INTEGRATED SUPPORT SYSTEM (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 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.

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.

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.

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 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 APPLICATIONS 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.

CAPTER 6 - THE COPY/VERIFY AND CREATE REFERENCE FILE UTILITIES

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.

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.

SORT DISK CATALOG

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.

DISK DUMP

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

DECOMPRESS UTILITY

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.

LIST/CROSS-REFERENCE UTILITY

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.

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.

COMPRESSION UTILITY

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 otected. 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.

RECONSTRUCT INDEX UTILITY

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.

OVERVIEW OF THE KFAM SYSTEMS

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.

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.

Subroutines are available to perform the following tasks.

TYPE AND NAME	FUNCTION
General Purpose	
OPEN	Open specified User File and companion Key File.
CLOSE	Close User File and companion Key File.
Random Access	
FINDOLD	Locate specified key in the Key File set User File Current Sector Address to record in User File with that key
Key Sequence Access	
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 secto
FINDLAST	Locate record with highest key in User File; set the User File Current Sector Address to that sector.
Add and Delete	

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.

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.

OVERVIEW OF PROGRAMMING AIDS

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 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 r 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.

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.

FREE UNUSED SECTORS

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.

ALLOCATE D*TA 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.

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.

OPEN/CLOSE OUTPUT

These subroutines open for output, and subsequently close, disk data iles 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

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.)

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 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.

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.

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.

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.

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 ompting 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.

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.

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.

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

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 r operator verification.

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.

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$.

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.

SORT-3

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.

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. ncluded 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.

performs most file maintenance tasks automatically with each marked subroutine call. Data files meeting KFAM-5 record requirements are easily converted to KFAM-5 files using a supplied utility program.

Sorting disk file records is accomplished by a highly versatile subsystem called SORT-4. A short user written set-up module loads the SORT-4 subsystem and provides the necessary sort parameters. Three types of sorts including tag, key, and full-record sorts are available. Input records can be included in or excluded from sorted output, and up to ten ascending or descending order sort key fields control output record order. A variety of input record and input file formats are fully supported.

ISS functions in conjunction with (1) the 2200 CPU's multiple file access capabilities (device table), (2) Automatic File Catalog statements and 2200 disk data hierarchy, (3) programming in the powerful, high-level BASIC or BASIC-2 languages, and (4) the automatic polling performed by the multiplex controller hardware (if applicable).

Multiplexed File Access

Files residing on a multiplexed disk drive may be concurrently accessed by multiple CPU's. Marked subroutines provide a controlled CPU file access system with four available access modes and password protection for both KFAM and non-KFAM files. KFAM supports record protection which allows concurrent CPU access to a file being updated. Features normally associated with small computer data base managment systems are thus provided by ISS in conjunction with 2200 hardware without the complexity of an operating system.

Hardware Requirements

1. ISS requires dual platter handling capability with at least one diskette, flexible disk, or minidiskette drive. All minidiskettes, flexible disks, diskettes, or disks must be formatted and cataloged.

2. ISS requires a Wang 2200C CPU equipped with Options 2 and 5, a 2200T, or a 2200VP.

3. All ISS Utility Programs require 12K of memory except Program Compare which requires 16K. The KFAM-5 Utility Programs require 16K. The SORT-4 system requires 8K, unless a KFAM file is being sorted in which case 12K is required.

APPLICATION BULLETIN NO. 24

IMPORTANT:
For installations using multiplexed disk drives, ISS Release 3.2 is recommended over ISS Release 2; also, only ISS-3.2 is operable on 2200VP's and available on minidiskette.

INTEGRATED SUPPORT SYSTEM (ISS) RELEASE 3.2

I. SOFTWARE ANNOUNCEMENT

Application Bulletins #11, #14, and #20 introduced the Integrated Support System (ISS) Releases 1 and 2. Since the release of ISS-2, ISS has undergone major revision and expansion. The result of this revision is ISS Release 3.2, which is now available. ISS Release 2.1 will still be maintained and available. Users of ISS who have purchased an ISS software maintenance agreement will obtain ISS Release 3.2 free of charge.

II. OVERVIEW

The Integrated Support System Release 3.2 minimizes the programming costs necessary to extend and customize the data handling capabilities of a Wang 2200 CPU equipped with a direct access storage device such as disk, flexible disk, diskette, or minidiskette, by providing a comprehensive set of standard functions typically required in a disk or diskette based data processing and programming environment. It consists of 45 marked subroutines, 21 utility programs, 2 subsystem routines, and system access software that links ISS software components and user application programs. ISS software is available on diskette, flexible disk and minidiskette, and may be copied to a single hard disk (if available) from any issued medium. ISS program storage and loading is supported by all four media.

Programming Support

Marked subroutines eliminate repetitious, detailed programming tasks otherwise required of an application programmer, and provide a simple interface between user application programs and a wide range of potentially complex disk-related or operator-related tasks.

Utility programs provide the ability to list, cross reference, compress, and decompress files containing program text (program files) and compare two program files on a line-by-line basis. Also, the presence of other supplied utility software eliminates the need for user written programs to perform those functions.

A unique indexed sequential access method, called KFAM-5 (K[illegible]
Access Method), is available for data files where [illegible]
rapidly accessed in non-sequential (random) [illegible]
subroutines handle file access, record [illegible]
record access by key, and record [illegible]

4. A printer is recommended for all utility programs, but is only required for the Disk Dump and List/Cross Reference ISS Utility Programs and the KFAM-5 Utility Programs. Other hard copy output devices may be used; however, because top-of-form use is not supported, multi-page output is not recommended.

5. Only hard disk (fixed/removable disk drive) is recommended as a storage medium for multiplexed files.

6. An ECN (engineering) change is required for the Multiplex Controller if Multiplexed files are used. Customers with a Preventative Maintenance contract receive the ECN free of charge. Customers without this service agreement will be billed for the ECN change. Hog mode now occurs immediately ($GIO hog).

   The ECN numbers required are listed below:

MULTIPLEX CONTROLLER	ECN NUMBER
Model 2224	5727
Model 2230MXA	5636B
Model 2230MXL	5720
2200 Workstation (WS)	5719

Distribution Media

ISS Release 3.2 is available on minidiskette, flexible disk and diskette. All versions may be copied to, and thereafter loaded from, a single hard disk (if available). Once copied to hard disk, ISS may be copied to any medium.

ISS-3.2 MEDIUM	PACKAGE NUMBER
three diskettes (Models 2270, 2270A)	195-0032 - 3
three flexible disks (Model 2240)	195-0032 - 2
seven minidiskettes	195-0032 - 8

III. NEW FEATURES OF ISS RELEASE 3.2

Those not familiar with ISS Release 2 should read "IV. FUNCTIONAL SUMMARY" (on a following page) before reading the following text.

System Access Software

Enhancements to ISS system access scftware include the following:

1. Operational CPU start-up procedures allow selection of either (1) a Warm Start, whereby existing system information is not updated, or (2) a Cold Start, whereby the date and other system information may be changed. Cold Start Information, such as disk or printer addresses, is used by ISS and KFAM Utility Programs. (The ISS menu hierarchy has changed slightly.).

2. The CPU memory diagnostic is bypassed if the CPU is a 2200VP and optional on other CPU's.

3. Messages can be sent to a specific CPU or all CPU's in the same multiplexed environment, after which conversation-like message transfer may occur.

4. Application programs may have any program file name.

ISS Utility Programs

Enhancements to the ISS Utility Programs include the following:

1. The List Utility has been removed because the same functions are available with the List/Cross Reference Utility. The Disk Sort Utility has been removed because of similar functions available with SORT-4. The following ISS Utility Programs have been added and provide functions supporting multiplexed environments, program development, and media conversion: (a) File Status Report, (b) Program Compare, and (c) Copy Tape to Disk.

2. Copy/Verify, Decompress, List/Cross-Reference, Compress, and Program Compare allow file name specification within entered alphabetic limits (RANGE). Copy/Verify input files are now specified by an INPUT MODE selection; also, whether the output files REPLACE existing files, or are ADDed to the output disk, or both, is determined by the OUTPUT MODE. The List and the Cross Reference options of List/Cross Reference have also been improved, and have BASIC-2 (2200VP) compatibility.

3. Sort Disk Catalog now allows the disk catalog index to be sorted by index sector sequence. The summary of disk space USED, FREE, and ALLOCATED is now output for the chosen file category in sectors. Up to 340 files may be output per list.

Key File Access Method Release 5 (KFAM-5)

Improvements to the Key File Access Method (KFAM) system include the following:

1. KFAM-5 provides major enhancements in the area of multiplexed (multiple CPU) file access. KFAM-5 supports one non-multiplexed mode and four multiplexed modes. Its non-multiplexed mode provides KFAM-3 characteristics within the framework of the same KFAM system that also supports enhanced multiplexed file access. KFAM-5 retains the Shared and Exclusive access modes found in KFAM-4, and also includes the Inquiry and Read Only access modes.

   A CPU which opens a file in the "Inquiry" mode may only read within the file specified, while other CPU's requesting the Inquiry (read), Read Only (read), or Shared (read/write) modes are granted access to the same file.

   A CPU which opens a file in the "Read Only" mode may only read within the file specified, while CPU's requesting the Inquiry (read) or Read Only (read) modes are also granted file access.

   A CPU which opens a file in the "Shared" mode may read and write within the file specified, while CPU's requesting the Inquiry (read) or Shared (read/write) modes are also granted file access.

   A CPU which opens a file in the "Exclusive" mode may read and write within the file specified. No other CPU may access the same file.

   In access modes where (1) multiple CPU access to a KFAM file is not allowed (Exclusive and non-multiplexed access modes), or (2) writing in the file is not allowed (Read Only access), KFAM-5 need not be concerned with record protection and hog mode options (neither are available under the circumstances), and Key File integrity. Thus, throughput characteristics are better in the Read Only, Exclusive, or non-multiplexed access modes than in the Inquiry or Shared modes where such checking features are necessary.

In the Inquiry and Shared access modes where both multiple CPU access and writing in the file can occur, KFAM employs a new technique using a busy/free flag to permit only one CPU access to the Key File at a time, thus preserving Key File integrity. Hog mode is therefore used by KFAM for only about 20% of average subroutine execution time (unlike KFAM-4's 100%). Record protection and hog mode options are available and the KDR (Key Descriptor Record) is read/written more frequently. Therefore, concurrent multiple CPU access to a file being updated (by one or more CPU's) is fully supported in the Inquiry or Shared access modes.

2. New subroutines have been added to the KFAM-4 set and include FINDPREVIOUS, RE-OPEN, WRITE RECOVERY INFORMATION, and SET-UP. Also, the following ISS Disk subroutines are also selectable from the KFAM "Build Subroutine Module" menu: Multiplex Open, Multiplex End, Multiplex Close, Set/Release Hog Mode, and Search Catalog Index.

3. Most KFAM-5 subroutines are identical to their KFAM-4 counterparts, whereas the following KFAM-5 subroutines require new argument lists and/or perform enhanced functions: Open, Delete, Findnew, Findnew (Here), Release, and Close.

4. The KFAM-5 version of Build Subroutine Module has been modified operationally. Subroutines are now selected from the displayed list and included in the output module (a program file) by depressing the appropriate Special Function key.

5. The "Disk Copy and Reorganize" Utility has been removed. Both Key Files and User Files now contain an END record (end-of-live-data) which is maintained by KFAM. Therefore, the ISS Utility Copy/Verify is used instead to copy and optionally reallocate file space, but must be followed by the KFAM Utility "Reallocate KFAM File Space" to adjust internal KFAM (KDR) pointers. A back-up disk copy may be obtained by using a COPY statement without using the Reallocate KFAM File Space Utility Program.

6. The Initialize KFAM File Utility allows blocked records written in BA mode as data file input (file type "B"). Minimum key length for any file is now 2 bytes. Scratched files may now be initialized. END records are written in the User File and Key File. The other set-up utility program, "Key File Creation" similarly handles END records, and because of the presence of an END record in the User File, entry of the last key is usually not required. A file password is requested when creating a new file.

7. The two Reorganize utilities have also been modified. The KFAM Utility Program "Reorganize KFAM File" is now called "Reorganize In Place," but is essentially the same as in KFAM-4. The KFAM program-controlled routine "Reorganize Subsystem" does not require user hog mode selection and variables S3 and S4 have been changed.

8. The "Reallocate KFAM File Space" Utility has been simplified to reflect changes now handled by Copy/Verify.

9. The contents of the Key File's Key Descriptor Record (KDR), with the exception of the Key File busy/free flag, are now stored in the next-to-last sector of the User File for use by the Key File Recovery Utility, should the Key File be accidently destroyed. This recovery information is written upon Closing a file if the "Close With Recovery Information" option was chosen during Build Subroutine Module, or after executing the "Write Recovery Information" subroutine. The KDR's contents have been changed.

10. "Print Key File" prints the current contents of the access table as well as the Key File. The access table is part of a multiplexed User File's catalog trailer record (last sector allocated). Any MOVE statement destroys this access table, which is used by KFAM files and multiplexed non-KFAM files as well, and thus destroys the file (use COPY or Copy/Verify instead).

ISS Subroutines

1. The general category of the ISS Screen/Disk Subroutines still applies. However, now they are subdivided into the Screen, the Disk, and the Translate Table Subroutines, each with their own menu. All use a subroutine selection operating procedure nearly identical to that used by KFAM's Build Subroutine Module.

2. The Screen subroutine Position Cursor has been modified to accommodate both 16 by 64 character and 24 by 80 character display screens. Data Entry provides operator Special Function Keys for reentering the current field, and has been enhanced to provide for numeric default value use. New Screen subroutines include Re-enter and Print.

3. The Disk subroutine Search Index now provides a return code identifying the file requested as active or scratched, data or program file, or not found. Free Unused Sectors updates the end-of-catalog as well as end-of-file when the file is the last file in the catalog area. Limits Next has been added, and returns the name of the next file in index sector sequence (same order as LIST DC command) and the file's status as scratched or active, data or program, or not found.

4. Disk subroutines, designed for shared file, multiple CPU use on multiplexed disk drives, are called the Multiplexed-File Open/End/Close Subroutines. Access modes include Inquiry, Read Only, Shared, and Exclusive. Each access method's rules about reading/writing in the file, provided with KFAM-5, do not apply. However, rules about a CPU being granted or denied file access based on its access mode and access modes already granted to other CPU's do apply.

5. Translate table subroutines are identical to their ISS-2 counterparts, with one exception: array TO$( ) is now Q9$( ).

SORT-4

SORT-4 enhancements include:

1. File formats supported now include both KFAM-4 and KFAM-5 files. TC (Telecommunications) files are supported by a variable length record format.

2. Input record formats supported by SORT-4 include, in addition to the DC mode, array-type blocking supported by SORT-3, packed arrays, contiguous packed records, certain variable length records, and BA mode.

3. SORT-4 allows a full-record sort on records up to 256 bytes, packed. It also allows a full-record sort with partial fields as sort keys.

4. SORT-4 will do a tag sort, which produces as its output the pointers to the original input records, instead of full records. The output file for the tag sort may be the sort work file, thus eliminating the need for a separate output file. Therefore, SORT-4 does not provide for a special output procedure.

IV. FUNCTIONAL SUMMARY

A brief summary of functions available with each ISS Software category is as follows:

System Access Software

To reduce the possibility of an operator entering the wrong CPU number during ISS start-up operation, a unique CPU number of 1,2,3, or 4 should be assigned to each CPU in an installation. (A "CPU" includes a 2200 WS, WCS, PCS, or other 2200 CPU's.) A label showing the assigned CPU number, type of CPU, and memory size should be placed near the CPU console screen for easy operator reference. Similarly, all peripherals should be labeled with their respective device addresses for operator reference purposes.

Each CPU must complete ISS start-up operation after the CPU is powered on, for instance, at the beginning of the processing day. To begin ISS start-up operation, the program module "START" is loaded from an ISS platter (disk, flexible disk, diskette, or minidiskette). START uses operator "prompts" to request entry of the CPU number and other information, which might include the peripheral addresses associated with this CPU's system configuration. CPU information, including the system configuration, is automatically maintained by ISS for each possible CPU (1-4). After successful entry of all requested start-up information, the ISS "system menu" is displayed.

The system menu allows easy selection of several displayed options, and links ISS software components and user application programs as well. System menu options allow the user to:

- load ISS support software contained on that or another ISS platter, such as a utility program or a group of marked subroutines,
- load other ISS system access software, for example, to send a message to one or all CPU's in the same multiplexed environment, or
- load an application program.

CPU start-up information ensures, for example, that a platter address entered during the operation of a utility program is a valid address. Other CPU information includes the date, which appears on printouts, and the printer address which determines if printed output is listed on a hard copy printer device or the CRT screen.

ISS Utility Programs

ISS Utility Programs are operator-controlled routines. Each processing operation performed by ISS Utility Programs (except Disk Dump) can be performed on multiple files. All are compatible with multiplexed files. Their functions are summarized below:

1. COPY/VERIFY - Copies files from one disk platter to another and verifies the copy. Media conversion can occur during copy by merely specifying the appropriate device addresses of the disk, diskette, flexible disk, or minidiskette drives. Additional sectors may be added to the copied files. Copied files may be renamed, or may replace existing files on the output platter. Files to be copied may be specified directly during Copy/Verify operation, indirectly by means of a reference file, or by means of alphabetical file name limits. Also, all files may be copied from a platter.

2. CREATE REFERENCE FILE - Creates a reference file which contains pairs of file name entries for indirect use by the Copy/Verify or Program Compare Utility Programs.

3. SORT DISK CATALOG - Prints a disk catalog index report, with files sorted (1) alphabetically by file name, (2) numerically by starting sector address, or (3) by file sequence in the index.

4. DISK DUMP - Prints the hexadecimal code and graphic character equivalents of the contents of any one disk file. In addition, the data file's contents may be printed with a field-by-field description.

5. DECOMPRESS - Copies a program file and in doing so breaks up all multi-statement lines, assigning a unique line number to each BASIC statement. Files may be specified by file name or by alphabetical file name limits. Also, all program files on a platter may be decompressed.

6. 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 four 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, one which identifies where marked subroutines are located, and one which associates all marked subroutines with the lines which refer to them. Files may be specified by file name or by alphabetic file name limits. Also, all program files on a platter may be listed/cross referenced.

7. COMPRESS - Reduces the size of source program files by eliminating REM (remark) lines, extra spaces, and inessential line numbers. Files may be specified by file name or by alphabetic file name limits. Also, all program files on a platter may be compressed.

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

9. FILE STATUS REPORT - Performs several functions tailored to a multiplexed disk environment, including closing one or all files open to a CPU, printing the CPU status of one or all files, and printing all files currently open to a CPU.

10. PROGRAM COMPARE - Compares two program files on a line-by-line basis, and indicates statements that do not match, if a statement number exists in one program but not in the other, if one program ends before the other, and whether they end with the same statement (statement numbers, device addresses, and file names are listed). The pairs of program files to be compared reside on different platters and may be specified directly by file name, indirectly by a reference file, by alphabetic file name limits, or all program files. With the latter two, files of the same name are compared.

11. COPY TAPE TO DISK - One to 99 files may be copied from cassette to disk. Up to 99 tape-resident files may be skipped before the first file is copied. Additional sectors may be added.

Key File Access Method (KFAM-5)

The Key File Access Method (KFAM-5) consists of the following utility programs:

1. INITIALIZE KFAM FILE calculates the required size of the Key File based on the estimated maximum number of records to be stored in the User File (data file), and catalogs and allocates the required space for the Key File. It stores vital information about the User File in the Key File, based on parameters supplied by the operator. It optionally catalogs and allocates space for the User File, if none exists.

2. KEY FILE CREATION should be run after Initialize KFAM File if the User File contains data records. Acceptable record formats include unblocked records occupying one or multiple sectors, array blocked records, and contiguous blocked records accessible in both "DC" or "BA" modes. It reads the User File and creates in the Key File an entry for each User File record. After completing Key File Creation with a User File containing data, or after running Initialize KFAM File with a User File containing no data records, KFAM subroutines may be used to add, delete, and update User File records and their corresponding entries in the Key File.

3. Although the KFAM subroutines are the heart of the KFAM system and perform most of the file maintenance, the REORGANIZE UTILITIES also may be required. REORGANIZE UTILITIES delete spaces left by deleted records and rearrange the order of the User File records according to ascending order of their keys. A new Key File which reflects the new User File is then constructed. There are two Reorganize Utilities:

   The REORGANIZE SUBSYSTEM is a program-controlled routine which reorganizes a file by outputting a new and reorganized User File and Key File. Optionally, 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, and may load a program upon its completion.

   REORGANIZE IN PLACE is a utility program which reorganizes the User File and Key File in place. It is used only when a file is so large that adequate output files could not be mounted when the file is reorganized.

4. The ISS Copy/Verify Utility and the REALLOCATE KFAM FILE SPACE Utility can be used together to copy a KFAM file and increase or decrease the amount of disk space allocated to the file. Use of Reallocate KFAM File Space is required after any KFAM file (User File and Key File) is copied by Copy/Verify. The Reorganize Subsystem may also be used to simultaneously copy, change file space allocation, change the file name, and reorganize a KFAM File.

5. PRINT KEY FILE: This utility prints the complete contents of the multiplexed Access Table and the current 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.

6. Recovery Utilties: The 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 multiplexed files there is an additional recovery utility called RESET ACCESS TABLE. KFAM-5 maintains in the User File trailer record information which indicates the CPU's accessing the file. This information is called the "Access Table" and its contents may be printed by PRINT KEY FILE. This Access Table will contain erroneous information when 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.

7. The KFAM Conversion Utilities: These utility programs are provided to convert files from KFAM-3 to KFAM-5, and from KFAM-4 to KFAM-5.

8. KFAM Subroutines may be selected from the BUILD SUBROUTINE MODULE Utility Program. The selected subroutines are written to disk in a program (module) file. KFAM subroutines are marked subroutines that automatically perform most maintenance tasks. Unlike similar access methods, records may be added to KFAM files in random order of their keys; the Reorganize Utilities may be used to reorganize a KFAM file's key order to ascending key record order if key sequence access is required.

General Purpose KFAM Subroutines:

OPEN	— Opens specified User File and companion Key File.
CLOSE	Closes User File and companion Key File.
RE-OPEN	Changes the access mode of a currently-open multiplexed KFAM File.
WRITE RECOVERY INFORMATION	Without closing the file, writes current file END record at end of active data in the User file, and recovery information in the next-to-last sector. Both would normally occur only when a file is closed.

SET-UP — Required with BUILD SUBROUTINE MODULE which breaks up subroutines into various modules. SET-UP initializes KFAM internal common variables, and is required before any subroutines are called.

Random Access KFAM Subroutines:

FINDOLD — Locates a specified key in the Key File and sets the User File Current Sector Address to the record in the User File with that key.

Key Sequence Access KFAM Subroutines:

FINDFIRST — Locates the record with the lowest key in the User File and sets the User File Current Sector Address to that sector.

FINDPREVIOUS — Locates the previous record in the User File in logical key sequence and sets the User File Current Sector Address to that sector.

FINDNEXT — Locates the next record in the User File in logical key sequence and sets the User File Current Sector Address to that sector.

FINDLAST — Locates the record with the highest key in the User File and sets the User File Current Sector Address to that sector.

Add and Delete KFAM Subroutines:

FINDNEW — Adds a specified key to the Key File, allocates space for a new record in the User File, and sets the User File Current Sector Address to that sector. Adds one to the record count.

FINDNEW (HERE) Adds a specified key to the Key File and sets the User File Current Sector Address to the sector where the new record is to be written. It is normally used to change the key of a deleted record; therefore, it is normally preceded by a DELETE. Adds one to the record count.

DELETE Removes the specified key from the Key File and sets the User File Current Sector Address to the record that has the deleted key. Subtracts one from the record count.

Special Purpose KFAM Subroutines:

RELEASE Allows a User File record, previously protected by one CPU, to be accessed by any CPU. Also releases hog mode.

KFAM also provides a multiplexed file access system with four available access modes, security features including file Password protection and record protection in applicable access modes, and in general, rapid access to randomly-dispersed records. A non-multiplexed access mode is also provided.

ISS Subroutines

The general category of ISS Screen/Disk subroutines includes three groups of marked subroutines: the Disk, the Screen, and the Translate Table Subroutines. They are loaded into memory from displayed menus.

Disk subroutines, which simplify disk-related programming tasks, include the following:

1. Search Index: This Disk subroutine examines the Disk Catalog Index to determine if a particular file: (1) has been cataloged, (2) is scratched or active, and (3) is a data or program file.

2. Allocate Data File Space: This Disk 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 uniquencss of the file name and allows a minimum acceptable file size to be specified.

3. Free Unused Sectors: This Disk subroutine examines a selected file in a catalog area, de-allocates those sectors between the END trailer record 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 (reserved) sectors be maintained in the file area.

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

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

6. Limits Next: This Disk subroutine returns the name of the next file in the order of file entries in the disk catalog index. It also indicates whether the file is a program or data file and active or scratched.

7. Multiplexed File Open/End/Close: These Disk subroutines provide a controlled file access system for data files on a multiplexed disk drive. This access system is made possible by maintaining file access information in the file's (hardware-generated) catalog trailer record. Four access modes are available including Exclusive file access. Multiplexed file Open/End/Close subroutines support creation of a new file, accessing an existing file, changing access modes without closing a file, writing an END record, closing a file, and file password protection. A Set/Release Hog Mode subroutine is also included.

Screen subroutines, which simplify operator-related programming tasks, include the following:

1. Data Entry: This Screen 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 or default value.

2. Alphanumeric Input: This Screen 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.

3. Numeric Input: This Screen 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 replaces the prompting dashes.

4. Position Cursor: This Screen subroutine moves the cursor to any point on the CRT and, optionally, erases characters to the right of the new cursor position and also the lines below it. Both 64 x 16 and 80 x 24 character display screens are supported. Usually a PRINT or INPUT statement follows cursor positioning.

5. Date: This group of Screen subroutines converts and manipulates dates in Gregorian and Julian form. It includes a subroutine for operator entry of the date.

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

7. Print: This Screen subroutine allows a specified character to be printed a specified number of times.

8. Re-Enter: This (internal) Screen subroutine displays RE-ENTER to indicate invalid operator entries.

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

SORT-4 Subsystem (Disk Sort Standalone Routine)

SORT-4 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 dialogue that would otherwise be required for operator entry of the sort parameters. When sorting is complete, SORT-4 can load a specified application program module, and therefore can be used as a subsystem to an application program with its program linkage capabilities. It requires very little operator attention.

SORT-4 offers the following operational features:

1. For maximum efficiency, it uses the extended BASIC statements described in SORT STATEMENTS (Publication #700-3559).

2. The programmer may specify whether a key sort or a full-record sort is to be performed, or permit SORT-4 to decide. Both the key sort and full-record sort provide sorted output records which resemble their input record counterparts. In addition, a tag sort may be specified, in which case only the pointers to each input record's position on the disk are written into the output (or work) file, and not the records themselves.

3. SORT-4 operates in a multiplexed environment, under ISS-3 conventions.

4. Six input file formats are accepted:

   a) an ordinary cataloged data file,

   b) a BAS-1 data file,

   c) a data file opened and closed with ISS OPEN/CLOSE subroutines,

   d) a KFAM-3 file,

   e) a KFAM-4 file, and

   f) a KFAM-5 file.

5. 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 and sort keys may be partial fields, that is, a STR () function of an alphanumeric variable.

6. Input record formats supported include, in addition to the DC mode array-type blocking, the following record formats:

   a) Packed arrays, where the array-type blocking is packed for writing on disk, and written in either DC or BA mode.

b) Contiguous packed records, where each individual record is packed into a contiguous space within an alphanumeric array, and written on disk in either DC or BA mode.

c) Variable length records, packed into an alphanumeric array with either a one-byte length indicator (block size up to 256) or a two-byte length indicator (block size greater than 256). The block may be written in either DC or BA mode. TC (Telecommunications) Files are supported by a variable length record format.

d) Individual alphanumeric fields in records written in unpacked format, blocked or unblocked, may contain packed sub-fields.

In the above record formats, the field form of $PACK is supported. However, the internal and delimiter forms of $PACK are not supported. A record may contain either one packed array, or any number of packed fields. In addition to the formats defined for the field form of $PACK, Wang packed decimal format, signed and unsigned, is also supported; exponential as defined in the PACK statement is not supported.

Any combination of record format and file format is permitted, with the exception of BAS-1 format, which is allowed only with array-type blocking or packed arrays. Although KFAM itself does not support variable length records, SORT-4 will sort a KFAM file with variable length records.

7. When output files are written to a non-multiplexed disk drive after a full-record sort was specified, the mounting of the output platter may be deferred until the last pass, at which time the input platter may be removed. With a tag sort, deferred mounting is also allowed when the output file is not written to a multiplexed disk drive, and is not the work file. This permits sorting a full disk platter in a dual platter system.

8. 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.

9. SORT-4 treats arrays in input records as arrays. An input record may contain up to 255 fields, with each array element counting as one field, provided that the record can be described in not more than 60 table entries.

10. SORT-4 allows a full-record sort on records up to 256 bytes, packed. It also allows a full-record sort with partial fields as sort keys. In most cases a full-record sort will be faster than a key sort on the 2200VP.

11. Sorting of KFAM files should be faster with SORT-4 than SORT-3, because special-purpose subroutines have been written to access a KFAM file in FINDFIRST/FINDNEXT sequence.

12. A starting and ending key may be specified for sorting a KFAM file, instead of a starting record number and the number of records to be sorted.

[illegible]

WANG LABORATORIES, INC.

MEMO TO: BOB SOUCY

FROM: TOM CAMP

SUBJECT: ISS RELEASE 4.9 (PRE-RELEASE 5.0)

DATE: JUNE 18, 1979

ISS 5.0 will be available within the next few weeks. Complete ducumentation will be distributed shortly thereafter. This memo is intended to highlight the major new features of this 2200 software product. Please see that our field analyst organization is aware of this information.

For certain critical situations, we can distribute a pre-release version of the utilities and KFAM-7. Should you know of an urgent need, please contact me. We would of course, like to ship as few pre-releases as possible.

ISS Utilities

1. Disk Dump

   This utility now supports the CRT (for vertical dump only). The EDIT keys allow the user to step through the file.

2. Program Compare

   Also supports the CRT for error messages and now provides an error limit feature which will terminate the compare after a certain number of errors.

3. Alter Disk Index

   A new utility designed to display the disk index and delete unused files at the end of the catalog. It also allows for changing names and status (active, scratched) of the file.

KFAM-7

KFAM-7 in the Multiple Bank Pre-release version contains the following modifications.

1. It supports the 2200MVP with extended memory (up to 256K).

2. It supports the 2280 (Phoenix) disk.

3. The reset access table utility has been modified to clear internal tables and close all, or specified, KFAM files for a given station or all stations.

4. The print key utility now includes a CRT display option.

5. All utilities use input modules and display defaults.

The new release of KFAM-7 will require the following changes to application programs.

1. The last parameter in the OPEN statement (GOSUB' 230), user file device address, formerly not used in KFAM-7, will be used, and will be important in identifying the file. Both the user and key file device addresses are selected during KFAM OPEN.

2. Device addresses are limited, as follows:

   Diablo or floppy disk:

   310, B10
   320, B20
   330, B30
   350, B50
   360, B60
   370, B70

   Phoenix disk:

   B10 or D10, 310 or D11, D12 through D15
   B20 or D20, 320 or D21, D22 through D25
   B30 or D30, 330 or D31, D32 through D35
   B50 or D50, 350 or D51, D52 through D55
   B60 or D60, 360 or D61, D62 through D65
   B70 or D70, 370 or D71, D72 through D75

The open subroutine will check the user file and key file device addresses and return an error condition (Q$ = "X") if it is not one of the above.

NOTE: The disk addresses are now related by the KFAM subroutine 'OPEN' to the specified device numbers.

3. In the Multiple Bank version of the KFAM-7 subroutines, global variables reside in partition

"KFAMCOM" and not "KFAM". User programs accessing KFAM global variables directly, must be modified to select "KFAMCOM" and then select back to "KFAM" to access the subroutines.

4. The Single Bank Version of the KFAM-7 subroutines, model KFAM0107, requires a partition size of 9.5K, starting with Release 5.0.

Tom Camp

TC:pn

KFAM-7 SUBROUTINES, MULTIPLE BANK VERSION

On the 2200MVP with extended memory, global partitions can be accessed only within a bank, except for a 5K common area which can be accesssed by all banks. The KFAM subroutines will not fit in 5K and therefore must be duplicated once for each bank accesssing KFAM. But the KFAM global tables must appear only once, in order to control disk access, and these tables must be available to all partitions using KFAM. Therefore, in order to run on a 2200 MVP with extended memory, the KFAM-7 subroutines must be split into two sections. One section contains only global variables, and is stored in the 5K common area. The other section contains subroutines only, and is duplicated once for each bank accessing KFAM.

This split version of the KFAM-7 subroutines is being added to KFAM-7 in the form of two new modules. KFAM0307 will contain only subroutines, and will reside in a partition called "KFAM" on each bank. This version of the subroutines will be referred to as the "Multiple Bank version". KFAM0407 will contain only KFAM global variables, "KFAMCOM" in order for KFAM0307 to run.

This means that there are now three versions of the KFAM-7 subroutines:

1. KFAM0107, Single Bank version
2. KFAM0207, MUX version
3. KFAM0307, Multiple Bank version

The Single Bank version may be used on an MVP with up to 64K of memory not multiplexed. The MUX version must be used if the MVP (any size) is mutiplexed with another CPU (2200T, VP or MVP) to the same disk. If the MUX version is used, it must be used by all programs concurrently accessing any given KFAM file. The Multiple Bank version can be used on any size MVP, but not if it is multiplexed.

In most cases, no changes are required to user programs in order to use the Multiple Bank version of the KFAM-7 subroutines. The user selects the "KFAM" partition as before. KFAM selects "KFAMCOM" as necessary and selects back to "KFAM" before returning control to he user program. If the user program refers to KFAM global variables, then the user must select "KFAMCOM" and then select back to "KFAM" to access the subroutines.

The KFAM-7 utililties are modified to run with whatever version of the subroutines happen to be resident in the "KFAM" partition. They will also run on the VP by using overlays.

Module KFAM0307 requires a "KFAM" partion size of 8.25K per bank. Module KFAM0407 will require 2.75K (including partition overhead) in partition "KFAMCOM".

In some cases, the KFAM global tables may not be large enough to accomodate all programs accessing KFAM files on a 256K system. @T$(30)17 contains one entry for each KFAM file currently open. @V4$(30)5 contains one entry for each sector currently being protected, in any file, by any program. The penalty for table @T$() being full is error code Q$="S" being returned from the OPEN subroutines. If @V4$() is full, the program simply waits until there is a slot available. Either of these tables may be expanded upwards from 30 by simply changing the array dimension in module KFAM0407.

KFAM-7 will support a maximum of 16 "stations" as before. These "station" numbers can be terminal numbers, partition numbers, or any other numbers assigned by the user, as long as no two programs are accessing KFAM with the sam station number at the same time.

PHOENIX DISK

The KFAM-7 utilities will be modified to support the 2280 "Phoenix" disk (16-bit sector address). KFAM-7 will also support the device addressing scheme on the Phoenix disk, for upwards compatability, where B10 is the same as D10, 310 is the same as D11, B20 = D20, 320 = D21, and so forth.

To determine whether it is a Phoenix disk, KFAM uses the $GIO statement:

```
$GIO #n, (70A0400870B,X$)
```

If this returns HEX (DO) in byte 11 of X$, then it is a Phoenix disk. Values of HEX (CO) through HEX (CF) indicate another disk. Byte 8 of X$=HEX(00) if the operation was ok.

RESET ACCESS TABLE UTILTIY

This utility has been enhanced to support clearing a single station or all stations, all files or files specified by part or indirect mode from a configuration.

TABLES CLEARED

	KDR	TABLE OF OPEN FILES	QUEUE	PROTECTED SECTORS	PROGRAM HOG	MUX ACCESS TABLE
SINGLE		X	X	X	X	X
MUX	X				X	X
MULTIPLE		X	X	X	X	X

In addition, KFAMWORK will be cleared (at the system address) automatically, if 'ALL' mode is used, or if specified (at a given address) when 'PART' or 'INDIRECT' mode is used:

IN GENERAL	ALL MODE	PART, INDIRECT MODE
STATION N=0	CLEARS ALL FILES IN GLOBAL ACCESS TABLES FOR ALL STATIONS	CLEARS SPECIFIED FILES FOR ALL STATIONS
STAION N 0	CLEARS ALL FILES FOR STATION N	CLEARS SPECIFIED FILES FOR STATION N

Note: 'ALL' mode is not supported for the MUX version.

This utility should cover most situations where a particular station has crashed in the middle of a program. It will not close file which are opened only with MUX OPEN and not KFAM OPEN. It does not write "END" records or save recovery information. If the program has crashed in the middle of a FINDNEW or DELETE operation, it may be necessary to run KEY FILE RECOVERY to reconstruct the Key File.

PRINT KEY FILE

The utility now includes CRT output with the following options:

'3		-in case of KIE's too long for one screen, switch from hex key to remaining information and back.
'4	END	-display last KIR
'6		-display KDR information
'7	BEGIN	-display first KIR
'8		-specify KIR to display
'9		-redisplay current KIR
'10		-print current KIR (or KDR)
'11		-jump ahead 5 sectors
'12		-jump ahead 1 sector
'13		-jump back 1 sector
'14		-jump back 5 sectors
'20	END	-display last KIE in KIR
'23	BEGIN	-display first KIE in KIR
'25		-redisplay at current KIE
'27		-display next 5 KIE's
'28		-display next KIE
'29		-display previous KIE
'30		-display previous 5 KIE's

If the MUX sector of a particular file is not positively identified, the program displays:

```
FILE NOT FOUND
USE ISS FILE STATUS REPORT TO CLOSE FILES
```

This may be one file, or more. The operator should be provided with a list of opened files for every application, and use the ISS Utility to close all files for this station.

TABLE OF OPENED FILES, @T$()

The file identification scheme is table @T$() has been changed, first because of the equivalency of different device addresses on the Phoenix disk (B10=D10,310=D11, etc) and second because it is necessary to find the last sector of the User File for the CLOSE STATION utility, in order to reset the MUX access table.

FORMERLY the first 3 bytes of each entry of @T$() WERE:

Bytes 1-2: KDR sector (VO$)
Byte 3: Key file device address, packed. (HEXPACK 2nd and 3rd bytes of device address, and or HEX (80) if first byte is "B").

This field identifier is changed as follows:

Bytes 1-2: KDR sector (VO$)
Byte 3: Key File device address, packed. (If first byte of device address is "3", make the third byte = "1". Then HEXPACK 2nd and 3rd bytes of device address).

In addition the User file address and device number have been inserted into the table beginning at byte 4 as follows:

Byte 4-5: Last sector of User File, HEX.
Byte 6: User file device address, packed. (See above).

This new packing scheme for the device address insures that any given file location can result in only on packed file identifier:

Device Address	Packed (hex)
B10 or D10	10
310 or D11	11
D12, etc	12, etc.
B20 or D20	20
320 or D21	21
D22, etc	22, etc.

Hog mode addresses (8 added to second byte) are now rejected by the KFAM OPEN subroutine.

Note that where certain device addresses are equivalent for the Phoenix disk (B10 and D10, 310 and D11, etc), they may be specified either way from program to program.

The Use of MULTIPLE KEY FILES and DUPLICATE KEYS with KFAM-7

ISS RELEASE 5.1

KFAM has been developed to maintain a file of pointers (Key File) to a sequential file with records in random order based on some unique field within each record. Sometimes it is worth maintaining a second file of pointers (Secondary Key File) based on some other field within each record which may not be unique. An example might be an employee file for which a primary key file is maintained based on social security numbers (unique) as well as a secondary key file based on department numbers (duplicates allowed). Thus any individual employee's record can be located by finding the social security number and pointer in the primary key file. Also, the records for every member of a department can be located by finding all pointers corresponding to the department number in the secondary key file.

Although KFAM-7 does not maintain two identical keys at any time, a convention for building and updating a key file for duplicate keys is recommended and supported. For such a system, several stages are significant.

I. Initializing a KFAM file for duplicate keys:
The supported convention for duplicate keys requires that a key length 3 bytes longer than the actual key be specified when the file is initialized. The use of these three bytes is described below. One complication that may arise during initialization has to do with the checks made to ensure that the key does not exceed record or sector boundries. When DUPLICATE key type is specified, the checks mentioned are made on the actual key lengths, not the extended length which includes the three extra bytes. In addition, it must be decided whether or not recovery information should be written. This information contains a copy of key file parameters fron KDR (Key Discriptor Record) and is maintained in the next to last sector of the user file. Since only one set of recovery information can be maintained in this sector, it is probable that WRITE recovery information should be specified when initializing the primary key file and DON'T WRITE recovery information should be specified when initializing the secondary key file.

II. Opening a KFAM file:

The primary key file is opened first by a station. Since the user file is opened by the multiplex convention by the station, a special parameter (the negative of the key file number) is used to reopen the user file as the station does KFAM open on the secondary file(s). In the case of the single bank and multiple bank version of KFAM-7, open files are identified by their KDR (first sector, key file) sector address and packed device address. Hence each of the key files is uniquely identified even though several of them may be referring to the same user file. In this way, file access integrity is maintained.

EXAMPLE:

```
GOSUB '230 (1,1,2,1, "KFAMF010", 3, "D10", "D10") :REM, OPEN
PRIMARY
GOSUB '230 (2,3,4,-2, "KFAMF010", 3,"D10, "D10") :REM OPEN
SECONDARY
```

III. Adding a record:

When adding a record to the user file, the primary key must be added to the primary key file and the secondary key must be added to any secondary key file. In the case when the secondary key is not necessarily unique, the user must create such a unique key. The procedure recommended here is to enter the key in the primary key file using FINDNEW. This will produce a pointer T4$ indicating relative sector and record within sector. The secondary key can be created by concatenating this pointer with the secondary key field. Since the pointer is unique to the record, the secondary key thus created becomes unique. A FINDNEW (HERE) is then used to enter this created secondary key in the secondary key file. Note that this procedure requires that a key length 3 bytes longer than the actual secondary key has to be specified when the secondary key file is initialized.

NOTE: FINDNEW (HERE) is recommended for secondary key files since there will be no need to cross check the pointer obtained by FINDNEW on different key files. As before, if different files are to be used, the pointer must be set up in the new file's parameters prior to the call (see example).

```
EXAMPLE: GOSUB '233 (1,1,K1$,0) : REM FINDNEW FOR PRIMARY KEY
         STR(K2$, L+1,3)=T4$    : REM SET UP UNIQUE SECONDARY KEY
         STR(T5$(2),1,3)=T4$    : REM PREPARE TO SWITCH FILES
         GOSUB '234 (2,1, K2$,0): REM FINDNEW(HERE) FOR SECONDARY
```

IV. Deleting a record:

Using the same convention discussed under adding keys, deleting a record could be accomplished in the following manner. First, DELETE the key from the primary key file. Then obtain the unique secondary key by taking the secondary key field and concatenating the pointer T4$ obtained from the original DELETE. DELETE this unique key from the secondary key file. As always it is recommended that the deletion of the record in the user file be indicated by setting the first byte of the key field to HEX(FF). Since it may be necessary to use BUILD KEY FILE to create secondary key files at some point, the first byte of every secondary key field should be set to HEX(FF) when the record is deleted.

EXAMPLE:

```
GOSUB '231 (1,1,K1$): REM DELETE ON PRIMARY
STR(K2$,T4-3,3)=T4$ : REM SET UP UNIQUE SECONDARY KEY
GOSUB '231 (2,1,K2$): REM DELETE ON SECONDARY
```

V. Locating a record:

If a primary key is known, it is sufficient to use FINDOLD. If only a secondary key is known and that key may occur several times, a special procedure is required. The secondary key is concatenated with the lowest possible pointer, T4$=HEX(000000) and FINDOLD is called. This positions the routine at the first of the duplicate keys. There after FINDNEXT is used until the key obtained T7$ (not counting the last three bytes) changes, indicating a new secondary key value encountered.

EXAMPLE:

```
1000  STR(K2$,L+1,3)=HEX(000000)
1010  GOSUB '232(2,1,K2$): REM FINDOLD (ON SECONDARY)
1020  IF Q$="X" or Q$="B" THEN EXIT
1030  GOSUB '237(2,1)  :REM FINDNEXT (ON SECONDARY)
1040  IF Q$="E" THEN EXIT (END OF FILE)
1050  IF Q$="X" or Q$="B" THEN EXIT
1060  IF STR(K2$,1,L)<>STR(T7$,1,L)THEN EXIT (NO MORE KEYS)
1070  PROCESS RECORD
1080  GOTO 1030
```

VI. Protecting records while working on them:

When sector protection is required, in order to keep other stations from accessing a record during processing by a station, the sector is placed in a global table of protected sectors indicating the station and key file protected as well as the packed disk address and sector identifier (obtained by taking the trailer record of the users file SUBC the relative sector within the user file). The global table is available in the Single Bank and Multiple Bank version of KFAM-7 only. The MUX version keeps track of sector protection in the KDR (first sector of the key file) and is adequate as long as only one key file exists for a user file. If more than one key file exists for a user file, no mechanism exists to cross check KDR's and sector access integrity can be violated. For the mux version, the users will have to establish sector protection schemes of their own, such as maintaining a list of protected sectors or setting a protection flag byte on the record itself.

Note that the global tables of protected sectors in KFAM-7 has been modified so that a sector protected by a station via one key file can be accessed by the same station via a second key file. Should the user want to release the sector, it will be necessary to release it for all key files that have accesssed it by that station.

VII. Closing Files:

When it is time to close files, two consideration arise when dealing with multiple key files. First, only one set of recovery information is maintained (next to last sector, user file). This recovery information is used in case of accidental destruction by the utility KEY FILE RECOVERY, thus avoiding the need to initialize (with all KDR specifications). It is likely that the user will want this information to refer to the primary key file. A special parameter for CLOSE (negative file #) is used to indicate that the recovery is not to be written in the user file and should be used for closing the secondary key files. In addition, FINDNEW(HERE) never updates the number of sectors used in the user file. If an END record were to be written in the user file while closing a secondary key file built by FINDNEW(HERE) calls only (as recommended), it would be written in the first sector of the user file, destroying records and altering the actual numbers of sectors used. The same parameter specified above indicates that an END is not to be written in the user file.

EXAMPLE:

```
    GOSUB '239(1): REM CLOSE PRIMARY FILE
    GOSUB '239(-2): REM CLOSE SECONDARY FILE
```

VIII. Build Key File:

The BUILD KEY FILE utility can be used on an initialized key file (initialize does not affect an already existing user file). A parameter "KEY TYPE" (standard or duplicate) will indicate if the duplicate key convention described here is to be used. It can be run for the primary key files and all secondary key files. Note again that deleted records should have the first byte of all key fields set to HEX(FF). Otherwise BUILD KEY FILE will file keys on records deleted according one key but not another. Also note the need to specify and extra three bytes when initializing a key file containing duplicate keys. In addition to deleted records, there are other records which should not have keys added for them. These are indicated by the Findnew Sector Table as portions of sectors assigned to a station. When blocked records are involved, a station will be given a sector in the user file in which to put records until the sector is filled. A new sector will be assigned for additional insertions. The Findnew Sector Table is maintained in the KDR and a copy is maintained in the recovery information. Sometimes, especially with secondary key files which use Findnew (here) it is necessary to use the findnew sector table as stored in the recovery information to locate any unused portions of sectors assigned to the various stations. For this purpose, USE/DON'T WRITE recovery information should be used. On the other hand, WRITE/DON'T USE recovery information should typically used on the primary key file.

IX. Key File Recovery:

It is unlikely that a user would want to maintain recovery information in the user file for a secondary key file or for a key file allowing duplicate keys. However, for flexibility, the KEY TYPE (DUPLICATE or STANDARD) parameter has been included in the KEY FILE RECOVERY utility. Remember that whatever KDR information has been stored as recovery information in the user file will be used to recover the key file. (Note that a secondary key file built by FINDNEW(HERE) will not indicate any sectors used in the user file, hence an end record would be written in the first sector of the user file). Typically, the procedure to recover multiple key files for a user file would have several steps. First use KEY FILE RECOVERY to recover the primary key file. Next use INITIALIZE KEY FILE with the DON'T WRITE recovery information option to initialze any secondary key files. Finally use BUILD KEY FILE specifing USE/DON'T WRITE recovery information on the freshly initialized seondary key files.

X. Reorganizing KFAM files in the Multiple Key File, Duplicate Keys environment:

Two means of reorganization are supported for KFAM-7, the REORGANIZE IN PLACE utility and the reorganize subsystem. Both can be used to move the records into order in the user file and rebuild the key file as is required for a primary key file. After reorganizing a KFAM file according to its primary key, it will be necessary to rebuild any secondary key files (all the pointers will have to be changed since the records have been moved around). During this rebuilding stage, the user file records cannot be moved around again. The following procedures are provided for reorganization of Multiple key files.

1. Reorganizing via the KFAM-7 menu:
   Use the REORGANIZE IN PLACE utility to reorganize the primary key file. (NOTE: it is usually preferable to use the REORGANIZE subsystem since it is faster and no backup is required. REORGANZE in place should only be used if there is not enough space available for a work file). Next use INITIALIZE KEY FILE with the DON'T WRITE recovery information option to initialize any secondary key files. Finally use BUILD KEY FILE specifing USE/DON'T WRITE recovery information on the freshly initalized secondary key file.

2. Reorganizing under program control
   The same approach to reorganizing a file with multiple keyfiles is used for program control as is used via the KFAM-7 menu, i.e. reorganize according to the primary key file and then rebuild each of the secondary key files. For this purpose a new module has been added to the Reorganize Subsystem as well as several new parameters. 06$="B" indicates "BUILD KEY FILE" as opposed to "REORGANIZE". 07$="S" indicates that the recovery information and end record are not to be written for the user file (usually the case with secondary key files as indicated in the discussion on closing files). 08$="D" indicates that the duplicate key convention is to be used in reorganizing or building the key files. The following example indicates the way these parameters might be used to reorganize a KFAM file consisting of a user file, a primary key file with unique keys, and two secondary key files, one of which has duplicate keys.

KEY FILE #1	KEY FILE #2	KEY FILE #3
06$="C"COPY BACK	06$="B"BUILD KEY FILE	06$="B" BUILD KEY FILE
07$=" "WRITE RECOVERY	07$="S"SKIP RECOVERY	07$="S" SKIP RECOVERY
08$=" "STANDARD KEYS	08$="D"DUPLICATE KEYS	08$=" " STANDARD KEYS

Note that for Build Key File, parameters relating to Output user and key files are ignored. Also note that some of the recovery information is used by Build Key File under program control. Specifically, unused record slots in blocked sectors are flagged as deleted. This requires that the recovery information and end record are kept up to date. Such updating is done normally whenever a file is closed (DEFFN'239). In addition, one can use WRITE RECOVERY INFO after executing FINDNEW on the primary key file.

3. Alternatively the user may want to write a program to rebuild the secondary key files. This would begin by initializing the secondary key files. The easiest way to initialize a key file under program control is to copy the first two sectors of the initialized Key File to a separate data file and then recopy them to the original key file at initialization time. This restores the KDR and the first KIR of the key file. The next step is to open the primary and freshly initialized secondary key files. FINDFIRST is done on the primary file, producing the pointer T4$. The secondary keys can be determined from the record and concatenated with T4$. FINDNEW(HERE) is done on the secondary key files. The procedure is then repeated using FINDNEXT until Q$="E" is returned.

XI. Reallocate KFAM File Space:

REALLOCATE KFAM FILE SPACE is designed to update a KFAM key file and user file to reflect modified file sizes. In the case of a primary key file this involves updating the KDR in the key file and rewriting the recovery information and end record in the user file. In the case of the secondary key file only the KDR is updated. It is important for secondary key files that the recovery information and the end record are not written. (Again, note that a file built by FINDNEW(HERE) only will not indicate any sectors used and will write an end record in the first sector of the user file). The KEY FILE TYPE parameter should indicate PRIMARY or SECONDARY as appropriate. Note that as of ISS Release 5.1, the upper bounds of the User and Key files are updated whenever OPEN (DEFFN'230) is executed. Thus it is no longer necessary to use this utility following file size changes.

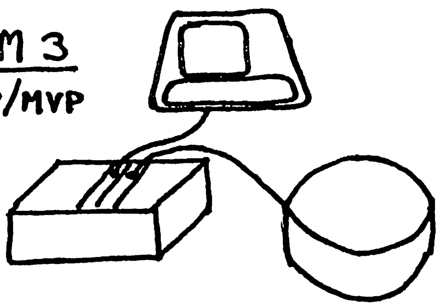
KFAM-7
TRAINING MATERIALS
ISS RELEASE 5.1

1. CONFIGURATION
2. STRUCTURE
3. TABLES - KDR/LOCAL/GLOBAL
4. LOGIC - QUEUE/SECTOR PROTECT
5. KFAM-7 UTILITIES
6. MULTIPLE KEY FILES/DUPLICATE KEYS
7. ISS 5.1 UTILITIES/SUBROUTINES

②

KFAM 3
T/VP/MVP

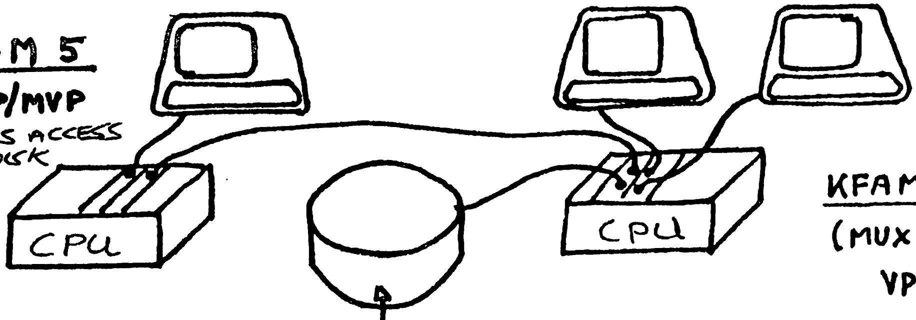
AINTAIN A
EY FILE INDEX
R A USER FILE,



NO ACCESS CONTROL TABLES

KFAM 5
T/VP/MVP

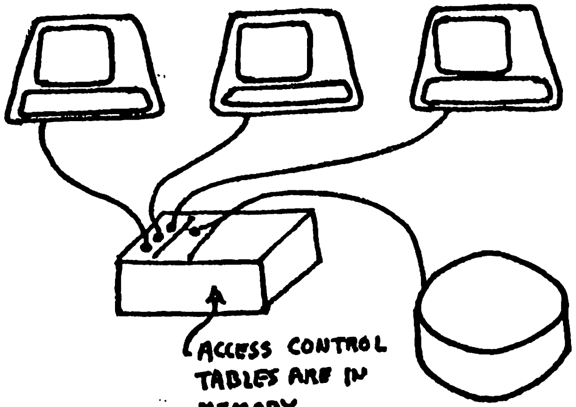
CONTROLS ACCESS
TO THE DISK
BY EACH
USER.



ACCESS CONTROL TABLES
ARE ON DISK

KFAM 7
(MUX VERSION)
VP/MVP

ALLOWS CPU'S TO
COMMUNICATE TO
EACH OTHER.



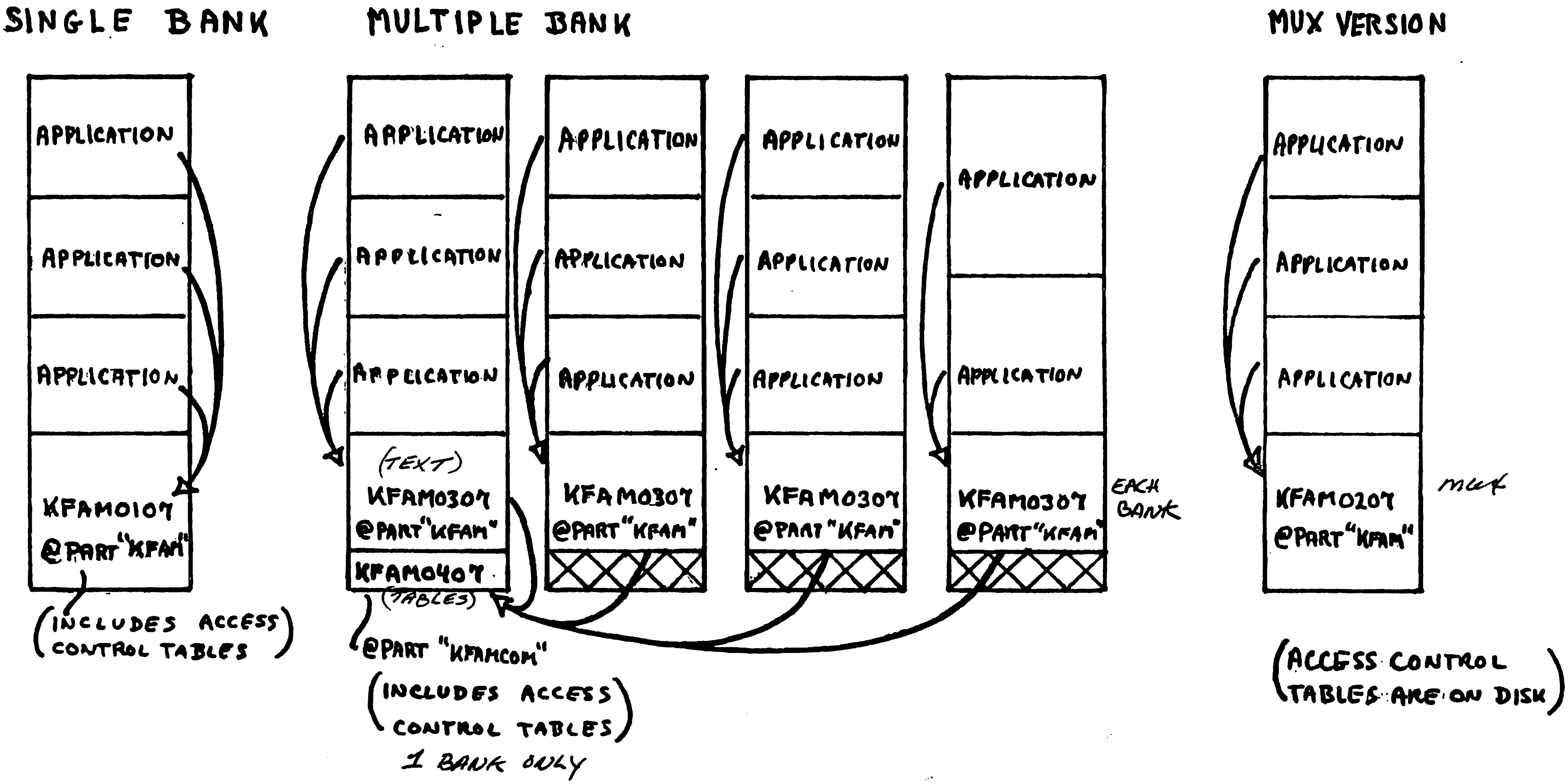
ACCESS CONTROL
TABLES ARE IN
MEMORY

KFAM 7

(SINGLE / MULTIPLE BANK VERSIONS)
VP/MVP

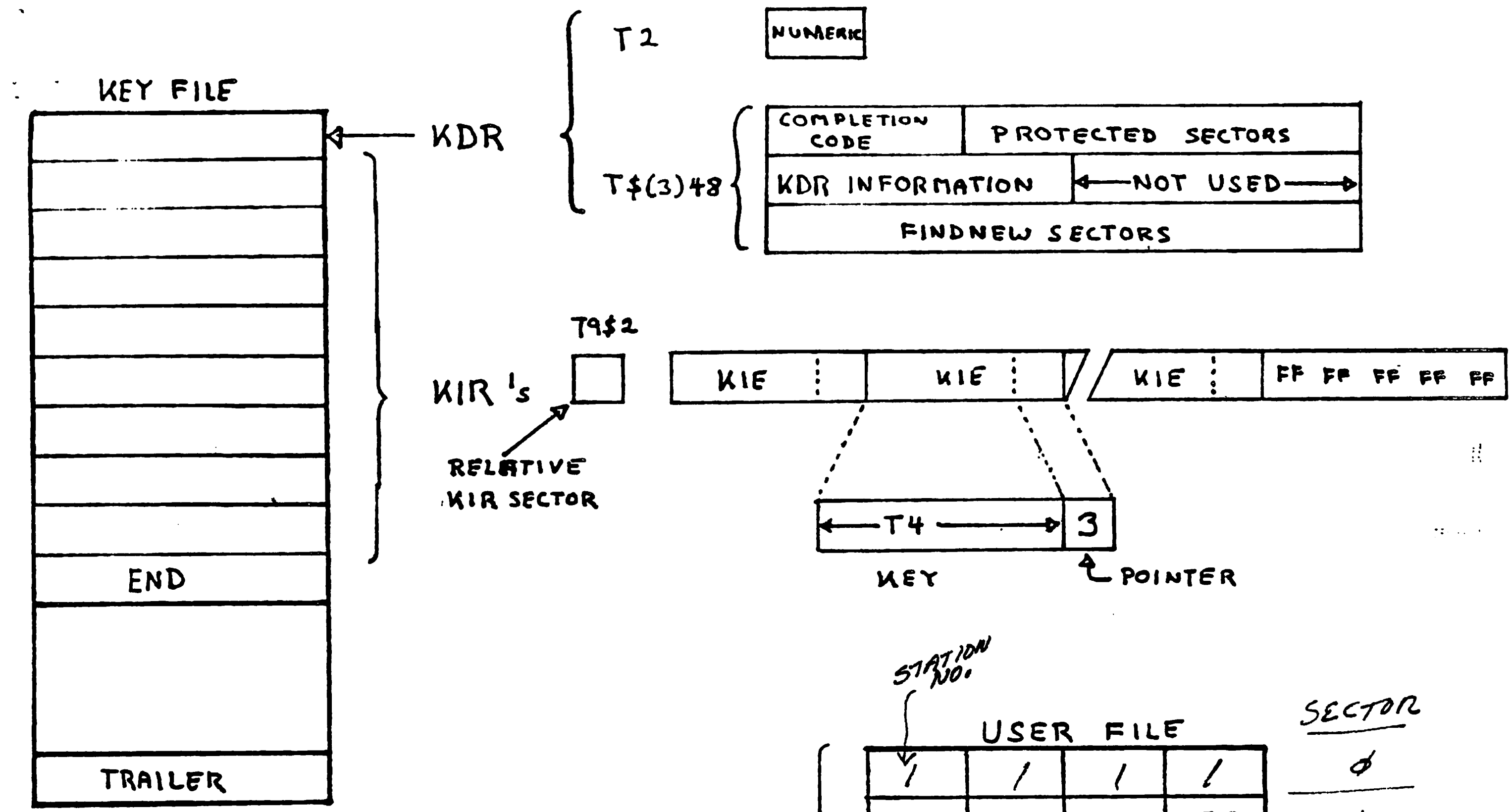
KFAM CONFIGURATIONS

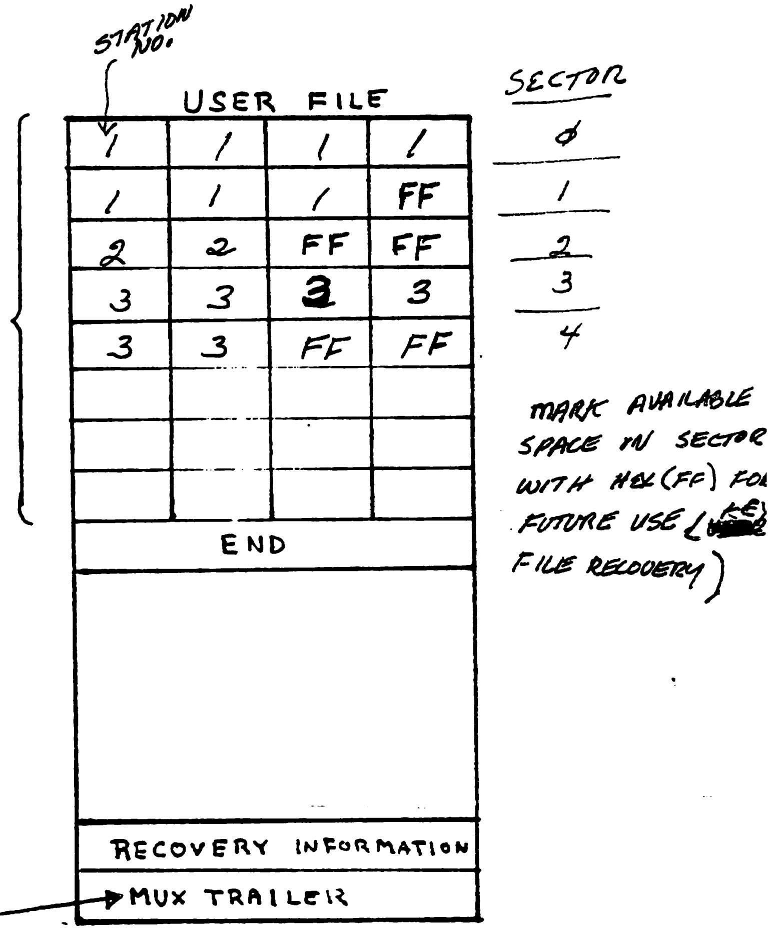
③ KFAM 7 MEMORY CONFIGURATIONS:



KFAM FILE LAYOUTS

KDR RECORD

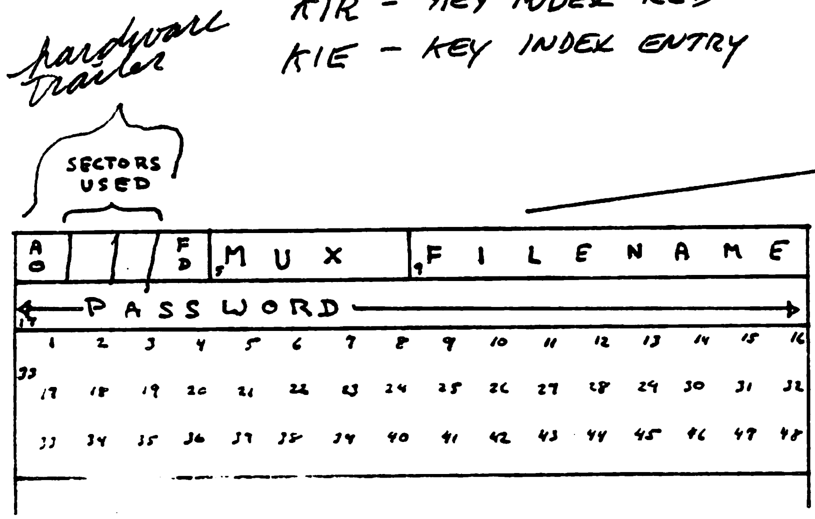




ACCESS MODES

	1	2	3	4	
ENQUIRY	Y	Y	Y	N	user read other's read
READ ONLY	Y	Y	N	N	user read others read
SHARED	Y	N	Y	N	user read & write other's read & write
EXCLUSIVE	N	N	N	N	user only other's excluded

KDR - KEY DESCRIPTOR RCD
KIR - KEY INDEX RCD
KIE - KEY INDEX ENTRY



FINDNEW (1)

ADDING KEYS FORCES KIR SPLIT AND A NEW LEVEL

SECTOR 1					SECTOR 2					SECTOR 3					(HLKIR)
4 0/1															(1)
4 0/1	8 0/2														(1)
4 0/1	6 0/3	8 0/2													(1)
⋮															⋮
2 0/4	4 0/1	6 0/3	8 0/2	10 1/1											(1)
2 0/4	4 0/1	5 1/2			6 0/3	8 0/2	10 1/1			2 1/	6 2/				(3)

sector

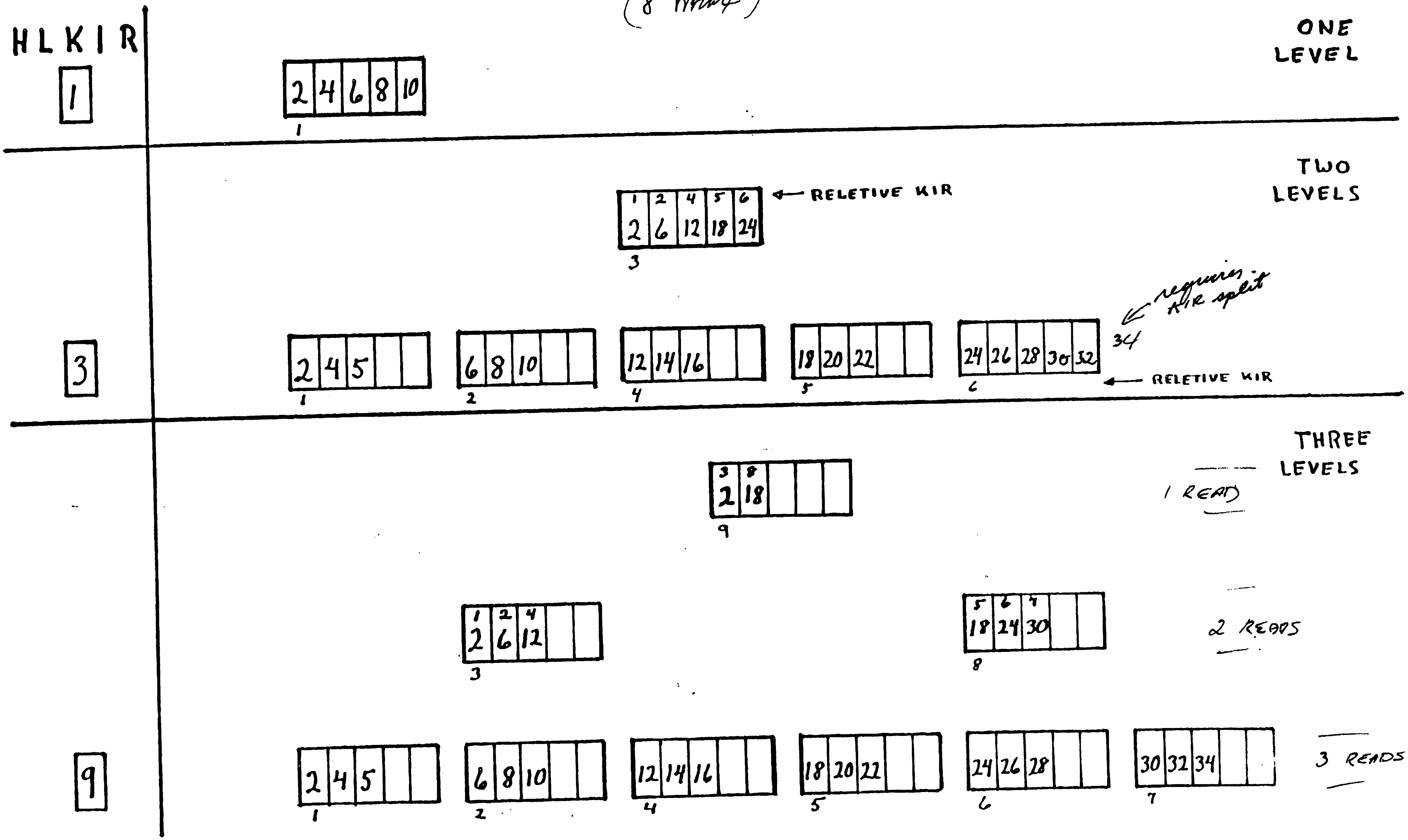
record no.

1ST entry in preceding sector.

FINDNEW(2)

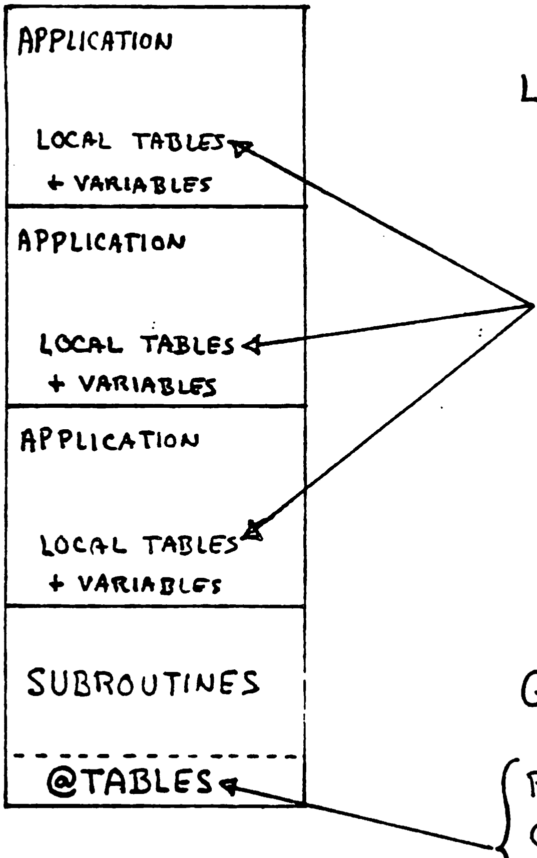
MULTIPLE LEVEL KEY FILES

(8 max)



LOCATION OF KFAM TABLES

(IN MEMORY)



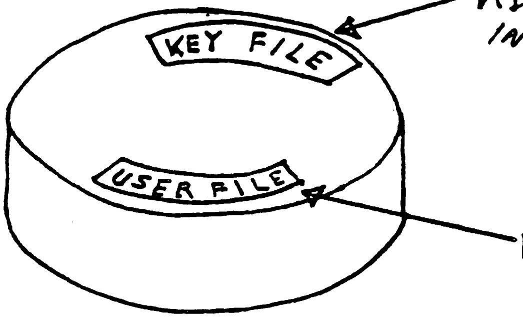
LOCAL TABLES (BY KFAM I.D.)

OLD PARAMETERS (FROM LAST CALL)	T5$(3)58
ACTIVE FILE INFORMATION	V0$(3)21
NAMES	V7$(3)8
DEVICE NUMBERS	T0$16

GLOBAL TABLES

PROGRAM HOG @T
QUEUE @Q$, @Q9$, @Q
TABLE OF OPEN FILES @T$(30)17
TABLE OF PROTECTED SECTORS @V4$(30)5

(ON DISK)

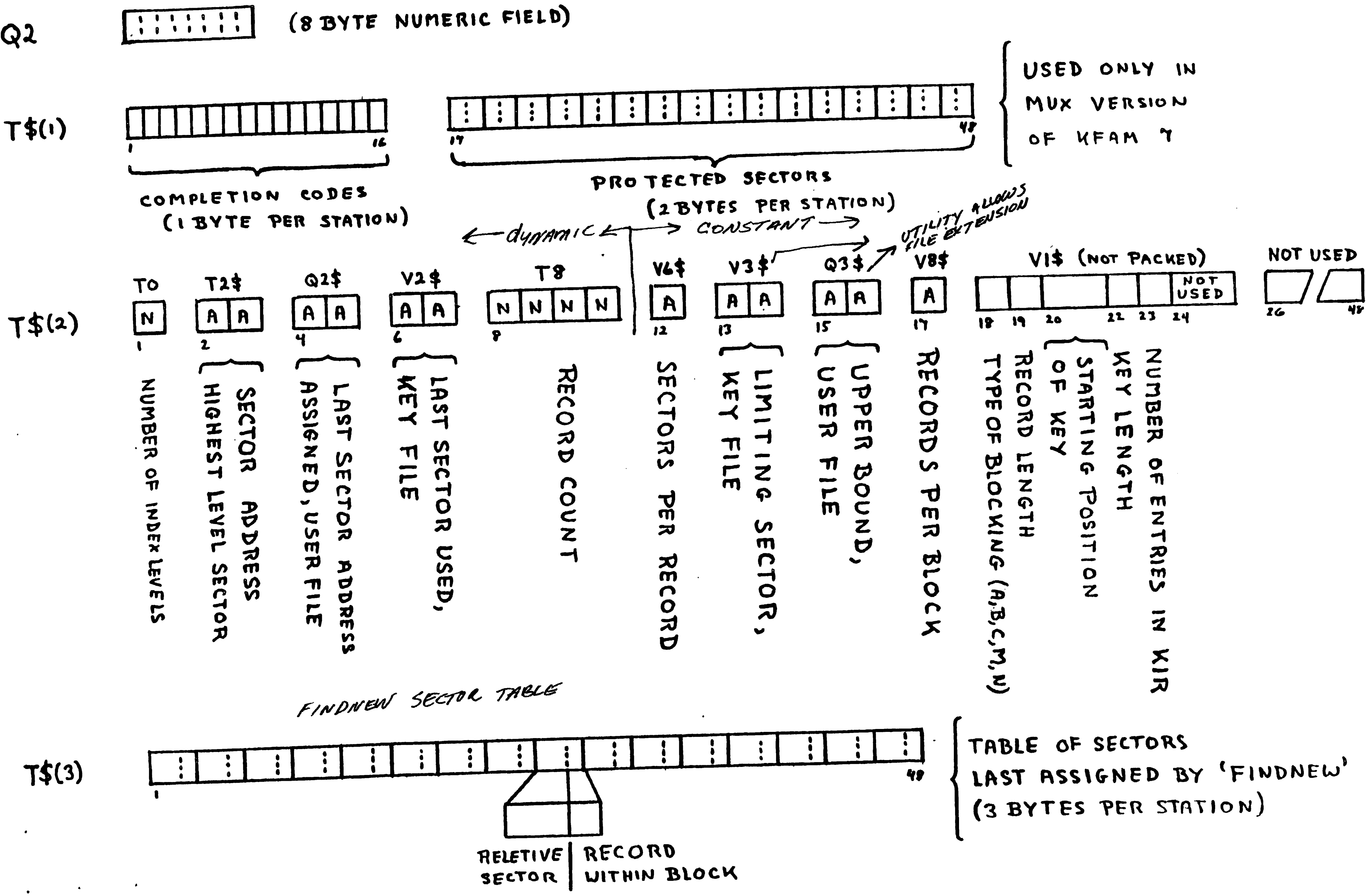


KDR TABLES
IN KEY FILE

MUX ACCESS TABLES
IN USER FILE

T$ (48) 3

KDR:

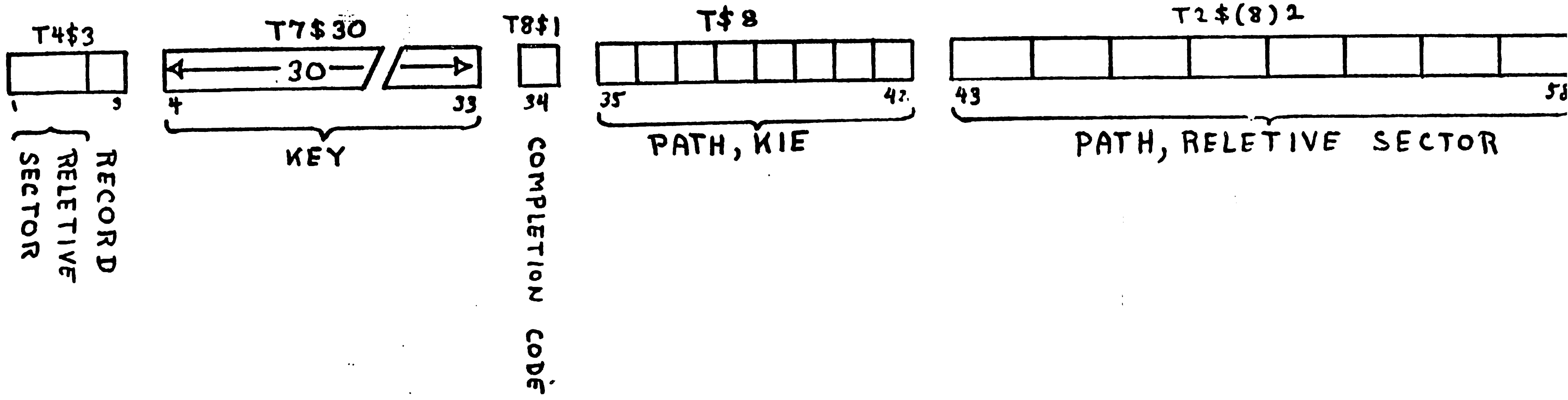

Q2
(8 BYTE NUMERIC FIELD)
T$(1)
1
16
COMPLETION CODES (1 BYTE PER STATION)
17
48
PROTECTED SECTORS (2 BYTES PER STATION)
USED ONLY IN MUX VERSION OF KFAM 7
← dynamic ← → CONSTANT →
UTILITY ALLOWS FILE EXTENSION
T$(2)
T0
N
1
NUMBER OF INDEX LEVELS
T2$
A A
2
SECTOR ADDRESS
HIGHEST LEVEL SECTOR
Q2$
A A
4
LAST SECTOR ADDRESS
ASSIGNED, USER FILE
V2$
A A
6
LAST SECTOR USED,
KEY FILE
T8
N N N N
8
RECORD COUNT
V6$
A
12
SECTORS PER RECORD
V3$
A A
13
LIMITING SECTOR,
KEY FILE
Q3$
A A
15
UPPER BOUND,
USER FILE
V8$
A
17
RECORDS PER BLOCK
V1$ (NOT PACKED)
NOT USED
18
19
20
22
23
24
TYPE OF BLOCKING (A,B,C,M,N)
RECORD LENGTH
STARTING POSITION
OF KEY
KEY LENGTH
NUMBER OF ENTRIES IN KIR
NOT USED
26
48
FINDNEW SECTOR TABLE
T$(3)
1
48
RELETIVE SECTOR
RECORD WITHIN BLOCK
TABLE OF SECTORS LAST ASSIGNED BY 'FINDNEW' (3 BYTES PER STATION)

⑥ LOCAL TABLES:

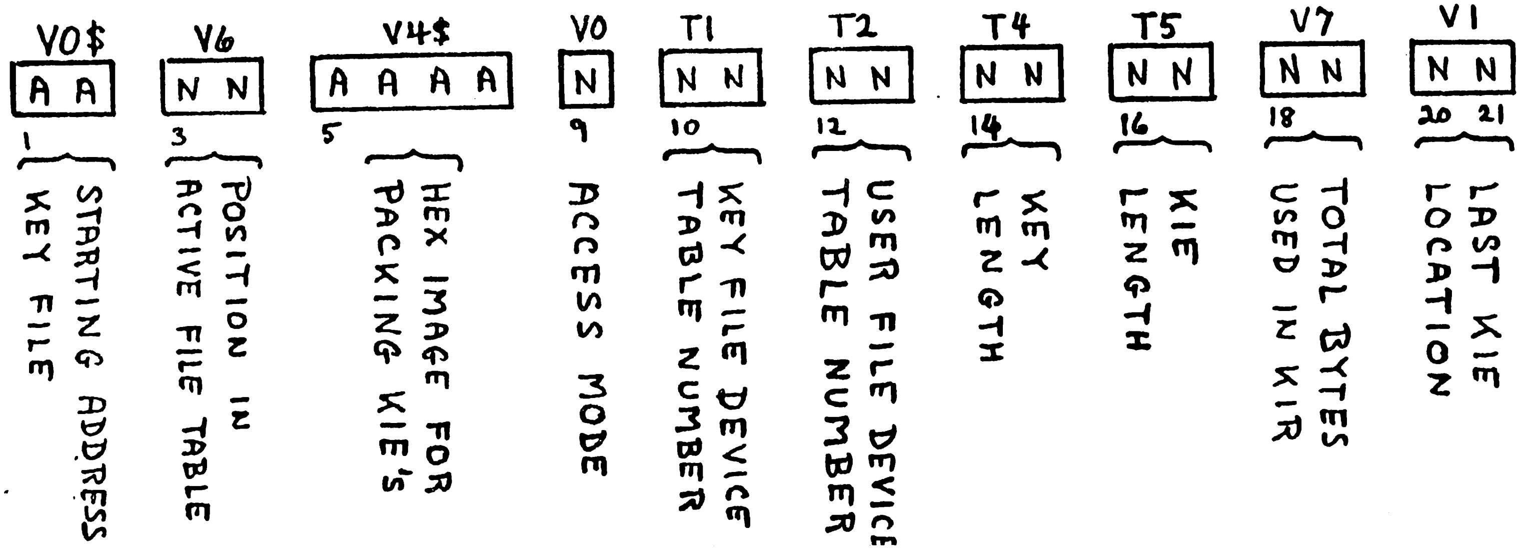
V7$(3)8 - TABLE OF ACTIVE FILE NAMES

TO$ 16 - TABLE OF ACTIVE DEVICE TABLE NUMBERS

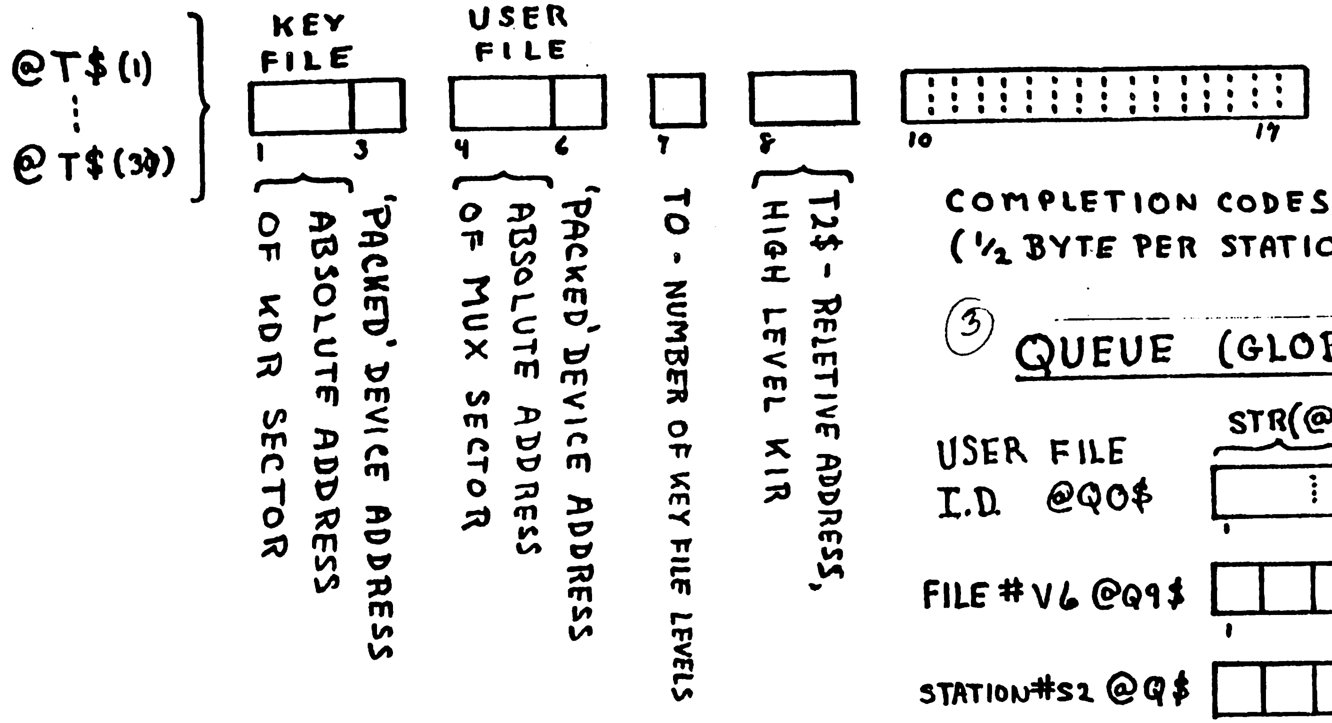
T5$ (3)58 - TABLE OF OLD PARAMETERS



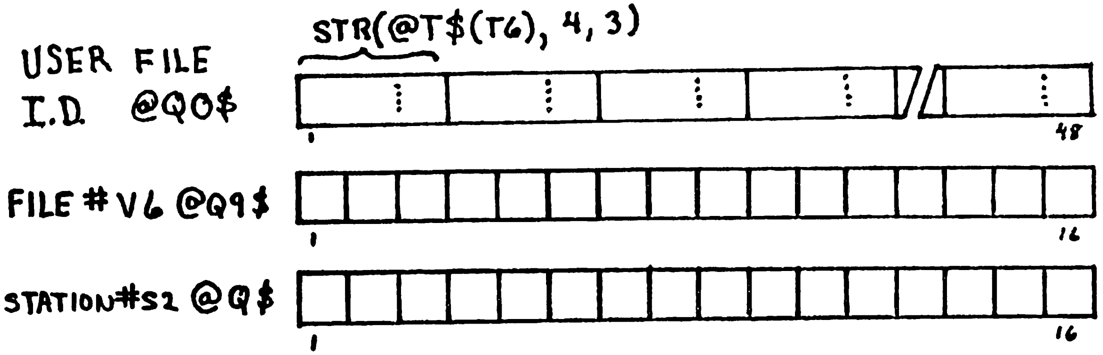
VO$(3)21 - TABLE OF ACTIVE FILE INFORMATION



GLOBAL TABLE OF OPEN FILES: ①

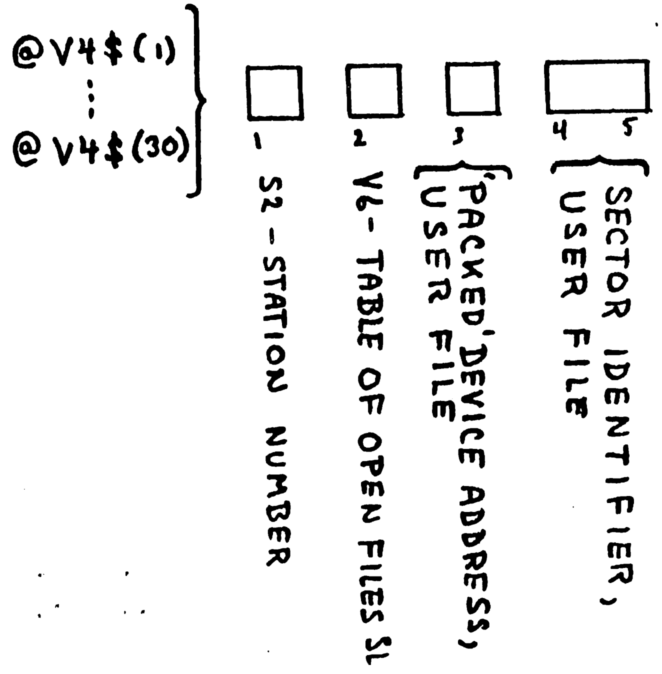


③ QUEUE (GLOBAL):



@Q = NEXT AVAILABLE POSITION IN QUEUE

② GLOBAL TABLE OF PROTECTED SECTORS:



④ PROGRAM HOG (GLOBAL):

@T = 0 - NOT HOGGED

@T = S2 - HOGGED BY STATION S2

REVIEW KFAM SUBROUTINES

SUBROUTINES FOR FILE ACCESS

'230 OPEN A KFAM FILE

'239 CLOSE A KFAM FILE

'213 RE-OPEN A KFAM FILE

ISS MULTIPLEXED FILE SUBROUTINES (FOR SEQUENTIAL FILES)

'217 OPEN A MULTIPLEXED FILE

'219 CLOSE A MULTIPLEXED FILE

'218 MULTIPLEX 'END' - SIMULATES A 'DATASAVE DC END' WITHOUT DESTROYING THE ACCESS INFORMATION

SUBROUTINES FOR KEY FILE UPDATE

'233 FINDNEW - INSERT KEY FOR NEW RECORD, RETURN WITH KFAM ASSIGNED USER FILE LOCATION

'234 FINDNEW (HERE) - GIVEN A RECORD'S LOCATION IN THE USER FILE, INSERT THE KEY INTO THE KEY FILE (USER ASSIGNS LOCATION POINTER)

'231 DELETE - GIVEN A KEY, REMOVE THE CORRESPONDING KEY INDEX ENTRY FROM THE KEY FILE

NOTE — FOLLOWING A 'DELETE', USER FILE SHOULD BE MARKED WITH A HEX(FF). ALSO, T4$ POINTS TO USER FILE LOCATION (RELATIVE SECTOR NO., AND RECORD NO. WITHIN SECTOR.)

SUBROUTINES FOR KEY FILE SEARCH

'232 FINDOLD - GIVEN A KEY, LOCATE THE RECORD

'235 FINDFIRST - FIND THE RECORD WITH THE LOWEST LOGICAL KEY

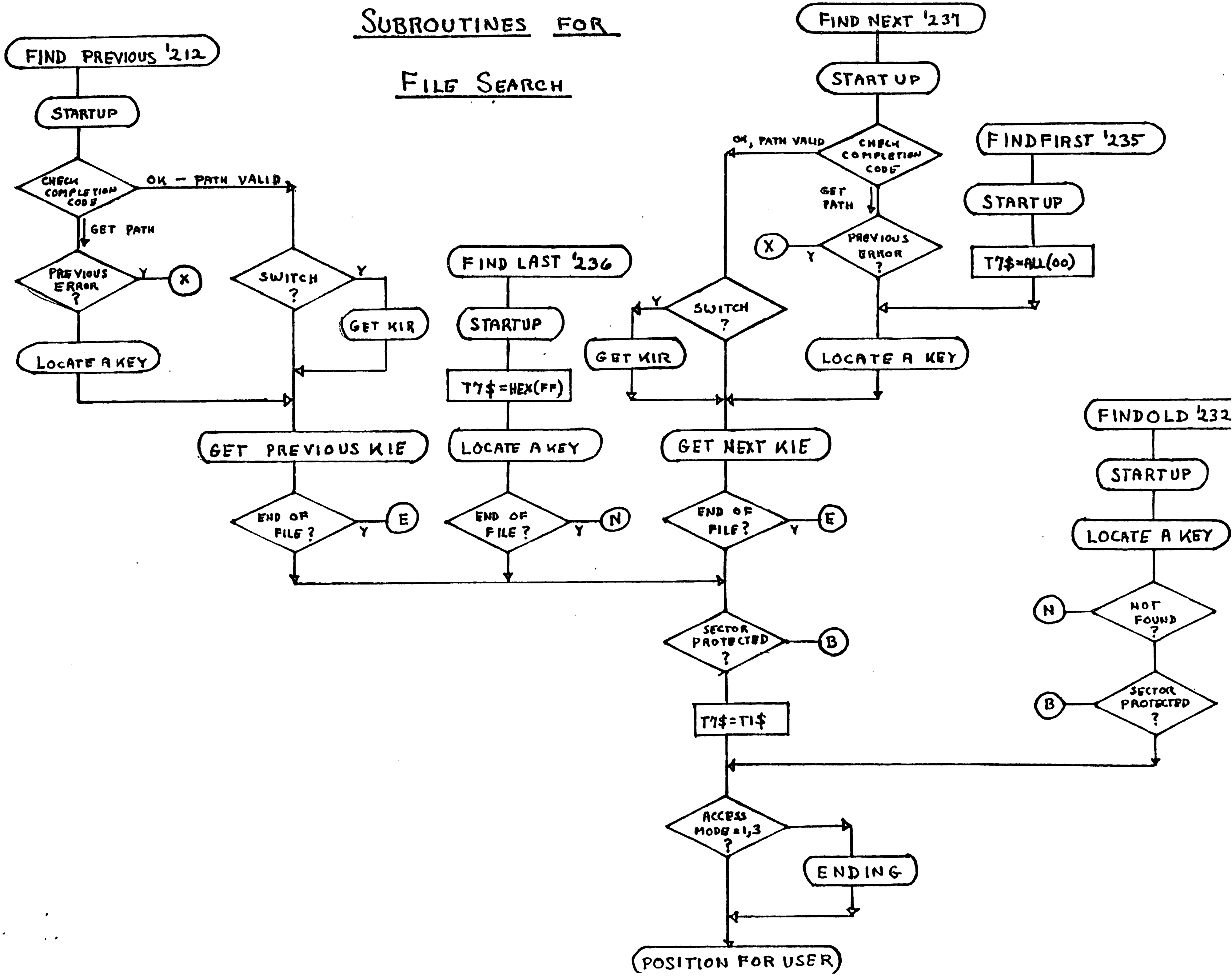
'237 FINDNEXT - LOCATE THE RECORD FOLLOWING THE PREVIOUSLY FOUND RECORD

'236 FINDLAST - FIND THE RECORD WITH THE HIGHEST LOGICAL KEY

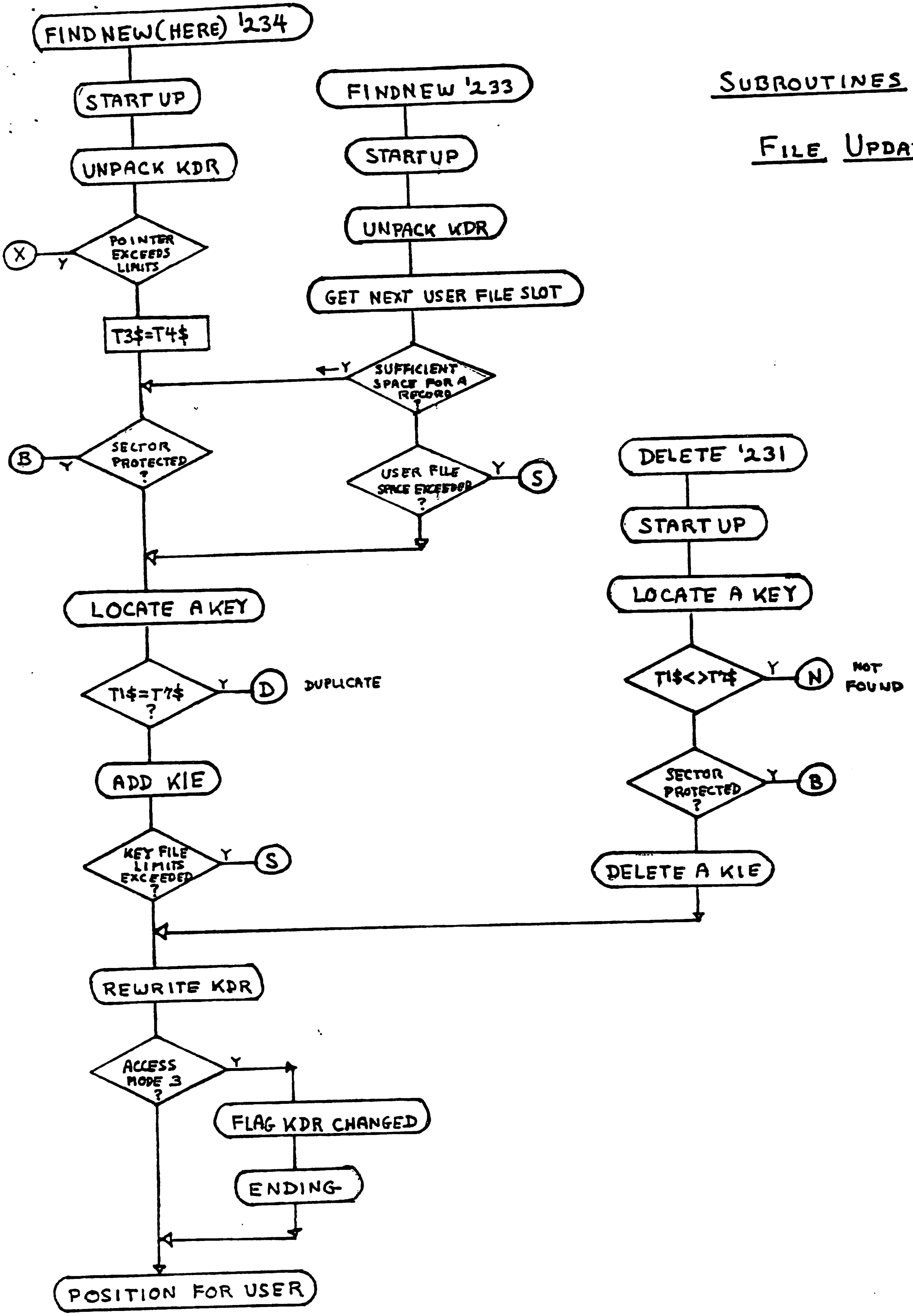
'212 FINDPREVIOUS - LOCATE THE RECORD PRECEDING THE PREVIOUSLY FOUND RECORD

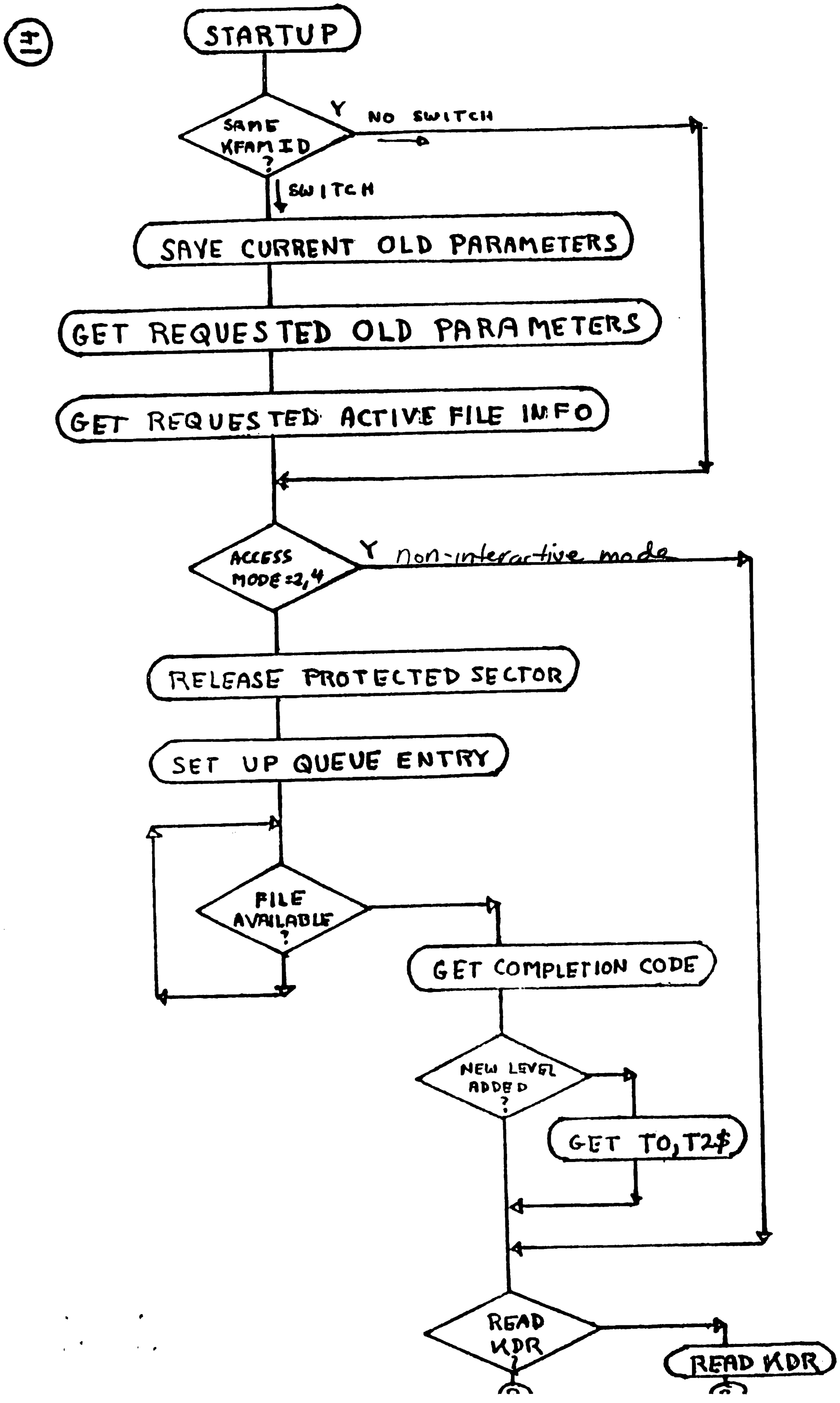
EXECUTION OF THESE SUBROUTINES ALWAYS PRODUCES A RETURN CODE AND OTHER VARIABLES THAT ARE AVAILABLE TO THE USER.

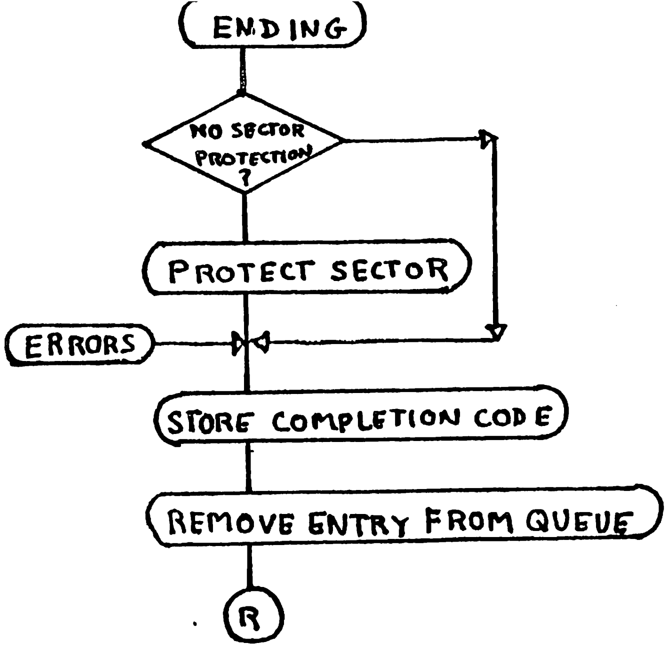
SUBROUTINES FOR FILE SEARCH



Subroutines for File Update




STARTUP
SAME KFAMID ?
Y
NO SWITCH
SWITCH
SAVE CURRENT OLD PARAMETERS
GET REQUESTED OLD PARAMETERS
GET REQUESTED ACTIVE FILE INFO
ACCESS MODE=2,4
Y
non-interactive mode
RELEASE PROTECTED SECTOR
SET UP QUEUE ENTRY
FILE AVAILABLE ?
GET COMPLETION CODE
NEW LEVEL ADDED ?
GET T0,T2$
READ KDR ?
READ KDR


ENDING
NO SECTOR PROTECTION ?
PROTECT SECTOR
ERRORS
STORE COMPLETION CODE
REMOVE ENTRY FROM QUEUE
R

KFAM-7 UTILITIES - 5.0

0 INITIALIZE KFAM FILES

1 BUILD KEY FILE

2 REORGANIZE IN PLACE

3 REALLO CATE FILE SPACE
DON'T NEED ON ISS 5.1

4 CONVERT TO KFAM-7
KFAM 3
KFAM 4
KFAM 5

5 PRINT KEY FILE

6 RESET ACCESS TABLES

7 BUILD SUBROUTINE MODULE
USUALLY 'GLOBAL' ON MVP

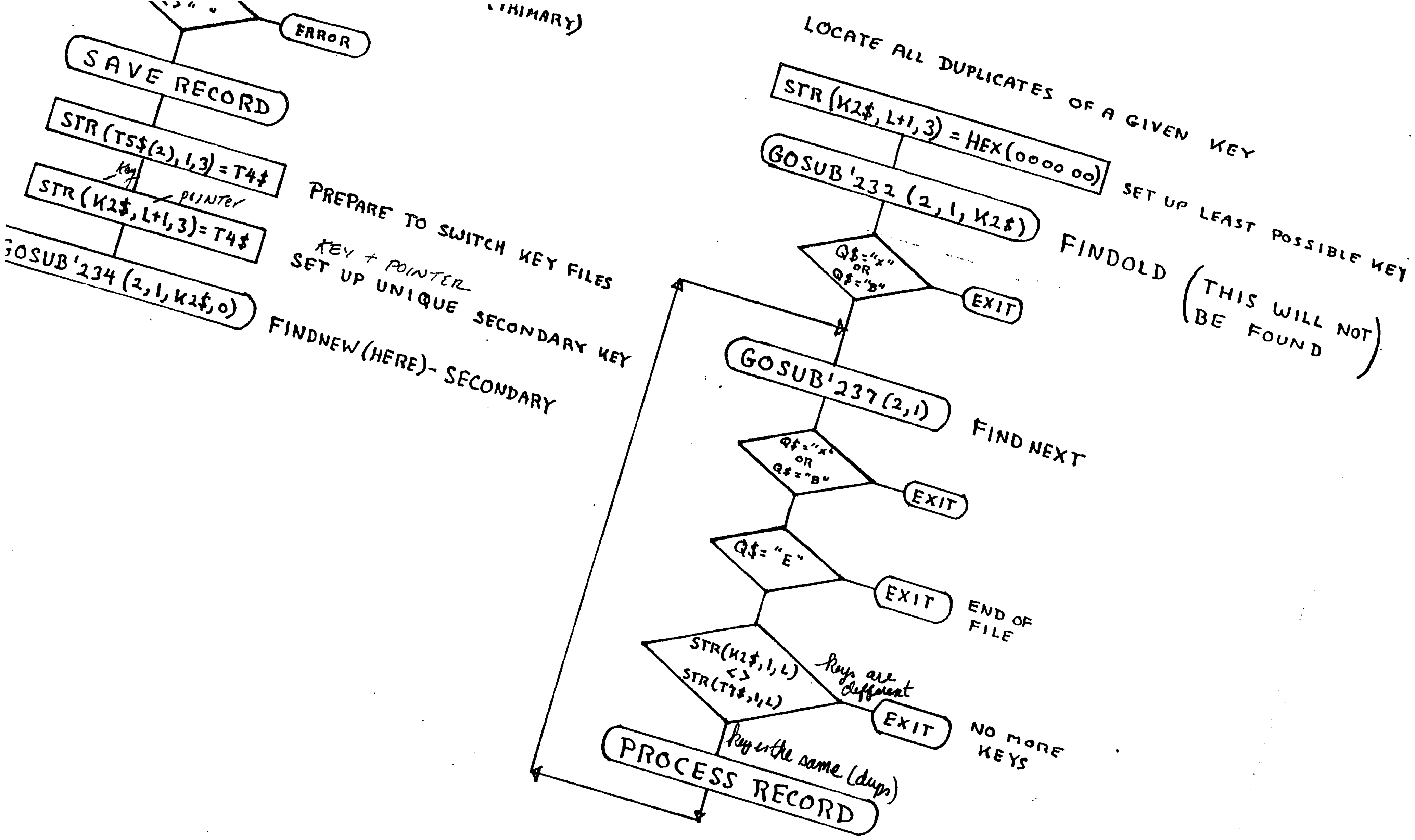
8 KEY FILE RECOVERY

REORGANIZE/REBUILD SUBSYSTEM

KFAM-7 UTILITIES (ISS 5.0)

Y = YES/REQUIRED
N = NO/ NOT REQUIRED
OR = OPTIONAL
NA = NOT APPLICABLE

	ACTIVE CATALOG STATUS					KEY FILE STATUS			RECOVERY INFORMATION					
	USER FILE	KEY FILE	KFAM WORK FILE	OUTPUT USER FILE	OUTPUT KEY FILE	FRESHLY INITIALIZED	INITIALIZED PREVIOUSLY	VALID KEY FILE	VALID KEY FILE PARAMETERS	VALID FIND NEW SECTORS	WRITES RECOVERY INFORMATION	USED FOR PRIMARY KEY FILE	USED FOR SECONDARY KEY FILE	SUPPORTS DUPLICATE KEY CONVENTION
INITIALIZE	OP.	OP.	N	NA	NA	N	N	N	N	N	OP.	Y	Y	Y
INITIALIZE('O)	OR	Y	N	NA	NA	N	Y	N'	N	N	OP.	Y	Y	Y
BUILD KEY FILE	Y	Y	N	NA	NA	Y	N	N	N	OP.	OP.	Y	Y	Y
REORGANIZE IN PLACE	Y	Y	Y	NA	NA	N	Y	Y	N	N	Y	Y	N	N
KEY FILE RECOVERY	Y	OR	N	NA	NA	N	N	N	Y	Y	Y	Y	N	Y
REORGANIZE (SUBSYSTEM)	Y	Y	N	OR	OR	N	Y	Y	N	N	OR	Y	N	Y
REBUILD (SUBSYSTEM)	Y	Y	N	NA	NA	N	Y	N	N	Y	OR	Y	Y	Y
REALLOCATE FILE SPACE	Y	Y	N	NA	NA	N	Y	N	N	N	OP.	Y	Y	NA
CONVERT TO KFAM-7	Y	Y	N	NA	NA	N	Y	N	NA	NA	Y	Y	Y	NA
PRINT KEY FILE	Y	Y	N	NA	NA	N	Y	N	NA	NA	N	Y	Y	Y
RESET ACCESS TABLES	Y	Y	N	NA	NA	N	Y	N	NA	NA	N	Y	Y	NA
BUILD SUBROUTINE MOD.	NA													


RIMARY)
ERROR
SAVE RECORD
STR(T5$(2),1,3)=T4$
PREPARE TO SWITCH KEY FILES
key
pointer
STR(K2$,L+1,3)=T4$
KEY + POINTER
SET UP UNIQUE SECONDARY KEY
GOSUB'234(2,1,K2$,0)
FINDNEW(HERE)- SECONDARY
LOCATE ALL DUPLICATES OF A GIVEN KEY
STR(K2$,L+1,3)=HEX(0000 00)
SET UP LEAST POSSIBLE KEY
GOSUB'232(2,1,K2$)
FINDOLD (THIS WILL NOT BE FOUND)
Q$="X" OR Q$="B"
EXIT
GOSUB'237(2,1)
FIND NEXT
Q$="X" OR Q$="B"
EXIT
Q$="E"
EXIT
END OF FILE
STR(K2$,1,L) <> STR(T7$,1,L)
Keys are different
EXIT
NO MORE KEYS
Keys the same (dups)
PROCESS RECORD

MULTIPLE KEY FILES / DUPLICATE KEYS

USER FILE

SECTOR	RECORD 1	RECORD 2
0	10000 AAA XXX	
1		
2		10002 AAA XXY
3		
4		
5	10001 AAB XXX	
6		

PRIMARY KEY FILE

KEY	POINTER
1 0 0 0 0	0 1
1 0 0 0 1	5 1
1 0 0 0 2	2 2

sector #
record #

SECONDARY KEY FILE (1)

KEY	POINTER
A A A 0 1	0 1
A A A 2 2	2 2
A A B 5 1	5 1

SECONDARY KEY FILE (2)

KEY	POINTER
X X X 0 1	0 1
X X X 5 1	5 1
X X Y 2 2	2 2

MAKE RECORD UNIQUE BY ADDING POINTER TO KEY TO MAKE A NEW KEY.

ISS UTILITIES - 5.1

0 COPY/VERIFY

1 CREATE REFERENCE FILE

2 LIST/CROSS-REFERENCE

3 COMPRESSION

4 DECOMPRESSION

5 SORT DISK CATALOG

6 DISK DUMP

7 FILE STATUS REPORT

8 PROGRAM COMPARE

9 RECONSTRUCT DISK INDEX

10 ALTER DISK INDEX

SCREEN/DISK SUBROUTINES
ISS 5.1

A. Screen Routines:

1. Position Cursor
2. Print Routine
3. Operator Wait
4. Date Routines
5. Data Entry

B. Disk Routines

1. Search Disk Index
2. Limits Next
3. Allocate Data File Space
4. Free Unused Sectors
5. Open/Close Output
6. Open/Close Input
7. MUX Open/End/Close
8. Select/Validate Addresses

C. Translation Tables

1. EBCDIC To ASCII
2. ASCII To EBCDIC

WANG LABORATORIES, INC.

MEMORANDUM

TO: ✓Jim McEvoy

FROM: Gerry Claggett

DATE: November 9, 1979

SUBJ: Modifications Incorporated in ISS 5.1

The following changes have been incorporated in ISS 5.1 (a maintenance release of ISS 5.0).

1. KFAM SUBROUTINES

A significant fault of the queue has been corrected (KFAM0107, KFAM0307) where in some instances a second station was allowed to begin to modify a key file concurrently with an initial station. The intention of the queue is to maintain file integrity by restricting access to a file to one KFAM call by one station at a time (interactive mode only).

The queue has been modified (KFAM0107, KFAM0307) so that it now refers to user files rather than key files. This is important only where multiple key files exist for a given user file. It insures that only one station will be modifying the user file at a time, eliminating the possibility of access by separate key files at the same time (interactive mode only).

The upper bounds for both the user and key files are automatically updated in the KDR by KFAM OPEN (DEFFN '230 - KFAM0107, KFAM0207, KFAM0307, KFAM230S). This is done based on the actual limits of the files at the time OPEN is executed, and takes into account record lengths for multiple sector records. This modification eliminates the required use of the KFAM utility REALLOCATE KFAM FILE SPACE following the change of key and user file sizes.

A minor modification has been made in RE-OPEN (DEFFN '213 - KFAM0107, KFAM0307) to hog the system while completion codes are updated. This was previously ommitted, allowing the remote possibility of two stations updating the completion codes concurrently when RE-OPEN was executed giving spurious results.

2. KFAM UTILITIES

INITIALIZE:

The option for duplicate and standard key types has been included with the default values. As before, it is only significant when checks are made on the key lengths. (KFAM107U, KFAM117U).

The option to write or not write recovery information has been included with the default values, Typically the recovery information is written when initializing the primary key and not written when initializing any secondary key files. (KFAM107U, KFAM117U).

The order of displayed default parameters has been adjusted to include these two additional items. They have also been included in the hardcopy printout of the parameters. (KFAM107U, KFAM117U).

Modification has been made to insure that all space in the key file is made available. Previously, the number of records requested was always used to determine the upper bound of the key file. Thus, when reinitializing using an exisiting key file, it was possible to have an erroneous upper bound established. (KFAM107U, KFAM117U).

BUILD KEY FILE:

The option to write and not use, or use and not write, the recovery information has been included in this utility to support building multiple key files. Typically WRITE/DON'T USE recovery information would be specified when building a primary key file, and USE/DON'T WRITE would be specified when building a secondary key file. Only the FINDNEW SECTORS portion of the recovery information is used. (KFAM207U, KFAM217U).

REORGANIZE/REBUILD SUBSYSTEM:

Modification has been made to correctly reactivate a scratched output key file if such a file has been specified. (KFAM3507).

Modification has been made so that the variable C1 is no longer erroneously left as a common variable at the conclusion of the subsystem execution. (KFAM3507).

PRINT KEY FILE:

Modification has been made to present a full screen of KIE's where possible. (KFAM617U).

RESET ACCESS TABLES:

This utility has been modified to support the addition of the variable QO$ to the queue. (KFAM717U).

3. ISS SUBROUTINES

FREE UNUSED SECTORS (DEFFN '227):
Modification has been made to this routine so that the high order bit for "next available sector in catalogued area" is no longer removed from the catalogue index. This was done previously to support disk platters formatted on the 2200T. Up to the 2280 (Phoenix) disk drives, the high order bit was not used by the VP and MVP. Now it is necessary to leave it intact for addresses up to 52,609 requiring the full eight bits. (ISS.227S).

DATE ROUTINES - CONVERT GREGORIAN TO JULIAN (DEFFN '221):
More efficient code has been incorporated in this routine. (ISS.220S).

4. ISS UTILITIES

COPY VERIFY:
Modification has been made to allow up to 65,534 extra sectors to be specified. Previously only 255 extra sectors were allowed. (ISS.000U, ISS.001U).

Modification has been made so that error messages encountered during this utility go to the specified output device. Prevously all error messages came to the CRT. (ISS.000U, ISS.001U).

Modification has been made so that when indirect mode is being used and the reference file specified includes a reference to itself, the output reference file will not be copied with its MUX trailer left open by the copying station. (ISS.001U).

CREATE REFERENCE FILE:
Deffn '15 has been disabled until appropriate in the input module, thus preventing unintended screen display. (ISS.010U).

CROSS-REFERENCE:
Modification has been made so that form feeds are no longer erroneously issued when "REM%↑" is encountered during the building of the cross-reference table. (ISS.022U).

LIST:

Modification has been made to correctly use REM%↑ in giving expanded print and form feeds. Previously an extraneous underline appeared. Also, it is no longer necessary to leave a blank space following REM% if "↑" is not used. (ISS.024U).

Modification has been made to process the statement "LIST' " if found in the text. Previously the program crashed on this statement. (ISS.024U).

Modification has been made to begin new lines as appropriate for text in trailing REM's. Previously the entire trailing REM was printed on one line. (ISS.024U).

CROSS REFERENCE:

Modification has been made to give proper form feeds and heading for marked subroutine cross reference listings. Previously the line count logic was faulty. (ISS.022U).

DECOMPRESSION:

Modification has been made to process the statement "LIST' " if found in the text. Previously the program crashed on this statement. (ISS.044U).

ALL UTILITY INPUT MODULES:

Modification has been made to make the use of PART mode easier. Subsequent to a return to "enter parameters" ('15), old entries are retained and correctly checked, including output name and extra sectors where specified.

It is now possible to specify up to 16 parameters in the input modules. Previously only 14 were allowed. (ISS.050S).

EXECUTION MODULES:

The statement "SELECT PRINT #4< S$(1)> " is no longer executed when the printer address (S$(1)) is equal to " ". Previously a crash could occur. (ISS.001U, ISS.031U, ISS.041U, ISS051U, ISS.061U, ISS.071U, ISS.081U, KFAM117U, KFAM617U).

5. SORT4

SORT400C:
Modification has been made to the hog device routine (DEFFN '215) to use $OPEN and $CLOSE instead of a $G10 statement. The $G10 statement did not work on the MVP or the Phoenix disk drive.

SORT490A:
Modification has been made to this SORT4 exit module so that the first executable line selects print to the CRT screen. Thus it is possible now to run SORT4 in background by selecting print "000" (output dump) and executing "$RELEASE TERMINAL" in the start up module.

6. LIMITATIONS:
Underlined characters in literals REM's and image statements are not supported in some ISS Utility programs (LIST, CROSS-REFERENCE, COMPRESSION, DEPRESSION, PROGRAM COMPARE).

cc: Pradeep Barthakur
Fred Fredericks

GC/nc
1220C

LONGER KEY → MORE READS (HIGHER TREE)
SHORTER KEY → LESS READS (LOWER TREE)

VACANT SPACE OR SPACE FROM DELETED RECORDS SHOULD BE MARKED WITH A HEX(FF) IN USER FILE TO ENABLE RECOVERY LATER. BY THE USER → OTHERWISE ONLY DONE WHEN BUILD UTILITY IS RUN.

USER FILE
* UPDATED AT FILE CLOSE
* UPDATED WHEN WRITE RECOVERY INFO SUBROUTINE IS RUN

AT FILE OPEN
* MAKES ENTRY IN "TABLE OF OPEN FILES" (GLOBAL)

TABLE OF OPEN FILES

1. INVENTORY FILE
2. CUSTOMER FILE
3. MASTER FILE

QUEUE TELLS WHICH STATION IS CALLING WHICH FILE IN TABLE IN FIRST-COME FIRST SERVE BASIS

~~ALSO CONTAINS TABLE OF PROTECTED SECTORS~~

GBS → HUNG UP ? COULD BE IN QUEUE OR IN HOG

KFAM

6/19/80

HISTORY:

KFAM 3 T/VP/MVP - CANNOT MUX DISK TO CPU'S

KFAM 5 T/VP/MVP - HAS COMMON CONTROL ACCESS TABLES TO ALLOW 1 USER TO ACCESS A PARTICULAR RECORD AT A GIVEN TIME. (SECTOR CONTROL)

KFAM 7 VP/MVP - SINGLE/MULTIBANK VERSIONS - CONTROLS ACCESS TO RECORDS IN A GIVEN SECTOR WITHIN A FILE.

MEMORY USE:

KFAMØ1Ø7 - KFAM7 TEXT & CONTROL TABLES FOR ALL PARTITIONS WHEN STORED IN A PARTITION WITHIN A BANK (GLOBAL)

CAN SELECT A "UNIVERSAL GLOBAL" TO STORE KFAMØ4Ø7 CONTROL TABLES → KFAMØ3Ø7 WILL STORE EACH TEXT IN EACH PARTITION.

KFAMØ3Ø7 - TEXT ONLY (≈ 10K) (STORED IN EACH BANK)
KFAMØ4Ø7 - ACCESS CONTROL TABLES (STORED ONCE IN 1 PARTITION IN 1 BANK).
KFAMØ2Ø7 - MAINTAINS ACCESS CONTROL TABLES ON KFAM7 MUX VERSION

KFAM - PUTS KEY IN INDEX & POINTS TO POSITION IN SECTOR TO WRITE IT; HOWEVER, USER MUST WRITE IT HIMSELF.

KFAM - SORTS KEYS IN KEY INDEX AND FORCES A 'KIR' SPLIT IF NECESSARY & PUTS HIGHER KEYS IN NEXT SECTOR.

HLKIR - 'HIGH LEVEL' KIR CONTAINS POINTER TO 1ST KEY IN EACH PRECEDING SECTOR & PLACES KEY IN APPROPRIATE SECTOR.

LEVELS OF KIE SUPPORTED

LEVEL	MAX KIE
1	$5^1 = 5$
2	$5^2 = 25$
3	$5^3 = 625$
↓	
8	5^8

IDEAS

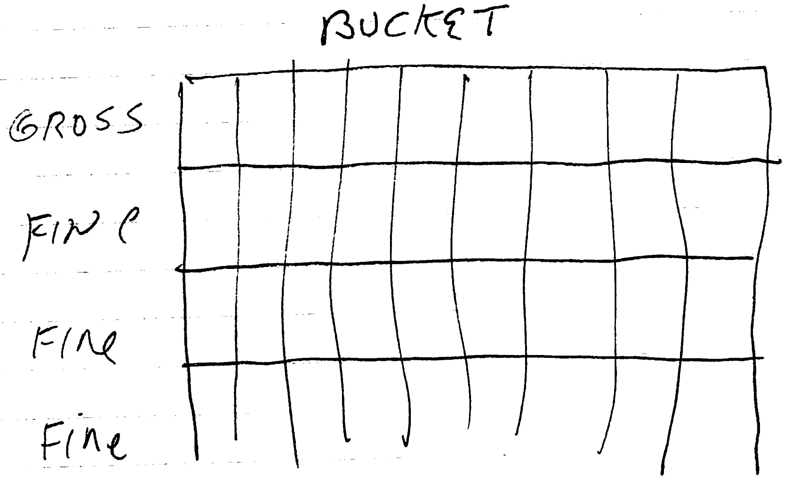
A) INQUIRY DATA ENTRY ACCESS SYSTEM
B) 1232 SECTORS (ALL ON 0-1023) 2 DISKETTES

HARDWARE

A) T, VP, MVP 32K
B) 80 x 24 screen
C) DEV 30K PART
D) RUN 17.5 GLOBAL (TERM RUN WITH LESS)

SOFTWARE

A) HIKAM
B) MAX REC LNGTH 1008 BYTES
C) MAX 128 FIELDS
D. COMPRESSION
  1. NUM 2:1
  2. A/N 4:3
E. RECORD PROTECTION AT RECORD LEVEL
F. KEY - MAX 58 BYTES (MAX 3 FIELDS, CONTIGOUS NOT REC.
G. MULTIVOL FILES
H. EXPAND FILE → READ SEQUENTIALLY + RECREATE



Key is hashed in algorithm to find home bucket
Key is inserted in FIS in sort order
GIS is updated with new key

If ~~key~~ home bucket is full, key overflows to next bucket

"RELEASE" — Releases sector from protection when changes (update) is complete.

GOSUB '234 - enables user to take advantage of unused space from deletes, etc.

GOSUB '233 - automatically places record pointer at next available location; regardless of open areas.

SECONDARY KEYFILES

MULTIPLE FILES — REORGANIZE PRIMARY FILES
— REBUILD SECONDARY FILES

KEY CAN BE 1-30 BYTES &

WRITING KFAM 7 PROGRAMS (APPLICATIONS)

REQUIREMENTS:

① KFAMØØØ7 (IN CORE → COMMON VARIABLES)

② S2 = STATION NUMBER = #PART OR # TERM

MORE > 1 CPU USE S2 = #PART + (X)

S2 = #PART
S2 = #TERM

③ SELECT @PART "KFAM"

④ GOSUB '230 (1, 1, 2, 1, F$, 3, " ", D1$, D2$)

1	1	2	1	F$	3	" "	D1$	D2$
KFAM ID	KEYFILE DEVICE#	USERFILE DEVICE#	KEYFILE #	USERFILE NAME	ACCESS CODE	PASS WORD	KEYFILE DEVICE ADDRESS	User FILE DEVICE ADDr.

OPENS FILE

⑤ IF Q$ <> " " THEN STOP "Error ON OPEN"

⑥ GOSUB '239 (1) CLOSES FILE

KFAM ID

ASSIGNMENTS

1. DESTROY — RECOVER KEY FILE SF'5,'8
2. DESTROY - INITIALIZE & BUILD KEYFILE SF'∅,'1
3. REORGANIZE IN PLACE SF'2
4. REORGANIZE SUBSYSTEM
5. BUILD SECONDARY KEYFILE

REORGANIZE/REBUILD SUBSYSTEM → USER WRITTEN → NOT ON MENU

MEMO

TO: George Levine
Wang Labs
110 East 59th St.
33rd Floor
New York City, NY 10022

FROM: Wendy MacGown
DATE: January 20, 1981

SUBJECT: Differences between KFAM-5 and KFAM-7

Citibank has expressed a desire to know why KFAM-7 is superior to KFAM-5, as often stated. I hope this memo will provide sufficient information.

The primary reason why KFAM-7 was originally introduced was to take full advantage of the MVP multi-user operating system. With the multi-user operating system, KFAM-7 subroutines are run in a background partition while the user's program is run in a foreground partition. The processing is faster and more efficient.

The subroutine programs KFAM0107 (single bank) and KFAM0307 (multiple bank) both access tables in core rather than on disk. KFAM-5 accesses tables on disk and was designed to run in one contiguous partition of memory. The KFAM-7 program, KFAM0207 (multiplexed) was designed to be used on the VP operating system and is basically the same as KFAM-5. An upgrade to KFAM-7 is not necessary if the user has a VP operating system and is running KFAM-5.

Subtle Differences:

KFAM-7 uses BASIC II, whereas KFAM-5 uses the earlier Wang BASIC.

Multiple key access to the user file does not work with KFAM-5 or with KFAM-7's KFAM0207. The protect table is contained in the key file itself, so when the user file is being accessed by one of the key files, the other key files don't know that the protect bit has been set. In KFAM-7 the protect table is in the global background partition, and can be accessed by all the key files.

In KFAM-7 the Reorganize/Rebuild Subsystem utilities will reorganize the user file based on one key file. If there are multiple key files, the secondary key files will not be mapped to the new user file. KFAM-7 however, allows for multiple key files. The Reorganize/Rebuild Subsystem set-up module can be written to map the secondary key files to the new user file.

CITIBANK

Citibank N.A.
399 Park Avenue
New York, N.Y.
10043

October 26, 1981

Mr. Al Goldman
WANG LABORATORIES
Mail Stop 1213
Lowel, Mass. 01851

Dear Mr. Goldman:

As I told you on our telephone conversation the other day, I am having KFAM-5 problems. You informed me that these errors could be caused by the fact that KFAM-5 was running on a Phoenix 80 meg disk drive. Also you suggested that we should change to KFAM-7. I had spent some time last year in Massuchuttes with Mr. Frank Sullivan trying convert from KFAM-5 to KFAM-7 but we were unable to do so. As you suggested, I am sending you the system so you can do the conversion. Enclosed you will find:

- program disk
- good database disk
- file definition
- system usage and KFAM changes

We would appreciate it if you would let me know when you receive the disks. There is very important data on these disks that we would like to keep track of. We will also be grateful if this data is kept confidential.

I will be looking forward to hearing good news from you soon and receiving a KRAM-7 system. My telephone no. is (212) 559-6282.

Sincerely yours

Caridad Ferro

CF/mb
Enclosures

TO: Elda Dilorenzo
FROM: Chuck Wilson
DATE: 07/13/83
SUBJECT: ISS Release 5.3 (new hash capability)

Enclosed is release 5.3 of ISS which contains the new 2200 hash capability. Along with this release there are reports showing program discrepancies between ISS 5.2 and ISS 5.3. I have noted what each error on the program compare is so we could concentrate our testing on those specific areas of the package. I have also enclosed catalog listings to verify those files that have changed in size and to assure that the correct number of data files and program files are placed on the diskettes as they go out.

The only file to change extensively is ISS.229S which is the "search index" routine. This routine allows a user to use MVP basic release 2.5 with the new catalog hashing algorithm which is not supported in earlier versions. There is a significant change in the size of this file, the original version of it in release 5.2 was approx. 1.5k bytes and it is now approx. 2.4k. It is approx. .84k larger now. Users may need to know this.

Other modules that have changed since release 5.2 are as follows:

ISS.031U -- Change was made so that HEX(80) as part of a valid line number would be masked as it resembles the keyword 'LIST' during compression. Without this change a program going through compression which included any line from 8000 to 8099 may stop prematurly with an unrecoverable error.

ISS.081U -- Change made to line 1710 for the same reason noted for ISS.031U update.

ISS.217S -- Change made to line 3880, "DEFFN'254" found on that line in the 5.2 version should have been "GOSUB'254". Without this change the List/Cross reference utility with the List option and all options specified would crash on line 8310 with a P48 because slot 3 of the device table has no valid disk address assigned.

KFAM3707 -- Change made to line number 4870 to initialize sector buffer in reorganize rebuild subroutine. Without change reorganize/rebuild would not functuon properly with certain KFAM files.

The file "KFAMREFS" has also been edited so that it may be referenced properly. The difference between this version and ISS 5.2 has to do with the last absolute sector of the file. This sector has been reconstructed for the new release and seems to be fine.

If there are any questions or problems please give me a call at ext 6565.

WANG LABORATORIES, INC.
ONE INDUSTRIAL AVENUE, LOWELL, MA 01851 • TEL: 617/459-5000, TWX 710-343-6769, TELEX 94-7421

MEMORANDUM

TO: Brian Weir

CC: Renee Plummer
Jim Mahoney
Duane Frunz

FROM: Ken Mailloux

SUBJECT: Probe F007149 - Wollaston Alloys

DATE: Nov. 16, 1984

I would like to emphasize the importance of unique station numbers when running multiplexed KFAM. The application that we looked at included a configuration named WAI-07. Loading this configuration into both CPUs is a potential problem. The application equates the KFAM variable S2 (station number) to #PART (partition number). Look at the screen dump of WAI-07 and notice the programs that are loaded and the partition numbers. The program "HELLO" that initializes the system and then loads "WAI" is in partition 2. The program "WAI" is loaded into partitions 6,9 and 12. If both CPUs are brought up under this arrangement then we have conflicts in partitions 3,6,9 and 12. The partition numbers are the same for both CPUs. As an example, CPU #1 part 3 has S2 =3 and so does CPU #2 part 3. The application will work sometimes until conflicting station numbers try to access the same file. The first one in will effectively hog the file and an access conflict error message will be signaled.

A better way would be to load separate configurations into each CPU as in the two examples. The user would not have to run partition status and perhaps release to a different partition to insure unique station numbers. At the user site you can check for conflicts by running @PSTAT on each CPU or print the value of S2 on each terminal when the main system menu is loaded.

Another potential problem is hitting the RESET key instead of Fn '31 and leaving the files open. A hardware problem that forces an exit without closing files would require resetting the access tables. The KFAM utility RESET ACCESS TABLES does not permit the ALL function and the user would have to know which files are accessed by a given module. This could be frustrating to a user not familiar with the details of the programs. Make sure they exit the modules the proper way.

As you have seen, an application that is designed to run on a Multi-Bank system must be modified to work in a Multiplexed environment. The ISS user manual explains most of the differences.

Good luck, and please call if you need additional information.

enclosures

0057Y

```
              *** WANG 2200MVP PARTITION GENERATION PROGRAM ***
Available memory: 61.00   56.00   56.00   56.00   56.00   56.00   56.00   56.00
Remaining memory:  0.00    0.00    0.00    0.00   56.00   56.00   56.00   56.00
                                                        LIST OF OPTIONS:
PARTITION SIZE(K) TERMINAL PROGRAMMABLE PROGRAM        SF'00 - clear partitions
    1       5.00      0         N                      SF'01 - clear device table
    2      10.00      0         N        KFAM0207
    3      23.00      1         Y        HELLO         SF'02 - divide mem. evenly
    4      23.00      2         Y
    5      10.00      0         N        KFAM0207      SF'04 - edit partitions
    6      23.00      3         Y        WAI           SF'05 - edit device table
    7      23.00      4         Y                      SF'06 - edit $MSG
    8      10.00      0         N        KFAM0207
    9      23.00      5         Y        WAI           SF'08 - load configuration
   10      23.00      6         Y                      SF'09 - save configuration
   11      10.00      0         N        KFAM0207      SF'10 - delete config.
   12      23.00      7         Y        WAI
   13      23.00      8         Y                      SF'15 - execute
```

WAI-07

```
Check configuration.  OK to execute (Y or N)?

:
```

```
              *** WANG 2200MVP PARTITION GENERATION PROGRAM ***
Available memory: 61.00   56.00   56.00   56.00   56.00   56.00   56.00   56.00
Remaining memory:  0.00    0.00    0.00    0.00   56.00   56.00   56.00   56.00
                                                        LIST OF OPTIONS:
PARTITION SIZE(K) TERMINAL PROGRAMMABLE PROGRAM        SF'00 - clear partitions
    1       5.00      0         N                      SF'01 - clear device tabl
    2      10.00      0         N        KFAM0207
    3      23.00      2         Y                      SF'02 - divide mem. evenl
    4      23.00      1         Y        HELLO
    5      10.00      0         N        KFAM0207      SF'04 - edit partitions
    6      23.00      4         Y                      SF'05 - edit device table
    7      23.00      3         Y        WAI           SF'06 - edit $MSG
    8      10.00      0         N        KFAM0207
    9      23.00      6         Y                      SF'08 - load configuratio
   10      23.00      5         Y        WAI           SF'09 - save configuratio
   11      10.00      0         N        KFAM0207      SF'10 - delete config.
   12      23.00      8         Y
   13      23.00      7         Y        WAI           SF'15 - execute

 Configuration 'CPU-2' loaded.  Edit which partition (default = 1 )?
```

```
              *** WANG 2200MVP PARTITION GENERATION PROGRAM ***
Available memory: 61.00   56.00   56.00   56.00   56.00   56.00   56.00   56.00
Remaining memory:  0.00    0.00    0.00    0.00   56.00   56.00   56.00   56.00
                                                        LIST OF OPTIONS:
PARTITION SIZE(K) TERMINAL PROGRAMMABLE PROGRAM        SF'00 - clear partitions
    1       5.00      0         N                      SF'01 - clear device tabl
    2      10.00      0         N        KFAM0207
    3      23.00      1         Y        HELLO         SF'02 - divide mem. evenl
    4      23.00      2         Y
    5      10.00      0         N        KFAM0207      SF'04 - edit partitions
    6      23.00      3         Y        WAI           SF'05 - edit device table
    7      23.00      4         Y                      SF'06 - edit $MSG
    8      10.00      0         N        KFAM0207
    9      23.00      5         Y        WAI           SF'08 - load configuratio
   10      23.00      6         Y                      SF'09 - save configuratio
   11      10.00      0         N        KFAM0207      SF'10 - delete config.
   12      23.00      7         Y        WAI
   13      23.00      8         Y                      SF'15 - execute

 Configuration 'CPU-1' loaded.  Edit which partition (default = 1 )?
```

WANG LABORATORIES, INC.
ONE INDUSTRIAL AVENUE, LOWELL, MA 01851 • TEL: 617/459-5000, TWX 710-343-6769, TELEX 94-7421

Memorandum

Ken
FYI

To: Max Mickiff
ATOM Latin America

From: Sheila D. Mitchell
Section Manager, VS Value Added/2200 Support

Subject: Technical report for TAC L5205000 and PROBE F009254

Date: 12 August 1985

CC: Mary Bowker
Duane Frunz
Henry Schinnagel

The attached technical report for TAC L5205000 and PROBE F009254 has been prepared by Duane Frunz to assist Sisteco with resolving the reported KFAM processing problem. The report is organized in the following manner:

1. History of problem since 7 Jan 85.
2. General description of Sisteco system.
3. Technical discussion of KFAM and 2200 CPU processing.
4. Seven (7) problems and recommendations for Sisteco system.
5. Appendices in support of problems and recommendations.

When the recommended system changes are made by Sisteco, we expect that Sisteco's problem will be resolved.

The duplication of the variable '@T' causing the system to hang is still an open question with the following issues:

1. Software (KFAM processing).
   - The software issue is that '@T' does not get cleared to zero. However, we have not been able to find rules for duplicating the problem using only the KFAM processing.
2. Hardware (cable installation).
   - The Hardware issue is that the cable can be installed with MUX to TERMINAL and TERMINAL to MUX. With cables reversed, the 2200 will continue to operate and only occasionally will cause the RESET signal to be sent to a different partition.
3. Operator training (using RESET).
   - The Operator issue is known to happen even after extensive operator training.

Also, enclosed are copies of the diskettes which we have received from Sisteco. If there are any additional questions or you should need more information, please call us.

Regards,

Sheila D. Mitchell

Sheila D. Mitchell

0059d:VS1002

History of Sisteco problem

KFAM based applications using the WANG 2200 have historically encountered performance and system hanging problems due to misunderstanding the procedures specified in the manuals. The three major problems with the manuals' descriptions are:

1. How shared (global) programs are used.
2. What is variable @T and how is it used.
3. Record (sector) protection.

The majority of reported problems have been resolved after the systems analyst implemented recommended changes based on understanding the manuals specifications. With a clarification of these principles, we believe that a majority of the problems with the Sisteco application will be corrected.

The problem with the KFAM system hanging due to the variable @T is an old problem that has eluded exact duplication rules. Also, the problem was always associated with an installation using four (4) or more terminals. So, the Sisteco system appeared to be an opportunity to obtain exact duplication rules with a two terminal system. Sisteco did prepare a special, stripped down, version of their system and supporting data. However, the following problems with the backups and instructions were encountered:

Backup dated 1/03/85:
1. Data file DF11F10 was received with a status of 'SHARED'
2. Data file ADDRESS1 showed that both terminals were using the program @DFP0010. Whereas the instructions stated that one terminal use program @DFP0010 and the other use @CCP0020.
3. Duplication not realized.

Duplicaton on 7/23/85:
1. Sisteco had problem at step 3. of instructions and cleared the variable '@T' without rebooting the system.
2. Sisteco encountered problem immediately upon using programs.
3. Data file ADDRESS1 showed that terminal two had been using the system.

Backup dated 7/23/85: Unable to duplicate per instructions.

The Technical Assistance Log dated 01/07/85 describes the problem as Sisteco understood what was causing the hang. However, their description contained a misunderstanding about the use of the variable '@T' but the problem with the program hanging at statements 6100 and 6110 while accessing file CCONTRA was well defined and consistent with the manual. The manual states that when an open is attempted and the status 'A' is received, then the program should attempt the open again. This is exactly what the program was doing but the cause for the file CCONTRA being unavailable had to be explained. An examination of the program @CCP0010 found numerous potential reasons for CCONTRA to be left open and hence unavailable to program @DFP0020.

The reasons for CCONTRA being left open by @CCP0010 is not a simple isolated problem. Therefore, all the potentially contributing problems have been identified and are being reported.

System overview of Sisteco processing

The system processes accounting data for many different types of transactions. The only portion of the system which we will discuss is the processing that affects two programs @DFP0010 and @CCP0020. Transactions added to the system by programs @DFP0010 (DISCHARGES) and @CCP0010 (INPUT/CHARGE) are processed basically in the same manner:

1. Operator selects program from menu.
2. System records program name and data in file CONTROL.
   2.a @CCP0020 opens KFAM files and is ready for step 3.
   2.b @DFP0010 opens KFAM files and displays program menu.
   2.c @DFP0010 accepts operator choice and is ready for step 3.
3. Operator enters customer number or zero (0) to finish.
4. Operator enters data values for transaction.
5. Using the operators' data, some values are computed, and the system performs the following processing:
   5.a Save one or more transactions in two KFAM data files.
      (@CCP0020 uses CC11F100 and CC11F110)
      (@DFP0010 uses DF11F100 and DF11F110)
   5.b Open file CONTROL, save values, and close CONTROL.
   5.c Open file CCONTRA, save values, and close CCONTRA.
   5.d Buffer processing:
      5.d.1 Increment buffer index
      5.d.2 If index more than six (6)
        THEN open special file, save values, and close file.
      5.d.3 Set current values in buffer.
       (@CCP0020 uses file ADDRESxx, with xx = partition number)
       (@DFP0010 uses ADDRESS1)
6. When step five (5) is completed, continue with step three (3)
7. When operator enters zero (0), return to menu
   7.a @CCP0020 closes KFAM files and returns to system menu.
   7.b @DFP0010 returns to program menu.
   7.c @DFP0010 closes KFAM files and returns to system menu
      after opertor enters zero (0) from the program menu.

This description is VERY over-simplified but in principle both programs repeat the steps three (3), four (4), and five (5) until the operator enters zero (0) for the customer number. With this understanding of the normal processing, we will discuss the different ways the system will appear to 'hang' because the program can not finish a step in the processing.

Before discussing the 'hanging' conditions caused by a program, we should note which terminals can use the programs and how the system responds to a data problem. The system allows any terminal to use a program and both terminals can be using the same program. However, the program @CCP0020 should be used only by terminal number one because when terminal two uses this program the data file ADDRES05 is used for step 5.d.3 and results in an error after the operator has entered the seventh transactions. When either program encounters a processing problem, a subroutine is used to close the KFAM files. This is an excellent design feature but problems can be created when an operator uses the subroutine by pressing function '31 on the keyboard and restarting the program after the subroutine is finished.

The Key File Access Method (KFAM) for the WANG 2200 is a collection of programs and utilities which can be used by an analyst for control and access of data. The access method used by KFAM is indexed files with two data files, one for the indices and one for the actual data. During the system design phase, the analyst must decide not only standard data structure issues but also how will the 2200 system be configured for the actual processing.

Configuring the 2200 system is a process in which a system administrator allocates the systems' user memory so that it will accomodate his particular programs (tasks). Depending on the system, the user may subdivide their total memory into anywhere from one (1) to a possible maximum of eight (8) BANKS. These BANKS are then subdivided into PARTITIONS which are specific allotments of user memory with the BANK. The memory assigned to a particular PARTITION is determined by the application program that will be performed in each PARTITION. A special type of PARTITION can be shared by other partitions when certain system conditions are satisfied.

The system condition for sharing a partition is that a program must execute the special instruction 'DEFFN @PART "..name.."'. When a program in another partition must use the shared partition, the program must execute a statement 'SELECT @PART "..name.."'. A second condition for defining a shared partition is to have the shared program require less than five kilobytes (5K) of memory and reside in the first partition. The literature refers to these different shared partition definitions as universal global and local global. That is, the small shared program in the first partition can be accessed by all (ie universal) partitions of the 2200 CPU whereas a larger shared program can be accessed only by the partitions (programs) within one (ie local) bank of memory.

KFAM uses special programs for the definition of universal and local global. The execution of the access method is performed by another program name KFAM0307 which must be in each bank of 2200 memory that will be using KFAM. Since this program is in each bank, the program KFAM0307 is referred to as the 'local global' program. Depending on how the analyst designs the system, each local memory bank may have any number of actual applications executing. When an application accesses data, KFAM must take care of updating the system values by using the local and universal global 'partitions'.

Sharing data between the different global memory areas is controlled within KFAM by using a flag. This flag says either 1. Yes, shared data is being accessed and changed by a program or 2. No, shared data is not being accessed and is available for another program to use the data. Technically, this flag has the variable name '@T'. With the flag saying only yes or no, a problem will occur when an application sets the flag to yes and does not complete processing and turn the flag back to no. This problem can occur when the system is interrupted by pressing the RESET key. The RESET key is the black button in the upper left corner of latest model keyboard and is the little silver button in the upper right corner of the older model keyboards. The black button has the word 'RESET' stamped upon it whereas the silver button has no label but has the word 'RESET' stamped on the keyboard.

When the RESET key is pressed, program execution is interrupted. This can create a problem when the program was using the shared program and the variable @T was set for hogging the shared program space.

Since the 2200 is a multi-tasking system, the analyst can choose to have one program (task) that is available to all the other programs within one bank of memory. When the 2200 has more than one bank of memory, the system can have a shared program in a special memory partition which can be accessed by all memory within the a single 2200. When the analyst has a system designed for a single 2200 and must change the system for two or more 2200s, the KFAM system allows a very minor change in the definition of which programs are shared. That is, with a single 2200, one program (KFAM0307) is shared in each bank (called local global) and one program (KFAM0407) is shared by all banks of memory (called universal global). Whereas, with more than one 2200 CPU only one global program is used (KFAM0207).

Summary of problems and recommended changes for Sisteco's KFAM system.

1. Problem: When function 31 is used, files CCONTRA and/or CONTROL can be left open by @@CCP0020.

Recommendation: a) Change logic of @CCP0020 in section 'ACTUALIZO' (lines 7620 to 7690) so that file is not open while other processing is being performed or b) Add a call to close multiplexed files ('219) to the subroutine ('31) that closes KFAM files.

2. Problem: When operator presses special function key zero (0) or one (1), the programs open, store data, and close the special data files CCONTRA and CONTROL.

Recommendation: Change the function numbers to larger numbers. If system design requires that operator use these function, then use a number that requires the SHIFT key. For example, by using special function twenty (20) instead of zero (0), the operator must hold the SHIFT key and the function key. If the system design does NOT require the operator to use these function, then use a number that can not be accessed from the key board.

3. Problem: Program @DFP0010 does not release the customer data (CLIEF100) that is shared with program @CCP0020. So, if both operators are using the same customer number, the operator using program @CCP0020 must wait for the operator using program @DFP0010 to use another customer number or finish processing.

Recommendation: a) Add the release ('238) function to program @DFP0010, or b) Change the find old request from access with record protection ('232(1,1,FO$)) to access without record protection ('232(1,0,FO$).

4. Problem: When both operators are using the same program, one operator may have to wait a long time for data to be saved because record protection is used for saving data.

Recommendation: After the data has been saved, use the release function.

5. Problem: Program @CCP0020 can take a long time to save data after the operator has entered one transaction.

Recommendation: Print a message that tells the operator the system is working with the data.

6. Problem: When system is configured, the printer must be selected and ready to print.

Recommendation: Change program MENUGRA by a) advising operator to prepare the printer, or b) removing code from MENUGRA and perform processing in the program that uses the printer.

7. Problem: Locating data with a key value and loading data which changes the variable. Then using the changed variable for processing.

Recommendation: Use a temporary variable and confirm that the loaded value is the same (correct) value.

Problem 1: When function 31 is used, files CCONTRA and/or CONTROL can be left open by @@CCP0020.

Recommendation: a) Change logic of @CCP0020 in section 'ACTUALIZO' (lines 7620 to 7690) so that file is not open while other processing is being performed or b) Add a call to close multiplexed files ('219) to the subroutine ('31) that closes KFAM files.

The section 'ACTUALIZO' from program @CCP0020 and the necessary supporting subroutines are attached. At statement 7630, we find that the file CCONTRA is opened but not closed. The file is closed at statement 7670 but if anything goes wrong during the other processing the file will be left open. That is, when something goes wrong a subroutine ('31) is called which closes the KFAM files but not the multiplexed files CCONTRA or CONTROL. The list of potential things to go wrong is:

A. Record not found in file CC11F100.
B. Record has wrong format for unpack statement.
C. File CONTROL can not be opened.
D. File CCONTRA can not be closed.
E. Operator presses RESET and function 31.

When problems A, B, or C happen, the program would use subroutine '31 to close the KFAM files and perform the statement END. After the 'END', the operator can restart the program by entering RUN and RETURN. So, the operator, instead of asking the system administrator about the problem, could enter more data until another problem occurred.

The problem D would happen only if another program had the disk hogged. This would probably be a rare occurence but is a possibility.

The problem E would happen when an operator assumed the system was 'hung again' instead of simplying taking much longer than normal to save the data. Problem number four discusses this in more detail.

```
 0010 REM %

 PROGRAMA @CCP0020

 7620 REM %

 ACTUALIZO

-7630 I1 = 10
    : GOSUB 7710 -
    : GOSUB 7900
    : GOSUB 7970
    : I1 = 0
    : GOSUB 7710 ————    CCONTRA LEFT OPEN
 7640 GOSUB 8570 ———(A) CC11FISO  FIND OLD DATA
    : GOSUB 7850 ————(B)- UNPACK DATA
    : X1=7
    : GOSUB 8660 ————(C) CONTROL LEFT OPEN
    :  IF A(3) <> 1 THEN 7650
    : G1(1) = G1(1) + A(21)
    : G0(2) =G0(2) + A(21)
    : B(5)= B(5) + A(21)
    : GOTO 7660

-7650 G1(2) = G1(2) + A(21)
    : G0(3) = G0(3) + A(21)
    : B(6) = B(6) + A(21)
-7660 G0(4)= G0(1) + G0(2) - G0(3)
    : B(7) = B(4) + B(5) - B(6)
 7670 I1 = 10
    : GOSUB 7740 ————————(D) CCONTRA  CLOSED
    : I1=0
    : X1=7
    : GOSUB 8690 ————————    CONTROL   CLOSED
 7680 GOSUB 7760 ——
 7690 FOR I = 15 TO 20 ——   SAVE DATA TO CC11FICC
    : A(I) = 0
    : NEXT I
    : RETURN
```

```
10 REM % PROGRAMA @CCP0020
 7960 REM %
```

ALTA TRANSACCIONES

```
-7970 PACK(######) STR(A$,5,3) FROM B9
    : GOSUB '233(3,1,A$,0)
    : IF Q$ = " " THEN 7990
    : IF Q$ ="B" THEN 7970
    : IF Q$ = "D" THEN 7980
    : GOSUB '100 ("ERROR EN FINDNEW - TRANSACCIONES",Q$,21,0,1)
    : GOSUB '31
    : END                                                   CLOSE KFAM
-7980 GOSUB '100("TRANSACCION YA INGRESADA"," ",21,0,1)
    : RETURN

-7990 INIT (FF) E0$()
    : INIT ("0") E1$(),E2$(),E3$()
    : IF Q = 1 THEN 8000
    : DATA LOAD DC #6,E0$(),E1$(),E2$(),E3$()
    : DBACKSPACE #6,1S
-8000 E0$(Q) = A$
    : E1$(Q)= H1$
    : E2$(Q) = H2$
    : E3$(Q) = H3$
    : DATA SAVE DC #6,E0$(),E1$(),E2$(), E3$()
    : GOSUB 7790
    : RETURN

10 REM % PROGRAMA @CCP0020
 7780 REM %
```

GRABO ADDRESS1

```
-7790 M1= M1 + 1
    : IF M1 > 6 THEN  GOSUB 7810             WHEN M1 = 7
 7800 STR(A$(M1),1,2)="04"
    : STR(A$(M1),3,13) = A$
    : STR(A$(M1),33,1) = " "
    : CONVERT S2 TO STR(A$(M1),34,2),(##)
    : CONVERT C1 TO STR(A$(M1),36,1),(#)
    : RETURN

-7810 M1=1
    : B1$="ADDRESXX"
    : CONVERT S2TO STR(B1$,7,2),(##)
    : GOSUB '217 (B1$,9,S2,0,4," ",M1$,0) OPEN  ADDRES02 OR ADDRESØ5
    : IF Q$=" " THEN 7830
    : IF Q$="A" THEN 7820
    : GOSUB '100("OPEN - ADDRESXX",Q$,21,0,1)
    : GOSUB '31
    : END                                          CLOSE KFAM
-7820 $BREAK
    : GOTO 7810                                    : END

-7830 DSKIP #9,END
    : DATA SAVE DC #9,A$()
    : GOSUB '218(B1$,9,M1$,0)
    : GOSUB '219 (B1$,9,S2,M1$,0)
    : INIT(FF) A$()
    : RETURN
```

```
10 REM % PROGRAMA @CCP0020
-8570 GOSUB '232(2,1,F0$)
    : IF Q$=" " THEN 8580
    : IF Q$= "B" THEN 8570
    : GOSUB '100("ERROR EN FINDOLD - SALDOS C/C",Q$,21,1,0)
    : GOSUB '31
    : END
```

CLOSE KFAM

: END

```
-8580 DATA LOAD DC #4,F6$(),F7$(),F8$()
    : DBACKSPACE #4,1S
    : B3=Q
    :  RETURN

10 REM % PROGRAMA @CCP0020
-7850 X=1
    : FOR I9 = 1 TO 4
    : UNPACK(-#########.##) STR(F7$(B3),X,6) TO B(3 + I9)
    : X = X + 6
    : NEXT I9
    : RETURN

10 REM % PROGRAMA @CCP0020
-8660 GOSUB '217("CONTROL",X1,S2,0,4," ",M1$,0)
    : IF Q$= " " THEN 8680
    : IF Q$ = "A" THEN 8670
    : GOSUB '100 ("ERROR EN OPEN - CONTROL",Q$,21,0,1)
    : GOSUB '31
    : END
```

CLOSE KFAM

: END

```
-8670 $ BREAK
    : GOTO 8660

-8680 DATA LOAD DC # X1,G0,G1$,G2$,G0(),G3$,G4$
    : RETURN
```

```
0010 REM %
```

PROGRAMA @CCP0020

```
-7710 GOSUB '217 ("CCCONTRA",8,S2,0,4," ",M1$,0)
    : IF Q$=" " THEN 7730
    : IF Q$ ="A" THEN 7720
    : GOSUB '100 ("ERROR EN OPEN - CCCONTRA",Q$,21,0,1)
    : GOSUB '31
    : END
-7720 $ BREAK
    : GOTO 7710
-7730 DATA LOAD DC #8,G2,G5$,G1(),G2(),G6$
    : IF I1 = 0 THEN RETURN
    : G2(1)=G2(1)+1
    : IF G2(1) > 9999 THEN G2(1) = 1
    : PACK (####) STR(A$,12,2) FROM G2(1)
-7740 DBACKSPACE #8,BEG
    : DATA SAVE  DC #8,G2,G5$,G1(),G2(),G6$
    : GOSUB '219 ("CCCONTRA",8,S2,M1$,0)
    : RETURN
```

CLOSE KFAM

: END

WAIT

RETURN

LEAVING CCONTRA FILE OPEN

CLOSED

```
0010 REM %
```

PROGRAMA @CCP0020

```
7890 REM %
```

EMPAQUETO CAMPOS TRANSACCIONES

```
-7900 PACK(###) STR(H1$,1,2) FROM A(2)
    : CONVERT A(3) TO STR(H1$,3,1),(#)
    : CONVERT A(4) TO I$,(######)
    : P1$=STR(I$,5,2)&STR(I$,3,2)&STR(I$,1,2)
    : CONVERT P1$ TO A(4)
    : PACK (######) STR(H1$,4,3) FROM A(4)
    : CONVERT A(5) TO STR(H1$,7,1),(#)
 7910 PACK(######) STR(H1$,8,3) FROM A(6)
    : PACK (############)STR(H1$,11,6) FROM A(7)
    : CONVERT A(8) TO STR(H1$,17,2),(##)
    : STR(H1$,19,1) = STR(F5$,3,1)
 7920 PACK(##) STR(H1$,20,1) FROM A(10)
    : PACK (######) STR(H1$,21,3) FROM A(11)
    : PACK (######) STR(H1$,24,3) FROM A(12)
 7930 PACK(##) STR(H1$,27,1) FROM A(13)
    : PACK (######) STR(H1$,28,3) FROM A(14)
 7940 X=1
    : FOR I9 = 1 TO 8
    : PACK (########.##) STR(H2$,X,5) FROM A(14 + I9)
    : X = X+5
    : NEXT I9
 7950 A(23)= 0
    : PACK(######.##) STR(H3$,1,4) FROM A(23)
    : A(24)= 0
    : PACK (########.##) STR(H3$,5,5) FROM A(24)
    : PACK (########.##) STR(H3$,10,5) FROM A(25)
    : RETURN
```

Problem 2: When operator presses special function key zero (0) or one (1), the programs open, store data, and close the special data files CCONTRA and CONTROL.

Recommendation: Change the function numbers to larger numbers. If system design requires that operator use these function, then use a number that requires the SHIFT key. For example, by using special function twenty (20) instead of zero (0), the operator must hold the SHIFT key and the function key. If the system design does NOT require the operator to use these function, then use a number that can not be accessed from the key board.

....

The special function button on the keyboard are numbered zero (0) to thirty-one (31). When a program can defined these functions as either a one line computation (or message) or as a complete subroutine. The operator can perform the function by pressing the key when the program has stopped or when the program is waiting for data entry. A processing problem may occur when the operator uses a function at the wrong time. For example, the Sisteco system uses function zero and one to open, save data, and close special data files CONTROL and CCONTRA. When this can happen is shown by looking at the screens from the two programs @CCP0020 and @DFP0010:

From program @CCP0020, after operator has entered zero to finish the processing:

```
-----------------------------------------------------------------
/1/        ATENCION: ANTES DE CANCELAR ESTE PROCESO VERIFIQUE
/2/        QUE LAS DEMAS PANTALLAS LO HAYAN HECO
/3/        DIGITE (RETURN) PARA CONTINUAR?
/4/   ?
/5/   ?
/6/   ?
/7/        DIGITE (RETURN) PARA CONTINUAR?
------------------------------------
```

The line number labels /1/ thru /7/ show what the operator sees on the screen. The lines /1/, /2/, and /3/ is normal processing. The message is translated to:

```
    ATTENTION: BEFORE CANCELLING THIS PROCESS VERIFY
    THAT THE OTHER SCREENS HAVE DONE IT.
    PRESS -RETURN- TO CONTINUE
```

So, if while the operator is checking that other terminals have finished, the special function key zero is mistakenly pressed, then the line /4/ will appear. The operator must press return twice to get lines /5/, /6/, and /7/. This could very confusing to the opertator and instead of pressing return, the operator might press RESET and function 31.

From program @DFP0010, after operator has entered one to start the processing:

```
-----------------------------------------
/1/       *** VALORES DIFERIDOS ***

/2/       0 - FIN DE LA TAREA
/3/       1 - ALTAS
/4/       2 - BAJAS

/5/       INGRESE OPCION?
/6/  ?
/7/  ?
/8/  ?
-----------------------------------------
```

The line number labels /1/ thru /8/ show what the operator sees on the screen. The lines /1/, /2/, /3/, /4/, and /5/ is normal processing when the program is entered from the menu. However, if the operator presses the function key one an extra time or happens to touch function key zero after pressing function one, then the extra '?' are displayed.

Appendix III

Problem 3: Program @DFP0010 does not release the customer data (CLIEF100) that is shared with program @CCP0020. So, if both operators are using the same customer number, the operator using program @CCP0020 must wait for the operator using program @DFP0010 to use another customer number or finish processing.

Recommendation: a) Add the release ('238) function to program @DFP0010, or b) Change the find old request from access with record protection ('232(1,1,FO$)) to access without record protection ('232(1,0,FO$).

When the operator using program @DFP0010 (DISCHARGES) enters a customer number and the other operator using program @CCP0020 (INPUT/CHARGE) enters the same customer number, then the operator using program @DFP0010 must wait for the operator using program @DFP0010 to finish processing. The program @DFP0010 will keep the data sector protected until either a different cusomter number is used or the operator finishes the program and returns to the system menu. Especially confusing is that the operator can return to the program menu (VALORES DIFERIDOS) and still have the record protected and keeping the other operator waiting.

Since both programs are only reading the data from the customer data file (CLIEF100), there is no apparent need for access with record protection because no update is performed. That is, data only read not changed and resaved. Therefore, reading without record protection would not affect the processing.

Appendix IV

Problem 4: Program @CCP0020 can take a long time to save data after the operator has entered one transaction.

Recommendation: Print a message that tells the operator the system is working with the data.

Program @CCP0020 has a section named 'MODULO CONTABLE' (lines 7520, 7610) which processes the data entered by the operator. The save process (step 5 of general description) will require ten cycles by the following sequence of statements and data conditions:

1. At line 7530, the screen is cleared.
2. With 'C2' is not one, branch to line 7540.
3. With 'C2' is not two, branch to line 7580.
4. At line 7580, a loop of ten is started.
5. Within the loop, when all values in array 'V()' are not zero, the 'ACTUALIZO' subroutine will be used ten times.

While this process is being performed, the operators' screen is blank with the cursor in the upper left corner. Therefore, the operator does not know if the system is hung or if the the program is still working.

Problem 5: When both operators are using the same program, one operator may have to wait a long time for data to be saved by the other program because record protection is used for saving the data.

Recommendation: After the data has been saved, use the release function.

....

The long time delay can be a result of:

A. One of the files being opened is busy.
B. The data sector for a record is busy.
C. The number of save cycles (step 5) is large.

Usually the save cycle (step 5) is performed only once. However, the save cycle may have to be used ten (10) times which results in a long delay which can appear to be a system hang.

The file 'ADDRESS1' contains data that reports:

1. Bytes 1 and 2 are '05'
2. Bytes 3,4,5,6 are account number.
3. Bytes 34 and 35 are partition number
   partition three (03) is terminal one (1)
   partition five (05) is terminal two (2)
4. Byte 36 is part of program being used.
   ('1' for ALTAS or '2' for BAJAS)

Using this information, we see that both terminal were using the program @DFP0010 and were entering data for account 1516.

Problem 6: When system is configured, the printer must be selected and ready to print.

Recommendation: Change program MENUGRA by a) advising operator to prepare the printer, or b) removing code from MENUGRA and perform processing in the program that uses the printer.

....

When the system is configured, the program MENUGRA is started in partition number three (3) for terminal number one (1). With the screen cleared, the printer is selected and output sent to the printer. However, the select statement has a zero (0) width, which is technically correct, but the operator does not see anything printed. So, the operator may not know the importance of the printer being turned on and selected. The operator can get the terminal to 'unhang' by pressing RESET, entering CLEAR, and doing LOAD RUN of another program.

This is a very minor issue but could have a major impact on the system because the operator will have experienced a time when it is O.K. to use the RESET key.

....

7. Problem: Locating data with a key value and loading data which changes the variable. Then using the changed variable for processing.

Recommendation: Use a temporary variable and confirm that the loaded value is the same (correct) value.

....

In program @CCP0020, the find old statement at line 5520 uses variable FO$ as the key value ('232(1,1,FO$). When the data is loaded at line number 5510, the same variable (FO$) is used to load the data. The data could have a problem and not have the same value in the user file as is in the key file. So, if they are different a problem would be created with this processing.

With the backup received from Sisteco, the data did not have this problem. We discuss this here only for completeness of the things which could cause a problem with the processing.

MEMORANDUM

TO: Janice Grandy SUBJECT: ISS Rel. 5.3

CC: Sheila Mitchell DATE: November 11, 1985

FROM: KM Ken Mailloux 2200/VS Value Added Product Support

The Integrated Support System (ISS) was updated to take advantage of the New Disk Catalog Hashing Algorithm introduced with Rel. 2.5 of the 2200 MVP Operating System. That is the basic difference between ISS Versions 5.2 and 5.3.

The current Version of the MVP OS is 2.6.2 and it also supports the New Hashing Algorithm. Release 5.3 is the compatible ISS.

The only file to change extensively was ISS.229S which is the "search index" routine. This routine allows a user to use the new disk catalog hashing algorithm which was not supported in earlier versions of the OS. There is a significant change in the size of this file, the original version (5.2) was approximately 1.5K bytes and with 5.3 it is approximately 2.4K. It is about .84K larger with 5.3, users may need to know this.

Other modules that have changed since Rel. 5.2 are:

ISS.031U - Change was made so that HEX(80) as part of a valid line number would be "masked" because it resembles the keyword "LIST" during compression. Without this change a program going through compression, which included any line from 8000 to 8099, could stop prematurely with an unrecoverable error.

ISS.081U - Change made to line 1710 for the same reason noted for the ISS.031U update.

ISS.217S - Change made to line 3880, the statement "DEFFN '254" found on that line in the 5.2 version should have been "GOSUB '254". Without this change the List/Cross Reference Utility with the List Option and all options specified would abort on line 8310 with a P48 error because slot #3 of the device table had no valid disk address assigned.

KFAM3707 - Change made to line number 4870 to initialize the sector buffer in the re-organize rebuild subroutine. Without this change the re-organize/rebuild subroutine would not function properly with certain KFAM files.

KFAMREFS - This file was edited so that it could be referenced properly. The difference between this version and the ISS 5.2 version has to do with the last absolute sector of the file. The sector was reconstructed for the new release.

If you have any questions please call me at 60381 at 59 Electronics Avenue MS 0115.

WANG LABORATORIES, INC.

KFAM-7 PRELIMINARY SPECIFICATIONS

KFAM-7 is a version of KFAM designed to run on the 2200 MVP. It is a modification of KFAM-5. File structures (Key File and User File) are exactly the same as in KFAM-5. No file conversion is required going from KFAM-5 to KFAM-7. Programs have been modified to take advantage of the design structure of the MVP. Minor changes will be necessary to convert user programs from KFAM-5 to KFAM-7.

2200MVP PROGRAMMING NOTE

On the 2200 MVP, any partition may have access to a common printer, usually specified as device 215. It is possible to have garbled printout as a result of more than one partition printing to device 215 at the same time. Therefore, applications programs must be changed for the MVP in one of the following ways:

1) A local printer can be attached to a given terminal and specified as device 204.

2) The common printer can be hogged by a particular partition for the duration of the printout, for example:

```
$OPEN/215
```

Or the printer can be assigned a device address and hogged that way:

```
SELECT #15/215
$OPEN #15
```

3) A program with infrequent printing can write the information to be printed on disk, then print it in a subsequent program, hogging the printer for a shorter period of time.

KFAM-7 PARTITION GENERATION CONSIDERATIONS

KFAM-7 operates under the Integrated Support System Release 3.7 and is only operable on a 2200MVP Central Processor. ISS-3.7 start-up operation requires that the following has occurred prior to loading the ISS "START" module.

1. The 2200MVP must have been Master Initialized.

2. All peripheral device addresses to be used as ISS device addresses must be set in the 2200MVP Master Device Table. Device addresses may be viewed and updated by choosing the EDIT DEVICE TABLE option during partition generation. Refer to the 2200MVP Introductory Manual for all partition generation instructions.

3. A partition configuration must be executed based upon the following guidelines. For KFAM-7, each partition must have at least a 9.0K memory size in order to run KFAM-7 utility programs. A 9.0K partition is also required for the global KFAM-7 subroutines, which are contained within the module "KFAM0107" (or a customized module created by BUILD SUBROUTINE MODULE). The user may either (1) LOAD and RUN "KFAM0107" (or customized equivalent) following each partition generation or (2) copy "KFAM0107" (or customized equivalent) to the 2200MVP Operating System Diskette and have the automatic bootstrap feature implemented to LOAD and RUN this global module (e.g., assign terminal 1 two 9K partitions with the higher partition number set to automatically bootstrap "KFAM0107").

4. After executing the partition configuration, the $RELEASE TERMINAL statement is available to attach a terminal to a different partition, which may be necessary to LOAD and RUN "KFAM0107" if the automatic bootstrap feature was not implemented.

5. ISS-3.7 start-up procedures may begin.

ISS-3.7 START-UP

ISS-3.7 start-up operation differs from ISS-3.2 start-up operation and much less peripheral checking occurs. A "STATION NUMBER" and not a "CPU NUMBER" is requested during ISS-3.7 start-up. Because ISS-3.7 operates only on the 2200MVP, it is recommended that the STATION NUMBER be the partition number currently in use. For each station number, a 10-sector system configuration file (station file) containing ISS-3.7 start-up default peripheral addresses may be created during ISS-3.7 start-up operation. The station file allows ISS-3.7 to load/store and update default (common variable) peripheral addresses automatically during ISS start-up operation for that station number.

The same potential problem exists as in ISS-3.2 for the station number, that is, if two partitions (or CPU's multiplexed to a shared disk) perform start-up at the same time and enter the same station (or CPU) number, the system configuration table's defaults may be changed without the station actually changing the defaults. Later, the station number entered is equated to common variable S2 and is available for use by all software running in that partition. The use of variable S2 by software may cause problems related to disk file access and updates if unique station numbers are not used.

In addition to those common variables read from the station file, ISS-3.7 equates other common variables based on internal tests during ISS start-up, e.g., memory size. ISS common variables are the same as in ISS-3.2, with the addition of CRT width, scalar S0.

ISS-3.7 start-up instructions follow.

1. LOAD and RUN "START".

2. In reply to the ENTER STATION NUMBER prompt, the following options are available:

   a. To view existing station files and their default values contained within, touch S.F. Key 00.

   b. To create a station file for a new station number, touch S.F. Key 16.

   c. Otherwise, enter the station number (1-48). If the number is not accepted (the prompt reappears) a station file does not exist on the ISS diskette in use for the station number entered. If accepted, see step 3.

3. In reply to the ENTER DESIRED FUNCTION prompt, if the displayed default values for DATE, PRINTER ADDRESS, DISK ADDRESSES, and ISS LOADING ADDRESS are <u>not</u> acceptable, enter the corresponding function number (1-4) and enter the value(s) desired for that function. When entering the PRINTER ADDRESS, the entry is not accepted if the printer is not ON and SELECTED. Any disk address or printer address entered <u>must</u> have been entered in the 2200MVP's Master Device Table, although Master Device Table and on-line disk address checking does not occur. Standard device addresses are displayed for the various address prompts and used for entry checking.

   When the displayed defaults are acceptable, the following options are available:

   a. To save the displayed defaults into the station file, touch S.F. Key 00.

b. To load an application program via the APPLICATION MENU and view all common variable values, touch S.F. Key 16. The default application loading address and file name are also displayed and may be modified. In reply to the APPLICATIONS MENU, enter 0(zero) to load the specified application program.

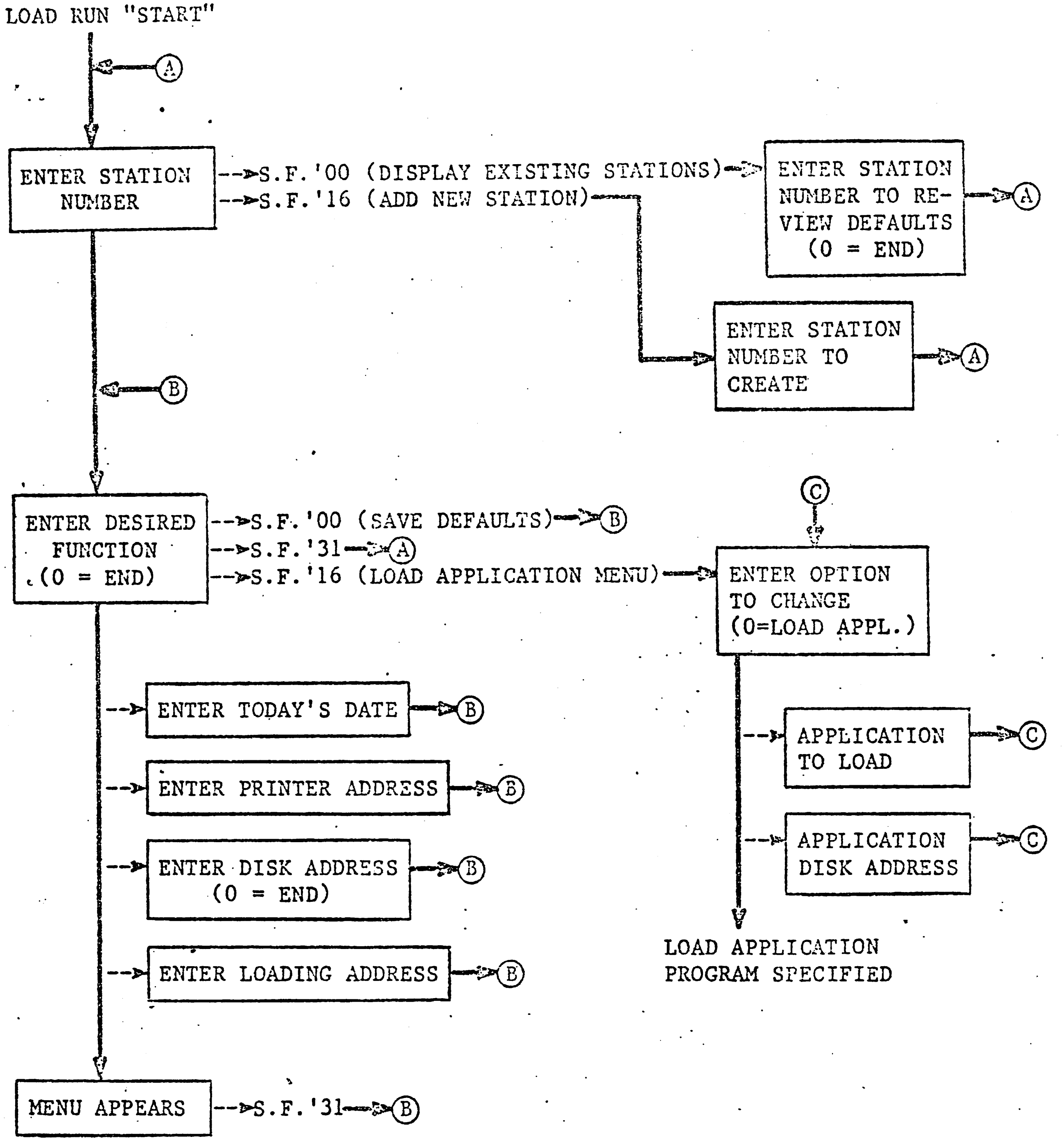
c. To obtain the KFAM-7 menu, enter 0(zero). If a fixed/removable disk contains the modules from the KFAM-7 diskette, the KFAM-7 menu appears; however, if the fixed/removable disk contains KFAM-7 and other ISS-3.7 components (when available), a SYSTEM MENU appears. This is part of the ISS-3.7 dynamic menu hierarchy which adjusts to the ISS-3.7 software on a ISS disk.

Refer to the illustration below for an overview of prompt linkages.

NOTE:

S.F. Key 31 is valid to return to the previous level prompt in reply to the KFAM-7 menu, the Applications menu, and the ENTER DESIRED FUNCTION prompt. Also, with all ISS-3.7 utility programs including KFAM-7, S.F. Key 31 is valid to abort a program, close any files open and return the KFAM-7 menu to the screen if the terminal is <u>not</u> in Edit mode (a blinking cursor indicates Edit mode is active and may be switched off by touching the EDIT key once).

ISS-3.7 START-UP FLOWCHART


LOAD RUN "START"
A
ENTER STATION NUMBER
S.F.'00 (DISPLAY EXISTING STATIONS)
S.F.'16 (ADD NEW STATION)
ENTER STATION NUMBER TO RE-VIEW DEFAULTS (0 = END)
A
ENTER STATION NUMBER TO CREATE
A
B
ENTER DESIRED FUNCTION (0 = END)
S.F.'00 (SAVE DEFAULTS)
B
S.F.'31
A
S.F.'16 (LOAD APPLICATION MENU)
C
ENTER OPTION TO CHANGE (0=LOAD APPL.)
ENTER TODAY'S DATE
B
ENTER PRINTER ADDRESS
B
ENTER DISK ADDRESS (0 = END)
B
ENTER LOADING ADDRESS
B
APPLICATION TO LOAD
C
APPLICATION DISK ADDRESS
C
LOAD APPLICATION PROGRAM SPECIFIED
MENU APPEARS
S.F.'31
B

Dashed lines indicate optional prompts. S.F.'31 is a valid reply to any prompt and moves from level C to prompt B or from level B to prompt A .

KFAM-7 SUBROUTINES

The KFAM-7 subroutines (module KFAM0107) are designed to be resident in a global partition named "KFAM," to be accessed by any other partition. Only the instructions, and not the variables, are resident in the global partition. This code may be executed by a number of calling partitions simultaneously.

Module KFAM0107 (or a user subset generated by BUILD SUBROUTINE MODULE) should be loaded into the global partition and run, to set up global variables, before being accessed by another partition.

The KFAM-7 variables (module KFAM0007) are overlaid or included in each calling partition. KFAM0007 (variables only) physically replaces KFAM0005 (variables and subroutines) in the application program. Each calling partition has its own set of KFAM variables, but they all share the same global KFAM subroutines.

The arrays on line 225 of module KFAM0007 may be dimensioned from 1 to 8 depending on the number of KFAM files to be accessed by this particular application. The default is 3.

The complete set of KFAM-7 subroutines will fit in a 9K partition, including 1122 bytes of MVP overhead. The KFAM-7 variables, per calling partition, occupy about 1000 bytes, plus 87 bytes per KFAM-7 file accessed.

The same software multiplexing scheme is used in KFAM-7 as is used in ISS-3 and KFAM-5. File access tables are maintained in the last sector of each data file to indicate which workstations are accessing the file, in which access mode. Subroutines MUX OPEN, MUX END, and MUX CLOSE are included in the KFAM-7 subroutine package, to control access to any file, whether a KFAM file or not.

In order to access the global KFAM, the user program must execute SELECT @PART "KFAM" at the start of the program and following any subsequent overlay.

The KFAM-7 subroutines are logically equivalent to the KFAM-5 subroutines, with the following exceptions:

OPEN(230): The device address of the User File is not used, but this parameter (last parameter in GOSUB'230) must be included, with either a real value or a dummy value.

OPEN returns Q$ = "S" if the internal table of open files (@T$(), see below) is full, in addition to error codes A, D, P, and X already defined.

Set and reset hog mode (215): This subroutine has been dropped. GOSUB'215 should be replaced with $OPEN or $CLOSE, to set or reset hog mode.

SET-UP (216): This subroutine has been dropped. The set-up logic is inherent in the loading of the KFAM partition and the calling partition. The KFAM common variables (KFAM0007) can be loaded and then overlaid by the application program, to save the space occupied by the COM statements.

MUX OPEN (217), END (218), and CLOSE (219): The device address parameter is not used in any of these subroutines. The parameter itself must be included, but it may be a dummy variable or value.

SEARCH CATALOG INDEX (229): This subroutine has been dropped. It is replaced by the BASIC-2 version of the LIMITS statement.

CLOSE (239): This subroutine has been changed slightly. Recovery information will not be saved if the access mode is (1) INQUIRY or (2) READ-ONLY. With access modes (3) SHARED or (4) EXCLUSIVE, recovery information will be saved. (If the subroutines have been tailored using BUILD SUBROUTINE MODULE, the option "CLOSE WITH RECOVERY INFO" must be specified to save recovery information with access modes 3 and 4.) To save recovery information regardless of access mode, use WRITE RECOVERY INFO (214) before closing the file.

All other subroutines are funtionally the same as in KFAM-5.

Access modes 1 (INQUIRY), 2(READ-ONLY), 3(SHARED), and 4 (EXCLUSIVE) are the same as in KFAM-5. Access mode 9 (not multiplexed) has been dropped.

The protect flag parmeter (dummy variable P) in subroutines FINDOLD, DELETE, FINDNEW, FINDNEW (HERE), FINDFIRST, FINDLAST, FINDNEXT, and FINDPREVIOUS may have the following values:

0 or 2 = no sector protection

1 or 3 = sector protection

The option to retain hog mode has been dropped in KFAM-7. This option is almost impossible to use in KFAM-5, because of the combination of hardware and software hogging of the Key File, and the various combinations of file accessibility that can result when multiple terminals are accessing multiple files. The same problems apply in KFAM-7, and therefore the option of holding hog mode has been dropped. It would mean complicating the program to support an option which would be rarely, if ever, used.

Similarly, in the MUX subroutines, the parameter to hold hog mode is ignored. MUX OPEN, MUXEND, and MUX CLOSE hog the file to do the necessary updates, and then release it. Hog mode may be set by the user before calling the particular subroutine, but will be released by the subroutine if another station has the program hogged, and released when the subroutine is finished.

The user may set and reset hog mode in user programs without danger of interfering with KFAM operations, but the general rule is that hog mode will be released in all the KFAM or MUX subroutines for the Key File, the User File, or both.

Two possible approaches are suggested for complex updating operations. One is to run the program with all files in EXCLUSIVE mode. The other is to do all KFAM operations first, setting protect flags as necessary and saving record pointers (T4$) if necessary, at then to set hog mode in the user program and do all the updating operations.

In KFAM-7, the word "station" replaces the term"CPU", as used in KFAM-4 and KFAM-5. The word "station" can refer to a terminal, or more specifically to a partition calling the global "KFAM" partition. In future versions of KFAM-7, the "station" could also be a separate CPU (2200T, VP, or MVP) multiplexed to the same disk as theMVP. The variable S2, which was formerly referred to as "CPU number," is now "station number," but its usage is the same despite the change in terminology.

KFAM-7 controls access to the disk internally, through tables in the global area, rather than by reading and writing the KDR, as is done in KFAM-5. This saves disk access time and also save time by not hogging the disk except on OPEN, RE-OPEN, CLOSE,. and WRITE RECOVERY INFO. All KFAM accesses go through the same global KFAM. It is important that the KFAM-7 subroutines should not reside in more than one global partition, because this module (KFAM0107) not only provides subroutines but also coordinates accesses to KFAM files.

In the global area is a table of open files, @T$(30)14. The number of table entries can be increased or decreased from 30, depending on the maximum number of KFAM files that can be open at any one time, by all stations on a given system. The contents of this table, per entry, are as follows:

START	LENGTH	CONTENTS
1	2	KDR sector (V0$) = starting sector of key file .
3	1	Device address of key file, compressed. Bytes 2 and 3 of device address are packed, then OR'ed with 80 if byte 1 = "B".
4	1	Number of index levels, T0, in IBM packed format.
5	2	Relative sector address of highest level index sector, T2$.
7	8	Per station 1 - 16, one half byte for internal completion code.

Note 1: The first 3 bytes above form a unique identifier for the particular key file being accessed.

Note 2: The internal completion code is used for two purposes, first to dtermine whether a particular file is open or closed for a particular station, and second to determine whether internal variables T0 (number of index levels), T2$ (sector address of highest level index), T2$() (path through index to current record, in terms of KIR sectors read), and T$ (path to current record, in terms of KIE starting location within KIR) are currently valid. Bit settings within the internal completion code are as follows:

0	normal completion, above variables valid.
1	path not defined (T2$() and T$), KDR OK.
2	path not defined (T2$() and T$), reread KDR (changed by another station).
4	index level added, get new values of T0, T2$ from table @T$(), above.
8	error condition, next and previous records not defined.
E	file open
F	file closed.

The bit settings above are unpacked to T8$. In access modes 1 (INQUIRY) and 3 (SHARED) internal completion codes are packed into table @T$(). In access modes 2 (READ-ONLY) and 4 (EXCLUSIVE), the only values appearing in table @T$() are "E" for open and "F" for closed.

An internal file ID, V6, is maintained by KFAM-7 to indicate the number of the entry for a given file within @T$(). @T$(V6) is the table entry for the particular file. The internal file ID should not be confused with the KFAM ID, the Key File Number, or the file numbers of the Key File and User File in the device table. The latter are specified by the user as OPEN parameters. The internal file ID is another number which is assigned by KFAM-7 for its own internal use, when the file is opened.

In the non-interactive modes (2 = READ-ONLY and 4 = EXCLUSIVE), KFAM-7 simply makes an entry in table @T$() to indicate that the file has been opened. It also stores TO and T2$ in the table @T$ (V6) to save reading the KDR when files are switched. It then operates very much as KFAM-3, where no interaction is possible, becuase in EXCLUSIVE mode no other CPU can access the file, and in READ-ONLY mode no other CPU can change the Key File.

The interactive modes are defined as 1 = INQUIRY and 3 = SHARED. KFAM-7 maintains a queue to regulate access to files in the interactive modes. The queue contains two entries, the station nubmer (S2), in hex, in one byte of @Q$, and the internal file ID (V6), in hex, in the corresponding byte of @Q9$. Upon entry to a KFAM-7 subroutine in an interactive mode, the station number and internal file ID are placed at the end of the queue. The queue (@Q9$) is then searched for the internal file ID, and the station number in the corresponding position of @Q$ has access to the file. All other stations requesting the file must wait. When the station accessing the file is finished, its entry is drpped from the queue, and the next station in the queue requesting that file is allowed to access it.

The queue allows access to the file on a first-come-first-served basis. This is an improvement over KFAM-5 which establishes no such priorities and gives all stations waiting an equal chance to get the file next, regardless of which one requested it first. It is possible, under KFAM-5, with bad luck, for one station to wait a long time, whereas under KFAM-7 luck is not a factor in gaining access to a file, and access is allowed from the various stations in the order that it was requested.

There is only one queue for all stations requesting all files. This does not inhibit different stations from accessing different files at the same time. For example, if the queue looks like this:

@Q$(station)	1	3	5	4	2
@Q9$(file)	4	2	1	2	3

Station 1 can access file 4, station 3 can access file 2, station 5 can access file 1, and station 2 can access file 3, all at the same time. Station 4 must wait to access file 2, until station 3 is finished.

Like KFAM-4 and KFAM-5, KFAM-7 maintains a table of protected sectors, protecting a sector of the User File from access by another station if the protect flag is set (parameter P = 1 or 3). But unlike KFAM-4 or KFAM-5, in KFAM-7 the table of protected sectors is stored in a global variable, @V4$(30)4, and not in the KDR. This global table contains all protected sectors for all stations accessing all files. The array dimension can be changed upwards or downwards from 30 if necessary. If the table is full, the station requesting sector protection simply waits until there is a vacant slot in the table. Sector protection is only effective in the interactive modes (1 and 3).

The contents of @V4$(), per entry, are as follows:

START	LENGTH	CONTENTS
1	1	Station number, S2, hex
2	1	Internal file ID, V6, hex
3	2	Protected sector = STR(T4$,,2)

NOTE: If a User File is being accessed by 2 or more Key Files, a protected sector as accessed through one Key File will not be recognized by a subroutine accessing the file through another Key File. KFAM does not support multiple Key Files per User File, but it is possible to design a protection scheme external to KFAM. This is also true for KFAM-4 and KFAM-5.

Through the use of these internal tables, reading and writing the KDR and hogging the disk are kept at a minimum. Instead of the KDR being the communications link between different stations accessing the same file, this communication information is held internally, thus eliminating disk access time and making throughput more efficient.

Global variables are also used as program constants and working variables, in order to cut down on the space required for KFAM variables in the user partitions. In order to update global variables, the program is hogged at certain critical times. This means that there are certain points in module KFAM0107 which can only be executed by the one station or partition which has gained access, as opposed to the normal case where the code may be executed at the same time (logically) by several different stations. At these critical points, all stations other than the one which has gained access must wait.

Points at which the program is hogged are as follows:

MUX OPEN (217)
MUX END (218)
MUX CLOSE (219)
OPEN (230): Setting up table @T$(),
   executing MUX OPEN (217)
CLOSE (239): Setting table @T$(),
   executing MUX END (218) and MUX CLOSE (219)
RE-OPEN (213): Executing MUX OPEN (217)
WRITE RECOVERY INFO (214): Executing MUX END (218)
FINDNEW (233) and FINDNEW (HERE) 234):
   Whenever a KIR sector is split, or about one time in eight, depending on key length and other factors.
Any subroutine: Adding or deleting a queue entry,
   updating internal completion codes

The times when the program is hogged are either infrequent or brief, and therefore should not slow down performance very much.

KFAM-7 UTILITIES

The KFAM-7 Utilities are the same as the KFAM-5 Utilities, with the following exceptions:

The KFAM-7 Utilities require the KFAM-7 subroutines to be loaded in a global partition named "KFAM". The utilities then run from another partition, accessing the global "KFAM" as required. The utilities require a partition size of 9K, in addition to the 9K partition requried for the KFAM-7 subroutines.

If the subroutines are tailored using BUILD SUBROUTINE MODULE, it should be noted that the following subroutines are required for the KFAM-7 Utilities:

230 OPEN
232 FINDOLD
234 FINDNEW(HERE)
235 FINDFIRST
237 FINDNEXT

CLOSE WITH RECOVERY INFO

The screen-handling subroutines for the KFAM-7 Utilities assume an 80-character CPU.

The words "CPU" or "CPU #" have been replaced with the word "STATION" in all screen displays.

The work file "KFAMWORK" in EXCLUSIVE mode is used by all KFAM-7 Utilities to generate code. Therefore, it is possible in all utilities to get the message "WORK FILE NOT AVAILABLE" if the work file is being used by another station. The work file is also used throughout utilities KEY FILE CREATION and KEY FILE recovery to store error messages to be printed. Therefore it is impossible to run another KFAM-7 Utility concurrently with either of these.

INITIALIZE KFAM FILE (KFAM1007)

If the option to get a hard copy printout of file specifications is taken, the message WAITING FOR PRINTER will be displayed on the screen if the printer is not available (not turned on, not selected, or hogged by another partition). The printer is hogged by KFAM1007 while printing file specifications, and then released.

KEY FILE CREATION UTILITY
KEY FILE RECOVERY

In the common module KFAM2007 which is shared by both these utilities, the following changes have been made:

The prompt "TURN ON PRINTER, KEY RETURN(EXEC) TO RESUME" has been eliminated. Error messages for duplicate keys and unreadable records are written on the work file "KFAMWORK" instead of printed directly on the printer. The file "KFAMWORK" is held in EXCLUSIVE mode for the duration of the program, thus making it impossible for any other KFAM-7 Utility to operate concurrently. There is a limit to the number of error messages that can be contained in the work file, and therefore the following program stop has been added: "STOP WORK FILE FULL".

The message "NO DUPLICATE KEYS" at the end of the program has been dropped. If there are no error messages, nothing is printed. If there are error messages, module KFAM2107 (new) is loaded to print the error report.

KFAM2107 displays "WAITING FOR PRINTER" if the printer is not available (not turned on, not selected, or hogged by another partition). That message goes away when the printer becomes available. It hogs the printer while the error report is being printed, and then releases it.

If the user has designated the screen as the printer, S$(1) = blank, KFAM2007 stores the error messages in disk file "KFAMWORK", and KFAM2107 then displays them one at a time on the screen, followed by a STOP. This is different from KFAM-5 which displays duplicate keys and unreadable sectors as they occur.

The message "ERROR ## LINE ####" is eliminated. Any errors other than disk read errors will result in a "hardware" error display.

CONVERT KFAM-3 TO KFAM-7
CONVERT KFAM-4 TO KFAM-7

Note that the conversion programs operate exactly the same as in KFAM-5, and also that there is no need to convert KFAM-5 to KFAM-7, since file formats are exactly the same.

PRINT KEY FILE

"WAITING FOR PRINTER" is displayed if the printer is not available. The program hogs the printer for the duration of its operation. Keying HALT/STEP and Special Function 31 will terminate the operation.

Note that protected sectors and internal completion codes are not stored in the KDR in KFAM-7 and therefore are not shown on the report. Protected sectors are shown as HEX(FFFF) and internal completion codes as "Z".

RESET ACCESS TABLE

This program resets access information in the last sector of the User File and in the KDR in the Key File, as in KFAM-5. It also resets access information for this file in internal tables @T$(), @V4$(), @Q$, and @Q9$.

Other stations may be accessing other files while this program is being run, but no other station should be accessing the file being reset.

BUILD SUBROUTINE MODULE

The prompt "NUMBER OF FILES (1 - 8)" has been removed. Module KFAM0007, line 225, should be modified manually for the number of files to be accessed by a particular application, by changing the array dimensions to a number from 1 to 8. The default is 3.

BUILD SUBROUTINE MODULE is probably not as useful in KFAM-7 as in previous versions of KFAM, because the KFAM subroutines are global, and must include any subroutine to be accessed by any calling partition while the subroutine module is resident in memory. If certain subroutines like FINDPREVIOUS are never used or if the machine is dedicated to a specific application using a specific set of subroutines, BUILD SUBROUTINE MODULE can be used to advantage, shortening the necessary size of the global partition. But in the general case it is perhaps best to load KFAM0107 intact into the global partition, and not use this utility.

The number of options has been shortened from KFAM-5. The choices are as follows:

Fn	Choice
01	230 OPEN
02	231 DELETE
03	232 FINDOLD
04	233 FINDNEW
05	234 FINDNEW(HERE)
06	235 FINDFIRST
07	236 FINDLAST
08	237 FINDNEXT
09	212 FINDPREVIOUS
10	238 RELEASE
11	239 CLOSE
12	CLOSE WITH RECOVERY INFO
13	213 RE-OPEN
14	214 WRITE RECOVERY INFO
15	217 MUX OPEN
16	MUX OPEN NEW
17	218 MUX END
18	219 MUX CLOSE
29	CANCEL
30	PROCESS
31	RETURN TO MENU

Special Functions 1 to 18 select the respective subroutines to be included in the subroutine module to be built. Subroutines which are called by other subroutines are automatically included, as in KFAM-5. For example, CLOSE WITH RECOVERY INFO automatically includes 239 CLOSE, 218 MUX END, and 219 MUX CLOSE.

CANCEL, PROCESS, and RETURN TO MENU operate the same as in KFAM-5, except that CANCEL is now Special Function 29 instead of 01, as in KFAM-5.

Special Function Keys not included in the above list are ignored.

At the bottom of the screen display, the number of files selected has been dropped. It reads simply "MODULE XXXXXXXX", where XXXXXXXX is the module name selected by the user.

TECHNICAL INFORMATION

Variable V: The variable V is not written as part of the KDR record. Its primary purpose is to serve as a flag to indicate which subroutine is executing. Its secondary purpose is to indicate whether the current version of the KDR is necessary to execute the subroutine. If V is greater than 4, the current version of the KDR must be in memory in order to execute the subroutine. Values of V are as follows:

1 = FINDNEXT (237)
2 = FINDPREVIOUS (212)
4 = default, FINDOLD (232),
FINDFIRST (235), FINDLAST (236)
RELEASE (238)
5 = RE-OPEN (213), WRITE RECOVERY INFO
(214), CLOSE (239)
6 = DELETE (231)
7 = FINDNEW (HERE) (234)
8 = FINDNEW (233)

V is used as a working variable (access mode) in OPEN (230).

KDR Record: The KDR consists of a numeric variable (flag) followed by array T$(), as in KFAM-5, except that the flag is not used and is always set to zero.

Variables Q4$ and V5$ have been replaced by T3$ (current record pointer for FINDNEW). This change should be made in the description of the KDR.

V0$(3)21, internal storage, per file: The hex image for unpacking is changed to @Q8$20. Packed variable Q0$ has been dropped, and variable V6 has been changed. Contents are now as follows:

VARIABLE	START	IMAGE	DESCRIPTION
V0$2	1	A002	Absolute starting sector of Key File.
V6	3	5002	Internal file I.D.
V4$4	5	A004	Hex image for unpacking entry from KIR, HEX(A0XXA003), where XX = Key Length
V0	9	5001	Access mode
T1	10	5002	File #, Key File.
T2	12	5002	File #, User File.
T4	14	5002	Key length
T5	16	5002	KIE length = T4 + 3.
V7	18	5002	KIR bytes used = INT(240/T5)*T5
V1	20	5002	Last key location = V7-T5+1

T5$(3)58, internal storage: The hex image for unpacking is changed to @T5$10. The contents of T5$() and @T5$ remain the same.

KFAM-7 SYSTEM DISK

The KFAM-7 System Disk should contain the following modules:

MODULE	DESCRIPTION
KFAMWORK	Work file for the KFAM-7 utilities. This is a data file of 15 sectors. All other modules are program files.
KFAM0007	KFAM-7 variables, to be included in user program.
KFAM0107	KFAM-7 subroutines, to be loaded in global partition.
START	Starting module for KFAM-7 utilities
ISS.OnnD	ISS start-up station file for station nn (1-48)
ISS.001M	ISS start-up functions
KFAM-7	KFAM-7 utilities, menu and initial dialog
KFAM1007	INITIALIZE KFAM FILE
KFAM2007	KEY FILE CREATION UTILITY, also called by KFAM9007, KEY FILE RECOVERY
KFAM2107	KEY FILE CREATION UTILITY and KEY FILE RECOVERY, print duplicate keys and unreadable sectors
	REORGANIZE IN PLACE:
KFAM3007	start up
KFAM3107	Generate code
KFAM3207	Reorganize
KFAM4007	REALLOCATE KFAM FILE SPACE
KFAM5007	CONVERT FROM KFAM-3 to KFAM-7, or KFAM-4 TO KFAM-7
KFAM6007	PRINT KEY FILE
KFAM7007	RESET ACCESS TABLE
KFAM8007	BUILD SUBROUTINE MODULE (uses KFAM0107 as input file)
KFAM9007	KEY FILE RECOVERY, start up (uses KFAM2007 to build Key File)
KFAM9907	Close files (called by all KFAM-7 utilities)
	REORGANIZE SUBSYSTEM (stand-alone):
KFAM3507	Start up, open files
KFAM3607	Generate code
KFAM3707	Reorganize, parts 1 and 2
KFAM3907	Part 3, close files
ISS.REFB	ISS Copy/Verify Reference File

ISS/KFAM	ISS 2 - KFAM3/4	ISS 3 - KFAM/5	ISS 5 - KFAM/7
Diskettes	3 diskettes ISS Utilities Disk Support KFAM-3	3 diskettes or 7 minis ISS Utilities KFAM-5 ISS Screen Disk / Sort 4	3 diskettes ISS Utilities KFAM-7 ISS Screen Disk / Sort 4
Utilities	COPY/VERIFY Sort disk Disk dump Decompress List-cross reference Compression Reconstruct index Create reference file	COPY/VERIFY Sort disk Disk dump Decompress List-cross reference Compression Reconstruct index Create reference file Program compare Copy tape to disk	COPY/VERIFY Sort disk Disk dump Decompress List-cross reference Compression Reconstruct index Alter index Create reference file
	KFAM-3 is one CPU KFAM-4 is multiplexed	* For multiplexed environment * File status report	File status report
Sub-routines	Search catalog index Allocate data file space Free unused sectors Datentry Open/close output Open/close input Alpha input Numeric input Position cursor Date Operator wait	Search catalog index Allocate data file space Free unused sectors Datentry Open/close output Open/close input Alpha input Numeric input Position cursor Limits next Multiplexed open end close Date Print Operator Wait Re-enter	Select validate disk Search catalog index Allocate data file space Free unused sectors Datentry Open/close output Open/close input Limits next Multiplexed open end close Print
Memory	KFAM-3 12K KFAM-4 16K Sort-3 8K/KFAM file 12K	ISS Utilities 12K Program comp. 16K KFAM-5 16K Sort-4 8K/12K	Limits next Multiplexed open end close Print Re-enter
Sort	Must have DC End Records must be all in same format Very specific data Uses canned utilities	File types 1. Ordinary cataloged data 2. BAS-1 3. ISS open/Close sub. 4. KFAM-3 5. KFAM-4 6. KFAM-5 Set up module	File Types 1. Ordinary cataloged data 2. BAS-1 3. ISS Open/close sub. 4. KFAM-3 5. KFAM-4 6. KFAM-5 or 7 Set up module

KFAM Utilities	Reorganize utilities deletes reorders 1.Reorganize subsystem 2.Reorganize KFAM file	Reorganize utilities deletes reorders 1.Reorganize subsystem 2.Reorganize KFAM file	Reorganize utilities deletes reorders 1.Reorganize subsystem 2.Reorganize KFAM file
	Adjust files 1. Reallocate KFAM file space (same disk 2. Disk copy & reorg		Adjust files 1. Reallocate KFAM file space (same disk) 2. Disk copy & reorg
	Print key file	Print key file	Print key file
	Key File Recovery	Key file recovery	Key file recovery
	Reset Access Tables KFAM-4 only	Reset Access Tables	Reset Access Tables
	Conversion Utilities KFAM-3: KFAM-1 to KFAM-3 KFAM-2 to KFAM-3 KFAM-4 KFAM-3 to KFAM4	Conversion Utilities KFAM-3 to KFAM-5 KFAM-4 to KFAM-5	Convert to KFAM-7 KFAM-3 to KFAM-7 KFAM-4 to KFAM-7
	Initialize KFAM-3/4 Keyfile create	Initialize KFAM file Key file create	Initialize KFAM file Build key file

Upgrading from KFAM-5 to KFAM-7

Customers who upgrade their 2200VP system to a 2200LVP or 2200MVP system will also want to upgrade their KFAM applications to take full advantage of the multi-user system. Two questions that are ususlly asked by the customer are as follows:

1. What will I gain by upgrading?
2. How do I upgrade?

This article will answer these two questions.

What will I gain by upgrading?

The primary reason why KFAM-7 was originally introduced was to take full advantage of the MVP multi-user operating system. With the multi-user operating system, KFAM-7 subroutines are run in a background partition while the user's program is run in a foreground partition. This makes more efficient use of user memory.

The subroutine programs KFAM0107 (single bank) and KFAM0307 (multiple bank) both access tables in core rather than on disk. KFAM-5 accesses tables on disk and was designed to run in one contiguous partition of memory. The KFAM-7 program, KFAM0207 (multiplexed) was designed to be used on the VP operating system and is basically the same as KFAM-5. An upgrade to KFAM-7 is not necessary if the user has a VP operating system and is running KFAM-5 (an exception to this is the presence of the Phoenix disk drive).

KFAM-7 uses BASIC II, whereas KFAM-5 uses the earlier Wang BASIC.

Multiple key access to the user file does not work with KFAM-5 or with KFAM-7's KFAM0207. The protect table is contained in the key file itself, so when the user file is being accessed by one of the key files, the other key files don't know that the protect bit has been set. In KFAM-7 the protect table is in the global background partition, and can be accessed by all the key files.

In KFAM-5 the Reorganize Subsystem utilities will reorganize the user file based on one key file. If there are multiple key files, the secondary key files will not be mapped to the new user file. KFAM-7 however, allows for multiple key files. The Reorganize/Rebuild Subsystem set-up module can be written to map the secondary key files to the new user file.

How do I upgrade?

For both MVP and VP versions:

1. Make a back-up copy of the user application.

2. CLEAR memory and then LOAD in user application.

3. CLEARP all KFAM subroutine text.

4. CLEARP all KFAM COMmon variables.

For the VP version:

1. RENUMBER the user program to start at location 4000, if necessary.

2. LOAD in KFAM0207 (subroutines).

3. LOAD in KFAM0007 (variables).

4. On line 40 take out:

```
DEFFN @PART "KFAM"
$RELEASE TERMINAL
$BREAK
```

5. On line 40 add:

```
GOTO 4000
```

6. In GOSUB'230 (OPEN) make sure that the variables are set properly. For example: The access method must be 1,2,3 or 4. KFAM-7 does not support access mode 9 (not multiplexed). Change all instances of access mode 9 to access mode 4 (exclusive).

7. Save the converted user application either under the same name as the previous KFAM-5 user application, if it has been saved elsewhere, or save it under a new name.

For the MVP version:

1. LOAD in KFAM0007 (variables).

2. Create a new line at line 50, (50 SELECT @PART "KFAM")

3. IN GOSUB'230 (OPEN) make sure that the variables are set properly. For example: The access method must be 1,2,3 or 4, KFAM-7 does not support access mode 9 (not multiplexed). Change all instances of access mode 9 to access mode 4 (exclusive).

$$T

2200 ISS - CONFIG. GUIDELINES/DEPENDENCIES

$$T
02/28/92

Information regarding support of select discontinued products can be found in the Discontinued Product Support section.

SYSTEMS SUPPORTED
CS-D, CS-N, CS/386-D/N, CS/386 TURBO

HARDWARE REQUIRED
Any currently supported 2200 workstation
Disk space allocated is approximately 1MB

SOFTWARE REQUIRED
No prerequisites or requirements

PERIPHERALS SUPPORTED
2536DW workstation or PC 200/300 series, IBM XT, AT and compatibles with PC2200 emulator

LITERATURE

Part Number	Title
700-5010A	2200 ISS Release 5 User Manual
700-5560A	2200 ISS Release 5 Reference Card

DATA SHEETS/MANUALS

Part Number	Title
700-6161	2200 ISS Data Sheet

$$T

2200 ISS - DISCONTINUED PRODUCT SUPPORT

$$T
02/28/92

The following represents a list of select discontinued products that are currently supported. This list is not all inclusive.

- o The discontinued CS-10D thru CS-80N, LVP, MicroVP, MVP, SVP and VP systems support ISS.
- o The PC240, 280 and 380 support 2200 ISS with PC2200 emulator as well as the 2436DW and 2436WP workstations.

$$T

2200 ISS - ADDITIONAL INFORMATION

$$T
02/28/92

- o To run the PC2200 terminal emulator, which allows a Wang PC 200/300 series or IBM XT or AT compatible PC to emulate a 2536DW workstation, operating system 3.3 or BASIC-2/386 1.0 is required.

$$T

$$T

2200 ISS - DESCRIPTION/STRATEGY

02/28/92

195-0052-x ISS (Release 5.2)

(x) = Media Type

The Integrated Support System (ISS), developed for the Wang 2200 Series product line, is a highly versatile software system which provides a wide range of programming and utility support through its file access software, utility functions, and pre-defined subroutines. The utility programs are user-controlled routines which allow program files to be copied, compressed, decompressed, listed, sorted, cross-referenced, and compared to other files. Special purpose utility functions allow creating, editing, or printing a reference file, as well as displaying or printing the contents of a data file. Screen/Disk subroutines perform standard programming tasks related to either user/screen or program/disk interaction and greatly reduce an application programmer's need to write repetitious, detailed routines.

The Key File Access Method (KFAM), an indexed sequential access method, offers rapid access to data by means of subroutines capable of handling both random and sequential record access. The versatile SORT subsystem, a major function of the ISS utility system, supports a variety of both record and file formats for sorting records.

ISS begins with system start-up procedures, displays current system information allowing for its modification, and maintains a hierarchy of menus which furnish access routes to both ISS support software and user-written programs. The start-up procedure makes standard system data available to all software in the system.