O$:INIT(00)S0$:INIT(FF)Y2$:M0$=S7$:Y3$=S8$:C=0:O=1:Y0=P2
800GOSUB 6600:IF S1$(Y0)<S0$THEN 6360:S0$=S1$(Y0):R$=R1$(Y0)
840DATA LOAD DA T#1,(R$,X1$)A0$(),A1$(),A2(),A3()
860X=VAL(STR(R1$(Y0),3)):G0$(O)=A0$(X):G1$(O)=A1$(X):G2 (O)=A2 (
X):G3 (O)=A3 (X)
1000 REM ************************************************
1010 REM SPECIAL OUTPUT ROUTINE
1020      IF STR(G1$(O),30) <> "X" THEN 6010:REM OUTPUT TO DISK?
1030      PRINT G0$(O),G1$(O),G2(O),G3(O)    :REM PRINT RECORD
1040      GOTO 6100   :REM DELETE RECORD FROM OUTPUT FILE
1050 REM ************************************************
6010GOSUB 6500:C=C+1
6100ADDC(Y3$,Y2$):IF Y3$>HEX(000000)THEN 800
6120IF O=1THEN 6300
6140INIT(FF)G0$(O)
6150GOSUB 6500:GOTO 6120
6300DATA SAVE DC $#3,END :DATA SAVE DC CLOSE#3:PRINT "INPUT RECO
RDS",R2:PRINT "OUTPUT RECORDS",C:COM CLEAR M$:IF M$= " "THEN 634
0:LOAD DC T#6,M$
6340STOP "END OF SORT"
6360STOP "SEQUENCE ERROR"
6370STOP "NO OUTPUT FILE"
6500O=O+1:IF O<=ROTHEN 6540
6520DATA SAVE DC $#3,G0$(),G1$(),G2(),G3()
6530O=1
6540RETURN
6600Y0=Y0+1:IF Y0<=P2THEN 6640:DATA LOAD DA T#2,(M0$,X1$)M0$,S1$
(),R1$():Y0=1
6640RETURN
```

This special output procedure deletes from the output file all records in which the 30 character second field ends with an "X". Deleted records are output to the CRT. The file is a blocked file and, therefore the subscript (0) is added to all variable references.

Writing An Entirely New Output Procedure

If none of the input records are to be written to a disk output file, then it is necessary to write a new output procedure, and not simply add statements to SORT303A as shown above. Specifically, if, in the set-up module, C$ is assigned "X" indicating no output file, then SORT303A may not be used.

When a special output procedure is loaded by the SORT-3 system, SORT-3 has created special sort records in the work file. These sort records are blocked in a way determined by SORT-3. Each sort record contains a sort key, and the sector address (and block location, if blocked) of the key's input record in the input file. The sort records within each block are in key sequence at the time the output procedure is called. Each block of sort records carries an additional chain pointer to the next, key sequential, block of sort records. The general task of an output procedure, then, is to read the sort records in sequence, and, using the pointers contained in each sort record, access the input file.

Critical initial values are passed to the output procedure in COM variables as follows:

O$8	=	output file name, if any
RO	=	records per block in input file
P2	=	sort records per sort (work) file block
S7$2	=	sector address of first block of sort records (in binary)
S8$3	=	number of sort records (in binary)

In addition, the following arrays are COM, and dimensioned to receive the sort records:

S1$(P2)K	=	sort keys, where P2 = sort records per block, and K = key length
R1$(P2)3	=	dimensioned to receive pointers to input records, where P2 = sort records per block, and each element of R1$ receives values as follows: Bytes 1 and 2: sector address of input record (in binary). Byte 3: location of input record within block (in binary). Byte 3 is undefined for unblocked input records.

File number #2 contains the device address of the sort (work) file. File number #1 contains the device address of the input file.

COM CLEAR M$ can be executed to designate as non-common all variables COM designated by the SORT-3 system.

A block of sort records should be read as follows:

```
DATALOAD DA T#2, (S7$,X1$) S7$, S1$(), R1$()
```

In each block of sort records, S7$ is the starting sector address (chain pointer) of the next key sequential block of sort records. There is *no* end of file marker in the sort (work) file. End-of-file must be determined by counting the number of sort records, and comparing it to the total (S8$). (Note, S8$ contains the number of sort records, not the number of sort record blocks.)

A special output routine that prints a file is shown in Example 33-3. The input records were written as:

```
DIM A$(4)8, B$(4)30, C(4), D(4)
.
.
.
DATA SAVE DC A$(), B$(), C(), D()
```

Example 33-3 A Special Output Routine That Prints The Sorted File

```
110 REM (PRINTOUE) SPECIAL OUTPUT ROUTINE - PRINTS SORTED FILE
120     SELECT PRINT 215(80)
130     DIM A$(4)8, B$(4)30, C(4), D(4), F1$3
140     F1$ = HEX(FFFFFF) :REM USED TO DECREMENT S8$
160 REM READ SORT RECORD
170     DATA LOAD DA T#2, (S7$,X1$) S7$, S1$(), R1$()
180         FOR A1 = 1 TO P2
190           REM READ BLOCK FROM INPUT FILE AND PRINT RECORD
200           DATA LOAD DA T#1, (R1$(A1),X1$) A$(),B$(),C(),D()
210           R=VAL(STR(R1$(A1),3)) :REM R = RECORD SUBSCRIPT
220           PRINTUSING 280, A$(R), B$(R), C(R), D(R)
230           ADDC (S8$,F1$) :REM DECREMENT S8$
240           IF S8$ = HEX(000000) THEN 270 :REM DONE?
250         NEXT A1
260     GOTO 170
270     SELECT PRINT 005 (64)
275     COM CLEAR M$ :REM MAKES SORT-3 COM VARIABLES NON-COM
280 % ########   ##############################   ##,###,###.##
  ##,###,###.##
```

33.8 ERROR MESSAGES AND RECOVERY PROCEDURES

Error Message	Explanation/Recovery
ERR 47	File number not selected. RECOVERY: Include SELECT statement in set-up module.
ERR 80	File not found. RECOVERY: All program files and data files must be mounted at the start of the sort. (Exception: deferred mounting of output.) STOP INVALID FORMAT File format must be 0, 1, 2, or 3. RECOVERY: Correct file format in set-up module.
STOP INPUT INVALID	No END record found on format 0 input file, or file is not a data file. RECOVERY: SORT-3 will not sort this file.

STOP WRONG INPUT FILE	Format 1, header record does not contain correct file name. RECOVERY: Rerun the program, using the correct input file.
STOP ERROR OPENING KFAM-3 FILE	Error detected in KFAM-3 OPEN subroutine. RECOVERY: Rerun. If error persists, check key file number in set-up module.
STOP KFAM FINDFIRST ERROR	No records in file. RECOVERY: SORT-3 will not sort this file.
STOP TOO MANY FIELDS	More than 55 fields per record, if unblocked, or 38 fields per record, if blocked. RECOVERY: The program will not sort this file.
STOP INVALID RECORD FORMAT	The sample record being examined (first record of the input file) is not in the proper format for a data record. RECOVERY: The program will not sort this file.
STOP NOT BLOCKED AS SPECIFIED	The sample record, or block of records, being examined, does not have the same number of records per block as specified in the set-up module, or the blocking is not in array form. RECOVERY: Correct records per block in set-up module. If blocking is not in array form, the program will not sort this file.
STOP STARTING RECORD TOO HIGH	Starting record # specified is greater than the number of records in the file. RECOVERY: a. Correct set-up module. b. There is nothing to sort.

STOP INVALID STARTING RECORD	Starting record number specified is less than 1 or not an integer. RECOVERY: Correct set-up module.
STOP INVALID NUMBER OF RECORDS	Number of records specified was less than 1, or not an integer, or not a number and not "ALL". RECOVERY: Correct set-up module.
STOP INVALID RECORDS PER BLOCK	Records per block specified was less than 1, or greater than 255, or not an integer. RECOVERY: Correct set-up module.
STOP INVALID NUMBER OF KEY FIELDS	Number of key fields specified is less than 1, or greater than 10. RECOVERY: Correct set-up module.
STOP INVALID KEY FIELD NUMBER	Key field number is less than 1, greater than the number of fields in the record, or not an integer. RECOVERY: Correct set-up module.
STOP INVALID PARTIAL FIELD	Starting byte less than 1, number of bytes less than 1 or greater than remaining bytes in the field. RECOVERY: Correct set-up module.
STOP SORT KEY TOO LONG	The total length of the sort key exceeds 64 bytes. RECOVERY: The program will not operate with a sort key longer than 64 bytes.
STOP INVALID ASCENDING/DESCENDING	Ascending and descending keys incorrectly specified. RECOVERY: Correct set-up module.
STOP FULL RECORD SORT NOT POSSIBLE	A full-record sort has been specified but is not possible because the record is too long, memory space is too small, work file space is too small, or a partial field has been specified for the sort key. RECOVERY: Correct set-up module.

STOP NO ROOM TO SORT	Memory size specified is less than 8K, or input record (or block) is so large that there is not sufficient work space in memory.
	RECOVERY: Correct set-up module.
	The program will not sort very large records (or blocks) if there is not enough work space in memory.
STOP WORK SPACE TOO SMALL	The area specified for the sort work file is too small.
	RECOVERY: Create a larger work file. Correct set-up module, if necessary.
STOP DEFERRED MOUNTING INVALID	Deferred mounting of the output file may only be specified for a full-record sort.
	RECOVERY: Correct set-up module.
OUTPUT SPACE TOO SMALL	A previously cataloged output file is too small to contain the present output.
	RECOVERY: Correct set-up module or create a larger output file.
STOP NO RECORDS	There are no records to be sorted.
	RECOVERY: None.
STOP NO OUTPUT FILE	No output file has been specified.
	RECOVERY: Correct set-up module.
STOP SEQUENCE ERROR	The final output of the sort is not in sequence. This could be caused by a hardware or software malfunction.
	RECOVERY: If the problem persists, notify Wang Laboratories.
ERR 01	Memory capacity exceeded.
	RECOVERY: Correct memory size (see Section 33.3).
ERR 43	Input records, or blocks of records are not all of identical format.

	RECOVERY: The program will not sort this file.
ERR72	Disk read error.
	RECOVERY: Rerun the program.
ERR 85	Disk write error.
	RECOVERY: Rerun the program.
STOP KFAM READ ERROR	KFAM FINDNEXT returns error code "X".
	RECOVERY: This should not happen.

APPENDIX B
SORT-3 VARIABLE CHECK-OFF LIST

NUMERIC SCALARS
FORMAT = MN

N \ M	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Δ	S	S	S			S	S	S	S	S	S	S	S	S	S	S	K	S	I	K	S	K		S	S	S
1													S	S	S	S	I	S	I	K		K		S	S	
2													S	S		S	S	S	S	K				S		
3						RESERVED										S	S	S	S	K						
4						FOR INPUT										S		S	S	K						
5						AND OUTPUT										S				K						
6						RECORD										S	S			K		K			R	
7						DEFINITION										S	S	S		K		K			R	
8																	S		S	K		K			R	
9																S	S		S	K		K				
0													S					S	S	K		K			S	

NOTE:

I	=	USED BY ISS
R	=	USED BY RPL
S	=	USED BY SORT-3
K	=	USED BY KFAM

NUMERIC ARRAYS
FORMAT = MN(

N \ M	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Δ		S									S			S						K						
1																				K						
2																										
3						RESERVED																				
4						FOR INPUT																				
5						AND OUTPUT																				
6						RECORD																				
7						DEFINITION																				
8																										
9																										
0																										

ALPHA NUMERIC SCALARS
FORMAT = MN$

N \ M	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Δ		S	S	S		S	S	S	S	S	S	S	S	S	S	S	K	S	S	K				S	S	S
1														S		S	I	S	S	K	S	K		S	S	
2													S	S		S	K	S	S	K		K		S	S	S
3						RESERVED								S		S	K		S	K		K		S	R	S
4						FOR INPUT								S			K			K		K		S	R	
5						AND OUTPUT														K		K		S	R	
6						RECORD														K		K		S		
7						DEFINITION												S	S	K		K				
8																S		S	S	K		K		S	R	
9																	K	S		K		K		S		
0													S	S		S		S	S	K		K		S		

ALPHA NUMERIC ARRAYS
FORMAT = MN$(

N \ M	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Δ				S														S	S							
1																		S	S	K				S		
2																				K	S					
3						RESERVED										S			S	K				S		
4						FOR INPUT													S							
5						AND OUTPUT													S							
6						RECORD													S							
7						DEFINITION										S			S							
8																S			S							
9																			S							
0																	K	S	S	K		K				

To help us to provide you with the best manuals possible, please make your comments and suggestions concerning this publication on the form below. Then detach, fold, tape closed and mail to us. All comments and suggestions become the property of Wang Laboratories, Inc. For a reply, be sure to include your name and address. Your cooperation is appreciated.

700-3657A

TITLE OF MANUAL: **INTEGRATED SUPPORT SYSTEM USER MANUAL**

COMMENTS:

Fold

Fold

(Please tape. Postal regulations prohibit the use of staples.)

WANG

Fold

FIRST CLASS
PERMIT NO. 16
Tewksbury, Mass.

BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

– POSTAGE WILL BE PAID BY –

WANG LABORATORIES, INC.
836 NORTH STREET
TEWKSBURY, MASSACHUSETTS 01876

Attention: **Marketing Department**

Fold

Cut along dotted line.

Printed in U.S.A.

WANG LABORATORIES (CANADA) LTD.
49 Valleybrook Drive
Don Mills, Ontario M3B 2S6
TELEPHONE (416) 449-2175
Telex: 069-66546

WANG EUROPE, S.A.
Buurtweg 13
9412 Ottergem, Belgium
TELEPHONE 053/704514
Telex: 26077

WANG DO BRASIL COMPUTADORES LTDA.
Rua Barao de Lucena No. 32
Botafogo ZC-01 20,000
Rio de Janeiro RJ, Brasil
TELEPHONE 226-4326, 266-5364
Telex: 2123296 WANG BR

WANG COMPUTERS (SO. AFRICA) PTY. LTD.
Corner of Allen Rd. & Garden St.
Bordeaux, Transvaal
Republic of South Africa
TELEPHONE (011) 48-6123
Telex: 960-86297

WANG INTERNATIONAL TRADE, INC.
836 North Street
Tewksbury, Massachusetts 01876
TELEPHONE (617) 851-4111
TWX 710-343-6769
Telex: 94-7421

WANG SKANDINAVISKA AB
Pyramidvaegen 9A
S-171 36 Solna, Sweden
TELEPHONE 08/27 27 95
Telex: 11498

WANG NEDERALND B.V.
Damstraat 2
Utrecht, Netherlands
(030) 93-09-47
Telex: 47579

WANG PACIFIC LTD.
902-3 Wong House
26-30, Des Voeux Road, West
Hong Kong
TELEPHONE 5-435229
Telex: 74879 WANG HX

WANG INDUSTRIAL CO., LTD.
110-118 Kuang-Fu N. Road
Taipei, China
TELEPHONE 784181-3
Telex: 21713

WANG GESELLSCHAFT M.B.H.
Murlingasse 7
A-1120 Vienna, Austria
TELEPHONE 85.13.54, 85.13.55
Telex: 74640 Wang a

WANG S.A./A.G.
Markusstrasse 20
CH-8042 Zurich 6, Switzerland
TELEPHONE 41-1-60 50 20
Telex: 59151

WANG COMPUTER PTY. LTD.
55 Herbert Street
St. Leonards, 2065, Australia
TELEPHONE 439-3511
Telex: 25469

WANG ELECTRONICS LTD.
Argyle House
Joel Street
Northwood Hills
Middlesex, HAG ILN
TELEPHONE Northwood 28211
Telex: 923498

WANG FRANCE S.A.R.L.
Tour Gallieni, 1
78/80 Ave. Gallieni
93170 Bagnolet, France
TELEPHONE 33.1.3602211
Telex: 680958F

WANG LABORATORIES GmbH
Moselstrasse 4
6000 Frankfurt AM Main
West Germany
TELEPHONE (0611) 252061
Telex: 04-16246

WANG DE PANAMA (CPEC) S.A.
Apartado 6425
Calle 45E, No. 9N. Bella Vista
Panama 5, Panama
TELEPHONE 69-0855, 69-0857
Telex: 3282243

WANG COMPUTER LTD.
302 Great North Road
Grey Lynn, Auckland
New Zealand
TELEPHONE Auckland 762-219
Telex: CAPENG 2826

WANG COMPUTER PTE., LTD.
37, Hill Street
Singapore 6, Republic of Singapore
TELEPHONE 333641, 321791
Telex: RS 23987 GENERCO

WANG COMPUTER SERVICES
836 North Street
Tewksbury, Massachusetts 01876
TELEPHONE (617) 851-4111
TWX 710-343-6769
Telex: 94-7421

DATA CENTER DIVISION
20 South Avenue
Burlington, Massachusetts 01803
TELEPHONE (617) 272-8550

WANG LABORATORIES, INC.
836 NORTH STREET, TEWKSBURY, MASSACHUSETTS 01876, TEL. (617) 851-4111, TWX 710 343-6769, TELEX 94-7421

Printed in U.S.A.
700-3657A
6-76-1M
Price: see current list