INFORMATION TO USERS

This material was produced from a microfilm copy of the original document. While the most advanced technological means to photograph and reproduce this document have been used, the quality is heavily dependent upon the quality of the original submitted.

The following explanation of techniques is provided to help you understand markings or patterns which may appear on this reproduction.

1. The sign or "target" for pages apparently lacking from the document photographed is "Missing Page(s)". If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting thru an image and duplicating adjacent pages to insure you complete continuity.

2. When an image on the film is obliterated with a large round black mark, it is an indication that the photographer suspected that the copy may have moved during exposure and thus cause a blurred image. You will find a good image of the page in the adjacent frame.

3. When a map, drawing or chart, etc., was part of the material being photographed the photographer followed a definite method in "sectioning" the material. It is customary to begin photoing at the upper left hand corner of a large sheet and to continue photoing from left to right in equal sections with a small overlap. If necessary, sectioning is continued again — beginning below the first row and continuing on until complete.

4. The majority of users indicate that the textual content is of greatest value, however, a somewhat higher quality reproduction could be made from "photographs" if essential to the understanding of the dissertation. Silver prints of "photographs" may be ordered at additional charge by writing the Order Department, giving the catalog number, title, author and specific pages you wish reproduced.

5. PLEASE NOTE: Some pages may have indistinct print. Filmed as received.

Xerox University Microfilms
300 North Zeeb Road
Ann Arbor, Michigan 48106
VOSBURY, Newman Arthur, 1932-
A DATA RETRIEVAL SYSTEM WITH HIGH STORAGE
EFFICIENCY AND A COMPUTER BASED INSTRUCTIONAL
SYSTEM.

The University of Oklahoma, Ph.D., 1973
Information Services, information storage and
retrieval systems

University Microfilms, A XEROX Company, Ann Arbor, Michigan
THE UNIVERSITY OF OKLAHOMA
GRADUATE COLLEGE

A DATA RETRIEVAL SYSTEM WITH HIGH STORAGE EFFICIENCY AND A COMPUTER BASED INSTRUCTIONAL SYSTEM

A DISSERTATION
SUBMITTED TO THE GRADUATE FACULTY
in partial fulfillment of the requirements for the
degree of
DOCTOR OF PHILOSOPHY

BY
NEWMAN ARTHUR VOSBURY
Norman, Oklahoma
1973
A DATA RETRIEVAL SYSTEM WITH HIGH STORAGE EFFICIENCY AND A COMPUTER BASED INSTRUCTIONAL SYSTEM

APPROVED BY

Richard V. Andree

James A. Layne

Stephen Figurski

Viktas B. Guslinskas

Thomas F. Hill

DISSERTATION COMMITTEE
ACKNOWLEDGEMENT

I would like to acknowledge the help and support of many people. In particular, Dr. Richard V. Andree has given much encouragement as a friend and teacher. The courses I have had from Drs. Payne, Gylys, Feyock, and Thompson have contributed much to my growth. Bev Dvorak has been most patient in typing the manuscript and very helpful in suggestions of form. Last, but not least, my family. My wife, Jane, has typed and proofread and put up with being a computer widow. My son, Larry, helped test the writing of lessons in SCRIPT.
Filmed as received
without page(s) iv.

UNIVERSITY MICROFILMS.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>LIST OF TABLES</th>
<th>vii</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF ILLUSTRATIONS</td>
<td>viii</td>
</tr>
</tbody>
</table>

## CHAPTER

I. INTRODUCTION ................................................................. 1  
Motivation for the project. Two components: an information storage and retrieval system (STRIPE) and a computer based instructional system (SCRIPT). Reasons for these two parts. Goals of the system. Examples of the uses of the systems.

II. CONCEPTUAL DESCRIPTION OF STRIPE ................................. 6  

III. FUNCTIONAL DESCRIPTION OF STRIPE ............................... 18  
The data model, simplified and in detail. File descriptions. The execution model. Program descriptions.

IV. CONCEPTUAL DESCRIPTION OF SCRIPT ............................... 46  
1. Hardware ................................................................. 46  
2. Software - An Overview ............................................... 47  
3. Data Storage, message file, decision, tables, system parameters, in-core data ............................................... 50  
4. Ease of Use .............................................................. 54  

V. FUNCTIONAL DESCRIPTION OF SCRIPT ............................... 58  
Program units.

VI. EVALUATION ................................................................. 65  

BIBLIOGRAPHY .......................................................................... 69
# LIST OF TABLES

<table>
<thead>
<tr>
<th>TABLE</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Performance of Hash Algorithm</td>
<td>12</td>
</tr>
</tbody>
</table>
**LIST OF ILLUSTRATIONS**

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Simplified diagram of the file system</td>
<td>20</td>
</tr>
<tr>
<td>2.</td>
<td>Category file illustration</td>
<td>27</td>
</tr>
<tr>
<td>3.</td>
<td>Descriptor file illustration</td>
<td>27</td>
</tr>
<tr>
<td>4.</td>
<td>Relationships of hardware and software units in SCRIPT</td>
<td>49</td>
</tr>
<tr>
<td>5.</td>
<td>Example of a state/action table, simplified</td>
<td>50</td>
</tr>
<tr>
<td>6.</td>
<td>State/action table with message vector</td>
<td>51</td>
</tr>
<tr>
<td>7.</td>
<td>SCRIPT decision table and message vector form</td>
<td>56</td>
</tr>
<tr>
<td>8.</td>
<td>SCRIPT message list form</td>
<td>57</td>
</tr>
</tbody>
</table>
A DATA RETRIEVAL SYSTEM WITH HIGH STORAGE EFFICIENCY AND A COMPUTER BASED INSTRUCTIONAL SYSTEM

CHAPTER I

INTRODUCTION

The motivation for this work stems from experience using a computer in a small college. Typically, one finds the following conditions. The computer is small, such as an IBM 1130 or an NCR 100. The use of the machine is divided between academic and administrative work. The administrative work is often hampered by the lack of sufficient memory. The academic work involves running programs in batch from some computer classes, a few math classes, a few natural science classes and a smattering of others. It does not seem economically feasible to have much, if any, online work through terminals. Most instructors feel they are too busy to learn how to use the machine. The few knowledgeable people running the computer center are too pressed for time to develop more efficient systems.

In this environment I was presented with a request to keep transcripts on file, which had been thought impractical on our size machine, an IBM 1130 with 8K by 16-bit core and a single disk holding about a million bytes. At the same time it was clear that more use of the computer should be made in instruction, but a simpler way to write the instructional modules must be found. Also, we needed to be able to teach more than one student at a time.

The foregoing led to the desire for two systems. One is an information storage and retrieval system generalized enough to allow it to be easily
installed for a wide variety of sets of data. The other is a generalized instructional system which would require little more of an instructor than the knowledge of his discipline.

Why should these be included in one package? First, because many small schools need both types of capabilities to justify having their computer. Second, because the hardware and many routines used in the instructional system lend themselves to remote access to the data retrieval system.

The goals for the information system are:
1. It should run on a computer with as little as 16K bytes of main memory and a single disk.
2. Storage should have at least 90% efficiency (defined in Chapter II).
3. Additions, deletions or modifications should be easily made at a rate of 1,000 per hour or more.
4. It should be easily adapted to differing formats of input.
5. It should have a query function in which selection and refinement of subsets of the data base can be made, sorting of the selected subset done, and listings in a variety of formats can be given.
6. The system should be expandable to handle extra disk drives.
7. It should be possible to add new fields to records without rebuilding the whole file.
8. The system should be relatively easy to transfer to a new computer.

The instructional system should meet these goals:
1. An instructor should be able to write a lesson easily - with no more than thirty minutes orientation to the system.
2. The system should be capable of original instruction or testing.
3. Feedback should be available to the instructor regarding the strength of the teaching module.
4. Up to eight terminals can be maintained in a time-sharing mode on a computer with 16K bytes of main memory.

5. The system should have an average response time of 3 seconds or less.

6. The cost in volume use should be around $1/hr. per terminal or less.

A search was made of existing systems that would at least approximately meet these goals. There was nothing available from IBM or known to CUETUG, an 1130 user's group. A very generalized information system, DRS, was available from Aeronautical Research Association of Princeton (now available from DNA, Inc.) which was very good. However, the storage efficiency was not great enough and a high volume of record modification was not practical. Another data storage system available from a government funded project and used by several 1130 installations also lacked in efficiency of storage, giving only about 50%. No good system for computer based instruction on the 1130 through terminals has existed. IBM has offered a communications capability on the 1130 but the cost was way beyond what would be possible for economical instruction. No version of Coursewriter or anything like it seems to exist for a machine of the size contemplated.

The information system developed is called STRIPE (System to Retrieve Information Packed Efficiently). The first version of STRIPE is now being used at Principia College. It has a storage efficiency of about 65%. The latest version has storage efficiency of up to 95%. Almost any kind of data may be held, integers, floating point numbers, or character strings. Execution times are good. The volume of data that can be handled is such that in an application such as at Principia (where cumulative academic records are kept) over 1,500 student records can be held in under a half million
bytes. (This means it can all fit on the standard single disk on an 1130.)
The system can be stretched to accommodate up to 2,500 records on a single
disk with degradation of execution. On a multi-disk system it could get up
to 13-15000 records. These records include all course information, names,
addresses, transfer credits, grade averages, and so forth. A complete evaluation
of STRIPE is given in Chapter VI.

Using STRIPE the user may obtain typical reports such as class lists,
grade reports, etc. in the above application. In addition he can inter-
actively query the system to select subsets of the master records. For
example, all the students with grade averages of 3.0 or better who are not
living in dorms could be selected. Statistics, such as maximum, minimum,
average, sum, and standard deviation can be requested. Selected subsets may
be further refined by applying additional criteria to them. They may also
be sorted on any fields and listed.

The instructional system is called SCRIPT. This is not an acronym -
just somewhat descriptive of the way an instructor would write a lesson
module. Simply put, the instructor writes a script containing information and
questions he would give the student. The student's response determines the
next part of the script to be given him.

SCRIPT is running and proven easy to use. A seventh grader and a
college instructor have each written lesson modules after less than half an
hour's instruction in the use of the system. Any subject matter can be
taught if the teaching can be in the form of a dialogue with questions and
answers. Teletypes or teletype compatible CRT's can be used. With a different
interface other terminals may be used.

Both SCRIPT and STRIPE will be made available to users. STRIPE will be
put into the library of the College and University Eleven Thirty User's Group (CUETUG). It should be a valuable addition to the group's software resources. SCRIPT will be available to accompany the interface which makes the system possible. The interface was loaned by Logicon, Inc. The device takes input from the terminals and puts it into buffer storage locations in the 1130 memory via cycle stealing. The interface also provides interrupts on reception or transmission of terminate characters, reception of a 'break' signal, and detection of a disconnect or parity errors.

In this work many concepts have been explored and exploited. Various data structures for intra-and inter-record organizations such as inverted lists, singly and doubly linked lists, etc. have been used. Language design and interpretation techniques were implemented in the query language as well as in the instructional system. In the instructional system simple time-sharing was implemented. Garbage collection techniques had to be evaluated and implemented. Searching and sorting techniques including hash coding, "Shell sort," "Quick sort," and others were tested, improved, and used.

In addition attention was given to the overall system concept outside the computer. To a great extent the computer system is adaptable to the user's way of doing things. This is illustrated in adapting the existing format of input cards, in leading the user when putting data or commands in through the console, and in the ease with which an instructor can use SCRIPT.
CHAPTER II

CONCEPTUAL DESCRIPTION OF STRIPE

1. Efficiency

The measurement of storage efficiency is straightforward. The ratio \( M/U \) is used where \( M \) is the minimum space theoretically required for a record, and \( U \) is the space actually used. Two problems arise in applying this definition.

First, should the space used by a master file record include some prorated portion of auxiliary files such as index and parameter files? Only the space actually used in the master record will be counted here. This simplifies the evaluation and makes it possible to evaluate master files for which the supporting files are unknown as is the case in the comparisons made in the last chapter. Also, in typical systems the volume of data in the master file is so much greater than that in the auxiliary files that it would make only a trivial difference in the above ratio to include them.

The second problem lies in deciding which fields are necessary data required to complete the information, and which are overhead. This can be subjective, but the differences in the ratio will be small.

2. Techniques Used to Achieve Efficiency

Variable length records

The master file contains variable length records. This is desirable for data bases in which the record may need to contain more data as time passes. This means the record must start out with some initial size and increase in size as needed. The increase could be determined dynamically
or set as a fixed increment. In STRIPE a fixed increment is used, stored as a system parameter.

Since variable length records are used, an index to the master file must be used. This means a pointer system must be defined. In STRIPE storage is on disk, so the pointer will involve the sector and the word within the sector. Here a design decision must be made. It would be desirable to have pointers fit in one 16-bit word. How many bits should be used for the sector and for the word? If we use 12 bits for the sector then there can be up to $2^{12} - 1 = 4095$ sectors. This leaves 4 bits for the word pointer. There are more than 16 words per sector, but if the word pointer is made to represent 20 word blocks it can reference up to 320 words. This fits in well with the IBM 1130 used in the project. The exact numbers can easily be changed to fit the specifications of a given computer. For a data base such as the one which started the project in which transcripts and other data are kept, about 2 records per sector would be possible. This would allow a file of up to 8190 records. Since this is designed for small schools, it should be adequate. Obviously, a change to a 2 word pointer could be made easily if necessary for a bigger file.

Generalized record descriptors

One source of waste in record storage is the allocation of space in records for fields that don't exist in some records. To allow each record to contain only the fields that actually are filled in, the following approach is used.

Records are considered in a generalized way as made up of four parts:
1. A mandatory block of fixed fields, present in all records.
2. An optional block of fixed fields which may or may not be present in
3. A variable field which may be of length zero. Several sub-fields may be included.

4. A block composed of zero or more groups of fixed fields. These will be referred to as the mandatory block, optional block, variable field, and variable block.

More generalization could be introduced, but this seems to allow for any reasonable type of record without introducing too much overhead.

Field embedding

One of the biggest reasons for low storage efficiency in many information structures is the use of more bits than necessary to store a given datum. As an example in the extreme, it is not unusual to find a whole word used to indicate sex when one bit would do. To avoid this problem the following approach is used.

Each block is considered as a string of bits. Fields are embedded in the string, allocated just enough space for the values to be entered.

The result is that no field takes more space than needed. To keep track of this, a pointer system describing record layout is kept in the system. This is stored once in a parameter file so it does not introduce any significant overhead.

3. System Parameters

A system parameter file is maintained. In it are kept the above mentioned field bit pointers, sizes of various files, block sizes, a character conversion table, etc. This allows non-duplication of data as well as effectively providing communication between different units of the system.
4. Character String Packing

Character data is stored as a subfield in the variable field. The subfield is prefaced with one word which tells which subfield it is. Also, the number of 16-bit words used in the subfield is given. The data is stored 3 characters per word, allowing for 40 possible characters. A special routine from the IBM commercial subroutines is used. If more than 40 different characters are used then they would be stored 6 bits per character for up to 64 characters, 7 bits for up to 128 characters, etc.

5. Structure of the Data

Each record has a prime key. Using the hashing algorithm described below on this key, a record's entry in an index is accessed. The index, of course, points to the record. If necessary, the record may be found by a sequential search of the index for a hash of the name, if any, of the record.

Records may be in one or more of various categories defined for the database. A file for these categories is maintained. Each category record contains a list of pointers to the master records in that category - an inverted list.

The groups that may be added to the fourth block of a record may have descriptors associated with them. A descriptor file is maintained. As with the categories, each descriptor includes an inverted list of pointers to the master records with groups associated with that descriptor. A descriptor might be a course description, parts description, etc.

6. Hash Code Algorithm

As mentioned above, hashing is used to access the master index. Hashing
is also used to access the descriptor file, though no index is needed since it can be arranged for the records to be of fixed length. It is also used in the garbage collection routine to keep track of the new addresses of the master records. The algorithm used is one of the best available, the linear quotient hash algorithm. A general description of it is given in (1). A comparison of it with other hash algorithms is given in (2).

The specific application of the algorithm here is as follows. To insert a new record or access an existing one, a sequence of pointers must be generated. Call them \( p_0, p_1, \ldots, p_n \) where there are \( n \) records in the file. The \( p_0 \) record is examined. If it is available for insertion or is the record being sought, as appropriate, then it is used. If not, we call the situation a collision. In such a case the \( p_1 \) record is examined and so on until either an appropriate record is found or \( n \) records have been processed. In the latter case either there is no more room for insertion, or the record sought does not exist in the file. Justification for this follows.

The sequence \( (p_i) \) is generated using what we call the prime key, \( K \), such as an identification number, and a divisor, \( D \). \( D \) will be a prime number here, though it could be done otherwise. The quotient, \( Q \), is computed as the integral part of \( K/D \). If \( Q \equiv 0 \mod D \), \( Q \) is set to 1. With the preliminaries out of the way, \( (p_i) \) is defined recursively as:

\[
p_0 = \text{remainder of } K/D + 1
\]

or

\[
= K - \left\lfloor K/D \right\rfloor * D + 1
\]

\[
p_{i+1} = (p_i + Q) \mod D + 1
\]

By the definition \( p_i \leq D \). Essentially then, \( D \) is the size of the file. A result in number theory shows that all the integers 1, 2, \ldots, \( D \) will be included
in \( (p_i), i = 0, 1, \ldots, D-1 \). A sketch of the proof of this is given in the appendix.

Divisors other than prime numbers might out perform a prime number in some cases. Nonprimes might also probe each file slot if necessary for many sets of keys. But using a prime number for \( D \) guarantees satisfactory performance regardless of the nature of the keys.

7. **Performance of the Hash Routine**

The performance of a hash routine depends partially on the nature of the keys used. The tests made on this one are on a set of 1,000 keys which are 6-digit student alpha numbers, ranging between 500000 and 999999. Eight hundred thirty eight were assigned by a registrar's clerk from an alpha table and 162 were artificially generated to fall between the existing ones in order to fill out the data set.

In Table 1, Load Factor is the percent of slots in the file used up. Data is gathered for entries from the three thirds of the data relative to the order in which they are entered. The average and maximum number of probes required to locate a record are given for each third as well as the overall figure. This can be compared to figures for the binary search technique in which 9 probes would be needed when the load factor is 50\% and 10 probes required for higher loads.

The efficiency of this accessing technique begins to drop seriously at a 90\% load with this data. An index file is being used in which each entry is only a few words. It does not use a significant amount of space to allow at least 10\% more slots in the index file than we will have entries.

A formula for the best average number of probes needed to access an entry in a hash table is given by Morris (3).
It is

\[ E = - \log(1-\alpha)/\alpha \]

where \( \alpha \) is the load factor. For \( \alpha = .8 \) this gives \( E \approx 2.01 \). The overall figure for \( \alpha = .8 \) in Table 1 is 2.14, which seems to be a reasonable approach to the optimum.

### TABLE 1

**PERFORMANCE OF HASH ALGORITHM**

<table>
<thead>
<tr>
<th>Load Factor</th>
<th>First</th>
<th>Third</th>
<th>Second</th>
<th>Third</th>
<th>Last</th>
<th>Third</th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td>50%</td>
<td>1.05</td>
<td>2</td>
<td>1.18</td>
<td>4</td>
<td>1.68</td>
<td>7</td>
<td>1.30 2</td>
</tr>
<tr>
<td>60%</td>
<td>1.06</td>
<td>3</td>
<td>1.49</td>
<td>6</td>
<td>2.21</td>
<td>8</td>
<td>1.55 8</td>
</tr>
<tr>
<td>70%</td>
<td>1.07</td>
<td>3</td>
<td>1.33</td>
<td>6</td>
<td>2.64</td>
<td>11</td>
<td>1.68 11</td>
</tr>
<tr>
<td>80%</td>
<td>1.09</td>
<td>4</td>
<td>1.62</td>
<td>7</td>
<td>3.82</td>
<td>21</td>
<td>2.14 21</td>
</tr>
<tr>
<td>90%</td>
<td>1.09</td>
<td>4</td>
<td>1.82</td>
<td>8</td>
<td>4.90</td>
<td>29</td>
<td>2.60 29</td>
</tr>
<tr>
<td>93%</td>
<td>1.09</td>
<td>4</td>
<td>2.37</td>
<td>11</td>
<td>6.41</td>
<td>32</td>
<td>3.29 32</td>
</tr>
<tr>
<td>95%</td>
<td>1.10</td>
<td>4</td>
<td>2.36</td>
<td>11</td>
<td>6.88</td>
<td>32</td>
<td>3.45 32</td>
</tr>
</tbody>
</table>

8. **Logical and Physical Files**

In order to use more than one disk for storage of the master records, there must be more than one physical file. STRIPE allows as many as five separate physical files making up the logical files. These files are given the symbolic numbers 1, 11, 21, 31, and 41. These numbers are specially chosen for the calculation of the right physical file when given the sector of the logical file.

The calculation of the proper physical file is done using as input the logical file sector and word block together with five values from the parameter
file. These give the number of sectors in each of the five possible physical files. The file number is built by starting with 1 and adding a multiple of 10 as indicated by comparing the logical sector to the parameters.

9. Limitations of the System

With the previously mentioned decision to use pointers consisting of a 12-bit sector pointer and a 4-bit pointer to the relative 20 word block within the sector there is a limitation of 4,095 sectors in the logical file. This would handle several thousand students in a typical student information system with 4 year transcripts included. If it were necessary to handle a larger file either word blocks of larger size - 40 or 80 words - would have to be used so that 13 or 14-bit sector pointers could be used, or 2-word pointers could be used. The latter would require additional space in pointer files such as the inverted lists kept in categories and descriptors. Straight forward programming changes would be necessary.

As presently written the system can be used with up to 5 disk drives. Assuming that peripheral files - index, descriptors, etc. - are kept on the disk which contains the operating system and all stored routines, the other drives would contain the master file. Advantages in speed would be found because less travel of the disk read/write arms would be necessary. Actually, the capacity of this system is only limited by the capacity of the hardware. It can use as many disk drives as can be run on the computer. On a 5-disk 1130 15,000 students can be handled.

10. Flexibility

There is great flexibility in types of data which can be stored. If records are virtually all character, the mandatory block may consist of very few fields, and the variable field will be large. We might have records
which consist of just the mandatory block if every record is fixed and numeric.

It is not unusual for the users of an information structure to find that their master file area has become full. They may have to totally rebuild their file. In STRIPE the procedure is easy. To understand the procedure involved as well as other aspects of the system to follow, we should first look at the parameter or communications file. During execution it is held in core.

For our purposes here we need to be aware of the following parameters.

NSLMF - the number of sectors in the logical master file
MFDIV - the divisor for the hash algorithm for the master file
NS1,NS11,NS21,—NS41 - the number of sectors in the physical files with symbolic numbers 1,11,21,31,41.

NWMB - number of words in the mandatory block
NWOB - number of words in the optional block
NWGP - number of words in the fixed groups in the variable block
NFMB - number of fields in the mandatory block
NFOB - number of fields in the optional block
NFGP - number of fields in each group in the variable block
BP1,BP2,—BPM - pointers to the bits at which the fields of the mandatory block, optional block, and variable field begin. (BP1 = 0)

GP1,GP2,—GPN - pointers to the bits within a group at which fields of the fixed groups in the variable block start. (GP1 = 0)

The parameter file includes many other quantities, some set at the time of generation of the system as are the above, and some dynamic, but the above are all that are needed for the moment.

Continuing with the problem of enlarging a full file, the first step is
to define new file space. If there has previously been 1 physical file for the master records it will have been symbolically represented as file 1. In this case the logical and physical file are the same. Now, the new file space will be an extension of the logical file. It will have symbolic number 11. The one logical file will be composed of two physical files. Whenever I/O on the logical file is to be performed the pointer to the location in the logical file is converted to a pointer into the physical files using calculations on the logical pointer and NS1,NS11,...NS41. (The system could easily handle more than five physical master files by including more of these parameters.) If we already had physical files 1 and 11, the new one would be 21.

The second step is to add file definition statements describing the new file space to all programs which reference the master file and recompile them.

Third, the parameter for the number of sectors in the physical files must be updated.

Fourth, the new file space must be initialized to identify it as free space. Then the system is ready. In terms of time it could all be done in half an hour or so. The new file space may be on the same disk as the old or on a disk on a different drive.

Sometimes after an information system has been in use for a while, it is found necessary to add new fields to the records. Again, with many systems a complete rebuilding is necessary. With STRIPE it is only necessary to update the parameters for number of words and fields in the various blocks and the field bit pointers, and then to run a program which moves the information in each record over to make space for the new field. This would involve a few minutes, human time, and a fairly small amount of machine time,
since most records will have some available space.

Fields may be defined but unused by simply allocating zero bits for the field. That is, we could have

\[ \text{BP7} \ 38 \quad \text{BP8} \ 41 \quad \text{BP9} \ 41 \quad \text{BP10} \ 49 \]

for field bit pointers. This means that field 7 begins with bit 38, field 8 begins at bit 41, field 9 begins at bit 41, etc. Thus field 8 has no bits. If it is expected that in the future field 8 will be used, it should be taken into account at the time of generation. By this process the extra storage space will not be used until required.

A useful feature of this technique is the ability to overlay definition of fields. As an example, we might wish to add a group to a student's record representing a course he has enrolled in giving department number, course number, section, and date. When grades are given if it were not desired to keep the section code, the grade and course credit codes could be put in its place. The field bit pointers for a group might look like:

\[ \text{GP1} \ 0 \quad \text{GP2} \ 5 \quad \text{GP3} \ 15 \quad \text{GP4} \ 23 \quad \text{GP5} \ 32 \quad \text{GP6} \ 15 \quad \text{GP7} \ 19 \quad \text{GP8} \ 23 \]

In this arrangement the department number begins with bit 0 and includes 5 bits. Course number begins at bit 5 and includes 10 bits, section at bit 15 with 8 bits, and date at bit 23 with 9 bits. The fifth field is a dummy representation to give the end of the fourth (date) field. The grade code, field 6 and the course credit, field 7, overlay the section. Field 8 is another dummy entry. Programs which were designed to operate on courses being taken currently would be able to refer to the section field while programs to enter the grade would refer to the grade and credit fields.

11. Modularity

STRIPE is modular in various ways. Input-output routines can be changed
to fit the needs of a particular application. The file accession algorithms are independent of the programs which add, delete, modify and retrieve records. Since the system is configured with parameters, the sizes of various files as well as the size and layout of master records can be changed without changing the programming.
CHAPTER III

FUNCTIONAL DESCRIPTION OF STRIPE

1. The Data Model

Simplified Model

First we look at a simplified picture of the model of the file system. Though this leaves out a host of details it should provide a good framework to fit the details into later.

STRIPE makes use of six permanently stored files which are essential to its operation. Another four files are used in an auxiliary status for the storage of standard messages, abbreviations, etc. In a given application there might be more or fewer of these auxiliary files. The six essential files are for: 1) master records, 2) category descriptions, 3) descriptions for the types of groups that might be appended to master records, 4) parameters for the system, 5) index entries for the master records, and 6) a working space file.

The master record has 4 parts as described in Chapter I, a mandatory fixed block, an optional fixed block, a variable field, and a block of zero or more fixed field groups. Each master record is identified by a key number. This could be a hash of an alpha field if an inherent identification number is not present.

It may be desirable to be able to categorize master records according to a given field contents. To this end the category file contains a name of the category and an inverted list of pointers to each master record in that
category. The maintaining of categories may be omitted by turning off a switch in the parameter file.

The descriptor file contains descriptions related to the groups of fixed fields which may be added to the end of the master records. For each descriptor an inverted list is kept of master records having that kind of group. Use of this file may also be turned off with a switch in the parameter file.

The parameter file carries a great variety of information such as the size of files, pointers to the beginning of the chain of free areas and the end-of-file free area, character tables and so forth. In the following examples only a few of the parameters are used. (See Figure 1.)

The work file is not used in the following example. It is used for temporary storage in garbage collection, selection of subsets of master records, etc.

This example starts with the parameter file. It tells us there are 6 records in the master file. The index has 7 slots. (The physical size may be 7 or more, but if the divisor used in the hash algorithm is 7, that will be the effective size.) The field in the master records used as the criterion for categories is field 4. The first of the chained free areas begins at address 40. The free area which goes to the physical end of file begins at address 65.

In this simplified model the addresses are in terms of word blocks. The free area beginning at block 40 is 7 blocks long as given in word 8. Word 9 points at the next free area at block 54. That free area is 6 blocks long and is the end of the chain since the pointer is 0. The free area at block 65 is not chained. It extends to the end of the file and is pointed to by the parameter file. The first record has key = 103. Its entry in the index
### Parameter File

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of master records</td>
<td>6</td>
</tr>
<tr>
<td>Size of index</td>
<td>7</td>
</tr>
<tr>
<td>Category field</td>
<td>4</td>
</tr>
<tr>
<td>Pointer to free area</td>
<td>40</td>
</tr>
<tr>
<td>Pointer to end-of-file free area</td>
<td>65</td>
</tr>
</tbody>
</table>

### Descriptor File

<table>
<thead>
<tr>
<th>Descriptor</th>
<th>Key</th>
<th>Inverted list</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>20</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>D2</td>
<td>15</td>
<td>31 0 0 0</td>
</tr>
<tr>
<td>D3</td>
<td>1</td>
<td>31 47 0 0</td>
</tr>
<tr>
<td>D4</td>
<td>0</td>
<td>0 0 0 0</td>
</tr>
</tbody>
</table>

### Category File

<table>
<thead>
<tr>
<th>Category</th>
<th>Key</th>
<th>Inverted list</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>31</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>C2</td>
<td>1</td>
<td>20 60 0 0</td>
</tr>
<tr>
<td>C3</td>
<td>47</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>C4</td>
<td>15</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>C5</td>
<td>0</td>
<td>0 0 0 0</td>
</tr>
</tbody>
</table>

### Figure 1. Simplified diagram of the file system.

#### Master File

<table>
<thead>
<tr>
<th>Address</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>103</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>D1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>105</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>D2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>111</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>D1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>113</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>D2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Word 8 = 7; word 9 = 54; other words = '$/'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>47</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Word 8 = 6; word 9 = 9; other words = '$/'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>54</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Word 8 = 6; word 9 = 0; other words = '$/'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>97</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>C1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>111</td>
<td></td>
<td></td>
<td></td>
<td>Word 8 = 32000; word 9 = 0; other words = '$/'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Index

<table>
<thead>
<tr>
<th>Key Pointer</th>
<th>Address</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>105 15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>113 31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>97 60</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>115 47</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>103 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>111 20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
file is in the sixth slot. We can calculate this as the proper slot by using the hash algorithm. Using a divisor of 7, the quotient $\frac{103}{7} = 14$ with remainder of 5. The pointer should be $5 + 1 = 6$. This record also has field 4 = C2. Checking the category file we see that category C2 has the pointer 1 in its inverted list. This master record also has a couple of groups of types D1 and D3. In the descriptor file we see that the inverted lists for D1 and D3 each have pointers to block 1. In the illustration each record is shown having all four parts for ease in labeling, but of course this is not necessarily the case. Each of these files would probably have more data in the records, such as names of categories, descriptor types, etc.

A detailed breakdown of the record contents on each file is presented next. Notice that in each case some data is system data while the bulk of it is relevant to the data base.

2. File Detail

As described above there are master, category, descriptor, parameter, index, and work files, as well as a standard message file which are basic to the system. In the sample system for a school registrar, described in the appendix, there are also files for department and letter grade symbols, names of school requirements, and descriptions of the school requirements. These special files are optional.

In the file descriptions the following terminology is used. A pointer to a record or free area in the master file is a 16-bit word of two parts, the relative sector and relative word block. The relative sector is the sector relative to the beginning of the logical file where the first sector is numbered 1. The relative word block is relative to the beginning of the sector. Block size is 20 words.
Master file

The master file, while logically one file, may be made up of from one to five physical parts. These are symbolically referred to as files 1, 11, 21, 31 and 41, as explained earlier.

In succeeding sections there will be references made to field pointers and word pointers within records. These would be calculated when fields and blocks of master records must be accessed. There is a significant difference between the two types of pointers. The field pointer to a given field is always the same, even if some preceding fields do not happen to be present as would be the case for a field in the block of groups when the optional block is not present. On the other hand the word pointer to the beginning of the variable field, for instance, depends on the presence or absence of the optional block.

In a given record the layout for the group is repeated for as many groups as there are in the record. The field number of the \( i^{th} \) field in the \( n^{th} \) group would be:

\[
NFMB + NFOB + 1 + (n-1) \times (\text{no. of fields in group}) + i
\]

As an example the field number of descriptor key 3 in group 6 would be:

(assuming 7 fields per group counting the dummy and 45 fields in the mandatory, optional, and variable fields blocks)

\[
46 + (6-1) \times 7 + 3 = 84
\]

If the system were searching a record for some specific data in the 4th field of a group, it would start with field 50, then 57, 64, etc. until the last group was checked. Since the beginning of the mandatory block has the number of groups in the record, this can be done easily.

The storage efficiency of this scheme can be evaluated as follows. The first four fields of the mandatory block use up 30 bits. Each block ends on a
## MASTER RECORD DETAIL

<table>
<thead>
<tr>
<th>Field No.</th>
<th>Description</th>
<th>Bits**</th>
<th>Pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Begin Mandatory Fixed Block</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Record Status</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>bit 0: optional block</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>bit 1: variable field</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>bit 2-6: unused</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>bit 7: error</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Length of variable field in words</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>Number of words in record</td>
<td>9</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>Number of groups of fields in block of groups</td>
<td>6</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>(Any more fields are application specific. Assume the rest are as in the sample in the appendix.)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A real mode number</td>
<td>32</td>
<td>195</td>
</tr>
<tr>
<td></td>
<td><strong>NFMB</strong>: Unused</td>
<td>13</td>
<td>227</td>
</tr>
<tr>
<td></td>
<td>(This contains the number of bits needed to make the block end on a word boundary.)</td>
<td></td>
<td>14 words</td>
</tr>
<tr>
<td></td>
<td><strong>Begin Optional Fixed Block</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>NFMB+1</strong></td>
<td>4</td>
<td>240</td>
</tr>
<tr>
<td></td>
<td><strong>NFMB+NFOB</strong></td>
<td>12</td>
<td>340</td>
</tr>
<tr>
<td></td>
<td>(Unused)</td>
<td></td>
<td>7 words</td>
</tr>
<tr>
<td></td>
<td><strong>NFMB+NFOB+1</strong></td>
<td>--</td>
<td>352</td>
</tr>
<tr>
<td></td>
<td>Variable Field</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(Each subfield begins with a header containing the subfield number in the first byte and the length in words of the subfield in the second.)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>NFMB+NFOB</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Begin Block of Groups</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+2</td>
<td>Descriptor key 1</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>+3</td>
<td>Descriptor key 2</td>
<td>10</td>
<td>6</td>
</tr>
<tr>
<td>+4</td>
<td>Descriptor key 3</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>(Data fields)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Last data field in group</td>
<td>8</td>
<td>40</td>
</tr>
<tr>
<td>+NFGP</td>
<td>Dummy</td>
<td>0</td>
<td>48</td>
</tr>
</tbody>
</table>

*See abbreviations in Chapter II.

**These figures are examples for illustration only.
word boundary which means the expected number of spare bits is 8 (for 16-bit word machines) for the mandatory optional blocks. Each subfield in the variable field uses 16 bits of overhead. Also, since each is stored 3 characters per word there may be 0, 5, or 10 bits wasted for an expected value of 5. Groups end on word boundaries, also. However, to avoid wasted space various ploys such as lumping groups into super groups can be used so that only a bit or two per group is wasted. The space used for a full record is given by:

\[ S = (\text{Mandatory words} + \text{optional words} + \text{number of groups} \times \frac{\text{words group}}{\text{field words}} + \text{variable field words}) \times 16 \]

The expected amount of waste space will be (assuming 2 bits per group):

\[ W = 30 + 8 + 8 + (\text{number of subfields}) \times 21 + \text{number of groups} \times 2 + 16 \times \frac{\text{words per group}}{2} \]

The efficiency is

\[ E = \frac{S - W}{S} = 1 - \frac{W}{S} \]

For the sample in the appendix the number of words in the mandatory block is 14. In the optional block there are 7 words. The variable field might have an average length of 30 words with 3 out of 4 possible subfields. Twenty-five groups of 3 words each might be present. Then

\[ S = (14 + 7 + 25 \times 3 + 30) \times 16 = 2016 \]
\[ W = 30 + 8 + 8 + 3 \times 21 + 25 \times 2 + 16 \times \frac{3}{2} = 183 \]
\[ E = 1 - \frac{183}{2016} = 91\% \]

**Free Areas in the Master File.**

There are two types of free areas, those left by the moving or deletion of records and the end-of-file free area. The e-o-f area extends from the end of the last record to the end of the logical file. The words of th:
area are set to the EBCDIC code for '$/' except for the 8th and 9th words. Word 8 will be 32000 unless the size of the area is less than 32000 words. In that case word 8 gives the size of the area. Word 9 is 0. The parameter file contains a pointer to the beginning of this area. The other free areas also are set to '$/' except for words 8 and 9 in each area. Word 8 will be the size in words of the free area. Word 9 will be a pointer to the next free area. End of the chain of free areas is signified by a zero pointer. The parameter file contains a pointer to the first free area as well as a pointer to the e-o-f free area.

Category File

The records in this file are in linked 12 word blocks. The first word of each block is the pointer to the next block, with zero being the end-of-chain. The first block contains the description of the category in words 2-11. The succeeding blocks contain the pointers to records in that category. The category number is the record number of the header block of a record. The headers are stored sequentially. (See Figure 2.) The parameter file has a pointer to the free blocks available. The parameter file also shows which field in a master record is to determine the category.

Descriptor File

This file is similar to the category file with the exception that a more complex header block is given, and the records are 32 words long. The data in the header block is packed bit-wise if it is numeric. Alphanumeric data may be packed 2 or 3 characters per word. The first word of the header is the identification number of the descriptor. This may be a hash of other fields, as in the sample. The second 16 bits is the pointer to the beginning of the inverted list. Two fields are used for the count of the number of
<table>
<thead>
<tr>
<th></th>
<th>101</th>
<th>Category one</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>102</td>
<td>Category two</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>100</th>
<th>200</th>
<th>Last category</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>102</td>
<td>201</td>
<td>20</td>
</tr>
<tr>
<td>103</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note that category 002 has 2 blocks used in the inverted list.

Figure 2. Category file illustration.

<table>
<thead>
<tr>
<th></th>
<th>D1</th>
<th>500</th>
<th>Descriptor data</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>D2</td>
<td>501</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>D3</td>
<td>502</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>500</th>
<th>0</th>
<th>pointers to master records</th>
</tr>
</thead>
<tbody>
<tr>
<td>501</td>
<td>651</td>
<td></td>
</tr>
<tr>
<td>502</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>651</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3. Descriptor file illustration.
records allowed with this descriptor, if applicable. The rest of the header is the descriptor data. The blocks for the inverted list are as in the category file. The parameter file points to the beginning of the free blocks so that when an inverted list needs more space it can be allocated.

Parameter File

A concise way to explain the parameter file is to list the fields in it. Each field is one 16-bit word.

<table>
<thead>
<tr>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Pointer to e-o-f free area, sector and word*</td>
</tr>
<tr>
<td>2</td>
<td>Pointer to first free area, sector and word*</td>
</tr>
<tr>
<td>3</td>
<td>Number of sectors in logical master file</td>
</tr>
<tr>
<td>4</td>
<td>Number records in master file</td>
</tr>
<tr>
<td>5</td>
<td>Master file garbage collection flag</td>
</tr>
<tr>
<td>6</td>
<td>Descriptor chain flag, 1 = off, 2 = on</td>
</tr>
<tr>
<td>7</td>
<td>Divisor for master file hash</td>
</tr>
<tr>
<td>8</td>
<td>Divisor for master move index (GARBG)</td>
</tr>
<tr>
<td>9</td>
<td>Sectors in file 1 (drive 0)</td>
</tr>
<tr>
<td>10</td>
<td>Sectors in file 11 (drive 1)</td>
</tr>
<tr>
<td>11</td>
<td>Sectors in file 21 (drive 2)</td>
</tr>
<tr>
<td>12</td>
<td>Sectors in file 31 (drive 3)</td>
</tr>
<tr>
<td>13</td>
<td>Sectors in file 41 (drive 4)</td>
</tr>
<tr>
<td>14</td>
<td>No. words in mandatory fixed block</td>
</tr>
<tr>
<td>15</td>
<td>No. words in optional fixed block</td>
</tr>
<tr>
<td>16</td>
<td>No. words in each group, variable block</td>
</tr>
<tr>
<td>17</td>
<td>Initial size of record in words (mult. 20)</td>
</tr>
<tr>
<td>18</td>
<td>Record increment when enlarged (mult. 20)</td>
</tr>
<tr>
<td>19</td>
<td>No. fields in mand. fixed block</td>
</tr>
<tr>
<td>20</td>
<td>No. fields in optional fixed block</td>
</tr>
<tr>
<td>21</td>
<td>No. fields in groups in variable block</td>
</tr>
<tr>
<td>22</td>
<td>Size in records of descriptor file</td>
</tr>
<tr>
<td>23</td>
<td>Descriptor file garbage collection flag</td>
</tr>
</tbody>
</table>

*Pointer format is used: bits 0-11 for sector and 12-15 for word.
Word Description
24 Divisor for descriptor file hash
25 Number in descriptor file
26 Number of categories (faculty)
27 Size of category file in records
28 Field in master record that holds category and
    serves as flag for category chains: on if pos.,
    off if Ø or neg.
29 Number of possible subfields in variable field
    in master record
30 "Available" pointer for category file
31 Number of BA requirements
32 Number of BS requirements
33 Number of types of majors
34 Input device number
35 Output device number
36 Current date, qtr/year code
37 "Available" pointer for descriptor file
38 Number fields in descriptor records
39 Number words in descriptor records
40-50 Unused
51-79 Pointers to first bit of field for mandatory fixed
    block as shown in detail for master file, relative
    to beginning of record
80-96 Pointers for fields in optional fixed block
97 Pointer to bit at beginning of variable field
98-107 Pointers to bits at beginning of fields in groups
    in variable block, relative to beginning of group
108-125 Unused
126-165 Character table for use in A1A3, etc.
    (EBCDIC codes stored in A1 form)
166-200 Bit pointers for fields in descriptor records (not
    implemented yet)

Index File

The index file uses 5 words per record.

Words 1-3: key fields in records

Word 4: pointer to master record
Word 5: hash code composed from name field of master record  
(The name is presumably first subfield in variable field.)

The divisor used in the hash algorithm determines the effective size of  
the index. For systems with 320 words per disk sector, there will be 64 records  
per sector. STRIPE uses prime numbers for the divisors and sectors might  
as well be used as fully as possible. Therefore, the divisor should be chosen  
to be the largest prime less than 64 times the number of sectors to be used.  
For example, if we need about 1000 index entries we might as well allocate  
16 sectors, giving 1024 records. Then we will use a divisor of 1021, giving  
that number of effective records.

System Message File  
A set of 20 character messages used to briefly identify errors, give  
system information, direct user actions, etc.

Working Space File  
Disk space is provided for temporary storage used in garbage collection,  
selection of subsets of the master records, etc.

Any other files are special purpose. Those used in the sample system  
shown in the appendix are explained here since similar files would be likely  
to exist in many applications.

Requirements File  
In the sample application a student academic data base is kept. There  
are requirements for graduation such as the number of foreign language credits,  
and number of upper level courses. This file holds a description of these  
school requirements for both the B.A. and B.S. degrees.

The records are fixed, 60 words per record. The B.A. requirements start  
at record 1, the B.S. at record 21. The number of different requirements for
a B.A. and for a B.S. degree is in the parameter file. Each record is a string of up to 60 numbers. These describe a matrix. The string of numbers is of the form

\[ IR, IC, N_1, C_{1,1}, C_{2,1}, \ldots, C_{IR,1}, N_2, C_{1,2}, C_{2,2}, \ldots, C_{IR,2}, \ldots, N_{IC}, C_{1,IC}, C_{2,IC}, \ldots, C_{IR,IC}. \]

IR and IC give the number of rows and columns. Thus, the form of the matrix is:

\[
\begin{array}{ccc}
IR & IC \\
N_1 & N_2 & \ldots & \ldots & N_{IC} \\
C_{1,1} & C_{1,2} & C_{1,IC} \\
C_{2,1} & C_{2,2} & C_{2,IC} \\
\vdots & \vdots & \vdots \\
C_{IR,1} & C_{IR,2} & C_{IR,IC} \\
\end{array}
\]

The \( C_{ij} \)'s are department and course numbers of courses which satisfy the requirement. The \( N_j \) at the head of the column tells how many of the courses in the column are needed to satisfy the requirement. The requirement may be satisfied, then, by finding at least one column, \( j \), such that there are \( N_j \) courses in the master record which match the department and course of an entry in that column. The \( C_{ij} \)'s are of the form \( +ddccc \) where \( dd \) is a department number, \( 0<dd<31 \), and \( ccc \) is a course number, \( 0<ccc<767 \). If an entry is of the form \( dd999 \) then any course from department \( dd \) will do. This eliminates the bother of long lists of courses. If all but a few courses in department \( dd \) will do, we can have entries of the form \( -ddccc \), meaning this particular course must not be used in satisfying the requirement. The \( dd999 \) following would then accept all other courses.

This technique would seem to be a useful one to use in many cases where
31
records must be evaluated.

Requirement Title File
For output purposes, a file of titles of the requirements is kept.

Department Abbreviations and Grade Symbols File
This file has as its \( i^{th} \) entry the letter designation of department \( i \).
Also, the letters used in grade symbols are kept.

3. The Execution Model
The process of defining the files, determining system parameters, and initializing the files is described in the appendix. For the present we assume the preliminaries to be done.

Access to the system is through an executive program EXEC. Its purpose is to bring the system parameters into common core storage shared by all program units, to determine from the user what function he wants, and to link to that function. The common area of core is labelled as follows:

COMM - an array (165) containing the system parameters.

FLPTR - an array (4) used in passing pointers such as sector and word pointers for reading records or word and bit pointers needed to access fields in master records.

STURC - an array (340) used to hold the current master record being accessed.

IERFL - a word used as an error flag on return from various subroutines

BUFF - an array of length 41 or more used to pass data from subprograms, such as those for reading cards, to main programs. (In main program QUERY, this array is replaced by several scalars and arrays peculiar to the inquiry process.)

There are other elements in COMMON in certain special programs, but the
above is always at the beginning of COMMON.

EXEC links to the following basic system programs as well as other special purpose programs satisfying needs of a specific application.

ADCAT adds a new category. If the system parameters call for the maintaining of categories of master records, this unit must be executed before master records are added.

ADDSC adds a new descriptor for the groups that could be added to the variable block at the end of the master records. If the system parameter calls for the maintaining of lists of records with given descriptor groups, this must be run before any such descriptor groups are added to records.

In both ADCAT and ADDSC an initial block of space for inverted files is chained to the record. Subsequently, the inverted list is maintained by the subroutine INVRT. The inverted lists contain the sector/word pointers to records.

ADMAS adds a record to the master file. Only the mandatory block and variable subfield 1 is set up. The steps are:

1. Read from the category file the category numbers on file.
2. Make sure there is space in the master file for a record.
3. Call RMA  SC, a routine which reads master record cards and puts the information into BUFF. (RMA is a routine local to an application.)
4. If RMA indicates the end-of-data the program writes COMM back into disk storage and links back to EXEC. Otherwise go on to step 5.
5. HSHFL finds an index entry. If the key is a duplicate go to 3.
6. Next the chained free areas, pointed to by parameter 2 of COMM, are examined until a free area big enough for a record is found. If one is found greater than or equal to the standard initial length, but too small for two
33

records, the whole area is used for the new record. If no chained area exists
or is big enough, then the record is put into the end of file area pointed
to by COMM(1), which is the pointer to the e-o-f free area.

7. If COMM(28) is positive its value is taken as the field in the master
record which determines the category. The pointer to the record is entered
into that category's inverted list by subroutine INVRT. If COMM(28) is zero
go on to step 8. If INVRT indicates an error such as no room in the cate­
gory's list or nonexistent category, ignore this record and go to step 3.

8. The fields of the master record are packed from BUFF into STURC by
routine IPFLD. Only the mandatory block is filled in.

9. The first subfield - probably a name - is put into STURC.

10. Housekeeping details are filled in, such as length of variable field,
and zeroing out unused parts of the record.

11. Put the index entry in. Increase the count of the number of master
records.

12. Write the record into the master file using routine WMASR.

13. Update the free area pointers.

14. Repeat from step 3.

ADVBL adds groups to the variable block at the end of the records. The
steps of the program are:

1. The routine RVBLC is called to return a prime key, or identification
number, of a master record together with a group of fields to be appended to
the record. (RVBLC is a local routine.)

2. If RVBLC indicates end-of-data the last master record accessed, if
any, is returned to storage. Then the parameter file is returned to disk
storage and a link to EXEC is executed.

3. On the first call to RVBLC the record corresponding to the prime
key is brought into core by routine RMASR.

4. On succeeding calls to RVBLC, if the key is different from the one returned on the last call, the last record is returned to disk storage, then the record for the new key is brought into core by RMASR.

5. Check the physical length against the active length of the record. If there is not enough space in the record for a new group, the routine RECMV is called. (It will move the record to the end-of-file area, enlarge it, update parameters and pointers, and return to the calling program.)

6. The field number for the first field of the group is calculated.

7. If the parameter file indicates that descriptor inverted lists are to be maintained this is done by calling HSHFl to access the descriptor and INVRT to add the record pointer to the list.

8. If all this has worked without error, the group is added to the in-core record. Then go back to step 1.

GPD RP deletes a group from the variable block of a record. This program operates as follows:

1. The descriptor file hash keys are read into an index in core. The field pointer to the beginning of the variable block of groups is calculated.

2. A card is read giving the key of a master record and the basic key of a descriptor. If end-of-data is encountered a link back to EXEC is performed. (Alternatively, there might be a card with a master key followed by cards with descriptor keys indicating groups to be dropped.)

3. The master record is accessed through HSHF2. The number of groups in that record is found and used to determine the field number of the last group.

4. The descriptor key of the record is examined. If it does not match the card data the group drops back by one group. If there is another group to
be checked, repeat this step. If there are no more groups an error message is given and control goes back to step 2. Go to step 5 if there is a match.

5. If the groups being dropped represent something like courses in a student record a date field in the group may be checked to be sure it matches the current date in the parameter file, since a normal drop of a course in a previous quarter would not be allowed.

6. The number of groups is decreased by one. The groups are repacked, wiping out the dropped group. The record is returned to the disk file.

7. If the parameter file indicates that descriptor lists are being kept, then the count is reduced by one in the descriptor record and the pointer to the master record is zeroed out of the descriptor's inverted list.

8. Go back to step 2.

DROP deletes a record from the master file.

1. A card is read with the key of a master record. If end-of-data is encountered a link to EXEC is performed.

2. The master record is accessed through HSHF2. The pointer to that record is saved for returning the area to the free chain.

3. The index file entry is emptied. The pointer word of the entry is set negative rather than zero. This is necessary to keep the continuity of the collision chains in the hash table.

4. If the parameter file indicates that categories of master records are maintained, this record's pointer is replaced by a zero in the category's inverted list.

5. If the parameter file indicates that descriptor lists are kept, then each group in the record is examined and its entry in the associated descriptor list is zeroed.
6. Finally, the record area is entered into the free area chain. The record length is saved. The record area is filled with '/$/'. The length of the area is put into word 8. The current parameter file pointer to the free chain is put into word 9. The pointer to this area is put into the parameter file. The newly formed free area is written into the disk file. Go back to step 1.

GARBG compresses the master file so that all free areas are combined into one end-of-file free area.

1. A temporary file for storing an index to the changes in addresses is initialized. A core array to be used for pointers to the free areas is initialized.

2. Pointers to the free areas, including the end-of-file area, are put into the core array. A count of the number of free areas is kept.

3. The array of free area pointers is sorted into ascending order. The moving up of records starts at the first free area.

4. The pointer to the master record following this free area is calculated. The record is read and then written at the beginning of the free area.

5. An entry in the move index is made using hash coding on the old pointer. Both the old pointer and the new one are entered.

6. A new free area pointer is calculated. The pointer to the next master record is calculated. A check is made to see if that pointer is up to the next free area. If so, the pointer to the next master record is recalculated.

7. Each time another free area is "passed" in packing the file the count of free areas is decreased. When it reaches zero the end-of-file free area has been reached.
8. The parameter file entries pointing to free areas are updated. The end-of-file free area is initialized.

9. The pointers in the inverted lists of categories and descriptors are updated using the hash function on the move index file. The master record index is similarly updated.

10. The parameter file is returned to disk and a link to EXEC is performed.

QUERY allows four operations of inquiry into the master file. The first operation is selection of a subset of the master file by using the command SELECT. Pointers to the selected records are stored in a disk file working set. Selections may be made by specifying a criterion expression whose BNF description is as follows:

<crit exp> ::= <crit> | <crit> AND <crit exp> | <crit> OR <crit exp>
<crit> ::= <field><relation><value exp> | NOT <crit>
[field] ::= F<num>
<num> ::= a string of digits
<relation> ::= LT | LE | EQ | GE | FT | EX
/value exp> ::= a numeric value | <field exp> | '<string>'
<string> ::= any characters
[field exp] ::= <field> | (<field><op><field>) | (<field><op><num>) | (<num><op><field>) | <num>
<op> ::= + | - | * | /

The meaning of the above relations are mostly obvious. EX means that the given field exists.

The important semantics have to do with determining the operands of NOT, AND, and OR. The order of operation is strictly right to left. This means that the left operand of the verbs AND and OR is the criterion immediately to
the left of the verb. (There is no left operand for NOT.) The right hand operand for any of these verbs is the result of the entire expression to the right of the verb. Of course, the value of any criterion such as F10 LT250 is either 1 or 0. Blanks embedded within the three elements of the expression are not allowed. Blanks between them are permissible.

The right to left order is used for two reasons. First, the programming is greatly simplified since a priority table does not have to be consulted and stacking of operands is avoided. Second, the user of STRIPE is apt to be unfamiliar with conventions of priority with respect to Boolean operators. With this arrangement only one idea must be understood.

Refinement of a selection can be made by using the command REFINE and giving criteria as described above. After a selection or refinement the number of records selected is given on the console.

The selected set can be sorted into ascending order by using the command ARRANGE. Fields are specified as in the selection: F<num>. The fields may or may not be separated with commas or blanks. Up to 10 sort fields may be specified. The sort significance goes from left to right. The result of this command is a working set of pointers sorted into the desired order.

The currently selected records may be listed on the printer specified in the parameter file by using the command LIST. The whole record may be listed by specifying ALL. Specific fields of the record may be listed by specifying up to 10 numeric fields (to be locally implemented).

The last function in QUERY is to give some simple statistics concerning fields of the selected records by using the command, STATS. One, two or three fields may be specified. For each one the sum, average, standard deviation, maximum and minimum is given.

One other command is used, EXIT. It performs a link back to EXEC.
The specific steps in executing LIST, STATS or EXIT are relatively trivial and will be passed over here. Execution of REFINE is virtually the same as SELECT except that only records already in the working set are examined. Therefore, the attention is on the more interesting aspects of SELECT and ARRANGE.

SELECT

The program which performs the SELECT command is called SLECT. For space reasons SLECT is a main program linked to by QUERY. Its first job is to process the input of selection criteria. As an example of a criterion we could have

\[ \text{FIOLT (F16 - F11) AND F5 GE 235.6} \]

In this criterion expression there are two criteria and a Boolean verb. SLECT will find the first criterion, FIOLT(F16-F11). A coded version of it will be placed in an array, CRIT, which is in common core storage. The coded version contains the field number, the relationship, the type of value, and a pointer to the value array where that value is stored. The criterion is coded into one word as follows:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-5</td>
<td>Field number</td>
</tr>
<tr>
<td>6-8</td>
<td>Relation</td>
</tr>
<tr>
<td>9-10</td>
<td>Type of value</td>
</tr>
<tr>
<td>11-15</td>
<td>Pointer into value array</td>
</tr>
</tbody>
</table>

The codes for relations are:

<table>
<thead>
<tr>
<th>Code</th>
<th>Relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Less than: LT</td>
</tr>
<tr>
<td>2</td>
<td>Less or equal : LE</td>
</tr>
<tr>
<td>Code</td>
<td>Relation</td>
</tr>
<tr>
<td>------</td>
<td>----------------</td>
</tr>
<tr>
<td>3</td>
<td>Equal : EQ</td>
</tr>
<tr>
<td>4</td>
<td>Greater or equal : GE</td>
</tr>
<tr>
<td>5</td>
<td>Greater than : GT</td>
</tr>
<tr>
<td>6</td>
<td>Exists : EX</td>
</tr>
</tbody>
</table>

The codes for value types are:

<table>
<thead>
<tr>
<th>Code</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Integer number</td>
</tr>
<tr>
<td>1</td>
<td>Real mode number</td>
</tr>
<tr>
<td>2</td>
<td>Field expression</td>
</tr>
<tr>
<td>3</td>
<td>String</td>
</tr>
</tbody>
</table>

The pointer to the value array gives the position in an array, IVAL, or equivalently, VAL*, in which the actual numeric value, string value or field expression is stored.

In storing values all numeric values are stored in VAL in real mode. The pointer will be to the equivalent location in IVAL. String values will be stored in IVAL. The first location for the string will contain the length of the string. Field expressions will always be stored as a sequence of pairs: (operation, field) or (operation, number). The first field or number in the expression is assumed to have a + applied to it unless a - is used. By this means of storage evaluation can be done by "accumulator" technique. An accumulator is initialized to 0. The first value is added or subtracted. The succeeding (op, val) pairs indicate an operation to be performed with the accumulator as the first operand and the value as the second. The result, of course, is left in the accumulator. The operation codes are 1, 2, 3, or 4 for add, subtract, multiply, or divide. A negative operation

*By use of EQUIVALENCE, VAL and IVAL occupy the same space.
code means the following element is a number rather than a field. When
this field expression is evaluated it will be strictly left-to-right.

Now, back to the main stream of interpreting the criterion. In the above
eexample

F1OLT (F16-F11)

would be coded into CRIT with the code for F16-F11 being entered into IVAL.
A pointer to this entry in CRIT would be placed in an expression array XP.
Then the AND would be detected. A numeric code for this would be placed in
XP, following the pointer to the first criterion in CRIT. To distinguish
between criteria pointers and Boolean codes, the Boolean verbs are coded
as negative numbers. The codes are:

<table>
<thead>
<tr>
<th>Code</th>
<th>Verb</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(</td>
</tr>
<tr>
<td>-1</td>
<td>)</td>
</tr>
<tr>
<td>-2</td>
<td>AND</td>
</tr>
<tr>
<td>-3</td>
<td>OR</td>
</tr>
<tr>
<td>-4</td>
<td>NOT</td>
</tr>
</tbody>
</table>

(Parenthesized Boolean expressions are allowed.) Next in the example the
criterion F5 GE 235.6 is entered in CRIT with the value 235.6 in VAL. The
pointer to CRIT is entered in XP.

When it is determined that there are no more criteria, a right parenthesis
(code, -1) is placed at the end of the coded expression in XP. (A left
parenthesis was initially placed at the beginning.) Now the expression must
be sorted into an appropriate order for execution. The order chosen for
evaluating the Boolean expression is strictly right-to-left.

The sorting process is as follows. The array XP, containing the coded
Boolean expression, such as
Symbols \((\text{Crit(1) AND Crit(2)})\)  

<table>
<thead>
<tr>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>-2</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>-1</td>
</tr>
</tbody>
</table>

is scanned to find the point where the left parenthesis is balanced, in the above case at the fifth element. Subroutine POLST is called which sorts the elements between the parentheses so that positive numbers (which are operands, or criteria pointers) come first from left-to-right in their same relative order, followed by the negative numbers in reversed order. Then the outer parentheses are erased by replacing them with -99's. The above would become 

-99,1,2,-2,-99

In this process parenthesized inner groups are left intact and treated as a single criterion, being sorted as are the single positive numbers.  

\[\text{e.g. } 0,1,-2,0,2,-3,3,-1,-1\]

becomes 

-99,1,0,2,-3,3,-1,-2,-99

POLST would be called again and this example would become 

-99,1,-99,2,3,-3,-99,-2,-99

The -99's will be ignored, so this is equivalent to 

1,2,3,-3,-2

The meaning of this is: 

\[\text{R = Criterion 2 OR Criterion 3 (2,3,-3)}\]

\[\text{LV = Criterion 1 AND R (1,R,-2)}\]

\text{LV would be the resultant value, assuming that the criteria 1,2,3 all are evaluated to 0 or 1.}

This process of repeatedly using POLST on XP until all parentheses have been resolved sorts XP into a reverse Polish string compatible with the strict right-to-left order of operations.

Finally, in the selection process the criteria must be evaluated against
a specific record. If the resultant value is one, the pointer to the record is saved in the working set. In evaluating a specific criterion as coded in CRIT, a function IRLVL is called. The process consists of making a copy of XP. This copy is gradually reduced until finally there is just a 0 or 1 left.

The process of placing or retrieving pointers in the working set is done through a subroutine IRTRV. This routine has internal buffers in which are images of blocks of the working set. This reduces the time needed in accessing the disk. Without the buffers the disk arm would be moving constantly between the master file and the work file.

ARRNG

Called as a subroutine from QUERY, it is the name of the program which performs the ARRANGE command. ARRNG first determines which fields are to be used as sort criteria. The fields are specified by the Fxxx form shown previously. As many fields as can be given with 20 characters may be specified. The fields may be run together or separated by blanks or commas. The fields are taken to be given in left-to-right order of significance. Then the actual sorting algorithm is entered to examine the records pointed at by the work set and rearrange the pointers into the desired order.

The choice of algorithm for this sorting is critical. There is no way to anticipate any inherent order already existing. Since the number of records to be sorted may be quite large, it cannot be done in core, even as a key sort. Bringing the necessary keys into the working set along with the pointers was rejected since to allow space for up to 10 keys in the working set would be costly in space used. To bring in one key at a time and do a separate sort on each in a radix sorting process would be prohibitively
time consuming for cases with several keys. Clearly an algorithm which allows the use of multiple keys and minimizes comparisons must be used. The two tried first are the "Shell" sort, or diminishing increment algorithm and the "Quicksort," or partition exchange algorithm (Knuth (7)).

As an example of the Shell process, a sort of 100 items took twenty minutes with increments of 63,31,15,7,3,1 and accessing the working set one pointer at a time. With a small buffer in the accessing of the working set this dropped to fifteen minutes. Changing the increments to 40,13,4,1 brought the time down to 10.5 minutes due to making two fewer passes. Drastically increasing the size of the buffer used in accessing the working set reduced the time to five minutes.

The same set of items was sorted using the "Quicksort" algorithm. The time was virtually the same. It became clear that while Quicksort required fewer interchanges it was making more accesses of the master records. The saving of time on the one hand was cancelled out on the other.

More effort on reducing the number of accesses of the master records is needed. Since the primary goal of this project was to improve storage efficiency, the improvement of the sorting algorithm will be deferred. The currently used methods seem to be a reasonable compromise between speed and storage space. All other methods would require significantly more space (except a "bubble" sort which wouldn't be considered).

MODRC

This program allows modification of master records in batches. Any of the four general parts of a given record may be changed. If the change involves expanding the size of the record, it may be moved. A code is initially given for each record to indicate which part of the record is to be
changed. This code is provided by the locally written routine, RDMDC, which reads the modification card.

**UTLTY**

A variety of subroutines are available to UTLTY for investigating all of the files in the system. They can list and/or patch any part of any file. This is used primarily for debugging purposes. However, it may also be used for data listings.
CHAPTER IV

CONCEPTUAL DESCRIPTION OF SCRIPT

1. Hardware

The hardware used for SCRIPT consists of the IBM 1130 with 16K bytes of core memory and single disk, a Logicon I/O Network (LI/ON) which allows direct access to main memory via cycle stealing, and one or more terminals. Teletypes were used. However, almost any type of terminal could be used with a different interface.

The relationship of the hardware is very simple as shown in this diagram.

```
Terminal <-> LI/ON <-> CPU
```

The terminal communicates directly with the LI/ON interface when its line adapter is input-enabled. The LI/ON in turn puts the character received directly into a buffer in core reserved for that line. On receipt of certain special characters, such as BREAK or LINE FEED, an interrupt is issued to the CPU (indicated by the dashed line). At that point software takes over.

The CPU can send characters to the LI/ON and thence to a terminal when that terminal's line is output-enabled. Upon transmission of an end-of-transmission code, the LI/ON issues an interrupt to the CPU.

The buffers associated with each line are used for both input and output. By analyzing the characters input from the terminal, putting the appropriate response characters in the buffer, and synchronizing these actions through the interrupt system, SCRIPT can communicate with the user at the terminal.
2. Software - An Overview

The beginning of the chain of programs which comprise the SCRIPT system is the interrupt level subroutine, ILS03. Upon occurrence of the level interrupt associated with the LI/ON a hardware generated branch is executed to ILS03. When that routine determines that the interrupt came from the LI/ON it branches to a multiple use routine, LMCC. LMCC contains routines of a utility nature such as initiating read or write to a given terminal, test whether a line is busy, mask and unmask interrupts, etc. LMCC also contains the routine to respond to the interrupts.

ILS03 and LMCC were provided by LOGICON. They were modified some what for this system.

When LMCC has determined the cause of the interrupt it branches to USER, passing the interrupt data along. USER puts the interrupt data into a common area where it can be available for the FORTRAN programs. This data includes the number of the terminal causing the interrupt, the cause of the interrupt, and the character count (if applicable). USER then branches to the routines BREAK and TERM to set the interrupt flags.

The main routine servicing the interrupts is SERV, which services interrupts due to the reception or transmission of an end-of-transmission character (EOT). If the EOT character came on a transmission, SERV responds by either setting up another message to the terminal if certain indicators warrant it or setting up a return to the main program. If the interrupt occurs on reception of characters from the terminal, the characters are analyzed and appropriate action is taken such as calling MODUL to send a response to the terminal.

The function of the main program is primarily to poll the terminals. In turn each terminal is scanned by checking a word in its assigned partition to see if it is active. If not, a routine is called to see if the terminal
has been turned on. If it has, an appropriate message is sent, the partition 
is set to indicate it is active and the line is left in receive mode.

The LI/ON has a clock that counts each three seconds. This can be 
read by the program. At the start of each cycle of polling the clock is 
read. For each active terminal the time since the last communication with 
that terminal is checked. If it is too long a warning message is sent. If 
the terminal still doesn't respond after a reasonable time, it is hung up.

Figure 4 illustrates some of the relationships of the hardware and the 
various elements of the software.

The dashed lines between the LI/ON and the CPU indicates the interrupt 
signal. The dotted line to ILS03 signifies the hardware generated branch 
to the interrupt level subroutine when the LI/ON has issued the interrupt. 
The wavy lines indicate the sending of messages to terminals via the LI/ON.

An explanation of the process by which the program units can determine 
the appropriate response to input from the terminal requires an explanation 
of the data structure. Thus, the next topic should be the structure of the 
data storage.

3. Data Storage

There are four files of primary concern to the operation of SCRIPT. The 
first is the message file. All messages for all lesson modules are stored 
here. They can be referenced by their sequential record number. The second 
is the file of decision tables. These provide the logic governing the 
responses to inputs from the terminals. A full description is given below. 
The third file is an index to the modules in the system. The last is a 
system parameter file.
Figure 4. Relationships of hardware and software units in SCRIPT.
Message file

When a lesson is added to the system the set of messages is stored. During execution of a module these can be retrieved and sent to terminals as directed by the logic of the module's decision table.

The messages are stored in a form such that they are ready for immediate transmission to a terminal as soon as they are transferred to the buffer for that terminal. This means that the characters have been converted to the appropriate code for the terminal, such as ASCII for teletypes. Also, they have been packed two characters per word, put into the order required by the hardware, and had characters for line feed, carriage return and EOT added.

The program unit which accesses and transmits these messages is MODUL - discussed later.

Decision tables

A decision table (or state/action table) uses a current state (initially 1) and an input to determine the current action (message to be transmitted) and the next state.

<table>
<thead>
<tr>
<th>Input State</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NS = 2</td>
<td>NS = 2</td>
<td>NS = 1</td>
</tr>
<tr>
<td></td>
<td>MESS = 1,5</td>
<td>MESS = 6,8,2</td>
<td>MESS = 7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input State</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>NS = 1</td>
<td>NS = 3</td>
<td>NS = 3</td>
</tr>
<tr>
<td></td>
<td>MESS = 3</td>
<td>MESS = 2,9</td>
<td>MESS = 4</td>
</tr>
</tbody>
</table>

Figure 5. Example of a state/action table, simplified.
In this example, with an initial state of 1, an input of 1 would cause messages 1 and 5 to be transmitted, and the next state would be 2. An input of 2 would then bring up messages 2 and 9, and the next state would be 3, and so on.

The actual storage of the decision tables is in two parts. There is a set of pointers to messages which is called a message vector, and there is the decision table itself. The preceding example would have the form:

**Message Vector**

<table>
<thead>
<tr>
<th>Position</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message pointer</td>
<td>1</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>2</td>
<td>7</td>
<td>3</td>
<td>2</td>
<td>9</td>
<td>4</td>
<td>12</td>
<td>11</td>
<td>10</td>
</tr>
</tbody>
</table>

**Decision Table**

<table>
<thead>
<tr>
<th>Input</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2,2,1</td>
<td>2,3,3</td>
<td>1,1,6</td>
<td>NS,NCM,VPTR</td>
</tr>
<tr>
<td>2</td>
<td>1,1,7</td>
<td>3,2,8</td>
<td>3,1,10</td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>1,1,11</td>
<td>2,1,12</td>
<td>3,1,13</td>
<td></td>
</tr>
</tbody>
</table>

Figure 6. State/action table with message vector.

The entries in the table are of the form (NS,NCM,VPTR). NS = next state. NCM - number of consecutive messages. VPTR = pointer to the position in the message vector where the pointer to the first message is found. If NCM > 1, then the next message pointer is in position VPTR + 1 and so on. In the
example, if the state = 2 and input = 1, then the next state will be 2; there
will be 3 messages transmitted; and the first message pointer is in position 3
of the message vector. We see that messages 6, 8, and 2 should be sent.

The bottom line of the table is labeled H. It enables the student to
receive help during the lesson. If he types $H$ the action indicated in this
entry will be used. The state remains the same.

If there are $n$ rows in the table for a lesson, the student will be able
to respond with answers 1 through $n-1$. Other digits will be rejected.

The three components of each entry in the table are packed into one
word. The numbers of bits allocated to each are such that

\[ 0 \leq NS \leq 31, \quad 0 \leq NCM \leq 7, \quad 0 \leq VPTR \leq 127 \]

The message vector and the table are stored as one long string of numbers.
The message vector goes first so that the values in VPTR will be minimized.
As it is the message vector may not be pointed at with VPTR > 127. The
message vector could be as long as 134 positions since VPTR = 127 and NCM = 7
would be permissible.

Index file

To access a particular module the position in the message file where
its messages start and the position in the decision table file where its
message vector and table begin must be known. This is kept in the index file.
Also the length of the message vector, and the number of rows and columns in
the table are kept. The entry for the module with identification $i$ is kept in
record $i$.

System parameters

The primary system parameters are:
1. The time increment used to determine when a non-responding terminal should be warned or cut off.
2. Number of entries in the index.
3. Pointer to the end of the decision table file.
4. Pointer to the end of the message file.
5. Physical size of the decision table file.

These items allow new lessons to be added and terminals to be controlled. This data is kept in a file on disk.

In-core data

The other part of the data structure is the main memory allocated to each terminal. The input/output buffer for each line already has been referred to. From the viewpoint of the LI/ON the first word of each buffer is that one with the lowest address. The first word is used by the hardware to indicate a character count and for flags indicating the status of the terminal. Each terminal also is allocated an area of core that is referred to as a terminal partition. It is a small allocation of 15 words. Here the data describing the current status of the terminal is found. The information consists of:

1. Bit flags to indicate active or inactive, teach or calculate mode, or whether a time warning has been given.
2. The time limit for next communication.
3. The number of messages that should be sent to the terminal.
4. The pointer to the next position in the message vector.
5. A pointer to the beginning of the decision table for this terminal.
6. A pointer to the beginning of the messages on disk for this module.
7. The current state in the lesson for this terminal (as applied to the decision table).
The eight other words are reserved to meet future needs of the instructor.

4. **Ease of Use**

As mentioned in the introduction SCRIPT can be used with great ease. The forms in Figures 7 and 8 may be used. The three parts of the forms are:

1) the decision table, 2) the message vector, and 3) the list of messages.

The decision table contains the logic which enables the system to determine what messages to transmit after a given input. The rows of the table correspond to the inputs. The columns correspond to states. Each entry in the table will specify the next state in the lesson which the user will be in. Each entry in the table also contains message information in two parameters. One is the number of consecutive messages to be sent. The other is a pointer into the message vector where the actual message numbers are kept.

The reason for this arrangement for indicating messages is to enable the decision table to be composed of single word entries.

To write a lesson for SCRIPT the instructor creates a script, or a series of groups of messages. Each group of messages should include a question requiring a multiple choice response. The individual messages are numbered in the message list. For each group of n messages to be sent a string of n consecutive message numbers is entered in the message vector. The decision table is filled in by indicating for each state and input what the next state (NS) should be, how many consecutive messages (NCM) should be sent, and where in the message vector those message pointers start (VPTR).

The initial state must be noted. The initial state of the lesson is 1. All the entries for state 1 should indicate the initial messages to be sent. (See appendix B for examples.)
The use of the last row of the table is special. This means that for normal input the choices should be from 1 to at most 8. The table should have one more row than the largest number of possible inputs in any one question. (Of course, the number of columns is the number of states.) The bottom row is used for extra help. In any state the entry in the bottom row should give help messages to be given to the student when he types $\text{SH}$.

As an example, look at the illustration in Figures 7 and 8. The initial state of 1 has entries for inputs 1 and 2 of (2,2,1) for (NS,NCM,VPTR). Only inputs of 1 or 2 should be made. (The system will reject 3-9.) For either input there will be 2 messages sent, starting with the number one entry in the message vector. The message vector shows that the messages to be sent are 1 and 2. From the message list they are:

"ZWEI UND ZWEI IST FUNF, NICHT WAHR?
1)JA 2) NEIN"

If the student types $\text{SH}$ he will remain in the same state and get the single message

"NO HELP NEEDED".

However, when he responds say, 1, meaning JA, the table entry is (3,1,3). The single message has its pointer at position 3 in the message vector. That gives message 6, VERSUCHE BITTE WIEDER. He is then in state 3. A response of 2 for NEIN will give him 3 messages whose pointers start at position 4 in the vector. They are messages 3,9, and 2.

The lesson may come to a logical end or keep cycling back through previous states.
### SCRIPT Decision Table

<table>
<thead>
<tr>
<th>States</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Legend

<table>
<thead>
<tr>
<th>MS</th>
<th>NCM</th>
<th>VPIR</th>
</tr>
</thead>
</table>

For tables with more than 15 states attach a second form to the right edge of this one.

### SCRIPT Message Vector

<table>
<thead>
<tr>
<th>1, 2, 6, 3, 9, 2, 7, 4, 5, 10, 8, 11, 12, 13, 14, 15</th>
</tr>
</thead>
</table>

Figure 7. SCRIPT decision table and message vector form.
### SCRIPT Message Form

<table>
<thead>
<tr>
<th>MESS. NO.</th>
<th>MESSAGES (less than 61 characters total)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ZWEI: UND ZWEI IST FÜNF: NICHT WÄHRE?</td>
</tr>
<tr>
<td>2</td>
<td>1) JA 2) NEIN</td>
</tr>
<tr>
<td>3</td>
<td>DU HAST RECHT</td>
</tr>
<tr>
<td>4</td>
<td>DIESE IST VIER.</td>
</tr>
<tr>
<td>5</td>
<td>DIESE IST FÜNF.</td>
</tr>
<tr>
<td>6</td>
<td>VERSCHE BITTE WIEDER.</td>
</tr>
<tr>
<td>7</td>
<td>ACH: DU LIEBER, LERNE ZAHLEN.</td>
</tr>
<tr>
<td>8</td>
<td>DAS IST ALLES.</td>
</tr>
<tr>
<td>9</td>
<td>DO YOU WANT THE NEXT LESSON?</td>
</tr>
<tr>
<td>10</td>
<td>THEN REQUEST LESSON G2.</td>
</tr>
<tr>
<td>11</td>
<td>NO HELP NEEDED.</td>
</tr>
</tbody>
</table>

**Figure 8.** SCRIPT message list form.
CHAP TER V

FUNCTIONAL DESCRIPTION OF SCRIPT

1. Program Units

There are program units which initialize the files, add new lesson modules, and perform other housekeeping functions. These are not very interesting and will be left to the last.

POLL

This is the main program—main in the sense that it is the vehicle for calling other units into play. POLL does not hold much of the logic. Its primary functions are to determine what lesson the users desire, sense what terminals are ready, set the lines of the ready terminals to receive, time the users to warn them if there is no communication, and to set terminal partition parameters initially. POLL has one other valuable function, providing a place to branch from and return to when interrupts occur.

All the other program units are subroutines or functions called into action by other program units.

ILSØ3

This is the interrupt level subroutine provided by LOGICON to go with the LI/ON. It was modified to mask out interrupts from the terminals while an interrupt is being processed.

LMCC

When ILSØ3 determines that an interrupt has come from the LI/ON it
branches to the interrupt servicing portion of LMCC. LMCC sets flags giving
the cause of the interrupt and passes them on to the program USER.

LMCC also involves sections used by other assembler written routines
which are FORTRAN callable to receive or transmit messages, test for busy,
mask interrupts, etc.

LMCC was modified to provide one extra function, that of turning off a
busy flag for a particular line.

USER

When branched to from LMCC, this assembler routine puts the information
about the interrupt into COMMON, then determines the cause of the interrupt,
and branches to the particular routine needed. This and the following
routines are not from LOGICON.

TERM & SERV

These routines are used if an interrupt occurs due to the reception
or transmission of an end-of-transmission signal. (This is a line feed in
messages coming from the terminal. It is an ASCII EOT-numeric value, 4- in
transmissions from the computer.)

TERM does the immediate servicing to handle the interrupt. The user's
time limit is updated if the line is in receive mode and a transaction stack
is set. In transmit mode the transaction flag indicator is flagged. SERV
does the actual logic processing of interrupts that are stacked. It does
so when the interrupt state is past.

If the EOT came in transmit mode SERV checks parameter 3 in the partition
for the line associated with the interrupt. That will indicate whether there
are further messages to be sent. If so, it branches to MODUL (see below) and
the return from MODUL sets a flag to indicate transmission is needed. If
parameter 3 indicates no further messages are pending a flag is set to put the line in receive mode. In any case a return to USER, LMCC, ILS03, and finally back to the main program is made.

If the EOT comes when the line is in receive mode, an analysis of the input must be made. First, a check is made to see if the user is in instruction mode or calculate mode. If the latter, then a branch to CALC is made. If in instruction mode, then the acceptable inputs are single digits indicating a multiple response or a control command of the form $c$. The c may be one of the letters E,H,C,R,S, or L meaning exit from the module, give some extra help, go into calculate mode, return to instruction mode, save his partition, or load his partition from disk, respectively. Only the first several characters are scanned; if all are blanks, a retype is requested.

If the first non-blank character is neither a digit nor $, the user is asked to retype his response. If the character is $, the next character is examined. If that character is E then the line is deactivated. If it is H, then MODUL is called with an argument of zero. This will signify MODUL to take the action given in the bottom row of the lesson decision table for the current state of the user. A C after the $ will cause the mode to be set to calculate. An R will set the mode flag back to instruction mode.

An $S will cause the terminal partition to be stored on disk. The mode and state of the lesson will remain the same, and then the lesson continues. The saving is done under the user's special identification number. There is no attempt at security to prevent users from using the wrong number. Only one partition will be saved under a given number. An $L will cause the partition for that user to be loaded from disk. Upon completion the mode and state of the lesson as it applied to that terminal will be the same as when the partition was saved. However, if the lesson module currently in
core is not the same one the user will be notified.

If a digit is the first non-blank character its value is passed to MODUL. Upon return a flag is set indicating that transmission of the message set up by MODUL is needed.

In any case when TERM is entered the time parameter in the partition is updated to indicate the next time a time warning would be given.

**BREAK**

BREAK can be used to interrupt transmission or reception of messages or to gain the system's attention when the terminal is turned on. When first entered, the routine sets the time parameter, sets the line not busy, and then determines whether the line was already active or not. If already active a flag is set to put the line in receive mode, and it returns. If the line has not been previously active, the partition active flag is set, a system message is set up - such as READY. TYPE #ID--, a flag is set to indicate transmission is needed, and it returns.

**MODUL**

This unit is given the number of the input response. This together with current state determines an entry in the decision table. The entry is broken down to produce the next state, the number of consecutive messages to send, and the pointer into the message vector where the message pointers start.

Then the first message is retrieved and put into the appropriate terminal buffer. If the number of messages is greater than 1, the next one is retrieved and inserted in the buffer. Idle characters are inserted between messages to overlay the first EOT and provide time for a carriage return at the terminal. The message vector pointer is increased by one and the count
of the number of consecutive messages to send is decreased by one each time a message is loaded into the buffer.

At the entry to MODUL the message count is checked and, if it is positive, the next message is sent. In this case there is no change of state or access of the decision table.

MESG

This little routine simply loads a system message into the buffer for a terminal. It sets the busy flag off, and then initiates transmission of the message.

CALC

At present CALC is a very rudimentary program to analyze an expression consisting of a function and argument or two numbers with an infix operator. The operators used are +,-,*,//,and**. The functions are LOG, SIN, COS, and EXP. Arguments and operands are simple numbers. No variables are used. When an expression is evaluated its result is transmitted. The purpose of the routine is only to allow simple calculation to help in giving answers to questions in the lesson modules.

PSAVE and PLOAD

These routines are used by TERM when the special commands $S and $L are given. PSAVE will store certain pointers from the terminal partition in the disk file for partitions. They will be stored in the record corresponding to the identification number under which the user signed on. PLOAD reverses the process.

By using $S and later $L the user can leave in the middle of a lesson and come back later to continue where he left off.
When new lessons are to be added to those already on disk, this stand-alone program reads the input, updates the lesson index, packs the data into its stored form, and enters it on the disk.

The data to be provided should be in the following form.

1. Message vector card(s): The format is 20I4. A field of zero (or blanks) serves as the end-of-vector signal. If the vector is exactly a multiple of 20 in length, then an extra blank card should be added so there will be a zero field read.

2. Dimension card: The number of rows and columns of the state/action table is given. Format is 2I4.

3. Decision table cards: The entries for the decision table are punched one to a card. For each entry NS, NCM, and VPTR are punched in 3I4 format. The cards are put in column order. The number of entries is the product of the numbers of rows and columns.

4. Message cards: Each message is punched on one card with the message number in columns 1-3 and the message in columns 5-65. A card with message number of zero signals end of the set of messages.

The decision table entries are packed into one word and entered into a disk file. The messages are translated into ASCII, packed two characters per word, and entered into the disk file. Both the decision table entries and the messages are filed according to a displacement factor obtained from the system parameter file. These displacements are put into an index entry for the lesson module. The displacement factors in the system parameters are updated to point to the end of the last entries.
INIT

This program simply initializes the system files prior to adding lesson modules. The symbols needed to recognize terminal inputs are entered into the system parameter file. Various system messages are read in, converted to ASCII, packed, and entered in the system parameter file, also.
CHAPTER VI

EVALUATION

1. **STRIPE**

The goals for the information system were given in the introduction. The degree to which they were met is as follows:

1. The system is running on an IBM 1130 with a single disk having a capacity of 1 M bytes and 16K bytes of main storage.

2. On one data base of student records using real data the storage efficiency is between 92% and 95%. The variation depends on whether records have been just enlarged and have some unused space or are essentially full. Suppose records had the form of a mandatory block of 112 bits and a variable field with 10 subfields each averaging 40 characters. The overhead would be 30 bits in the mandatory block and 160 bits in the subfields. Total bits in the variable field would average $10 \cdot (40 \cdot 5 + 16) = 2300$. Total bits would be $2300 + 112 = 2412$. The efficiency would be $(2412 - 190)/2412 \approx 92\%$.

3. Additions, modifications are easy to make in a batch process. The times observed for additions of new records have ranged from 750 to 1200 records/hour. This timing is dependent on whether category lists are maintained, the physical positioning of the files, the size of the master file, and whether multiple disk drives are used. A dramatic decrease in movement of the disk arm will be observed when the master file is on one drive and the other files are on another. Timing in the neighborhood of 1800 to 2400 records added per hour should be achievable.
4. Input is not of a generalized format. Hence it is not as easy to adapt to a new format as in some systems. On the other hand the user can be totally flexible in writing his input routines which outweighs the disadvantages. This means that his main responsibility is to put the data into the proper slots in the common I/O buffer.

5. The query function has been described. It works reasonably efficiently, is easy to use, and is capable of expansion to other functions.

6. As previously explained the system does handle multiple disk drives.

7. The flexibility exists to add new fields to records even where those fields were not originally planned for. The logical file may be enlarged without rebuilding the existing file.

8. To transfer the system to a new computer having a FORTRAN compiler would involve writing new bit manipulation routines in assembler. There are few of these, so the task would not be hard.

STRIPE has met the goals set for it, with the possible exception of the goal of easy adaptability to differing input formats. This was not done because in developing the system it was found that generalizing the input significantly slowed program execution. Still, the writing of short I/O routines seems to be a relatively easy task.

To get a better picture of how STRIPE compares with other file systems in use a questionnaire was sent out to members of CUETUG. Thirty-five descriptions of administrative files were evaluated. The mean efficiency was 51.5%. The range was from 21% to 80%. A less formal investigation of file structures in more sophisticated computer centers seemed to indicate a slightly higher efficiency. However, on the average, it is unlikely to be much more than 60%.

The significance of these figures lies in this. With broad use of the
techniques used in STRIPE, the effective storage space at many computer centers would increase by half.

2. SCRIPT

The goals for SCRIPT were also given in the introduction.

1. Several instructors have learned to write SCRIPT lessons in less than 30 minutes. A seventh grader has done the same.

2. The system can be used for either instruction or testing. The automatic recording of scores has not been implemented.

3. Diagnostic information relative to the lesson's clarity are provided. When the student asks for help (using $H$) a record is kept of the state and input at which help was needed.

4. SCRIPT is running now set up for 8 terminals. Memory space is not available for more in the 16K byte system.

5. Response time for an 8 terminal system has only been calculated. The hardware available for development included only one teletype. The interface has just two line adapters. To exceed 3 seconds average response time with 8 terminals in use, the average service time would have to be above .375 seconds. Since the equipment operates on an interrupt system, once a message is sent to a terminal the computer can go on to service another. Even if every user made a request for service each second, the system could handle it.

6. The cost of using any system such as SCRIPT would be prohibitive unless well used. If SCRIPT were used as much as 8 sessions a day for 200 days per year, the cost would be about 70¢/session. This figure is arrived at by allocation about 50% of the cost of the basic computer to the terminal use. This amounts to about $35,000. Nearly all the cost of the terminals and
the interface is, of course, allocated to the academic use. This comes to $16,000. The total is $51,000. If we use only 5 years to prorate this cost, the yearly amount of $10,200 divided by 12,800 sessions gives the above cost per session.

This cost calculation is made with the assumption that between sessions other work could be run. It should be recognized that no consideration has been given to costs of development of lesson modules. In SCRIPT or any other system this will be significant if it is not spread over a large volume of users. Sharing of lessons among educational institutions is very necessary. On the credit side it can be pointed out that by using some hardware from other sources than the mainframe manufacturer considerable savings can be made - as much as $300 per month. This would amount to enough to pay for a lot of development.

On balance, then, though there is further work to be done, the goals have been met.
BIBLIOGRAPHY


Other Sources:
Numerous discussions at conferences of CUETUG and the Association for the Development of Instructional Systems added ideas in this development.
APPENDIX A

PROOF THAT THE LINEAR QUOTIENT HASH ALGORITHM
WILL PROBE ALL SLOTS OF A FILE IF THE DIVISOR IS PRIME

Assume the divisor, D, is a prime integer and K is the principal key. Q = \lfloor K/D \rfloor. \lfloor x \rfloor means FLOOR(x). If this makes Q = 0, set Q = 1. Let p₀ be the remainder when K is divided by D, or

\[ p₀ = K - \lfloor K/D \rfloor \cdot D \]

Then we recursively define

\[ p_{i+1} = (p_i + Q) \mod D + 1 \]

Without losing generality we can assume 1 ≤ Q ≤ D since the pᵢ's are calculated modulus D.

Obviously, the sequence p₀, p₁,...,pᵢ,...,pᵢ₋₁ = \{pᵢ\} will have to start duplicating by the D+1 term. We need to show it can not duplicate earlier.

By the construction of \{pᵢ\} there will be subsequences S₁, S₂, etc. of \{pᵢ\} in which the sequence has been increasing in say, Sᵢ, decreases going to Sᵢ₊₁, increases through Sᵢ₊₁, decreases going to Sᵢ₊₂, etc. As an example, consider D = 11, Q = 3, and p₀ = 2.

\{pᵢ\} = \{2,5,8,0,3,6,9\}

We see that duplicating begins on the 12th element. There are three subsequences as described.

S₁ = \{2,5,8\} \quad S₂ = \{0,3,6,9\} \quad S₃ = \{1,4,7,10\}

It is easily seen that the number of these subsequences is Q. The number of elements in each Sᵢ is either \lfloor D/Q \rfloor or \lfloor D/Q \rfloor + 1.

Now, suppose the first elements of Sᵢ and Sₖ are equal. (Taking the case
of the first elements should suffice.) That is, \( s_{j1} = s_{k1} \). That means \( s_{j1} \) was obtained by going \((k-j)\) times around the "modular clock," starting from \( s_{k1} \), or

\[
s_{j1} \equiv s_{k1} + (k-j) \cdot D
\]

Now if

\[
s_{j1} = p_i \text{ and } s_{k1} = p_m,
\]

then

\[
s_{j1} = p_i \equiv p_m + (m-i) \cdot Q \equiv s_{k1} + (k-j) \cdot D
\]

and we have

\[
(k-j) \cdot D = (m-i) \cdot Q
\]

\[
D = \frac{m-i}{k-j} \cdot Q
\]

If \((k-j) = Q\), then \( p_i \) and \( p_m \) are \( D \) elements apart which is as it should be for duplication. But if \((k-j) \neq Q\) where there are \( Q \) of those subsequences then \( D \), an integer, is equal to the product of two integers, and, thus, \( D \) is not prime. The case where \( Q = 1 \) is trivial.
APPENDIX B

SAMPLE SCRIPT LESSON IN LIMITS OF INFINITE SERIES
### SCRIPT Decision Table

<table>
<thead>
<tr>
<th>States</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>13</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>15</td>
</tr>
</tbody>
</table>

### Inputs

<table>
<thead>
<tr>
<th>Inputs</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>6</td>
<td>8</td>
<td>7</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>7</td>
<td>1</td>
<td>7</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Legend

<table>
<thead>
<tr>
<th>NS</th>
</tr>
</thead>
<tbody>
<tr>
<td>NCM</td>
</tr>
<tr>
<td>V PTR</td>
</tr>
</tbody>
</table>

### SCRIPT Message Vector

```
1, 2, 3, 4, 33, 5, 6, 9, 10, 11, 12, 7, 5, 13, 14, 6, 22, 12, 34, 15, 20, 16, 17, 18, 6, 21, 22, 12, 19, 5, 20, 18, 30, 31, 29, 18, 23, 24, 27, 30, 50, 60, 70, 80
```
<table>
<thead>
<tr>
<th>MESS. NO.</th>
<th>MESSAGES (less than 61 characters total)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>WE'VE SEEN THAT SOME SERIES SUM</td>
</tr>
<tr>
<td>2</td>
<td>1/3 + 1/9 + 1/27 + ... CONVERGE</td>
</tr>
<tr>
<td>3</td>
<td>WHAT IS THE LIMIT OF THIS ONE?</td>
</tr>
<tr>
<td>4</td>
<td>1/1 2/3 4/5 6/7</td>
</tr>
<tr>
<td>5</td>
<td>LET'S TRY THAT AGAIN.</td>
</tr>
<tr>
<td>6</td>
<td>THAT'S RIGHT.</td>
</tr>
<tr>
<td>7</td>
<td>SINCE SOMETHING IS ADDED TO 1/3, THE SUM MUST BE BIGGER.</td>
</tr>
<tr>
<td>8</td>
<td>IT SEEMS WE MUST GO BACK TO THE PREVIOUS LESSON.</td>
</tr>
<tr>
<td>9</td>
<td>WE'VE ALSO SEEN THAT SOME SERIES SUCH AS</td>
</tr>
<tr>
<td>10</td>
<td>1/2 + 1/3 + 1/4, DIVERGE TO INFINITY.</td>
</tr>
<tr>
<td>11</td>
<td>NOW CONSIDER 5 - 5 + 5 - 5 + ...</td>
</tr>
<tr>
<td>12</td>
<td>DOES IT CONVERGE OR DIVERGE?</td>
</tr>
<tr>
<td>13</td>
<td>OK, WHAT IS THE LIMIT?</td>
</tr>
<tr>
<td>14</td>
<td>1/0 2/5 3/-5 4/10 5/-10 6/16) NONE</td>
</tr>
<tr>
<td>15</td>
<td>IF SO, EVENTUALLY THE SUMS GET ARBITRARY</td>
</tr>
<tr>
<td>16</td>
<td>CLOSE TO THAT LIMIT, AND REMAIN THERE</td>
</tr>
<tr>
<td>17</td>
<td>WILL THE SUMS EVER STAY WITHIN 1 OF THE LIMIT?</td>
</tr>
<tr>
<td>18</td>
<td>YES, 2: NO:</td>
</tr>
<tr>
<td>19</td>
<td>IN HOW MANY TERMS?</td>
</tr>
<tr>
<td>20</td>
<td>THEN DOES IT CONVERGE?</td>
</tr>
<tr>
<td>21</td>
<td>THAT MEANS IT DIVERGES.</td>
</tr>
<tr>
<td>22</td>
<td>THEN HOW ABOUT (5 - 5) + (5 - 5) + (5 - 5) + ...</td>
</tr>
<tr>
<td>23</td>
<td>YOU SEEM TO HAVE THE RIGHT IDEA</td>
</tr>
<tr>
<td>24</td>
<td>THAT IS ALL TODAY.</td>
</tr>
<tr>
<td>25</td>
<td>GET SERIOUS!</td>
</tr>
<tr>
<td>MESS. NO.</td>
<td>MESSAGES (less than 61 characters total)</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>26</td>
<td>TO GET THAT ANSWER YOU MUST HAVE CHANGED THE SERIES.</td>
</tr>
<tr>
<td>27</td>
<td>THIS SHOULD LEAVE YOU THINKING ABOUT ASSOCIATIVITY AND</td>
</tr>
<tr>
<td>28</td>
<td>COMMUTATIVITY IN INFINITE SERIES</td>
</tr>
<tr>
<td>29</td>
<td>BUT ISN'T ((5 - 5) + ((5 - 5) + \ldots) EQUAL TO (5 - 5 + \ldots) BY ASSOC. LAW?</td>
</tr>
<tr>
<td>30</td>
<td>NO, THERE IS ALWAYS ANOTHER 0 OR 5, SO IT CAN'T</td>
</tr>
<tr>
<td>31</td>
<td>STAY WITHIN 1 OF EITHER</td>
</tr>
<tr>
<td>32</td>
<td>FOR ANY FINITE NUMBER OF TERMS NO HELP PROVIDED</td>
</tr>
<tr>
<td>33</td>
<td>THAT IS</td>
</tr>
<tr>
<td>34</td>
<td>NO HELP PROVIDED</td>
</tr>
<tr>
<td></td>
<td>ANSWER 1 OR 2</td>
</tr>
</tbody>
</table>
The following is a short segment of a lesson written by a seventh grade student. It took less than 20 minutes explanation to show him how to use SCRIPT. (No corrections have been made. This is just as he wrote it the first time.) This is the result exactly as it came out on the teletypewriter.

READY* TYPE #ID
#6
TYPE 1 TO BEGIN
1
GEOMETRY
IF YOU TOOK THE DEGREE OF
EACH ANGLE IN A TRIANGLE AND ADDED THEM
TOGETHER, WHAT WOULD YOU HAVE?*
1)180 2)360 3)270 4)NONE OF THESE
2
WRONG TRY AGAIN
1)180 2)360 3)270 4)NONE OF THESE
$H
NO HELP NEEDED
1
RIGHT DO YOU WANT THE NEXT LESSON
1) YES 2) NO
$S
1
LINES AND PLANES LESSON TWO
HOW MANY PLANES CAN INTERSECT TWO POINTS IN A
LINE AND 1 POINT NOT ON THE LINE
$L
2
THAT'S ALL THANK YOU
APPENDIX C

DESCRIPTION OF STRIPE APPLIED TO
AN ACADEMIC RECORD SYSTEM

To illustrate the use of STRIPE the use of the files and the layout of files in those fields will be shown. This gives more detail to the explanations given in Chapter II.

The purpose of this application of STRIPE is to hold the academic cumulative record of students in a college.

1. Category File

This file concerns faculty members who are advisors. The advisor is arbitrarily assigned a number which is used as the record number of his entry in the file. Records in the file are of two types. The primary type contains a pointer to the first secondary type record chained to the primary, a count of the number of advisees for this advisor, and the advisor's name. The second record contains a pointer to the next secondary record, if any, and pointers to records in the master file of advisees. All these records are 12 words each. For both the secondary record chains and the master record pointers, zero means null. These records are not bit packed since the nature of the information is such that there would be little advantage.

2. Descriptor File

This file contains information about courses. It, too, has primary and secondary records.
The primary records list information about the course itself as shown in the table below, including a pointer to a secondary record. Primary records are bit packed.

The secondary record's first word is a pointer to the next secondary record, if any. The other words are pointers into the master file to students enrolled in the course. Secondary records are one field/word.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>No. of bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Numeric code for course</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>Pointer to secondary record</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>Record status</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>Department number</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>Course number</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>Section</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>Course credit</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>Quarter and year</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>Number enrolled</td>
<td>8</td>
</tr>
<tr>
<td>10</td>
<td>Maximum number allowed</td>
<td>8</td>
</tr>
<tr>
<td>11</td>
<td>Hour given</td>
<td>4</td>
</tr>
<tr>
<td>12</td>
<td>Instructor code number</td>
<td>7</td>
</tr>
<tr>
<td>13</td>
<td>Building and room</td>
<td>16</td>
</tr>
<tr>
<td>14</td>
<td>Building and room</td>
<td>16</td>
</tr>
<tr>
<td>15</td>
<td>Exam period</td>
<td>13</td>
</tr>
<tr>
<td>16</td>
<td>i) Days given 16 characters</td>
<td>128</td>
</tr>
<tr>
<td></td>
<td>ii) Course title 16 characters</td>
<td>128</td>
</tr>
<tr>
<td></td>
<td>iii) Instructor name 10 characters</td>
<td>80</td>
</tr>
</tbody>
</table>

**TOTAL 480 bits = 30 words**

The quarter and year is coded with fall, winter, spring and summer having codes 1, 2, 3, and 4, respectively. Thus, the code for spring, 1973 is 373. For internal storage 150 is subtracted, and 223 is stored. For summer, 1973 the code is 473. For internal storage 470 is subtracted, and 3 is stored. This system will work until the year 2000.
The identification number of the course is a hash of the department, course and section. The system checks to be sure there is not a duplicate hash number when course records are entered. If there is, the section number is changed.

3. Master File

The contents of these records are given in the following table. Depending on the length of the variable field an initial size of 100 words provides space for nearly a year's worth of courses.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>No. of bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Record status: bit 0=opt. block present, bit 1=vbl field present, bit 7=error bit</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>Length of vbl field in words</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>Number of words in record</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>Number groups of fields in vbl block</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>Id. No. (first)</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>Id. No. (second)</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>Id. No. (third)</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>Degree</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>Sex</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>Class</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>First major</td>
<td>5</td>
</tr>
<tr>
<td>12</td>
<td>Second major</td>
<td>5</td>
</tr>
<tr>
<td>13</td>
<td>First advisor</td>
<td>7</td>
</tr>
<tr>
<td>14</td>
<td>Second advisor</td>
<td>7</td>
</tr>
<tr>
<td>15</td>
<td>First dorm.</td>
<td>5</td>
</tr>
<tr>
<td>16</td>
<td>Second dorm.</td>
<td>5</td>
</tr>
<tr>
<td>17</td>
<td>Course crd. att. here (x10)</td>
<td>11</td>
</tr>
<tr>
<td>18</td>
<td>C.C. achieved here (x10)</td>
<td>11</td>
</tr>
<tr>
<td>19</td>
<td>Total grade pts. (x10)</td>
<td>16</td>
</tr>
<tr>
<td>20</td>
<td>Grade pt. average (x10000)</td>
<td>14</td>
</tr>
<tr>
<td>21</td>
<td>C.C. achieved elsewhere (x10)</td>
<td>11</td>
</tr>
<tr>
<td>22</td>
<td>Total course cred. (x10)</td>
<td>11</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
<td>No. of bits</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>23</td>
<td>Academic status</td>
<td>4</td>
</tr>
<tr>
<td>24</td>
<td>Geographic region</td>
<td>6</td>
</tr>
<tr>
<td>25</td>
<td>School requirements met</td>
<td>16</td>
</tr>
<tr>
<td>26</td>
<td>Residence code</td>
<td>4</td>
</tr>
<tr>
<td>27</td>
<td>Real mode field</td>
<td>32</td>
</tr>
<tr>
<td>28</td>
<td>Unused</td>
<td>13</td>
</tr>
<tr>
<td>29</td>
<td>Birth date month</td>
<td>4</td>
</tr>
<tr>
<td>30</td>
<td>Birth date day</td>
<td>5</td>
</tr>
<tr>
<td>31</td>
<td>Birth date year</td>
<td>7</td>
</tr>
<tr>
<td>32</td>
<td>Date grad. (year) from H.S.</td>
<td>7</td>
</tr>
<tr>
<td>33</td>
<td>H.S. rank in class</td>
<td>10</td>
</tr>
<tr>
<td>34</td>
<td>Size of H.S. grad. class</td>
<td>10</td>
</tr>
<tr>
<td>35</td>
<td>Grade pt. ave. in H.S. (x1000)</td>
<td>14</td>
</tr>
<tr>
<td>36</td>
<td>SAT, verbal</td>
<td>10</td>
</tr>
<tr>
<td>37</td>
<td>SAT, math</td>
<td>10</td>
</tr>
<tr>
<td>38</td>
<td>Years of Eng. in H.S. (x2)</td>
<td>3</td>
</tr>
<tr>
<td>39</td>
<td>Years of Soc. Sci. in H.S. (x2)</td>
<td>3</td>
</tr>
<tr>
<td>40</td>
<td>Years of For. Lang. in H.S. (x2)</td>
<td>3</td>
</tr>
<tr>
<td>41</td>
<td>Years of Math. in H.S. (x2)</td>
<td>3</td>
</tr>
<tr>
<td>42</td>
<td>Years of Hist. in H.S. (x2)</td>
<td>3</td>
</tr>
<tr>
<td>43</td>
<td>Years of Nat. Sci. in H.S. (x2)</td>
<td>3</td>
</tr>
<tr>
<td>44</td>
<td>H.S. credits</td>
<td>5</td>
</tr>
<tr>
<td>45</td>
<td>Unused</td>
<td>12</td>
</tr>
<tr>
<td>46</td>
<td>Begin variable field</td>
<td></td>
</tr>
</tbody>
</table>

Variable field: 4 possible subfields
1) name, 2) birthplace, city & state, 3) address 1, 4) address 2
Form of each subfield: first word has sub-field number in first 8 bits, no. of words in subfield in second 8 bits. Lines of address separated by local code. Use * here.
Average length if all subfields are in might be (in words) name=5+birthplace=6+addr 1=15+addr 2=15. Total 42 words.
Master File detail continued:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>No. of bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>47</td>
<td>Dept.</td>
<td>6</td>
</tr>
<tr>
<td>48</td>
<td>Course no.</td>
<td>10</td>
</tr>
<tr>
<td>49</td>
<td>Section</td>
<td>8</td>
</tr>
<tr>
<td>50</td>
<td>Course credits (x10)</td>
<td>8</td>
</tr>
<tr>
<td>51</td>
<td>Grade</td>
<td>8</td>
</tr>
<tr>
<td>52</td>
<td>Date (quarter and year code)</td>
<td>8</td>
</tr>
<tr>
<td>53</td>
<td>Dummy</td>
<td>0</td>
</tr>
</tbody>
</table>

Summary of Space for Record

<table>
<thead>
<tr>
<th>Block</th>
<th>Bits used</th>
<th>Overhead</th>
<th>Minimum required</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>224</td>
<td>39</td>
<td>185</td>
</tr>
<tr>
<td>2</td>
<td>112</td>
<td>12</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>480</td>
<td>58</td>
<td>424</td>
</tr>
<tr>
<td>4</td>
<td>1744</td>
<td>16</td>
<td>1728</td>
</tr>
</tbody>
</table>

Efficiency if 160 word record is filled with 36 classes

\[
\text{Efficiency} = \frac{2437}{2560} = .952
\]

Efficiency could be as low as .95 if measured at the time a record has just been enlarged by 20 words and only 1 new group has been put in.

The other files are just as shown in Chapter III. Therefore, they are not repeated here.

Once the files and record layouts have been determined the files are defined for the monitor system. Then program INFIL is used to put system
messages and parameters into the files and to initialize the master file. Then ADCAT and ADDSC can be run to set up categories and descriptors.
APPENDIX D

PROGRAM LISTINGS FOR STRIPE

Main programs are listed first in the order described in the text. Subprograms are given in alphabetical order.
**INFIL**

INTEGER Sizin, DCLS L
INTEGER ECF (320), BLANK (38), CHAR (40)
INTEGER FIELD (97), COMM (50)
EQUIVALENCE ECF (ECF (320), DCLS L)
EQUIVALENCE ECF (IFXF. COMM (19)), (IVBL. COMM (21))
DATA K32/32000., K0/0./
DATA NCRD/IZ, Sizin/2.0, 1024./, ECF/320*5/./, BLANK/38*5/.*
DEFINE FILE 1 8160., 20, U, N1, 2 300, 12, U, N2, 3 800, 20, U, N3
DEFINE FILE 4 320, 1, U, N4, 5 1024, 5, U, N5, 6 (42, 15, U, N6)
DEFINE FILE 8 (96, 10, U, N8), 7 (40, 60, U, N7), 9 (50, 4, U, N9)

100 WRITE (1, 530)

930 FORMAT (TYPE WHICH FILE TO INITIALIZE*/1.2.3.4.5.6.7.8.9 OR 9 IN I 
+2 FORMAT*/10 FOR EXIT OR 11 FOR ALL FILES*
READ (6, 9921)

902 FORMAT (I2)
I ALL 1
GO TO (1., 2., 3., 4., 5., 6., 7., 8., 9., 1000, 1011 +)

1011 I ALL 2
C...FILE 1. IF MULTIPLE MASTER FILES USED, INITIALIZATION OF THEM
C...MUST BE ADDED
1 DO 10 I = 1, 8145, 16
10 WRITE 1 I ECF
WRITE (11) (ECF (J), J = 1, 17), K32, IZ
GO TO 100, 2 * I ALL

C...FILE 2
2 DO 20 I = 1, 100
20 WRITE (21) IZ, IZ, (BLANK (J), J = 1, 10)

C...LINK TOGETHER LAST 200 RECORDS IN CATEGORY FILE FOR FREE AREAS
DO 25 I = 101, 299
J = I +
25 WRITE (21) J, (ECF (K), K = 1, 11)
WRITE (23) 100, IZ, IZ, (ECF (K), K = 1, 11)
GO TO 100, 3 * I ALL

C...FILE 3
3 GO TO 30 I = 1, 647
30 WRITE (31) IZ, (BLANK (J), J = 1, 29)
25 I = 500, 800
35 WRITE (31) IZ, IZ, IZ, (BLANK (J), J = 1, 30)
GO TO (100, 400), I ALL

C...FILE 4
4 WRITE (1, 921)

921 FORMAT (*PUT COMM CARDS IN*)
PAUSE
400 READ (2, 920) CCW

920 FORMAT (20I)
READ (2, 920) FIELD
WRITE (41) CCW
WRITE (42) DCLS L

C...PUT IN ARRAY TO USE IN A1A3
DO 45 I = 126, 165
45 WRITE (41) CHAR (I-125)

C...PUT IN LAYOUT CF MASTER RECORD
KSM = 0
KK = 51 + IF XF + ICF
DO 46 I = 51, KK
WRITE (41) I, KSM

46 KSM = KSM + FIELD (I-50)
KSM = 0
KK = KK + IVBL + 1
DO 49 I = 5, KK
WRITE (41) I, KSM

49 KSM = KSM + FIELD (I-50)
GO TO 100, 5 * I ALL

C...FILE 5
5 DO 50 I=1,5
50 WRITE 5,1Z,1Z,1Z,1Z,1Z
GO TO 100,600,IALL
C***FILE 6
6 WRITE(1,919)
   919 FORMAT('PUT REQUIREMENT MSGS IN READER')
   PAUSE
C***MUST BE 42 REQUIREMENT MSG CARDS
   60 DO 60 I=1,42
   60 READ(NCRD,935)(BLANK(J),J=1,15)
   935 FORMAT(15A2)
   60 WRITE(6,1*(J))BLANK(J),J=1,15)
GO TO 100,700,AALL
C***FILE 7
C***PUT 20 BA FOLLOWED BY 20 BS CARDS. USE BLANKS FOR UNUSED RECORDS
7 WRITE(1,910)
   910 FORMAT('PUT REQUIREMENT DESC. IN READER')
   PAUSE
   70 DO 70 K=1,40
   70 READ(NCRD,912)(IDR,G,N)
   912 FORMAT(3(I2,1X))
   IF(IDR=1A1)71,71,61
   61 IF(NN=40)62,62,7
   62 IF(NN=58)63,63,7
   63 READ(NCRD,913)(BLANK(I),I=1,NN)
   913 FORMAT(1117)
   WRITE(7,1*(G))NN,(BLANK(I),I=1,NN)
   CONTINUE
   71 GO TO 100,800,AALL
C***FILE 8
8 WRITE(1,914)
   914 FORMAT('PUT ERR MSGS IN RDR')
   PAUSE
   80 DO 80 I=1,96
   80 READ(2,900)(BLANK(J),J=1,10)
   900 FORMAT(10A2)
   80 WRITE(81,1*(J))BLANK(J),J=1,10)
GO TO 100,1900,AALL
C***FILE 9
C***USE 50 CARDS. PUT IN BLANKS FOR UNUSED RECORDS
9 WRITE(1,915)
   915 FORMAT('PUT IN DEPT. ABBR. & LTR GRADE CARDS IN RDR')
   PAUSE
   900 DO 90 J=1,50
   900 READ(NCRD,916)(KA,KB)
   916 FORMAT(2A2)
   90 WRITE(91J,1*(J))KA,KB,BLANK(37),BLANK(38)
GO TO 100
1000 CALL EXIT END
** EXEC

INTEGER COMW (165), FLPTR (4), STURC (340), BUFF (67)
COMMON COMW, FLPTR, STURC, IERFL, BUFF
DATA I Y / 1, IN / N /
C...EXECUTIVE ROUTINE
C...BUFF (67) CORRESPONDS TO SPTR USED IN QUERY
DEFINE FILE 4 320.1, U, N4
BUFF (67) = 0
IERFL = 1
READ (4, 1) CCWW
4 WRITE (1, 900)
900 FORMAT ('SET SWITCHES')
CALL DATS *(*15, J)
IF (J = 1) 59.9, 5
5 WRITE 1, 901
901 FORMAT (0-EXIT / 1-ADD ADVISOR / 2-ADD COURSE / 3-A
CDD STUDENTS / 4-ADD COURSE TO STUDENT / 5-PUT GRADES / 6-REQUIREMEN
CT SHEETS / 7-GARbage COLLECTION / 8-ADD COURSE TO STUDENT.CRS.* / 9-DELETE MAST
*ER REC. / 10-DELETE MAST *ER REC. / 11-DELETE MAST *ER REC. / 12-QUERY / 13-UTILITY")
9 PAUSE
DO 10 I = 1, 16
CALL DATS * I-1, J
10 CONTINUE
GO TO 5
15 K = I-1
16 WRITE 1, 902 K
902 FORMAT ('SET SWITCH .12. RIGHT TYPE Y OR N AND THEN EOF'.
READ 6, 903 IA
903 FORMAT (A1)
IF (IA = 'Y') 20, 30, 20
20 IF (IA = 'N') 16, 5, 16
30 GO TO 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150, 160, 170, 180, 190, 1
50 CALL EXIT
60 CALL LINK (ACCAT)
70 CALL LINK (ACSTD)
80 CALL LINK (ACLASS)
90 CALL LINK (ADVEL)
100 CALL LINK (MDVEL)
110 CALL LINK (RQUR)
120 CALL LINK (RQFC)
130 CALL LINK (RQFT)
C...SET UP MODIFICATION PROGRAM TO MODIFY ANY PART OF RECORD
140 GO TO 190
150 CALL LINK (GPDP)
160 CALL LINK (CRCP)
170 CALL LINK (QURY)
180 CALL LINK (UTLT)
180 CONTINUE
WRITE (1, 904)
904 FORMAT (NOT AN OPERATIONAL ROUTINE')
GO TO 4
END
**ADCAT**

**THIS ROUTINE ASSUMES FACULTY IS ASSIGNED NUMBERS 1-VALUE OF COMM(27)**

```plaintext
INTEGER COMM(165),FLPTR(4),STURC(340),BUFF(41)
INTEGER ADVSR(10),BLANK(15)
COMMON COMM,FLPTR,STURC,IERFL,BUFF
DATA K99/0, BLANK/15*1648/,KEOD/**/
EQUIVALENCE (ADVSR(1),BUFF(2)),(NUM,BUFF(1)),(KEST,BUFF(41))
EQUIVALENCE (K99,COMM(34)),(KVAIL,COMM(30))
DEFINE FILE 2(300,12,U,N2),4(320,1,U,N4),89(96,10,U,N8)
```

**RECORD LAYOUT OF CATEGORY RECORDS**

**RECORD 1** POINTER TO INVERTED LIST

**RECORD 2** NUMBER IN LIST

**RECORD 3** BEGINNING OF NAME(10 WORDS)

**INVERTED LIST IS MAINTAINED IN THE RECORDS FOLLOWING THE CATEGORIES.**

**CHAIRS ARE IN FIRST WORDS.** IN ACTIVE RECORDS THE FOLLOWING

**RECORDS:**

5 CALL HCATC
   IF(KEST-KEOD)6,99,6
6 IF(NUM)999,5,13
10 READ(2*NUM)11
   IF(I)<999,11,999
11 COMM(CM=CM(CM(26)+1)
   WRITE(2*NUM)KVAIL,K99,ADVSR
   KSV=KVAIL
12 CALL ERR(25)
   IERFL=1
   99 WRITE(A'1)COMM
   CALL LINK EXEC
   999 CALL ERR(18)
   IERFL=1
   GO TO 5
END
```
** ADDSC

```
INTEGER COMM(165),FLPTR(4),STURC(340),BUFF(41)
INTEGER CRINX(797),BUFF1,BUFF2
INTEGER ENCSF,SZCRS
COMMON COMM,FLPTR,STURC,IERFL,BUFF
EQUIVALENCE (BUFF2,BUFF(2))
EQUIVALENCE (KCODE,BUFF(41)),(BUFF(1),BUFF1),(KVAIL,COMM(37))
EQUIVALENCE (ENCSF,COMM(25)),(SZCRS,COMM(22)),(IDIV2,COMM(24))
DATA KECC**'*,IZ/0/
DEFINE FILE 3(800,30,U,N3),4(320,1,U,N4),8(96,10,U,N8)
C...ADD NEW DESCRIPTOR RECORDS
.C...FIRST READ IN INDEX OF DESCRIPTOR IDS FROM DISK FOR HASH1 TO USE
DO 2 I=1,10
2 READ(31)CRINX(I)
5 IF(ENCSF-SZCRS)20,10,10
10 COMM(23)=1
CALL ERR(35)
GO TO 999
C...RDSCC FILLS IN BUFF. BUFF(1) IS THE KEY NO. OF THE DESCRIPTOR.
20 CALL RDSCC
IF(KECC=KCODE)40,999,40
40 CALL HASH1(BUFF1,CRINX,IR,IDIV2,1)
C...HASH1 WILL NOT PUT IN A DESCRIPTOR WITH DUPLICATE KEY.
41 CALL ERR(10)
IERFL=1
CALL STACK
GO TO 20
42 CRINX(IR)=BUFF1
BUFF2=KVAIL
KP=KVAIL
IF(KP)10,10,70
70 READ(3*KP)KVAIL
WRITE(3*KP)12
WRITE(3*IP)(BUFF(I),I=1,30)
ENCSF=ENCSF+1
GO TO 5
999 WRITE(4*1)COMM
CALL LINK(EXEC)
END```
**ADMAS**

C.....ADDS A NEW MASTER RECORD TO THE DATA BASE
REAL XBUFF(20)
INTEGER CCVM(16S),FLPTR(4),STURC(340),BUFF(41)
INTEGER SZSTU, DCLSL, ADVX(100), NAME(9), ENSTF, MPT(12)
COMM CCM, FLPTR, STURC, IERFL, BUFF
EQUIVALENCE (NCADV, CCM(27)), (IDIV, CCM(7)), (INSIZ, CCM(17)),
+ (ICM(14), CCM(14)), (NCFLD, CCM(19)), (CMM(1), ICM(1),
+ (CCVM(2), ICM(2)), (SZSTU, CCM(33)), (NOSTD, CCM(4))
EQUIVALENCE (BUFF(41), IFB41), (KVAIL, CCM(30)), (NAME(1), MPT(1))
EQUIVALENCE (ICM(3), CCM(3)), (ICM(20), CCM(28)), (CCM(5), ICM(5))
EQUIVALENCE (IS9, STURC(9)), (IS10, STURC(10)), (CCM(5), ICM(5))
EQUIVALENCE (BUFF(5), IBUF5), (BUFF(6), IBUF6), (BUFF(7), IBUF7)
EQUIVALENCE (STURC(8), IS8), (BUFF(1), XBUFF(1))
C.....THE EQUIVALENCING OF MPT AND STURC(321)-(332) IS ONLY
C.....NEEDED IF THE INITIAL RECORD IS NOT OVER 320 WORDS
C.....EQUIVALENCE (KPT(1), STURC(321))
DATA DCLSL/* */, KEOD/* */
DEFINE FILE 1(8160,20,UN),2(300,12,UN),3(320,1,UN),
DEFINE FILE 4 3,1(5,10,UN), 5 1024,5,UN, 8(96,10,UN)
C.....READ IN CATEGORY NUMBERS. NOTE THAT ADVX MUST BE DIMENSIONED BIGGER
C.....IF THERE ARE OVER 100 CATEGORIES
DO 3 I=1,100
READ(21) ADVX(I)
3 CONTINUE
IF (ENSTF-SZSTU+2) 10,10,10
10 ICM5=1
IERF 17
GO TO 1000
C.....RMASC FILLS IN BUFF. BUFF(I) CORRESPONDS TO FIELD(I) IN
C.....MANDATORY FIXED BLOCK FOR FIELDS 5 THRU NOFLD. IF
C.....NOFLD IS GT 30 A CHANGE MUST BE MADE. THE ENTRY FOR SUBFIELD 1
C.....OF THE VBL FIELD BEGINS IN BUFF(31). FIELDS 1 THRU 4 OF
C.....RECORD ARE COMPUTED IN THE PROGRAM, SO BUFF(1)-(4) UNUSED NOW.
20 CALL RMASC
GO TO (27,26), IERFL
26 IERFL=I
GO TO 999
C.....IFLG IS TC SHOW WHETHER THE FREE AREA
C.....IS BEING EXAMINED OR POINTED TO BY COMM OR BY A PREVIOUS AREA
C.....NEEDED FOR MAINTAINING FREE AREA CHAINS
27 IF (IFE41-KECD) 1027,995,1027
1027 IFLG=1
ICM5=IGB4(1CCM1,0,11)
ICM4=IGB4(1CCM2,12,15)
ICM3=CCM2
WM=IBUF5
NN=IBUF6
NNN=IBUF7
C.....MAKE SURE THERE IS A PLACE IN INDEX AND NOT A DUPLICATE
CALL HSMP2 WK, AN, NN, KREC, IDIV, I, IPTR
IF (KREC) 271,271, 28
271 CALL STACK
GO TO 20
20 CALL FICHSC)230,230,228
230 ICMSC=IGB4(1CCM1,0,11)
ICM4=IGB4(1CCM2,12,15)
ICM3=CCM1
228 CALL RMASC(I,CW)
C.....NEED TO DETERMINE IF THERE ARE AT LEAST 100 WORDS AVAILABLE
C.....TO START THE RECORD
A=IS8
N90=N/INSIZ+1
C.....IADD NEEDED FOR LATER CHECK ON WORD IN SECTOR
C.....ON WHICH RECORD STARTS
IADD=INS9/20
IF(N90=3) 29,29,30
30 N90=3
29 N*XSEC=IS9
**THIS SECTION** IS FOR A RECORD. GO TO NEXT FREE AREA.

**IFLG** = 2

**ICHSW** = **NXSW**
**ICM** = **NXWRD**

GO TO 26

**THIS SECTION** IF **IFLG** = 1 AND **N90** = 2. I.E. WE ARE IN THE

**AREA** POINTED TO BY **COMM** (2) AND THE NO. OF WORDS

**IS BETWEEN** **INSIZ** AND **TWICE INSIZ**.

**ICM** = **NXSW**
**CALL** **IPFLD** **N,X,3**
**IN** = **ICM** + 1

**IL** = **N**

GO TO 60

**CHECK** TC TO SEE IF **N** SHOWS REST OF FILE FREE

**IF** (**N** - 32000) = 43, 46

**IF** (ICM2 - ICM) = 29, 43

**STURC**(INSIZ + 8) = 32000

**GO TO 48**

**STURC**(INSIZ + 9) = **NXSW**
**CALL** **IPFLD** **INSIZ,X,3**
**IN** = **ICM** + 1

**IL** = **INSIZ**

**GO TO 60**

**CHECK** TC IF **N90** = **IFLG** = 2. **FREE AREA** POINTED AT BY

**AREA** SIZE **IS BETWEEN** **INSIZ** AND **TWICE INSIZ**. **LSTSP** IS SECT.

**HERE NEXT FREE AREA BEGINS. LSTWP IS REL. WO. IN THAT SECT.**

**THEY MUST BE PUT INTO THE PREVIOUS FREE AREAS POINTER.**

**LSTWP** = **INSIZ**
**CALL** **IPFLD** **N,X,3**
**IN** = **ICM** + 1

**IL** = **INSIZ**

**GO TO 60**

**CHECK** TC IF **N90** = **IFLG** = 2. **FREE AREA** POINTED TO

**BY PREV. AREA** SIZE **GE TWICE INSIZ**. **SECTOR** AND **WORD POINTERS**

**MUST BE PUT INTO THE PART OF THIS FREE AREA REMAINING FREE.**

**CALL** **IPFLD** **INSIZ,X,3**
**IF** (ICM - IADD) = 51, 52

**LSTWP** = **ICM** + **INSIZ**

**LSTSP** = **ICM** + 1

**53** **CONTINUE**

**CHECK** **N** **GE** 32000

**IF** (N - 32000) = 54, 55

**55** **IF** (ICM2 - 2STL + **INSIZ**) = 56, 57

**56** **STURC**(INSIZ + 8) = 32000

**GO TO 58**

**54** **STURC**(INSIZ + 9) = **NXSW**
**IN** = **ICM** + 1

**IL** = **INSIZ**

**GO TO 60**

**CONTINUE**

**NOW GET CATEGORY RECORD FILLED IN**
IF(ICW28)71,71,61
61 CONTINUE
   KADV=BUFF(ICW28)
   READ(2*KADV)IPT,NUM
   CALL INVRT(2,IPT,120,ICHSW,MPT)
   GO TO (71,68,1)IERFL
C***THE MEANS THERE IS NO MORE ROOM IN CATEG. FILE. GARBAGE COLLECT
68 CALL ERR(4)
   IERFL=1
   CALL STACK
   GO TO 20
C***NEW FLL INFORMATION INTO MASTER RECORD
71 GO TO 72 IF=3ACFDL
   CALL IPFLD(ELFF(I)+1),123,45,1)
C***THE VALUE 123,45 BEING USED JUST FOR TEST OF REAL MODE FIELD.
C***THAT ARGUMENT SHOULD BE 0 UNLESS FILLED IN FROM RMASC
72 CONTINUE
C***PUT IN NAME. NAME GOES INTO VBL FIELD. SINCE WE HAVE
C***NO OPTIONAL FIXED BLOCK AS YET, VBL FIELD FOLLOWS MAND. FIXED.
C***NO. OF A3 FIELDS IN NAME IS GIVEN IN BUFF(38)
   K=31
   II=ICW14+2
   III=ICW14+1+NAM
   GO TO 174 I=II,I
   STURC(I)=BUFF(K)
174 K=K+1
   IN=IN+1+NAM
C***SET RECORD TO INDICATE VBL FIELD
   CALL IPFLD 64,X,1
C***SET LENGTH OF VBL FIELD
   CALL IPFLD(NAMAV+1,X,2)
C***NUMBER OF GROUPS OF FIELDS IN VBL BLOCK
   CALL IPFLD 0,X,4
C***ZERO CUT RECORD STARTING WITH UNUSED WORDS. IN IS FIRST
C***WORD UKUSEC IN FIXED PART OF RECORD.
   DC 75 K=IN,IL
   STURC(K)=0
   GO TO (175,173),IERFL
C***FILL IN INDEX FILE AND ADD 1 TO NUMBER OF STUDENTS
175 WRITE(5*KREC)MM,NN,NNN,ICHSW,BUFF(39)
   NSTD=NSTD+1
C***PUT RECORD ON DISK
C***KEY CF 1 FOR WMA Sr MEANS ALSO WRITE AN ADDITIONAL TEN
C***WORDS FOR CER FREE AREA CHAIN
   CALL WMA Sr(IKVS,1)
   GO TO (80,173),IERFL
173 IERFL=1
   CALL STACK
   GO TO 20
C***CHECK FOR UPDATING POINTER TO LAST RECORD
80 IF(ICHSC=1GV=(ICOM10.0,11))20,79,78
78 ICOM1=IPVE(ICHSC,ICOM1,0,11)
79 IF(ICHS=IADD81,B5,85)
81 ICOM1=IPVE(ICHSC+1,ICOM1,12,15)
   GO TO 86
85 ICOM1=IPVE(ICHSC+1,ICOM1,12,15)
   ICOM1=IPVE(ICHSC1,ICOM1,0,11)
   GO TO (20,82),IFLG
C***FILL IN FREE AREA CHAIN IF THIS WAS NOT FIRST ONE
82 CALL RWMA Sr(ISVSC,ISVWD)
   GO TO (84,83),IERFL
83 IERFL=1
   GO TO 20
84 IS9=LISTP
   IS10=LISTP
   CALL WMA Sr(ISVSC,0)
   GO TO 20
1000 CALL ERR IERR
   IERFL=1
999 CONTINUE
WRITE 4 1 COMM
ENSTF=IGVB(1CCW1,0,11)
NSLFT=SZSTU-ENSTF-1
WRITE(1,900)NSLFT
900 FORMAT(*THERE ARE NOW * I4 * SECTORS LEFT IN STU. FILE*)
CALL LINK EXEC
END
**ADVBL**

```plaintext
** INTEGER COMM(165),BUFF(41), CHAR(40), MWT(30) **
** COMMON COMM,FLPTR,STURC,IERFL,BUFF **
** EQUIVALENCE (IDIV2,COMM(7)),(IDIV1,COMM(24)),(NCRD,COMM(34)) **
** (KECD,BUFF(41)),(ICM16,COMM(16)),(ICM15,COMM(15)) **
** (ICM19,COMM(19)),(ICM20,COMM(20)),(ICM21,COMM(21)) **
** (ICM3,COMM(3)),(ID1,BUFF(1)),(ID2,BUFF(2)) **
** (ICM3,BUFF(3)),(STURC(1),IST) **
** EQUIVALENCE (KC,BUFF(5)),(KA,BUFF(6)),(KS,BUFF(7)) **
** EQUIVALENCE (CSPTR(4)),(STURC(1)) **
** DATA M,MM,NN,SS,CC **
** DEFINE FILE 5(1024,5,U,N5),9(50,4,U,N9),2(300,12,U,N2) **
** DEFINE FILE 1(8160,20,U,N1),4(320,1,U,N4),8(96,10,U,N8) **
** C... ADDS A NEW GROUP TO END OF MASTER RECORD **
** C...BUFF(1),(2),(3) ARE ID FIELDS **
** C...BUFF(4),(ICP21) ARE FIELDS TO PUT INTO RECORD **
** IFGL=1 **
** C...IT IS UP TO RVELC TO SET BUFF WITH FIELD DATA AND SET COMM(6) **
  1 CALL RVELC **
  2 IF(KECD-KAST)4,22,4 **
  3 IF(ID1-MM)20,10,20 **
  4 IF(ID2-MM)20,10,20 **
  5 IF(ID3-MM)20,35,20 **
  6 IF(ID4-MM)20,70,20 **
  7 IF(ID5-MM)20,105,20 **
  8 GO TO (30,25),IFLG **
** C...PUT PREVIOUS RECORD BACK **
  25 CALL IPFD NCFLD=X4 **
  26 CALL WMASR(ISCWD,0), IFGL=1 **
  27 GO TO (30,25),IFLG **
** C...FIND NEW RECORD **
  30 IF(KECD-KAST)32,999,32 **
** C...ADD XC WORDS IN HAND, PCT. VBL FIELD **
  31 CALL HSHF2(ID1,ID2,ID3,K,IDIV2,2,ISCWD) **
  32 IF(K)33,33,34 **
  33 CALL ERR(8) **
  34 WRITE(1,900)ID1,ID2,ID3 **
  35 FORMAT(REC.ID IS ',...,313) **
  40 GO TO 135 **
  36 CALL WMASR(ISCWD) **
** C...COMPUTE SPACE AVAILABLE FCR FIELD GROUPS **
** C...ADD XC WORDS IN HAND, PCT. VBL FIELD **
** C...PLUS XC. OF VBL BLOCK GROUPS TIMES WORDS PER GROUP **
  37 NCFLD=GFLD(4) **
  38 LVEL=GFCLD(2) **
  39 IP=1GBLIST(0,0) **
  40 CALL IERFL **
  41 GO TO (36,45),IERFL **
** C...COMPUTE CURRENT EFFECTIVE LENGTH OF RECORD, SEE IF THERE **
** C...IS CCMI FCR CAN MORE DESCRIPTOR GROUP **
  42 LS=ICM14+ICM15+IP*LVEL+NCFLD*ICM16 **
  43 IF(LS=LS-1CM16)40,50,50 **
** C...NEED MORE SPACE **
  44 CALL IPFD NCFLD=X4 **
  45 CALL RECMV(ISCWD,K) **
  46 GO TO (36,45),IERFL **
** C...COMPUTE FIELD NOS. FOR GROUP **
  47 IF(ICM19+ICM20+ICM21+NCFLD)50,60,60 **
** C...FILL IN DESCRIPTION INVERTED LIST IF ICM6=2 **
  48 GO TO (90,65),ICM6 **
** C...ID OF DESCRIPTOR IN BUFF(4)=KID **
```
65 CONTINUE
KEY=2
66 CALL HSMFL(KID,IR,IDIV1,KEY,CSPTR)
IF(IR)134,134,80
134 CALL ERR(7)
135 CALL STACK
N=0
GO TO 1
80 CALL INVRT(3,LSTPT,30,0,ISCWD,WPT)
READ(3*IR)KIC,IPNT,I1,I2,I3
N=IGVB(I3,8,15)
I3=IPVB(N+4),I3,8,15
WRITE(3*IR)KIC,IPNT,I1,I2,I3
GO TG(90,134),IERFL
90 IFLG=2
NCFLD=NCFLD+1
IERFL=1
DO 55 I=1,IC*21
CALL IFLD(BUFF(I+4),X,NF+1)
GO TG(95,91),IERFL
55 IERFL=1
WRITE(5*960,)
960 FORMAT(*ERR CK *,14)
95 CONTINUE
GO TG 1
999 WRITE(4*1)CGW
CALL LINK(EXEC)
END
** GPDRP

```fortran
INTEGER CEP, CRS, SEC, STURC(340), CRINX(497)
INTEGER CMP(165),ILPRT(4),MPT(30)
COMMON CMP, CMM, FLPRT, STURC, IERFL, BUFI
EQUIVALENCE (ICM15, CMM(19)), (ICM20, CMM(20)), (ICM21, CMM(21)),
  (ICLM, CMM(6)), (KDAT, CMM(36)), (ICM4, CMM(14)),
  (ICM16, CMM(16)), (IDIV1, CMM(7)),
  (IDIV2, CMM(24)), (NCRD, CMM(34)), (NPRNT, CMM(35))
DATA Kast/*/*
DEFINE FILE 1(816,20,U,N1).2(300,12,U,N2).3(800,30,U,N3).
  5(1024,S,U,N5).6(96,10,U,N6)
C***THIS CRCDP A GROUP FROM THE BLOCK AT THE END OF A MASTER RECORD
DO S = 1,497
5 READ(3*1)CRINX(1)
    WCVF = ICM19+ICM20+1
C***ID AND NAME CARD FOLLOWED BY SOME DROP CARDS
10 READ(NCRD,900)M,N,NN,DEP,CRS,SEC,KST
900 FFORMAT(213,T78.11,T26.12,T35.13,A1,T80,A1)
15 IF(KST-KAST)18,999,18
18 IF(M)26,26,20
20 CALL HSPF2(W,N,N,DEP,KRCRID,DIV1,2,NSNW)
    IF(KRCRID)21,25,25
21 CALL ERR(7)
22 CALL STACK
GO TO 23
23 CALL RMASK(NSNW)
GO TO 10
26 NGPS = GFLD(4)
C***COMPUTE FIELD NC. OF FIRST FIELD IN LAST GROUP
   NFLDS=(NGPS-1)*ICM21+1
   LFLD=LFLD+WCVF
   NFDS=LFLD-ICM21
30 IF(DEP-IG)35,31,35
   IG=GFLD(LFLD)
31 IF(CRS-IG)32,35
   IG=GFLD(LFLD+1)
32 IF(SEC-IG)33,32,35
   IG=GFLD(LFLD+2)
   IPVE(IG,6,2,7)
   IF(SEC-IG)33,32,35
C***CRCP BACK ONE GROUP AND TRY AGAIN
35 LFLD=LFLD-ICM21
   IF(LFLD=WCVF)36,36,30
36 CALL ERR(28)
37 CALL STACK
IERFL=1
GO TO 10
42 GFLD(LFLD+5)
   IF(IG-KCAT)47,50,47
47 CALL ERR(63)
GO TO 37
50 CONTINUE
C***MAKE SURE THIS IS NOT A COURSE IN PREVIOUS QUARTER
C***TAKE THIS CUT IF CATE CHECKING ISN'T NEEDED
42 GFLD(LFLD+5)
   IF(IG-KCAT)47,50,47
47 CALL ERR(63)
GO TO 37
50 CONTINUE
C***RESET NC. OF GCRCPS IN VBL BLOCK
55 CALL IPFLD(NGPS-1,X,4)
C***COMPUTE WORD PCINTER FOR REPACKING COURSES
   IG=GFLD(2)
   NWRCDS=ICM14+IGVE(STURC(I),0,0)*ICM15+IG
C***COMPUTE WHICH VBL GROUP IS CRCPPED
   LFLD=LFLD-WCVF/LICM21
C***COMPUTE WRC DCINTER TO IT AND TO END GROUP
   N1=NRDS+LFLD*ICM16+1
   N2=NGP*LFLD+LICM16-1
58 STURC(I)=STURC(I)
   CALL WMSR(NSNW,0)
GO TO(10,59), ICLCH
```
96

59  KID=IPVB(DEP,CRS.0.5)+SEC
   KEY=2
69  CALL HASH1(KIC,CRINX,IR,IDIV2,KEY)
   IF(IR)156,156,159
158  CALL ERR(10)
   GO TO 37
159  READ(3*IR)KIC,IPNT,I1,I2,I3
C.*REDUCE NC. ENROLLED BY ONE
   N=IPVB(13,8,15)
   13=IPVB(N-1,13,8,15)
   WRITE(3*IR)KIC,IPNT,I1,I2,I3
   CALL INVRT(3,IPNT,497,NSNW,0,MP0)
C.*INVERTED LIST AND REPLACE IT WITH 0
C.*IF OCCURRENCE OF NSNW IN THE
   GO TOC(10,37),IERFL
999  CALL LINK(EXEC)
END
**DRCP**

INTEGER COMM(165), FLPTR(4), STURC(340)
INTEGER DCLS,L,MPTR(30),BUFF(41)
COMMM,CMM,CCMM,COVM(34),FLPTR,STURC,IERFL,BUFF
EQUIVALENCE (KARD, COMM(34)), (IDIV1, COMM(7)), (IDIV2, COMM(24))
EQUIVALENCE (CCMM(28),ICMM(28), (COMP(6), ICMM(6))
EQUIVALENCE (KDAT, COMM(36)), (COMP(4), ICMM(4)), (COMP(2), ICMM(2))
EQUIVALENCE (IS8, STURC(8)), (IS9, STURC(9))
EQUIVALENCE (COMM(19), ICMM(19)), (COMM(20), ICMM(20)), (COMM(21), ICMM(21))
DATA DCLS/,9,/ DATA NEG.IZ/-1.0/
DEFINITE ARD.EL M. N
900 FORMAT(213)
10 CALL HSFID(KARD,0,IR,IDIV1,KEY,ISCDW)
20 WRITE(5,IR)NEG.IZ,IZ,IZ,IZ
30 CALL RPASR(ISCDW)
40 IF (KCCM(28),20,20,15)
50 K=GFLD(3CM(28))
60 CALL INVRT(2*K,12,ISCW,0, M PT)
70 GO TC(20,16),IERFL
80 IERFL=1
90 CALL STACK
100 GO TC(50,25),ICMM
200 NGPS=GFLD(4)
300 IF (NGPS),50,50,30
40 MOVFL=ICMM+1
50 MCFVL=ICMV+1
60 MCFDL=MCFL+1
70 NFEU=NCFL-1+NGPS
80 CONTINUE
90 CALL HSFIP(KF,IR,10,IV2,KF,IPTR)
100 IF (IR),50,40
110 CALL INVRT(3,IR,30,ISCDW,0, M PT)
120 GO TC(45,41),IERFL
130 IERFL=1
140 CALL STACK
150 NFEU=MFCFL-ICMV+1
160 NFEU=MFCDL-ICMV+1
170 CONTINUE
180 ENTER RECOROF AREA IN FREE AREA CHAIN
190 NSW=ICMM
200 ICMM=ISCDW
210 K=GFLD(3)
220 DO 60 I=1,K
230 STURC(I)=DCLS
240 IS8=K
250 NSW=ISW
260 CALL WMAR(ISVSW,0)
270 REDUCE COUNT OF NO. OF MASTER RECORDS
280 ICM4=ICM4-1
290 GO TC 1
300 WRITE(4,*1)COMW
310 CALL LINK( EXEC)
320 END
** GARBG **

```
INTEGER FREC(340), FLPTR(4), S(320)
INTEGER COMM(165), STURC(340), HUFF(41), MVINX(2), DOLSL, ST4,
+ P=1
*CRS(30), STX(5), IFREC(2,400), ST8, ST9
COMM CN COMM, FLPTR, STURC, IERFL, HUFF
EQUIVALENCE (STURC(8), ST8), (STURC(9), ST9)
EQUIVALENCE (KSIZ3, COMM(3), (S(1), IFREC(1,1))
EFS=IGVE(1,1), MVINX(1), MVINX(2), MVINX(3), STX(4), ST4),
+ (FLPTR(1), IF1), (FLPTR(2), KSC1), (FLPTR(3), IF2),
+ (FLPTR(4), KSC2), (COMM(1)), (COMM(7)), (COMM(8))
EQUIVALENCE (ICM6, COMM(6)), (ICM15, COMM(15)), (ICM14, COMM(14))
+ (ICM16, COMM(16)), (ICM36, COMM(36)), (ICM2, COMM(2))
DATA CCLSL/*
DEFINE FILE 1(8160, 20, U, N1), 2(300, 12, U, N2), 3(800, 30, U, N3),
+ 4(320, 1, U, N4), 5(1024, 5, U, N5), 6(640, 2, U, N6), 8(56, 10, U, N8)
C.. THIS COMPRESSES THE FILE SO THAT THERE ARE NO FREE AREAS EMBEDDED
C.. IN THE MASS CF RECORDS. THERE WILL JUST BE AT
C.. IN THE MESS OF RECORDS THERE WILL JUST BE THE ACTIVE AREA AND END
C.. CF FILE FREE AREA.
C.. IFREC IS DIMENSIONED (2,400) SO THAT EXPANDING OR CONTRACTING IT
C.. MNT AFFECT SMLL2
C.. FILE 9 IS TEMPORARY WS FILE FOR MOVE INDEX. REQUIRES 4 SECTORS
C.. FOR 800-920 STUDENTS
DO 5 I=1,2
DO 5 J=1,1400
5 IFREC(1, J)=0
C.. INITIALIZE MOVE INDEX
DO 10 I=1,500,160
10 WRITE(9,'(I5)')
   IP=1
   ISVLS=IGVE(1CM1, 0, 11)
   IFREC(1,1)=ISVLS
   IFREC(2,1)=ISVLS
   NFS=IGVE(1CM2, 0, 11)
   NF=IGVE(1CM2, 12, 15)
   NFS=ICM2
   KNFRE=1
   DO 7 K=2,400
   IF(NFSW)1998, 8, 1998
1998 KNFRE=KNFRE+1
   CALL RMASR(NFSW)
   GO TO(11, 2001), IERFL
2001 IERFL=2
   GO TO 999
11 IFREC(1, K)=NFS
   IFREC(2, K)=NF
C.. GET POINTER TO NEXT FREE AREA
   NFS=ST9
   NFS=IGVE(NFSW, 0, 11)
   NF=IGVE(NFSW, 12, 15)
   NFS=IPV8(NFS, NF, 0, 11)
   IP=IP+1
7 CONTINUE
C.. SORT IFREC 1NC ASCENDING ORDER
   CALL SHL2(IFREC, KNFRE)
   NFS=IFREC(1, IP)
   NF=IFREC(2, IP)
   NFS=IPV8(NFS, NF, 0, 11)
   IP=IP+1
C.. READ FIRST FILL AREA
   CALL RMASR(NFSW)
   NXFS=IFREC(1, IP)
   NXFS=IFREC(2, IP)
   NXFS=IPV8(NXFS, NXFS, 0, 11)
   IP=IP+1
   NWRCS=ST8
   KNFRE=KNFRE-1
C.. IF THIS FREE AREA GOES TO END OF FILE NOTHING MORE TO BE DONE
   IF(KNFRE)355, 355, 9
C.. CALCULATE NEXT STDNT PNTR
   CALL PTCA(L(NFSW,NWRS, NSTSW))
```
N S T S C = I G V ( N S T S W , 0 »11)
N S T W D = I G V ( N S T S W , 12 » 15)

14 C CONTINUE

C.. READ NEXT MASTER RECORD
20 CALL CR ( N S T S C , N S T W D )
STW = N S T W # 20
IRN = 320 + N S T W
READ ( I F 1 * K S ) ( F R E C ( I ) , I = 1 , I R N )
IRN = I R N + 1
READ ( I F 2 * K S ) ( F R E C ( I ) , I = I R N , 3 4 0 )

C.. EXTRACT NC, CF CARDS IN RECORD
K W D = I G V ( F R E C ( 1 ) , 1 5 » 1 5 ) * 2 5 6 + I G V ( F R E C ( 2 ) , 0 » 7 )
DO 5 0 I = 1 , K W D

50 S T U R C ( I ) = F R E C ( I )
V V 1 = N S T W
V V 2 = N S T W D

C.. KVINK CONTAINS CLD POINTER IN POSITION 1, NEW PTR IN POS. 2
C FOR FIND KVINK CONTAINS CLD POINTER, RETURNS NEW PTR
C FOR PLACE KVINK CONTAINS CLD AND NEW POINTERS
C KEY = 1 FOR PLACE OR FIND
C IDIV MUST BE PRIME AND SET FROM COMM
C KEY = 1 FOR PLACE OR FIND
C CALL H S M I X ( KVINK , IDIV * KEY)
C WRITE STUDENT BACK ON DISK
C CALL W M A S R ( N S T W , 0 )
C SAVE POINTER AND SIZE OF STUDENT FOR USE AT STMT 100
L S C = N S T W
L S W = N S T W D
K N F R E = 0
C CALCULATE PIXEL FOR FILL AREA
CALL P T C A L ( N S T W , K W O , N S T W D )
K W S = I G V ( N S T W , 0 » 1 1 )
K W F = I G V ( N S T W , 1 2 » 1 5 )
C CALCULATE PIXEL TO NEXT STUDENT TO MOVE
66 CALL P T C A L ( N S T S W , K W S , N S T S W )
N S T S C = I G V B ( N S T S W , 0 » 1 1 )
N S T W D = I G V B ( N S T S W , 1 2 » 1 5 )
C CHECK TO SEE IF WE ARE UP TO NEXT FREE AREA
IF ( N S T S C = N X F S W ) 1 4 , 7 0 , 7 5
70 I F ( N S T W - N X F W ) 1 4 , 7 5 , 7 5
C.. RECALCULATE PIXEL TO NEXT MAS.REC. FIRST EXAMINE THIS FREE AREA
75 CALL R M A S R ( N X F S W )
K N F R E = K N F R E - 1
I F ( K N F R E ) I 1 0 0 , 1 0 0 , 7 6
76 K W D = S T B
A X F S = I F R E X ( 1 , I P )
N X F S = I F R E X ( 2 , I P )
N X F S W = I P V B ( N X F S , N X F W , 0 » 1 1 )
I P = I P + 1
G O T O 6 6
C.. PATCH COMM POINTERS. FIRST CALCULATE END OF STUDENT RECS.
100 L S C W D = I P V B ( L S C , L S W , 0 » 1 1 )
C ALL P T C A L ( L S C W D , H N N , L S C W D )
I C M = L S C W D
I C M 2 = 0
C.. INITIALIZE FREE AREA
CALL R M A S R ( I C M 1 )
L S C = I G V B ( L S C W C , 0 » 1 1 )
L S W = I G V B ( L S C W C , 1 2 » 1 5 )
D O 1 1 0 I = 1 , 3 4 0

110 S T U R C ( I ) = C C L S L
I F ( L S C = K S I Z E + 1 0 0 ) 1 2 0 , 1 2 0 , 1 1 5
115 S T B = 3 2 0 0 + ( K S I Z E - L S C ) - L S W + 1
G O T O 1 2 1
120 S T B = 3 2 0 0
121 S T S = 0
C ALL W M A S R ( I C M 1 , 0 )
D O 1 3 0 I = 8 » 1 0
130 S T U R C ( I ) = D C L S L
J = L S C + 1
K S = K S I Z E = 1
D O 1 5 0 K = J , K S
100 IF(K-ISVLS)150,150,160
150 CALL *MAH(X,320)
C...NOW UPDATE POINTERS IN CATEGORY FILE
160 DO 210 I=101,300
  READ(2*I)BUFF(J),J=1,12
205 CONTINUE
  PMV=BUFF(J)
  KEY=2
  CALL *SHMIX(PMVX,DIV,KEY)
  BUFF(J)=PV2
208 CONTINUE
  WRITE(2*I)(BUFF(J),J=1,12)
210 CONTINUE
C...UPDATE COURSE POINTERS IF NEC.
GO TOC(300,255),150
255 DO 260 I=500,800
  READ(3*I)CRS
270 CONTINUE
  IF(CRS(J))270,270,260
260 PMV=CRS(J)
  KEY=2
  CALL *SHMIX(PMVX,DIV,KEY)
  IF(KEY)265,265,267
265 CRS(J)=0
GO TO 270
267 CRS(J)=PV2
270 CONTINUE
C...WRITE(3*I)CRS
C...FINALLY, UPDATE THE STUDENT INDEX
280 CONTINUE
300 DO 350 I=1,ICN
  READ(5*I)STX
310 CONTINUE
  IF(ST4)350,350,310
310 PMV=ST4
  KEY=2
  CALL *SHMIX(PMVX,DIV,KEY)
  ST4=PV2
  WRITE(5*I)STX
350 CONTINUE
355 NSLT=KSIZE-IGVE(INCN,0,11)
WRITE(1,800)NSLT
900 FORMAT(*THERE ARE NOW##14## SECTORS FREE IN STU FILE*)
999 WRITE(4*I)CCME
99 CALL LINK (EXEC)
END
**QUERY**

```
INTEGER C12
INTEGER LANG(6),PTR,OP(4),CRIT(12),IVAL(50),COMM(165),STURC(340).
+ FLPTR(4),ICF(4)
COMMON CCM*,FLPTR,STURC,IERFL,OP,CUR*IVAL,PTR
EQUIVALENCE (CRIT(12),C12)
DATA ICP/*11*,**/*1/*
DATA LANG/*'SE'*,RE*,LI*,AR*,ST*,EX*/
  EQUIVALENCE (CRIT(12),C12)
DATA ICP/*11*,**/*1/*
C***THIS LINKS TO CR CALLS THE Routines WHICH PERFORM THE INQUIRY OPER.
C***QUERY COMMANDS CONSIST OF SELECT,REFINE,LIST,ARRANGE,STATS,EXIT
DEFINE FILE 1(8160.20.6M1),5(1024.5.U,N5),4(320.1.U,N4),
+ 9(960.1.U,N9),8(96.10.U,N8)
C***FILE S IS TEMPORARY FOR WORKING SET
DO 2000 I=1,4
  2000 CP(I)=ICF(I)
  WRITE(1,500)
  900 FORMAT('SEtEC*oPINE,LIST',ARRANGE,STATS,OR EXIT')
      READ(6,901)K
      901 FORMAT(A2)
      DO 10 I=1,6
          IF(KL-LANG(I))10,15,10
      10 CONTINUE
      15 GO TO(100,200,300,400,500,600,1).1
      100 C12=1
          SPTR=0
          CALL LINK(SELET)
  200 C12=2
      CALL LINK(SELET)
      300 CALL LISTM
          GO TO 1
      400 CALL ARNG
          GO TO 1
      500 CALL STATS
          GO TO 1
      600 CALL LINK(EDIT)
END
```
**SELECT**

REAL VAL(25)
INTEGER CCLS, CP2, SZMAS, C12
INTEGER CRIT(12), KGT(3), AND(3), OR1, OR2, REL(2), RPAR, PER, SEMI,
+ LIKE(78), BLANK, STATE, QUOTE, OP(4), CRIT, TCRIT, VPTR, XP(25),
+ TCRIT(25), IVAL(50), XPTR, SPTR
COMMON COMM, FPTR, STURC, IERFL, GP, CRIT, IVAL, SPTR
EQUIVALENCE (COMM(19), ICM19), (COMM(20), ICM20), (COMM(3), SZMAS)
EQUIVALENCE (VAL(1), IVAL(2)), (BLANK, COMM(132)), (PER, COMM(164)),
+ (CR1, COMM(129)), (OR2, COMM(135)), (STURC(11)), IST
EQUIVALENCE (CP2), (CRIT(12), C12), (XP1), IXP1), (IS8, STURC(8))
DATA CCLS/0*/I2/0/
DATA REL/L1*/T*/E*/G*/Q*/H*/V*/E*/X*/...
+ DATA RPAR, LPAR, SEMI, QUOTE/*
+ defines file 1(8160, 20, U, M), 5(1024, 5, U, N5), 4(320, 1, U, N4),
+ 9(560, 1, U, N9), 8(96, 10, U, N8)
C**This gets and encodes the selection criteria, and then selects
C**those records which satisfy the criteria. The pointers to the
C**selected records are saved in the work file. SPTR is count of select
C**C12 used to indicate select or refine
3 XPPTR=2
1 IXP1=0
1 VPR=1
1 CRIT=1
1 STATE=1
1 MOVFL=ICM19+ICM20+1
1 CALL ERR(34)
1 READ(6), LINE
1
901 FORMAT(78A1)
1 C**FIRST, PROCESS INPUT
1 DO 1000 I=1,78
1 IF(LINE(I)=BLANK)4,1000,4
1 IF(LINE(I)=SENV)5,1000,5
1 5 GO TO(10,40,80,50), STATE
1 C**LOCK FOR FIELD. NF LE 0 MEANS ERROR
1 C**CN RETURN I POINTS TO LAST CHAR. OF FIELD
1 10 CALL FIELD(LINE, I, NF)
1 12 TCRIT=PVV(NF, TCRIT, 0, 5)
1 12 XPPTR=CRITP
1 12 SPTR=XPPTR+1
1 12 STATE=2
1 20 GO TO 1000
1 C**LOCK FOR LEFT PAREN.
1 20 IF(LINE(I)=LPR)30,21,30
1 21 KODE=0
1 20 GO TO 300
1 C**IS IT *ACT
1 30 I=I-1
1 30 DO 35 J=1,3
1 35 II=II+1
1 35 IF(LINE(I)=ACT(J))1050,35,1050
1 35 CONTINUE
1 35 KODE=-4
1 30 I=I+2
1 30 GO TO 300
1 C**LOCK FOR RELATION
1 40 DO 45 J=1,6
1 40 IF(LINE(I)=REL(1, J))45,41,45
1 41 IF(LINE(I)=REL(2, J))45,46,45
1 45 CONTINUE
1 46 TCRIT=PVV(J, TCRIT, 0, 8)
1 46 I=I+1
1 46 IF(J=6)800,200,1000
1 C**LOCK FOR RT. PAREN.
1 50 IF(LINE(I)=RPAR)60,55,60
55 KODE=1
GO TC 300
C...LCCK FCR AND
60 II=I-1
DO 65 J=1,3
II=I+1
IF(LINE(II)-AND(J))70,65,70
65 CONTINUE
KODE=-2
STATE=1
I=I+1
GO TC 300
C...LCCK FCR CR
70 IF(LINE(I)-CR1)1050,72,1050
72 IF(LINE(I+1)-CR2)1050,73,1050
73 KODE=-3
STATE=1
GO TC 300
C...LCCK FCR COMPARISON VALUE. IF FIELD EXPRESSION IT MUST
BE ENCLOSED IN PARENTHESES.
80 IF(LINE(I)-LPAR)81,100,81
81 IF(LINE(I)-RPAR)85,150,85
C...SHOULD BE NUMBER
85 CALL NUMB(LINE,I,X,ITYP)
C...NUMBR ANALyzES CHARACTERS IN LINE UP TO THE NEXT NON-NUMERIC
CHARACTER. BLANKS ARE IGNORED. NUMERIC VALUE IS RETURNED IN X.
C...I RETURNS AS PCINTER TO LAST CHAR. OF NUMBER.
C...ITYP IS 0 FOR INTEGER, 1 FCR REAL, 2 FCR ERROR.
C...PUT TYPE AND VALUE PCINTER IN CRITERIA
86 TCRT=IPBV(ITYP,TCRT)10,10
TCRT=IPBV(VPTR,TCRT)11,15
VPTR=(VPTR+2)/2
VAL(VPTR)=X
VPTR=VPTR+1
GO TC 200
C...TRANSLATE FIELD EXPRESSION. KOP INITIALLY INDICATES PLUS
100 IF(I+1)
KOP=1
TCRT=IPBV(2,TCRT)9,10
TCRT=IPBV(VPTR,TCRT)11,15
IF(LINE(I)-CP2)107,105,107
C...KOP INDICATES MINUS SIGN
105 KOP=2
107 IF(LINE(I)-CP1)110,120,120
C...NEGATING KCP INDICATES NUMERIC VALUE WILL FOLLOW IT
110 KOP=-KCP
IVAL(VPTR)=KCP
IVAL=VPTR+4
IP=VPTR/2
IVAL(IP)=X
GO TC 130
C...BEETTE BE FIELD
120 CALL FIELD(LINE,I,NF)
125 IF(NF)1050,1050,125
125 IVAL(VPTR)=IP
IVAL(VPTR)=NF
VPTR=VPTR+2
C...LCCK FCR RT. PAREN.
130 IF(LINE(I)-RPAR)135,140,135
C...CHECK FCR OPERATION
135 DO 136 KCP=1,4
136 IF(LINE(I)-CP(KCP))136,107,136
136 CONTINUE
GO TC 1050
140 IVAL(VPTR)=0
VPTR=VPTR+1
GO TC 200
**C. . . ENTER STRING**

150 TCRT = IPVB(3, TCRT, 9, 10)

TCRT = IPVB(VPTR, TCRT, 11, 15)

KNT = 0

153 I = I + 1

IF(LINE(I) = C.LCTE) 155, 160, 155

155 KNT = KNT + 1

IVP = VPTR + KNT

IVAL(IVP) = LINE(I)

GO TC 153

160 IPRT(VPTR) = KNT

V PTR = V PTR + KNT + 1

**C. . . ENTER CRIT IN CRIT**

200 CRIT(CRTP) = TCRT

CRTP = CRTP + 1

STATE = 4

GO TC 1000

C. . . ENTER KCD IN XF

300 X P(VPTR) = KCDCE

XPTR = XPTR + 1

1000 CONTINUE

C. . . NOW SCRT XF INTO TC PCLISH. FIRST PUT END PAREN.

1001 XP(VPTR) = -1

1155 K L = 0

IFLAG = 1

DO 1200 I = 1, XPTR

IF(XP(I)) = 155, 160, 165

1160 GO TC(I161, I162), IFLAG

1161 IFLAG = 2

IF = 2

J = 1

1162 KL = KL + 1

GO TC 1200

1165 GO TC(1200, 1170), IFLAG

1170 IF(XP(I)) = 1200, 1175, 1200

1175 KL = KL - 1

IF(KL) = 1200, 1160, 1200

1180 CALL PCLST(XF, J + 1, I - 1, TXP)

K L = 0

IFLAG = 1

XP(J) = -99

XP(I) = -99

1200 CONTINUE

GO TC(1203, 1155), IFLAG

1203 CONTINUE

C. . . NOW START READING RECORDS AND EVALUATING

IS = 0

GO TC (1204, 2000), C12

2000 IS = IS + 1

IF(IS = SPTR) = 2001, 1600

2001 CALL ITRV(ISCDW, IS, 2)

ISCDW = 155, 1175, 1200

IS = 155, 1175, 1200

GO TC 1205

C. . . POINTER FCR SECTOR 1, WORD BLCK 0

1204 ISCDW = 16

1205 CALL RMASR(ISCDW)

IF(STURC(I) = CCLSL) = 1220, 1208, 1220

1208 L = STURC(8)

C. . . ASSUMPTICK IS MADE HERE THAT IF POINTER IS IN WORKING

C. . . SET IT WILL NOT GET INTO FREE AREA

1210 CALL PTCAL(ISCDW, L, ISCDW)

IF(ISCDW(I11) = 1205, 1600, 1600

C. . . SEE IF THIS RECORD SATISFIES CRIT.

C. . . FIRST, SET TXP TC ZERC

1220 DO 1225 I = 1, TXP

1225 TXP(I) = 0

K = 0

XP(VPTR) = -98

DO 1400 I = 1, XPTR

IF(XP(I)) = 1235, 1235, 1230
1230  \( K = K + 1 \)
\( T X P(K) = XP(I) \)
GO TC 1400
1235 IF(XP(I)+99)1400,1400,1240
1240 ICP=XP(I)-1
C...EVALUATE EACH CRITERION
IF(TXP(K)-100)1260,1250,1252
1250 I1=0
GO TC 1261
1252 I1=1
GO TC 1261
1260 I1=IRLVL(TXP(K))
C...TAKE CARE OF ACT OPERATION
1261 IF(ICP-3)1262,1290,1262
1262 K=K-1
IF(K)1266,1266,1263
1266 TXP(1)=IRLVL(TXP(I))
GO TC 1401
1263 IF(TXP(K)-100)1270,1264,1265
1264 I2=0
GO TC 1275
1265 I2=1
GO TC 1275
1270 I2=IRLVL(TXP(K))
1275 GO TC(1260,1230),IOP
1280 TXP(K)=I1+I2+100
GO TC 1400
1300 IF(I1)1310,1310,1320
1310 I2=1330,1330,1320
1320 TXP(K)=101
GO TC 1400
1330 TXP(K)=100
1400 CONTINUE
1401 IF(TXP(1))1455,1455,1450
1450 GO TC(1455,2000),C12
C...SPTR IS INITIALIZED TO 0 IN QUERY PRIOR TO SELECT LINK
1455 SPTR=SPTR+1
CALL IRTRV(ISCWD,SPTR,1)
1495 GO TC(1500,1496),C12
1496 CALL IRTRV(12,IS,1)
GO TC 2000
1500 L=GFLC(3)
GO TC(1210,2000),C12
1600 CALL IRTRV(ISCWD,1,3)
GO TC(1602,1601),C12
C...ELIMINATE ZEROED POINTERS. MAKES IT BETTER FOR ARRNG.
1601 DO 1670 I=1,SPTR
IF(ISCWD)1760,1610,1700
1610 JJ=I+1
ICNT=1
DO 1650 J=JJ,SPTR
IF(ISCWD)1660,1620,1660
1620 ICNT=ICNT+1
1650 CONTINUE
C...THERE ARE ICNT ZEROED POINTERS IN A ROW
1660 JJ=I+ICNT
DO 1670 J=JJ,SPTR
CALL IRTRV(ISCWD,J,2)
CALL IRTRV(ISCWD,J-ICNT,1)
1670 CONTINUE
SPTR=SPTR-ICNT
1700 CONTINUE
1602 WRITE(1,903)SPTR
903 FORMAT(*SELECTED*)
1605 CALL LINK(QUERY)
1050 WRITE(1,902)
902 FORMAT(*SYNTAX NEAR *.12.*TH CHAR.*)
GO TO 1605
END
**ARRNG**

**SUBCUTINE ARRNG**

INTEGER SP, STACK(20, 2)
INTEGER LINE(20), COMM(165), FLPTR(4), STURC(340), OP(4), IVAL(50)
+ SPTR, CRIT(12)
COMMON COMM, FLPTR, STURC, IERFL, OP, CRIT, IVAL, SPTR
EQUIVALENCE (COMM(163), KOMMA), (COMM(132), KBLNK), (COMM(141), IEFF)
ICT=0

**MESSAGE ASKS FOR FIELDS ON WHICH TO SORT**

**FIELDS ARE TO BE ENTERED AS A SERIES OF FXXX’S, WITH OR W/O SEPARATION**

CALL ERR(61)
READ(6, 901) LINE
901 FORMAT(20A1)
DO 20 I=1, 20
LINE(I)=LINE(I)

**IGNORE BLANKS OR COMMAS**

IF(LINE(I)=KBLNK) S, 20, 5
5 IF(LINE(I)=KOMMA) 10, 20, 10
10 CALL FIELD(LINE(I), NF)
IF(NF) 12, 12, 12
12 WRITE(1, 902)
902 FORMAT('SYNTAX NEAR’, J2, 'CHAR**')
GO TO 999

**FIELD**

13 ICT=ICT+1
LINE( ICT)=NF
20 CONTINUE

**BEGIN SORTING**

CRIT(1)=0
CRIT(2)=0
SP=0

I=1
L=SPTR
500 CALL G(I, L, K, ICT, LINE)
I2=L-K-1
I1=K-I-1
IF(I1) 640, 640, 600
600 IF(I2) 640, 640, 610
610 SP=SP+1
IF(I2-I) 630, 620, 620
620 STACK(SP+1)=I
STACK(SP+2)=K-1
I=K+1
GO TO 500
630 STACK(SP+1)=K+1
STACK(SP+2)=L
L=K-1
GO TO 500
640 IF(I2) 670, 670, 650
650 I=K+1
GO TO 500
660 L=K-1
GO TO 500
670 IF(SP) 680, 680, 690
680 CALL ITRV(K, K, 3)
999 CONTINUE
WRITE(5, 903) CRIT(1), CRIT(2)
903 FORMAT(1X, 218)
RETURN

690 I=STACK(SP+1)
L=STACK(SP+2)
SP=SP-1
GO TO 500
END
** KCCPC

INTEGER CCNM(15), FLTR(4), STLRC(34), ELF(41)

INTEGER FFL(64), F1(16), F2(16), F3(16), F4(16), ELF5, ELF6, ELF7,
+ ELF8, ELF9, ELF10, ELF11, EFT, XFT

REAL XELFF(10)

CCNM = CCNM + FLTR + STLRC + IEPRFL + ELFF + XELFF

CALL + IFLAG(+1)

EQUIVALENCE (ELFF(I), IC1), (ELFF(2), IC2), (ELFF(3), IC3),
(ELFF(4), IC4), (CCNM(15), IC15), (CCNM(2), IC2)

CALL + FLTR(1), F1(1), F2(1), F3(1), F4(1)

CALL XELFF(10)

C... RCMD RETURN IC IN XELFF(1)-XELFF(2), ELFF(4) INDICATES MODIFICATION

C... TYPE 1-2-3 OR 4, THE BITS OF ELFF(5)-ELFF(8) INDICATE WHICH FIELDS

C... TC ACCIPY. THE FIELD VALUES ARE GIVEN IN ELFF(5)-(4C).

C... FLCATING FLD. VALUES ARE IN XELFF(1)-(1C)

C... MODIFICATIONS IN ANY ONE CYCLE CAN BE

C... 1) CHANGE OF TC 22 FIELDS IN MAX, CR CPT, ELCCS

C... 2) CHANGE ANY 1 VAR, SLEFIELC, SLEFIELD NC, AND CHAR COUNT GIVEN IN

C... ELFF(5) & (10). CRANS. PACKED IN A3 IN ELFF(11)--

C... 3) CHANGE FIELDS IN ANY ONE CYCLE

C... 4) ALL APLICAL ELOCK OR CHANGE ALL CPT. ELCEK FIELDS.

MFL = ICM + ICM42

10 IEPRFL = 1

CALL + DSCC

GC TC(12,545), IEPRFL

12 GC TC(25,15), IFLAG

15 IF (IC1 = 2) IC2 = 16, 2C

16 IF (IC2 = 2) IC1 = 17, 2C

17 IF (IC3 = 2) IC2 = 18, 2C

C... REFTEA LAST MASTER RECORD

20 CALL + MASH(IEW, IC)

21 CALL + PPAFL(ICI, IC2, IC3, KP, ICIV, 2, IEN)

20 CALL + STACK

30 GC TC IC

40 CALL + LASH(IEW)

C... SET FIELD PLACE

40 CALL + EFAN(ELFF, F1)

41 CALL + EFAN(ELFF, F2)

42 CALL + EFAN(ELFF, F3)

43 CALL + EFAN(ELFF, F4)

EPT = 5

XFT = 1

GC TC(44, ICC, 65, 2CC), ELF4

4E GC GC I = 1, NCL

I(PFL(I)) ICC, 6C

50 CALL + IFEGC(ELFF(EPT), XELFF(XFT, 1), GC TC(65, 2C), IEPRFL

C... CHECK TC TO SEE WHICH FTA TC INCREMENT

5E KAI = IC

I(CC4(K) = 32) EC, 6E, 6E

EE EPT = XFT + 1

GC TC EC

5E XFT = XFT + 1

60 CCN + IC

60 GC TC IC

C... CHECK IF CR CPLF FIELDS. CHANGES IN FIRST 2 FIELDS IN CRCLP ACT

C... ALLOWED. IN THIS CASE ELFF(5)-(11) SHOULD HAVE A DESCRIPTOR IC.

C... ELFF(12)-- WILL PAVE THE NEW FIELD VALUES.

C... IF CRFL TO FIRST FIELD OF CRCLF WITH GIVEN IC

C... IF(1G) IC3, IC2, 7C

C... KG AND MCL ARE BASIC FIELD MCS. CR CRFLS STARTING WITH FCLRH FIELD

7C KG = KCL + 1

MCL = KCL + IC2

7C GC EC I = KG MCL

IF (PFL(I) = 7E, 7E, 7E
72 \texttt{K=ICF+L}
\texttt{CALL IFFLC(ELFF(EFT),XELFF(XFT),K)}
\texttt{K=KCF+L}
\texttt{IF(CCN*(K)-32)} 74, 76, 76
74 \texttt{EFT=EFT+1}
\texttt{GC TC 76}
76 \texttt{XFT=XFT+1}
76 \texttt{L=L+1}
80 \texttt{CONTINUE}
80 \texttt{GC TC IC}
C...FLT in \texttt{VEL} SLEFILEC
100 \texttt{CALL FVCLF(ELFG,ELF1C,ELF1I)}
\texttt{GC TC(1C,1IC,2C),IERFL}
110 \texttt{CALL FCFAV(IS*,KR*)}
\texttt{GC TC(111,2C),IERFL}
111 \texttt{IERFL=1}
\texttt{GC TC IC}
200 \texttt{CALL ZACCF}
\texttt{GC TC(1C,2IC),IERFL}
210 \texttt{CALL FCFAV(IS*,KR*)}
\texttt{GC TC(1C,2C),IERFL}
555 \texttt{WRITE(4*I)CCN}
\texttt{CALL LINK(EXEC)}
END
**UTILITY**

INTEGER COMM(165), STURC(340), BUFF(100), FLPTR(4)
COMCN CCM, FLPTR, STURC, IERFL, BUFF
DEFINE FILE 1(8160, 20, UN1), 2(300, 12, UN2), 3(800, 30, UN3)
DEFINE FILE 4(1020, 1, UN4), 5(1024, 5, UN5), 6(96, 10, UN6)
DEFINE FILE 6(42, 15, UN6), 7(40, 60, UN7)
DEFINE FILE 8(50, 4, UN9)

***THIS CAN BE USED FOR A VARIETY OF UTILITY RECORD SEARCHING OR PATCHING***

WRITE(1, 900)

900 FORMAT(*1-LIST CR PATCH COMM*/,*2-LIST ERR, ABBRVS. OR ETC. FILES/*, *
+3-LIST OR PATCH INDEX*/,*4-LIST OR PATCH CATEG. FILE*/,*5-LIST/PC *
+H MASREC*/,*6-LIST CATEG. CHAIN*/,*7-LIST OR PATCH DESCRIPTOR*/,* *
+*0-EXIT*)

2 READ(6, 901) IGC

901 FORMAT(11)

4 IF(IGC=8) 4, 2, 2
5 GO TO 10, 20, 30, 40, 50, 60, 70, 99, 9100
10 CALL LPCCM
20 CALL LEARF
GO TO 1
30 CALL LPIAX
GO TO 1
40 CALL LPADV
GO TO 1
50 CALL LPMR
GO TO 1
60 CALL LADCH
GO TO 1
70 CALL LPCRS
1 WRITE 1, 902
902 FORMAT(0, 1, ..., CR 9)
GO TO 2
99 CALL LINK(EXEC)

END
The following main program is a special program used to print out a form of the student's transcript. The output includes a listing of what graduation requirements the student has not yet fulfilled.
**ROUIR**

```fortran
INTEGER NAME(10),STURC(340), SG(18),RO(16),NAM1(21),
+ CHAR(40),NAV2(15),COMM(165),DPNM(32,2),MPT(11)
INTEGER BUFF(41),FLPTR(4)
COMMON CCM,H,FLPTR,STURC,IERFL,BUFF
EQUIVALENCE(MPT(1),BUFF(1))
EQUIVALENCER(NFXF,COMM(15)),(ICPF,COMM(20))
EQUIVALENCER(APRT,COMM(35)),(NAM1(1),NAV2(1))
EQUIVALENCER(CHAR(1),COMM(126)),(NCFLD,COMM(21))
DEFINE FILE 1(8160,20,N,1),2(300,12,U,N),4(320,1,U,N),
+ 5(1024,S,U,N5),6(42,15,U,N6),9(50,4,U,N9),
+ 7(40,60,U,N7),8(96,10,U,N8)
C...THIS IS A SPECIAL PROGRAM FOR PRINTING OUT RECORDS IN
C...CATEGORY GROUPINGS
C...STUDENT TRANSCRIPTS ARE GIVEN WITH AN EVALUATION OF ALL SCHOOL REOS.
C...SET UP GRADE SYMBOLS AND DEP. ABBRS.
DO 4 I=1,32
  4 READ(9* I )CPNM( I,1 ) .DPNM(1,2)
DO 5 I =33,50
  5 READ(9* I )SG( I )
  6 CALL ERR(62)
  7 READ(6,908)IGC
908 FORMAT(11)
  GO TO TC (2.1,999).GO
  1 IFRST = 1
  2 ILAST = COMM(26)
  3 GO TO 8
  4 CALL ERR(50)
  5 READ(6,910)IFIRST
910 FORMAT(11)
  IF(IFIRST-CCMM(27))3,3,2
  3 ILAST = IFRST
  4 DO 100 I=IFIRST,ILAST
    5 READ(2* I ) I PNT, NUM, NAME
    6 IF(IPTAT(100,100,9)
    9 READ(2* I ) I PNT, MPT
    10 DO 90 J=1,1
    11 IF(MPT(J))90,90,40
    40 WRITE(NPRNT,500)NAME
900 FORMAT(*1ADVISOR*,5X,10A2)
  10 CALL RMASR(MPT(JJ))
CC= GFLD(22)/10.0
GPAA= GFLD(20)/1000.0
KK=21
C...GVELF(N,KARY,K)
C...K IS THE SUBFIELD NC. OF THE VBL FIELD IN STURC
C...KARY IS AN ARRAY IN WHICH RETURN IS MADE
C...K IS DIMENSION OF KARY
C...CN RETURN K WILL BE LENGTH OF SUBFIELD OR ORIGINAL LENGTH OF
C...KARY, WHICHEVER IS SMALLER
C...CALL GVELF(1,NAM,1,KK)
WRITE(NPRNT,901)NAM1,CC,GPA
901 FORMAT(*1X,21A1,5X,*CRS CRED='*F6.2,5X,**GPA='*F5.3//)
  IG=GFLD(25)
  CALL EXPAN(IG,R0)
  IIDE= GFLD(6)
  WRITE(NPRNT,204)
904 FORMAT(*1X,*REQUIREMENTS REMAINING*/)
  ISH=20*(IIDE-1)
  NRO=CCM*(IIDE+30)
C...PRINT CUT REQUIREMENT DESCRIPTION IF NOT SATISFIED
DO 70 L=1,NC
  IF(RC(L))65,65,70
  65 IREC=ISMAL
  70 CONTINUE
WRITE(NPRNT,906)
905 FORMAT(*1X,15A2)
906 CONTINUE
WRITE(NPRNT,906)
```

906 FORMAT(' //X, ' 'COURSE', '4X,' 'QTR/yr', '2X,' 'C.C.', '1X,' 'GD', '2X,' 'GP')
 IFIRST=WF+ICPF+2
 IGF=GFLD(4)
 IF(IGF).EQ.90,90,75
 75 LAST=IFIRST+ICF+NOFLD
 DO 31 J=IFIRST,LAST,NOFLD
 ICS=GFLD(J+1)
 IDT=GFLD(J+5)
 IF(J-IFIRST).EQ.22,23,23
 23 IF(.LT-KDT).EQ.21,25,21
 21 QAV=XSUM/CCSY
 WRITE(NPRINT,503)QAV
 903 FORMAT(' //X, ' 'T35,' 'QTR.AVE.', '5F5.3')
 22 KDT=IDT
 CCSYM=0.
 XGSYM=0.
 25 IF(IDT-20).GT.24,26,26
 24 IO=1
 IY=IDT+70
 GO TO 27
 26 IO=IDT/100
 IY=IDT-IO+100+50
 IQ=IO+1
 27 IG=GFLD(J+4)
 CC=GFLD(J+3)/10.0
 CCSYM=CCSYM+CC
 XG=(IG-9)*CC
 IF(XG).LT.29,30,30
 29 XG=0
 30 XGSYM=XSYM+XG
 WRITE(NPRINT,502)DPNM(IDP,1),DPNM(IDP,2),ICS,IQ,IY,CC,SG(IG),XG
 902 FORMAT(' //X, ' 'A2', '1X,' 'I3, ' '2X, ' 'I2, ' 'I3, ' 'F3, ' '2X, ' 'F3, ' '1)'
 CAV=XSUM/CCSY
 WRITE(NPRINT,503)QAV
 90 CONTINUE
 IF(IPNT).EQ.100,100,9
 100 CONTINUE
 GO TO(599,6,599),IG0
 999 CALL LINK(EXEC)
 END
SUBROUTINES USED IN STRIPE
**DRIVE**

SUBROUTINE DRIVE(IS,IW)
INTEGER CCMM(165),FLPTR(4),STURC(340)
COMMON CCMM,FLPTR,STURC,IERFL
EQUIVALENCE(IF1,FLPTR(1)),(K1,FLPTR(2)),(IF2,FLPTR(3))
+(K2,FLPTR(4))
C***FIND THE PHYSICAL FILE AND RECORD NUMBER CORRESPONDING TO THE O/GICAL
C***SECTOR AND WORD BLOCK POINTER. COMMON 9 THRU (13) SHOULD HAVE THE
C***CUMULATIVE TOTAL NO. OF SECTORS IN PHYSICAL FILES 1,11,21,31,41, RESP.
IF(IS)999,999,5
5 DO 100 I=9,13
 IF1=I+104*(I-9)
 K1=(IS-1)*16+I+1
 IF(IS-CCMM(I))10*11*100
10 IF2=IF1
 K2=K1+16-IW
 GO TO 200
11 IF2=IF1+10
 K2=1
 GO TO 200
100 CONTINUE
C***MESSAGE IF SECTOR NUMBER IS TOO BIG
 999 CALL ERR(30)
 IERFL=2
200 RETURN
**DRIVE**

SUBROUTINE DRIVE(IS,W)
INTEGER COMM(165),FLPTR(4),STURC(340)
COMMON COMM,FLPTR,STURC,IERFL
EQUIVALENCE(IF1,FLPTR(1)),(K1,FLPTR(2)),(IF2,FLPTR(3)), (K2,FLPTR(4))
C...Fine the physical file and record number corresponding to the O/GICAL
C...sector and word block pointer. COMM(9) thru (13) should have the
C...cumulative total no. of sectors in physical files 1,11,21,31,41, resp.
IF(IS)999.999.5
5 DO 100 I=9,13
   IF1=I+10*(I-9)
   K1=(IS-1)*16+I+1
   IF(IS-COMM(I))10,11,100
10 IF2=IF1
   K2=K1+16-I
   GO TO 200
11 IF2=IF1+10
   K2=I
   GO TO 200
100 CONTINUE
C...Check if sector number is too big
999 CALL ERR(30)
   IERFL=2
200 RETURN
END

**ERR**

SUBROUTINE ERR(K)
INTEGER WESS(10)
READ(*K)WESS
WRITE(1,500)K,WESS
900 FORMAT(13,2X,10A2)
RETURN
END
** EXPAN

* CALL EXPAN(IW,IR) PUTS THE BITS OF IW INTO THE 16 POSITION ARRAY IR (FORTRAN ORDER)

ENT EXPAN

EXPAN DC **
LDX 216
LDX II EXPAN
LD 11 0
SRX 16
LD L1 1
STX L 1
SLA 16
PLACE IN EXTENSION
STORE IT IN XR1
ZERO ACCUMULATOR

SHIFT SLT 1
STX L1 0
SLA 16
ZERO ACCUMULATOR
MDX 1 -1
MDX 2 -1
MDX SHIFT
MDM EXPAN +2
BSC I EXPAN
END
** FIELD 

SUBROUTINE FIELD(LINE,1,NF)  
INTEGER COMM(165),LINE(78)  
EQUIVALENCE (COMM(141),IEFF)  
CALL COMM  
C***RETURN -1 IF CHARACTER ISN'T F. ELSE RETURN NUMBER  
C***FILLING F IF VALID  
IF(LINE(I)-IEFF)10,20,10  
10 NF=1  
15 RETURN  
20 K=1  
C***CHECK NEXT CHAR. IF NUMERIC BREAK OFF NUMERAL AND ACCUMULATE VAL.  
23 LK=LINE(K)  
26 K=IGVB(LK,0,3)  
27 I=IGVB(LK,1,8)  
28 I=IGVB(LK,2,7)  
29 I=IGVB(LK,3,7)  
30 NF=NF*10+I  
GO TO 20  
30 I=K-1  
GO TO 15  
END  

** GFLD  

FUNCTION GFLD(I)  
INTEGER COMM(165),FLPTR(4),STURC(340),K  
EQUIVALENCE (COMM(141),IFP),FLPTR,STURC,IERFL  
CALL COMM  
C***IERFL IS LEFT WITH VALUE 2 IF IT FAILS  
GFLD=0.0  
C***GET WORD AND BIT POINTERS. IFPTR PUTS THEM INTO FLPTR  
CALL IFPTR(I)  
GO TO (10,40),IERFL  
10 IF(IFWD-ILWD)*IFBT-31)20,16,16  
16 I=15-IFBT  
17 M=IGVB(STURC(IFWD),IFBT,15),IGVB(STURC(IFWD+1),0,IFBT-1),  
+ 0,II)  
18 M=IGVB(STURC(IFWD+1),IFBT,15),IGVB(STURC(ILWD),0,ILBT),0,II)  
GFLD=X  
GO TO 40  
20 K=14-ILBT  
GFLD=IGVB(STURC(IFWD),IFBT,15),0,0,K1)  
+IGVB(STURC(ILWD),0,ILBT)  
RETURN  
30 GFLD=IGVB(STURC(IFWD),IFBT,ILBT)  
40 RETURN  
END
** GVBLF**

SUBRUTINE GVBLF(I1,KARY,KK)
C..II TELLS WHICH SUBFIELD
C..KARY IS RETURN ARRAY
C..Kk IS DIV. CF KARY. IT RETURNS AS THE LENGTH OF SUBFIELD.
C..C..DONT CALL WITH KK AS A CONSTANT. KARY IS BLANKED IF SUBFIELD
C..IS NOT PRESENT.
INTEGER KARY(I),CHAR(40)
INTEGER FLPTR(4)
INTEGER CCWY(165),STURC(340),BUFF(41)
COMMON CCWY,FLPTR,STURC,IERFL,BUFF
EQUIVALENCE (ICW14,COMM(14)),(ICW15,COMM(15)),(CHAR(1),COMM(126))
EQUIVALENCE (L,KSTAT)
DATA KBLNK/* */
C..SEE IF VEL FIELD IS PRESENT
KSTAT=STURC(I)
IF(IGVE(KSTAT,1,1))10,10,20
10 DO 1 IS=1,KK
15 KARY(I)=KBLNK
KK=0
RETURN
20 KDISP=ICW14
C..SEE IF CPT. BLOCK IS PRESENT
IF(IGVE(KSTAT,0,0))40,40,30
30 KDISP=KDISP+ICW15
C..LENGTH OF VEL FIELD, BEGINNING, AND END
40 L=FLD(L)
KEND=KDISP+L
KDISP=KDISP+1
C..START EXAMINING SUB-FIELDS
45 L=STURC(KDISP)
NSUB=IGVE(L,0,7)
LSUB=IGVE(L,8,15)
IF(NSUB-11)50,60,50
50 KDISP=KDISP+LSUB+1
IF(KDISP=KEND)45,10,10
C..MAKE SURE KARY IS NOT OVERFLOWED
60 L=LSUB+3
IF(L-KK)70,70,65
65 LSUB=KK/3
GO TO 71
70 KK=L
71 CALL ASA1(STURC,KDISP+1,LSUB+KDISP,KARY,1,CHAR)
75 RETURN
END
** HASH1

SUBROUTINE HASH1(KR, IRY, IR, IDIV, KEY)
  DIMENSION IRY(2)
  KEY determines whether this is a PUT, GET or GET CONTINUED
  GO TO (1, 2, 12), KEY
  1 KSUB = 0
     GO TO 3
  2 KSUB = KR
  3 K = 0
     KRR = IABS(KR)
     CALL MOD(KRR, IO, IR, IDIV)
     IF(IO + 1 - IDIV) = 6, 5, 6
  5 IO = 0
  6 K = K + 1
  7 IF(IRY(IR) - KSLB) = 4, 99, 4
  8 IF(KEY = 1) = 10, 8, 10
  9 IR = 0
  99 RETURN
  10 IF(K = 35) = 12, 12, 11
  11 IR = -1
     GO TO 99
  12 CALL MOD(IR + IO, IP, IR, IDIV)
     GO TO 6
END

** HSH1

SUBROUTINE HSH1(KR, IR, IDIV, KEY, CSPTR)
  INTEGER CSPTR(4)
  KEY = 1, 2, 3 FOR PUT, GET, OR GET CONTINUED
  GO TO (1, 2, 12), KEY
  1 KSUB = 0
     GO TO 3
  2 KSUB = KR
  3 K = 0
     KRR = IABS(KR)
     CALL MOD(KRR, IO, IR, IDIV)
     IF(IO + 1 - IDIV) = 6, 5, 6
  5 IO = 0
  6 K = K + 1
  7 READ(3*IR) KIC, CSPTR
     IF(KIC = KSUB) = 10, 99, 10
  99 CSPTR(4) = IVE(CSPTR(3), 2, 9)
     CSPTR(3) = IPVE(IVE(CSPTR(2), 9, 15), 0, 7, 13) + IVE(CSPTR(3), 0, 1)
     CSPTR(2) = IVE(CSPTR(2), 3, 8)
     RETURN
  10 IF(K = IDIV) = 12, 12, 11
  11 IR = -1
     GO TO 99
  12 CALL MOD(IR + IO, IP, IR, IDIV)
     GO TO 6
END
** HSHF2

SUBROUTINE HSHF2(N,NP,IR,IDIV,KEY,IPTR)
C***THIS ACCESSES THE MASTER INDEX. KEY=1,2,3 FOR PUT OR GET
K=0
X=N*1000.0+N
CALL MDDX(X,IC,IR,IDIV)
7 GO TO (1,2),KEY
1 XSSUB=O
GO TO 3
2 XSSUB=X
3 IF(IIC+1-IDIV)*5*6
5 IO=O
6 REAC(5*IR)M*,NN,**111IPTR
Y=M*1000.0+N
GO TO (18,17),KEY
17 IF(MM)30,999,30
18 IF(X-Y)25,29,25
25 IF(MM)99,30,30
29 CALL ERR(24)
GO TO 999
30 K=K+1
6 IF(Y-XSUB)10,99,10
9 RETURN
10 IF(K-35)12,12,999
12 CALL MOD(IIR+IC,IP,IR,IDIV)
GO TO 6
999 IR=-1
GO TO 99
END
**HSHIX**

SUBROUTINE HSHIX(INX, IDIV, KEY)
C... ASCII ALPHABET BACKUP FILE FOR MOVE INDEX IN GARBG
C... KEY=1 MEANS PUT ENTRY INTO INDEX. KEY=2 MEANS FIND AN ENTRY
C... INTEGER MINX(2), MV(2), IFLE(2), 160), HI
C... EQUIVALENT(MV(1), M), MV(2), N)
DATA LC=H1/0, 0/
C... SET POINTERS FOR IN CORE PART OF MOVE INDEX IF THIS IS FIRST CALL
IF (LO) 101, 101, 102
101 LO=1
HI=160
READ(9, LC) IFLE
102 IF (MINX(1)) 50, 50, 55
50 MINX(2)=0
GO TO 99
55 K=0
MINX(1) CALL MCD(KK, IC, IR, IDIV)
7 GO TO (12, 2), KEY
8 KSUB=0
GO TO 3
2 KSUB=K
3 IF (IG+1, IDIV) 6, 5, 6
5 IG=0
6 IG=1
GO TO 500
C... IF KEY IS A RECORD THAT WAS NOT MOVED... SET ITS NEW VALUE TO BE SAME
17 IF (MINX) 11, 18, 30
18 IF (KX=MV) 30, 29, 30
29 CALL ERR(24)
GO TO 999
30 KX=K
IF (KSUB=10, 97, 10
97 GO TO (95, 199), KEY
98 IG=2
GO TO 500
199 MINX(2)=N
99 RETURN
10 IF (K=IDIV) 12, 12, 11
C... IF POINTERS NOT IN INDEX LEAVE IT UNCHANGED WHEN KEY =2
11 GO TO (999, 1100), KEY
110 MINX(2)=MINX(1)
GO TO 99
12 CALL MCD(IR+IG, IP, IR, IDIV)
GO TO 6
999 KEY=-1
GO TO 99
500 IF (LC-IR)*(IR-HI) 210, 200, 200
200 =IR-LC+1
GO TO (204, 206, 10
204 K=IFLE(1, 1)
K=IFLE(2, 1)
GO TO (18, 17), KEY
206 IFLE(1, 1)=MINX(1)
IFLE(2, 1)=MINX(2)
GO TO 99
210 WRITE(9, LC) IFLE
LC=(IR-1)/160*160+1
HI=LO+159
READ(9, LC) IFLE
GO TO 500
END
** IFPTR **

SUBROUTINE IFPTR(I)
INTEGER COM(165),FLPTR(4),STURC(340)
COMMON COM,FLPTR,STURC,IERFL
EQUIVALENCE (STURC(1),IST),(STURC(2),IST2)
EQUIVALENCE (FLPTR(1),IFW0),(FLPTR(2),IFBT),(IFIX,COMM(19)),
(FLPTR(3),ILW0),(FLPTR(4),ILBT),(IOP,T,COMM(20)),
(PFXW,COMM(14)),(IDPW,COMM(15)),(NFVBL,COMM(21)),
(NFVBL,COMM(16))
***PRODUCE POINTER TO FIELD(1) OF MASTER RECORD
C..ROUTINE SHOULD RETURN FIRST WORD, FIRST BIT, LAST WORD, LAST BIT,
C..IN ARRAY FLPTR IN COMMON. IERFL IS 2 IF IT FAILS.
IF(IP=IGVBL(IST,0,0))
MFIC=FIX+1CPT
IF(IFIX)100,100,11
11 IF(I-FPIC)15,15,200
C..CHECK TO SEE IF CPT, BLOCK IS PRESENT
15 IF(IP)201,201,100
100 IF=IFIX
101 IFRST=CCMM(I)
ILAST=CCMM(I+1)-1
IF(IFRST-ILAST)105,105,102
102 IFCT=16
RETURN
105 IFWD=IFRST/16+1
ILWD=ILAST/16+1
110 IFBT=IFRST-(IFWD-1)*16
ILBT=ILAST-(ILWD-1)*16
RETURN
200 MFIC=MFIC+1
IF(I-MFIC)201,201,210
C..COPS, TRIED TO GET VBL FIELD
201 IERFL=2
RETURN
C..LOCKING AT FIELDS IN VBL BLOCK, FIRST COMP, START WORD OF BLOCK
210 IVST=PFX*ICPT*IP+IGVBL(IST,8,14)
C.. WHICH FIELD GROUP
II=IF-WFIC1
III=II/FFVBL
IV=II-II*FFVBL
IF(III)215,215,220
215 IIII=NFFVBL
C..NOW IVST POINTS TO BEG, OF GROUP OF FIELDS
C..III POINTS TO WHICH FIELD IN THE GROUP
220 KCPT=MFI+II+50
IVST=IVST+II*FFVBL*16
IFRST=CCMM(KCPT)+IVST
ILAST=CCMM(KCPT+1)-1+IVST
GO TO 105
END
** IPVB  IGVB

ENT  IGVB
ENT  IPVB
* CALL K=IGVB(IW,IF,IL)
* RETURNS THE VALUE OF BITS IF THRU IL INCLUSIVE
IGVB  DC
  LDX II  IGVB
  LD II  I
  STO  SLA
  SET UP SHIFT LEFT
  LD  S
  I2
  FOR
  A
  II
  SHIFT
  A
  SR
  TO THE
  STO  SRA
  RIGHT
  LD  II  0
  LD WORD WITH DESIRED BITS
SLA  NOP
SRA  NOP
ESC  L  IGVB,63
* CALL K=IPVB(IVAL,IW,IF,IL)
* RETURNS THE VALUE OF IW WITH BITS IF THRU IL
* REPLACED BY IVAL.  IW IS UNCHANGED.
IPVB  DC
  LDX II  IPVB
  LD II  1
  LOAD WORD TO BE FILLED
  STO  IW
  LD II  2
  LOAD FIRST BIT POINTER
  STO  IF
  LD II  3
  LOAD LAST BIT POINTER
  STO  IL
  LD  S
  FIFTH
  SET UP
  S
  IL
  LENGTH OF
  STO  RT
  RIGHT SHIFT.
  LD II  1
  IF
  S
  SIZE
  STO  A
  OF
  STO  SIZE
  BIT STRING.
  LDX  SIZE
  SET XR2 TO USE IN ERR CHECK.
  LD II  0
  LOAD VALUE TO BE PUT IN.
  STO  IVAL
  SLCA  2
  CHECK
  LD  SIZE
  MAGNITUDE
  S
  L
  2
  OF
  BNN  OK
  IVAL.
CALL ERR  IF IVAL TOO BIG BRANCH
DC  K36
TO ERR ROUTINE, THEN RETURN
LD  IW
ORIGINAL VALUE OF IW.
B
RET
* IF ERROR CHECK IS NOT WANTED TAKE OUT FROM
* LDX  2 16  TO HERE
* IF ERR ROUTINE IS USED, THE ABOVE ASSUMES
* CNE ARGUMENT.
OK  LD  IW
LOAD THE WORD TO BE CHANGED
LDX  12  RT
XR2 USED FOR SHIFT COUNT
SRT  2 0
ANY RT. HAND BITS TO BE
*  LDX  12 SIZE
XR2 USED FOR SHIFT TO ZERO
SRA  2 0
OUT BITS TO BE CHANGED
SLA  2 0
OR  IVAL
INSERT NEW BITS
LDX  12 RT
XR2 USED AGAIN FOR SHIFT
SRT  2 0
COUNT TO GET RT. HAND BITS BACK
RET  MDX L  IPVB
SET UP RETURN ADDRESS
ESC  1  IPVB
SR  SRA
SL  SLA
CNE  DC
1
FIFTHN  DC  15
IF  DC
** INVRT **

SUBROUTINE INVRT(IFIL, IPT, ISZ, CLDPT, NEWPT, MPT)

INTEGER CLDPT, MPT(1)
INTEGER CCMV(165), FLPTR(4), STURC(340), BUFF(41)
COMMON CCMV, FLPTR, STURC, IERFL, BUFF

EQUIVALENCE (CCMV(30), ICM30), (CCMV(37), ICM37)

C***IFIL IS THE FILE WHOSE INVERTED LIST IS TO BE CHANGED
C***IPT IS THE POINTER TO THE INVERTED LIST FOUND IN CAT. OR DSC FILE
C***ISZ IS THE SIZE OF THE ARRAY MPT. CORR. TO RECORD SIZE
C***CLDPT IS THE PTR VALUE TO BE CHANGED
C***NEWPT IS THE NEW PTR TO BE SUBSTITUTED FOR CLDPT.

IF CLDPT=0 WE ARE INSERTING A NEW PTR.
C***IF NEWPT=0 WE ARE DELETING AN OLD PTR.
78 READ(IFIL,IPT)(MPT(I), I=1, ISZ)
80 CONTINUE
IF(KPT) = CLDPT, 80, 85, 80
79 IPT = CLDPT
GO TO 78
82 IF(NEWPT) = 0, 83, 90, 83
83 CALL ERR(53)
IERFL = 2
RETURN
C***INSERTING NEW ENTRY. NEED ANOTHER BLOCK OF SPACE
C***CCMV(30) OR (37) POINTS AT THE NEXT AVAILABLE BLOCK.
90 K = IFIL + 1
GO TO (91, 92) * K
91 KIVAL = ICM30
GO TO 95
92 KIVAL = ICM37
95 IF(KIVAL) = 0, 96, 100
100 WRITE(IFIL, IPT) KIVAL
READ(IFIL, IPT) KIVAL
GO TO (101, 102) * K
101 ICM30 = KIVAL
GO TO 103
102 ICM37 = KIVAL
GO TO 103
103 DO 105 I = 1, ISZ
105 KPT(I) = 0
1 = 2
85 MPT(I) = NEWPT
WRITE(IFIL, IPT)(MPT(I), I=1, ISZ)
GO TO 84
END
**IPFLD**

SUBROUTINE IPFLD(K,X,I)

INTEGER COMK(165),FLPTR(4),STURC(340)

INTEGER S1,S2

COMMON COMK,FLPTR,STURC,IERFL

EQUIVALENCE (MFIX,COMM(19),(IOPT,COMM(20)),(MFXW,COMM(14)),
+ (IDPX,COMM(15)),(NFVEL,COMM(21)),(NVVEL,COMM(16)),
+ (FLPTR(1),IFWC),(FLPTR(2),IFBT),(FLPTR(3),ILWD),
+ (FLPTR(4),ILBT),(S1,STURC(1)),M2,Z,M2,M2,M2)

C...PUTS THE VALUE K OR X INTO FIELD I DEPENDING ON WHETHER

C...FIELD I IS 16 BITS OR LESS OR IS 32 BITS.

C...IERFL IS 2 IF IT FAILS

C...GET WORD AND BIT POINTERS

CALL IPFTR(1)

IF(IFBT-16)=999,999

5 GO TO(10,95),IERFL

10 IF(IFWD-ILWD)=11,20,20

11 IF((ILWD-IFWD)*16+ILBT-IFBT-31)=12,35

12 Z X

IF(IFBT=14,14,15

14 STURC(IFWD)=M2

STURC(ILWD)=M1

GO TO 25

15 IF(15-IFBT)

STURC(IFWD)=IPVB(IGVB,M2,0,11,STURC(IFWD),IFBT,15)

STURC(IFWD)=IPVB(IGVB,M2,161,15,IGVB,M1,0,11,0,IFBT-1)

STURC(ILWD)=IPVB(IGVB,M1,161,15,STURC(ILWD),0,ILBT)

GO TO 25

20 IF(IFBT-16)=21,999,999

21 STURC(IFWD)=IPVE(K,STURC(IFWD),IFBT,ILBT)

GO TO(999,999),IERFL

C...SET ERR OR FLAG IN RECORD

99 S1=IPVE(1,S1,7,7)

999 RETURN

30 K1=ILBT-IFBT+17

IF(K1-16)=32,40,40

32 IF(K-2=K1)=40,40,35

35 CALL ERR(17)

WRITE(1,900)K,1

900 FORMAT(216)

GO TO 99

40 K2=15-ILBT

STURC(ILWD)=IPVB(IGVB(K,K2,15),STURC(ILWD),0,ILBT)

K1=16-K1

K2=K2-1

STURC(IFWD)=IPVE(IGVB(K,K1,K2),STURC(IFWD),IFBT,15)

GO TO 25

END
FUNCTION IRLVL
REAL VAL(25)
INTEGER TCRTI
INTEGER IVAL(50), CRIT(12), OP(4)
INTEGER COMM, FLTTR, STURC, IERFL, RP, CRIT, IVAL
EQUIVALENCE COMM(21), ICN(21), COMM(20), ICM(20), COMM(19), ICM(19)
6 COMM(29), ICN(29), STURC(1), STUR1
C...RETURNS 1 CR 0 DEPENDING ON WHETHER THE CRITERION IS SATISFIED OR NO.
C...PNTS TO CRITERION DESCRIPTION IN CRIT
C...VAL AND IVAL HOLD THE COMPARISON EXPRESSIONS
EQUIVALENCE VAL(1), IVAL(2)
IF(y-100) 1, 28, 26
1 TCRTI CRIT
KF IGVB CRIT, 0.5
KR IGVB CRIT, 0.8
KP IGVB CRIT, 1.15
KT IGVB CRIT, 3.01
C...FIRST GET VALUE OF FIELD. IF FIELD IS NON-EXISTENT, VALUE OF
C...FUNCTION IS 0. IF A NUMERIC FIELD, FIRST FIND OUT
C...WHETHER FIELD IS IN MAND.*, OPT.*, VBL FIELD OR VBL BLOCK.
POF=ICM(19), ICM(20)
IF KP - IF (ICM(19)) 30, 30, 5
5 ISF=KF-IGVL
IF ISF 20, 20, 10
10 KSF=KF-ICM(29)
IF KSF 40, 40, 60
C...IS OPT BLCK PRESENT
20 IF IGVB STUR1, 0.40 25, 25, 30
25 IF KP - 6 28, 28, 28
26 IRLVL 1
27 RETURN
28 IRLVL 0
GO TO 27
30 IF KP - 6 31, 26, 31
31 X GFLD KF
GO TO 100
C...IS F THE SUBFIELD IN THE VBL FIELD
C...IF VBL FIELD NOT PRESENT LENGTH WILL BE ZERO
LK RETUNS AS LENGTH OF SUBFIELD BUT NOT
C...GREATER THAN PRESET LENGTH OF KARY
40 LK=20
CALL GVBLF(ISF, KARY, LK)
IF(LK) 28, 28, 300
C...FIELD IS IN THE GROUPS IN THE VBL BLOCK, CALCULATE
C...REDUCED POINTER GET NO. OF GROUPS
50 KSF=KSF/ICM(21), ICM(21)
NGPS=GFLD(A)
GO TO 400
C...NOW GET COMPARISON VALUE, FIRST CHECK THAT TYPE
C...IS RIGHT, KT SHOULD BE 1, 2, OR 3 FOR NUMERIC
100 GO TO(115, 115, 120, 120, 105), KT
105 CALL ERR(55)
GO TO 28
C...INTEGER VALUE
C...REAL MODE VALUE
115 KP=(KP+2)/2
XX=VAL(KP)
GO TO 200
C...FIELD EXPRESSION, POINTER INDICATES BEGINNING OF EXPRESSION
C...IN IVAL, EXPRESSION FORM IS AS FOLLOWS—AN OPERATON IS GIVEN
C...FCLLCWD BY EITHER A FIELD OR A NUMERIC VALUE. OPERATIONS ARE
C.../^/*CCOED AS 1, 2, 3, OR 4, RESP. IF THE OP IS FOLLOWED BY A
C...FIELD, THE CP CODE IS POSITIVE. IF A NUMERIC FOLLOWS, THE OP IS
C...NEGATIVE. AN CP OF ZERO ENDS THE EXPRESSION.
C...EVALUATION IS STRICTLY LEFT TO RIGHT.
120 XX=0, 0
125 IOP=IVAL(KP)
IF(IOP) 130, 200, 140
C•••NUMERIC VALUE
  130  IOP=-ICP
  131  KP=KP+4
  132  K=KP/2
  133  XK=IVAL(K)
  134  GO TC 150
C•••FIELD VALUE
  140  XK=GFIELD(IVAL(KP+1))
  141  KP=KP+2
  142  GO TC(155,160,165,170),ICP
  143  150  XX=XX+XK
  144  GO TC 125
  145  160  XX=XX-XK
  146  GO TC 125
  147  165  XX=XX+XK
  148  GO TC 125
  149  170  XX=XX/XK
  150  GO TC 125
C•••COMPARE VALUES, NUMERIC
  150  IF(X-XX)330,325,340
C•••COMPARE STRINGS. LK IS LENGTH FIELD STRING.
C•••LENGTH OF TEST STRING IS IN IVAL(KP)
  200  LT=IVAL(KP)
C•••COMPARE STRINGS. LK IS LENGTH FIELD STRING.
  200  IF(LK-LT)305,310
  201  305  LT=LK
  202  310  DD 320  I=1,LT
  203  320  IF(KARY(I)-IVAL(II))330,320,340
  204  320  CONTINUE
  205  325  GO TC(28,26,26,26,26,26,26,KR
  206  330  GO TC(28,26,26,26,26,26,26,26,KR
  207  340  GO TC(28,26,26,26,26,26,26,26,26,26,26,26,KR
C•••CHECK FCR FIELD IN EACH GROUP IN VBL BLOCK
C•••KSF=KSF+1=MCFI
  250  DD 500  I=1,6EPS
  251  IF(X=GFLD(KSF))500,26,500
  252  500  KSF=KSF+1=CM21
  253  GO TC 28
END
** IRTRV

SUBROUTINE IRTRV(ISW, I, KEY)
INTEGER HL, HH, HBUFF(200), LBUFF(200)
DATA LL, LH, HL, FH/32000, 32000, 0, 0/ IN/1/
C... THIS IS A BUFFER ROUTINE TO ACCESS THE SET OF POINTERS PUT ON FILE
C... BY SLECT AND USED BY LISTS. A RANG, STATS. KEY=1, 2, OR 3 FOR
C... GET, PUT OR RETURN BUFFER TO FILE. BUFFER IS IN
C... The PARTS FOR MORE EFFICIENT USE IN A RANG
GO TO (10, 10, 31), KEY
10 IF(LL-I)12, 15, 30
12 IF(LL-I)35, 15, 15
15 J=I-LL+1
GO TO (20, 25), KEY
20 LBUFF(J)=ISW
GO TO 99
25 ISW=LBUFF(J)
GO TO 99
C... NEED NEW LOW BUFFER. PUT PRESENT ONE BACK FIRST.
30 GO TOC(32, 31), IN
31 WRITE(9'LL)LELF
GO TOC(32, 32, 60), KEY
32 LL=(I-I)/200*200+1
LH=LL+195
GO TOC(34, 33), IN
34 HL=LL+200
MH=LH+200
33 REACC(9'LL)LBUF
GO TC(62, 10), IN
C... NEED NEW HIGH BUFFER. PUT PRESENT ONE BACK FIRST
60 WRITEC(9'HL)HBUF
GO TC(61, 55), KEY
61 HL=(I-I)/200*200+1
IN=2
MH=HL+199
62 REAC(9'HL)HBUF
GO TC 10
35 IF(HL-I)40, 45, 30
40 IF(HL-I)60, 45, 45
45 J=I-HL+1
GO TC(50, 55), KEY
50 HBUF(J)=ISW
GO TOC 99
55 ISW=HBUF(J)
99 RETURN
END
*ONE WORD INTEGERS
** LADCH

SUBCUrine LADCH
INTEGER ADVNM(10), CHAR(40)
INTEGER COMV(165), STURC(340), BUFF(41), FLPTR(4)
COMVCM COMV, FLPTR, STURC, IERFL, BUFF
EQUIVALENCE(COMV(27), ICY(27))
EQUIVALENCE(FLPNTM, COMV(39))
EQUIVALENCE(CHB(1), COMV(126))

C***PROC: CE A LIST OF PRIME KEYS OF RECORDS IN CATEGORIES
1 CALL ERR(62)
READ(6,901)IAN
901 CONTROL(11)
IF (IAN - 3)5,5,1
5 GO TO (10,50,99), IAN
10 CALL ERR(60)
READ(6,903)N
903 CONTROL(13)
NUM1=N
NUM2=N
GO TO 60
50 NUM1=1
NUM2=1
60 GO 100 J=NUM1, NUM2
READ 2 J IPN1, NUM1, ADVNM
WRITE(NP1,904 IPN1, NUM1, ADVNM)
904 FORMAT(1X, 2I4, 1X, 10A2)
64 IF IPN1 100, 100, 65
65 READ 2 IPN1, ADVNM
DO 90 I = 1, 11
IF ADVNM I 90, 90, 70
70 CALL RMASM(ADVNM(I))
N=GFHD(5)
N=GFHD(6)
KN=GFHD(7)
WRITE(5, 905) N, NN
905 FORMAT(1X, 3I3)
90 CONTINUE
IF IPN1 100, 100, 65
100 CONTINUE
GO TO 1
99 RETURN
END
** LEARF

SUBROUTINE LEARF
INTEGER COMM(165),STURC(340),BUFF(60),FLPTR(4)
COMMON COMM,FLPTR,STURC,IERFL,BUFF
EQUIVALENCE (NPRNT,COMM(35))
COMMON LIST FILES USED FOR AUXILIARY PURPOSES
1 CALL ERR(58)
READ 6,903 IFL
903 FORMAT 11
IFL IFL-5
9 RETURN
5 IF IFL-4 6,6,1
6 GO TO 600,700,800,9000, IFL
600 GO 42 I=1,42
READ(6*1)(BUFF(J), J=1,15)
WRITE(NPRNT,910)(BUFF(J), J=1,15)
910 FORMAT IX+15A2
42 CONTINUE
GO TO 1
700 GO 70 I 1,40
READ(7*1)BUFF
70 WRITE NPRNT,911 BUFF
911 FORMAT (IX+15I8,7))
GO TO 1
800 GO 80 I 1,64
READ(8*1)(BUFF(J), J=1,10)
80 WRITE(NPRNT,912)(BUFF(J), J=1,10)
912 FORMAT IX+10A2
GO TO 1
900 GO 20 I 1,50
READ(9*1)KA,KB,KC,KD
20 WRITE 5,900 KA,KB,KC,KD
900 FORMAT IX+4A2
GO TO 1
END
**LISTM**

SUBROUTINE LISTM

INTEGER ST1, DCLSL
INTEGER COMO(165), FLPTR(4), STURC(340), OP(4), CRIT(12), IVAL(50), SPTR
COMMON CCMM, FLPTR, STURC, IERFL, OP, CRIT, IVAL, SPTR
EQUIVALENCE (STURC(1), ST1), (COMM(125), DCLSL)

C...LIST RECORDS WHOSE POINTERS ARE IN WORKING SET
C...NEED TO ADD FACILITY TO LIST SPECIFIC FIELDS. USE CRIT AND IVAL

IF(SPTR)4,4,5
IF(SPTR)4,4,5
4 CALL ERR(57)
GO TO 99
5 DO 100 I=1,SPTR
CALL ITRV(ISCW0, I+2)
IF(ISCW0)10,100,10
10 CALL RMAVR(ISCW0)
IF(ST1=DCLSL)20,15,20
15 CALL ERR(6)
GO TO 100
20 CALL LISTM
100 CONTINUE
99 RETURN
END
**LCCGP**

FUNCTION LCCGP(I1,I2,I3)
INTEGER CCMX(165),FLPTR(4),STLRC(340)
CCMIX CCMX,FLPTR,STLRC,IERFL
EQUVALENCE (CCCMX(20),ICM20),(CCCMX(19),ICM19),(CCCMX(21),ICM21)
LCCGP=0
IF((CCMIX(15)+ICM20)+2)
IG=ERFL(4)
IFLGP=IF((IG-1)*ICM21)
IF(IIFLP(IFGFLP)S,10,10
RETURN
10 IF(I1-GLDF(IFLGP))TO,20,20
15 IF(I2-GFSL(IFLGP+1))TO,20,30
20 IF(I3-GFSL(IFLGP+2))TO,25,30
25 LCCGP=IFLGP
RETURN
30 IFLGP=IFLGP+ICM21
GC TC 1
END
**LPADV**

SUBROUTINE LPADV
INTEGER ADVSR(12)
INTEGER COMM(165),STURC(340),BUFF(41),FLPTR(4)
COMMON COMM,FLPTR,STURC,IERFL,BUFF
EQUIVALENCE (COMM(27),ICP27)
EQUIVALENCE (IPRT,COMM(35))
EQUIVALENCE (IPRT,ADVSR(1))
C.**LIST.CR PATCH CATEGORY FILE**
1 CALL ERR(56)
   READ(6,91))I
91 FORMAT(I1)
   GO TO (10,.100,99),I
10 CALL ERR(54)
   READ(6,91))I
   GO TO (11,.12),I
11 CALL ERR(52)
   READ(6,94))NI
   N2=NI
   GO TO 15
12 N1=1
   N2=ICP27
15 DO 50 I=M,N2
   READ(2*I)ADVSR
   IF(IPRT)50,50,30
   WRITE(IPRT,50)ADVSR
90 FORMAT(1X,2I4,1X,10A2)
30 READ(2*IPRT)ADVSR
   WRITE(IPRT,56)ADVSR
96 FORMAT(1X,12I7)
   IF(IPRT)50,50,35
50 WRITE(IPRT,97)
97 FORMAT(/)
   GO TO 1
99 RETURN
100 CALL ERR(51)
   READ(6,91))KK
   CALL ERR(50)
   READ(6,94))K
94 FORMAT(13)
   READ(2*K)ADVSR
   GO TO (101,.201),*KK
101 WRITE(1,95)ADVSR
95 FORMAT(2I4,1X,10A2)
   READ(6,95)ADVSR
202 WRITE(2*K)ADVSR
   GO TO 1
201 WRITE(1,96)ADVSR
   CALL ERR(49)
   READ(6,94)IF,IVAL
   ADVSR(IF)=IVAL
   GO TO 202
END
** LPCOM**

SUBROUTINE LPCOM
INTEGER CCMM(165),STURC(340),BUFF(41),FLPTR(4)
CCMM, CCMM, FLPTR, STURC, IEREF, BUFF
EQUIVALENCE NPRNT, COMM 35
C-COMPUTE CR PATCH PARAMETER FILE
READ(4*1)CCMM
1  CALL ERR(S6)
   READ(6,900)IGC
909 FORMAT(11)
   IF(IGC)1,1,2
2   IF(IGC-3)3,1
   CONTINUE
   GO TO(10,100,999), IGO
10  WRITE(NPRNT,910)CCMM
910 FORMAT(IX,10I10)
   GO TO 1
100 CONTINUE
   CALL ERR(47)
   CALL ERR(48)
   READ(6,91)K, IV
91 FORMAT(14)
   IF(K)5,999,5
   CCMM(K)=IV
   GO TO 1
999 WRITE(4+1)CCMM
RETURN
END
**LPCRS**

**SUBROUTINE LPCRS**

```fortran
INTEGER CB(30)
INTEGER CCMV(165),STURC(340),BUFF(41),FLPTR(4)
C EQUIVALENCE(C(1),BUFF(1)),(KID,BUFF(1)),(KCH,BUFF(2))
EQUIVALENCE(CCMV(24),IDIV),(IAN,BUFF(31)),(KSTAT,BUFF(32)),
(KD,BUFF(33)),(KN,BUFF(34)),(KS,BUFF(35)),
BUFF(6),IBF6),(BUFF(7),IBF7),(BUFF(8),IBF8),
BUFF(9),IBF9)
```

```fortran
C ...LST CR PATCH DESCRIPTORS. NOT GENERALIZED YET
1 CALL ERR(56)
READ(6,901)IAN
901 FORMAT(I11)
IF(IAN)11,12
2 IF(IAN-3)5,1
5 GO TO (10,100,999),IAN
10 CONTINUE
CALL ERR(46)
READ(6,901)IAN
IF(IAN)2,6,8
6 WRITE(1,906)
906 FORMAT('TYPE INCLUSIVE RECORD NOS. IN 213*)
READ(6,908)11,12
908 FORMAT(213)
GO TO 11
8 I=1
12=IDIV
11 DO 20 I=11,12
READ(3*I)CB
IF(KIC)15,20,15
15 KSTAT=IGVB(IBF3,0,2)
KD=IGVB(IBF3 ,3,8)
KN=IGVB(IBF3 ,9,15),IGVB(IBF4 ,0,1),7,13
KS=IGVB(IBF4 ,2,9)
KCC=IGVB(IBF4 ,10,15)
KDT=IGVB(IBF5 ,0,7)
KNB=IGVB(IBF6 ,8,15)
KMX=IGVB(IBF6 ,0,7)
KH=IGVB(IBF6 ,8,11)
KINST=IGVB(IBF7 ,0,2)
IAN=IGVB(IBF6 ,12,15)
KINST=IPVB(IAN,KINST,9,12)
KBLG=IPVB(IBF7 ,3,15),IGVB(IBF8 ,0,2),0,12
KRD=IPVB(IBF7 ,3,15),IGVB(IBF9 ,0,2),0,12
KEX=IGVB(IBF9 ,3,15)
WRITE(5,902)KID,KCH,KSTAT,KD,KN,KS,KCC,KDT,KNUM,KMX,KH,KINST,
KBLG,KRD,KEX,(BUFF(J),J=6,29)
902 FORMAT(/1X,1316,2A2,16/*/1X,8A2,2X,8A2,2X,8A2)
16 READ(3*KCH)CB
WRITE(5,905)CB
905 FORMAT(1X,1*E15)
17 IF(KID)120,20,17
120 KCH=KID
GO TO 16
20 CONTINUE
GO TO 1
100 CALL ERR(44)
READ(6,904)IR,IV
904 FORMAT(316)
READ(13*IR)CB
WRITE(3*IR)CB
100 CONTINUE
GO TO 1
999 RETURN
END
```
**LPIX

SUBROUTINE LPIX
INTEGER COMW(165), STURC(340), BUFF(41), FLPTR(4)
COMON, COMW, FLPTR, STURC, IERFL, BUFF
EQUIVALENCE (COMW(7), ICM7)
EQUIVALENCE (NPRNT, COMW(35))

C...LIST CR PATCH MASTER INDEX
1 CALL ERR(56)
   READ(6,930)IAN
930 FORMAT(11)
   GO TO (10,100,999), IAN
10 DO 50 I=1,ICM7
   READ(5*1),M,N,N,ISW,NKEY
   IF(M)727,50,727
727 WRITE(NPRNT,900),M,N,N,N,N,N,N,N
900 FORMAT(1X14,1X,313,2X,I6,17)
   CONTINUE
   GO TO 1
100 CALL ERR(AJ)
   READ(6,905)M,N
905 FORMAT(213)
   IF(M)101,100,905
101 IF(M)105,1,101
   READ(5*1),M,N,N,N,ISCWD,NKEY
   IF(M)=M)150,110,150
110 IF(N)150,160,150
   CONTINUE
   WRITE(1,906)
906 FORMAT(*NCT FOUND*)
   GO TO 160
160 WRITE(1,907),M,N,N,ISCWD,NKEY
907 FORMAT(*RECORD CURRENTLY*313,2X,16,17,3X,'TYPE NEW RECORD DIRECTLY
+ UNDERNEATH*/)
   READ(6,908),M,N,N,ISCWD,NKEY
908 FORMAT(313,2X,16,17+)
   WRITE(5*1),M,N,N,ISCWD,NKEY
   GO TO 1
999 RETURN
END
**SUBROUTINE LPWR**

INTEGER FLPTR(4)
INTEGER STURC(340),DOLSL,COMM(165),CHWR(120),BUFF(100)
COMM,COMM,FLPTR,STURC,IERFL,BUFF
DATA DOLSL,'**/
EQUIVALENCE (IDIV,COMM(7)),(CHAR(1),COMM(126)),(IFRS,STURC(1)),
+ (ICM19,COMM(19)),(ICM20,COMM(20)),(BUFF(1),IFLTR),
+ (ICM29,COMM(29)),(ICM21,COMM(21)),(COMM(16),ICM16)
EQUIVALENCE (IPRT,COMM(35))

**LIST OR PATCH MASTER RECORD**

CALL ERR(56)
READ(6,802)ILP
GO TO (7,7,99),ILP
CALL ERR(42)
READ(6,802)IGC
READ(6,803)ISC,ISW
IFPTR=IPVR(ISC,ISW,0,11)
IF(ISC)10,10,4
READ(6,800)M,N,N
IF(ISC)80,80,11
IF(ISC)313,313,313
CALL MSFT(M,N,N,K,IDIV,2,IPTR)
ISC=IGV3(IPTR,0,11)
ISW=IGV3(IPTR,12,15)
IF(K)6,6,4
CALL ERR(7)
GO TO 10
4 CALL RMASR(IPTR,3)
GO TC(200,10),IERFL
210 IF(IRFS-DOLSL)3,2,3
2 A=STURC(0)
GO TO 55

**FIRST PRINT CUT SECTOR, WORD, AND VBL FIELDS**
3 WRITE(*,501)ISC,ISW
901 FORMAT(1X,214)
CALL LRTR
GO TO 1000
55 WRITE(1,*),STURC(9),STURC(10)
903 FORMAT(1*,318)
1000 GO TC(10,1200),ILP
1200 CALL ERR(41)
READ(6,802)ILP
CALL ERR(40)
READ(6,807)IFW,IVAL
807 FORMAT(13,16)
GO TC(210,220),ILP
210 CALL IFPLD(IVAL,X,IFW)
GO TO 230
220 STURC(IFW)=IVAL
230 CALL WMASR(IPTR,0)
ILP=1
GO TO 200
99 RETURN
END
**LISTMR**

SUBROUTINE LISTMR

INTEGER COMM(165),FLPTR(4),STURC(340),BUFF(67),CHAR(40)
COMM(N,FLPTR,STURC,ERRFL,BUFF)

EQUIVALENCE (CHAR(1),COMM(126))

EQUIVALENCE (IDIV,COMM(17)),(CHAR(1),COMM(126)),(IFRS,STURC(1)),

+ (ICM19,COMM(19)),(ICM20,COMM(20)),(BUFF(I),IFLTR),

+ (ICM29,COMM(29)),(ICM31,COMM(21)),(COMM(35),NPRNT)

C***LIST THE MASTER RECORD CURRENTLY IN STURC

N=GFLD(3)

KL=67

J=1

76 CALL GVBFL(J,BUFF,KL)

IF(IFLTR=16448)78,77,78

78 WRITE(NPRNT,905)(BUFF(I),I=1,KL)

905 FORMAT(1X,80A1)

77 J=J+1

IF(J-ICM29)>76,76,300

C***NCW PRINT PANDO AND OPT. BLOCKS

300 J=1

IC=ICM19

K=1

12 IF(KK-ICM19)5,15,11

11 KK=IC

J=J+1

15 I1=0

DO 20 I=K,KK

II=II+1

20 BUFF(I)=GFLD(I)

WRITE(NPRNT,900)(BUFF(I),I=1,II)

900 FORMAT(1X,1018)

KK=KK+10

K=K+10

GO TO(12,25,12,35),J

C***SEE IF OPT. BLOCK IS PRESENT

25 IG=GFLD(1)

IC=ICM19+ICM20

IF(IGV0(1G,0,0))35,35,30

30 J=3

K=ICM19+1

KK=K+9

GO TO 12

35 CONTINUE

C***PRINT CUT EXPERIMENTAL REAL FIELD

Z=GFLD(27)

WRITE(NPRNT,906)Z

906 FORMAT(1X,F10.5)
**LIST**

SUBROUTINE LISTM

INTEGER COMM(165),FLPTR(4),STURC(340),BUFF(67),CHAR(40)
COMPK, CCMP, FLPTR, STURC, IERFL, BUFF

EQUIVALENCE (CHAR(1),COMM(126))
(+ (ICM19,COMM(19)),(ICM20,COMM(20)),(BUFF(1),IFLTR),
+ (ICM29,COMM(29)),(ICM21,COMM(21)),(COMM(35),NPRNT)

C....LIST THE MASTER RECORD CURRENTLY IN STURC
N=GFLD(3)
KL=67
76 CALL GVBLF(J,BUFF,KL)
77 IF (IFLTR-16448)77,77,78
78 WRITE(NPRNT,505)(BUFF(I),I=1,KL)
905 FORMAT(1X,80A1)
77 J=J+1
IF (J-ICM29)76,76,300

C....NOW PRINT MAND. AND OPT. BLOCKS
300 J=1
IC=ICM19
10 K=10
12 IF (KK-IC)15,15,11
11 KK=IC
J=J+1
15 II=0
DO 20 I=K,KK
II=II+1
20 BUFF(I)=GFLEC(I)
WRITE(NPRNT,500)(BUFF(I),I=1,II)
900 FORMAT(1X,101E0)
KK=KK+10
K=K+10
GO TO (12,25,12,35)J

C....SEE IF CPT. BLCK IS PRESENT
25 IG=GFLEC(1)
IC=ICM19+ICM20+1
IF (IGVB(Ig,0,0))35,35,30
30 J=3
K=IC+1
KK=K+9
GO TO 12
35 CONTINUE

C....PRINT OUT EXPERIMENTAL REAL FIELD
Z=GFLEC(27)
WRITE(NPRNT,906)(Z
906 FORMAT(1X,F10.5)

C....PRINT OUT VBL GROUPS
C....FORMAT USED HERE IS LOCAL
C....GET NUMBER OF GROUPS IN VBL BLOCK
NPS=GFLEC(4)
IF (NPS)100,1000,50

C....ICM21 IS NC. CF FIELDS PER GROUP
C....J IS FIELD NUMBER POINTER
50 J=IC
ICM=ICM21-1
IC3=ICM*3
DO 100 I=1,NPS,3
KK=1
DO 80 L=1,3
DO 70 K=1,IC
BUFF(KK)=GFLEC(J+KK)
70 KK=KK+1
80 J=J+1
WRITE(NPRNT,902)(BUFF(KK),KK=1,IC3)
902 FORMAT(1X,3(214,A1,316,SS))
J=J+IC3
100 CONTINUE
1000 WRITE(NPRNT,910)
910 FORMAT(/)  
RETURN  
END
**NUMBR**

SUBROUTINE NUMBR(LINE, I, X, ITYP)
INTEGER LINE(78), COMM(165), FLPTR(4), STURC(340), OP(4), PLUS,
*PER*
COMMON COMM, FLPTR, STURC, IERFL, OP
EQUIVALENCE (PLUS, CP(1)), (MINUS, OP(2)), (COMM(164), PER)
EQUIVALENCE (COMM(132), KBLNK)
C...CONVERT A STRING CF CHARACTERS TO NUMERIC VALUE; IGNORE LEADING BLANK!
C...I IS COLUMN CF LINE TO START EXAMINING. ON RETURN I IS COLUMN
C...OF LAST CHAR CF NUMBER. VALUE IS RETURNED IN X.
L=-1
K=1
IP=1
SIGN=1.0
SUM=0.0
PSUM=0.0
ITYP=0
2 IF (LINE(K)-KBLNK)=8.3
K=K+1
GO TO 2
3 IF (LINE(K)-MINUS)=5.4
4 IF (LINE(K)=PLUS)=6.7
5 SIGN=-1.0
7 K=K+1
C...CHECK FOR POINT
6 L=LINE(K)
IF (L=PER)=15,50,15
C...EXTRACT DIGIT
15 IZ=IGVE(LI,0,3)
20 IF (IZ=15)=25,20,25
25 GO TO(26,70), IP
26 ITYP=2
I=K
RETURN
30 SUM=SUM*10.0+IZ
IP=2
GO TO 7
50 K=K+1
L=LINE(K)
IP=2
IZ=IGVE(LI,0,3)
IF (IZ=15)=25,55,25
55 IZ=IGVE(LI,4,7)
IF (IZ=9)=60,60,25
60 Z=IZ
PSUM=PSUM+Z*10.0**L
L=L-1
ITYP=1
GO TO 50
70 X=SUM+PSUM
X=X*SIGN
I=K-1
RETURN
END
** MCD

SUBROUTINE MCD(K,IO,IR,IDIV)
C...MODULUS FUNCTION FOR HASHING. IO IS QUOTIENT OF K/IDIV. IR IS REM.
     IO=K/IDIV
     IR=K-IO*IDIV+1
RETURN
END

** MCDX

SUBROUTINE MCDX(X,IO,IR,IDIV)
C...REAL MODULUS FUNCTION CF MCD
     IO=X/IDIV
     XO=IO
     IR=X-XO*IDIV+1.0
RETURN
END
SUBROUTINE PCLSTXP,I,J,TXP)
INTEGER LP1,RP1
INTEGER XP(80),RP,PCNT,TXP(80)
INTEGER CCMW(165),FLPTR(4),STURC(340),BUFF(41)
COMMON CCMW,FLPTR,STURC,IERFL,BUFF
EQUIVALENCE (CCMW(34),KARD),(NPRTN,COMM(35))
C...THIS ROUTINE SCRTS A STRING REPRESENTING A BOOLEAN EXPRESSION
C...TO ORDER SUCH THAT THE ATOMS AND PARENCHEZED GROUPS
C...COME FIRST IN THE SAME RELATIVE ORDER FOLLOWED BY THE
C...OPERATION CODES IN REVERSE ORDER. ( IS CODED AS 0.
C...THE ATOMS ARE POSITIVE NUMBERS REPRESENTING CRITERIA DESCRIBED
C...IN CRIT.
KGM=1
5 NGMIN=J
JPREV=1
NSCRT=1
II=NGM
8 IF(XP(II))10,20,10
10 IF(XP(II))71,15,15
15 RP=II
GO TO 50
C...FIND BALANCING RT. PARENTHESIS
20 II=II+1
PCNT=0
21 IF(XP(III)+1)24,22,24
22 IF(PCNT)26,23,28
23 RP=III
GO TO 50
24 IF(XP(III))27,25,27
25 PCNT=PCNT+1
26 CONTINUE
CALL ERR(39)
RETURN
28 PCNT=PCNT-1
GO TO 27
50 IF(JPREV)55,55,70
55 LP=II
DO 500 LL=LP1,RP1
500 TXP(LL)=XP(LL)
ID=RP-LP
IDJ=LI+ID
KK=LP1
DO 400 LL=LP1,IDJ
XP(LL)=XP(KK)
400 KK=KK+1
ID=RP1-LP1
IDJ=RP1-ID
KK=LP1
DO 300 LL=ICJ,RP
XP(LL)=TXP(KK)
300 KK=KK+1
KK=LP1+RP-LP+1
IF(NGMIN=KK)65,65,60
60 NGMIN=KK
65 NSCRT=2
LP1=RP
RP1=RP
70 II=RP+1
GO TO 74
71 LP1=II
RP1=II
JPREV=-1
II=II+1
74 IF(II-J)8,8,75
75 GO TO (85,80),NSCRT
80 NGM=NGMIN
** PCLST
**SUBRCLINE PVBLF(ISFLD,KARY,N)**

**FUNCTION PVBLF(ISFLD,KARY,N)**

**SUBROUTINE PVBLF(ISFLD,KARY,N)**

**INTEGER KARY(1)**

**INTEGER CMW(165),FLPTR(4),STURC(340)**

**COMMON CMW,FLPTR,STURC,IERFL**

**EQUIVALENCE (STURC(1),IST)**

**EQUIVALENCE (CMW(15),ICM15), (CMW(14),ICM14)**

C***ISFLD IS THE NC. OF SUBFIELD TO PLT IN. KARY IS ARRAY HOLDING

C***IT. N IS NC. OF WORDS IN SUBFIELD NOT COUNTING HEADER.

C***SEE IF VEL FIELD IS PRESENT

IV=IGVE(IST,1,1)+1

C***GET WORD POINTER TO VEL FIELD POSITION

ID=ICM14+IGVE(IST,0,0)*ICM15+1

LVF=FLD(2)

GO TO (50,10), IV

C***IS THERE ALREADY SUBFIELD ISFLD IN VBL FIELD

10 LVEL=IC+LVF

12 IMC=STURC(IDC)

IS=IGVE(IMC,7)

IL=IGVE(IMC,8,15)

IF(IS-ISFLD)15,25,15

15 ID=ID+IL

IF(KTS-LVFL)12,20,20

20 IDIF=N

GO TO 30

C***CHANGE LENGTH PARAMETER FOR VEL FIELD

25 IF(IDIF)30,35,30

30 CALL SCWZ(IDC,ICIF)

GO TO (50,10), IERFL

C***PUT IN SUBFIELD

35 STURC(ID)=IPVE(N+1,IMC,8,15)

IF(IS=IC+1)

GO TO 100

100 RETURN

END
GO TO 5
C...NOW INVERT ORDER FROM NGM TO J POSITION
85 CONTINUE
NGWIN=NGM
JJ=NGWIN+(J-NGWIN)/2
L=0
DO 200 K=NGWIN JJ
LLL=J-L
XP(K)=XP(LLL)
XP(LLL)=LL
200 L=L+1
RETURN
END
SUBROUTINE Q(II,LL,F,ICNT,LINE)
C...GIVEN II AND LL AS POUNDS OF PARTITION
C...THE PARTITION IS PARTITIONED
C...F RETURNS AS PARTITION POINT
REAL VL(10)
INTEGER CP(4),CRIT(12)
INTEGER LINE(20),F,COMM(165),FLPTR(4),STURC(340)
COMMON CP,CW,F,STURC,IERFL
COMMON CP,CW
EQUIVALENCE (CPy(20),ICN20),(COMMy(19),ICM29)
EQUIVALENCE (COMMw(132),KELNK)
MOFL=ICK19+ICK20
MOFL=ICK19+ICK29
I=II
L=LL
F=II
IFLAG=1
CALL IRTRV(IscwD,1,2)
CALL RMASR(IscwD)
CRIT(2)=CRIT(2)+1
DO 100 N=1,ICNT
NF=LINE(N)
IF(NF-PFL)75,75,80
75 VL(N)=GFLP(NF)
GO TO 100
80 GO TO(85,100),IP
C...IP IS USED TO INSURE ONLY 1 STRING FIELD IS USED FOR SORT
85 NF=NF-PFL
IF(NF-ICK29)50,90,100
90 LK=6
CALL GVBLF(NF,AVL,LK)
IF(LK-6)93,93,98
93 LK=LK+1
DO 95 MM=LK,6
95 AVL(MM)=KELNK
98 IP=2
100 CONTINUE
NR=L
NRR=I
400 CALL IRTRV(LSCWD,NR,2)
CALL RMASR(LSCWD)
CRIT(2)=CRIT(2)+1
IP=1
DO 200 N=1,ICNT
NF=LINE(N)
IF(NF-PFL)150,150,150
150 IF(VL(N))=GFLP(NF))151,200,155
151 IF(IFLAG)300,200,200
155 IF(IFLAG)200,200,300
160 LK=6
GO TO(161,200),IP
161 NF=NF-PFL
IF(NF-ICK29)162,162,200
162 CALL GVBLF(NF,EVL,LK)
IP=2
IF(LK-6)165,175,175
165 LK=LK+1
DO 170 MM=LK,6
170 EVL(MM)=KELNK
175 DO 180 MM=1,6
180 IF(AVL(MM))=EVL(MM))151,180,155
180 CONTINUE
200 CONTINUE
IF(IFLAG)210,210,220
210 I=I+1
NR=J
GO TO 230
220 L=L-1
NR=L
230 IF(L=1)999,999,400
C...INTERCHANGE
300 CONTINUE
CRIT(I)=CRIT(I)+1
CALL IRTRV(LSCWD,ARR,I)
IFLAG=-IFLAG
IF(IFLAG)310,310,320
310 I=I+1
F=L
NR=I
NRR=L
GO TO 230
320 L=L-1
F=L
NR=L
NRR=I
GO TO 230
999 CALL IRTRV(lscwd,F,1)
RETURN
END
** PTICAL **

SUBROUTINE PTICAL(NSW,L,NXSW)

C ... NSW IS A SECTOR/WORD BLOCK POINTER
C ... NXSW WILL BE THE SEC/WD POINTER L WORDS FARTHER ON IN THE DISK
NS=IGVE(NSW,0,11)
NW=IGVE(NSW,12,15)
J=NW*L/20
JJ=J/16
NXS=NS+JJ
NXW=J-16*JJ
NXS=IPVB(NXS,NXW,0,11)
RETURN
END

** RCATC **

SUBROUTINE RCATC

INTEGER CCMY(165),FLPTR(4),STURC(340),BUFF(41)
COMKCN CCWY,FLPTR,STURC,IERFL,BUFF
EQUIVALENCE (KARD,CCMY)

C ... READ CATEGORY CARD
C ... ADVISOR NC. IN BUFF(1)
C ... ADVISOR NAME IN A2 IN BUFF(2)-(11)
C ... EOD FLAG IN A1 IN BUFF(41). USE **
1 READ(KARC,900)(BUFF(I),I=1,11),BUFF(41)
90C FORMAT(13,T10,10A2,T80,A1)
RETURN
END
** SUBROUTINE RDSCC **

INTEGER FIELD(15)
INTEGER COMM(165),FLPTR(4),STURC(340),BUFF(41)

GO TO CC&P+FLPTR+STURC+IERFL+BUFF

EQUIVALENCE (BUFF(2),HUFF2),(BUFF(3),BUFF3),(BUFF(9),BUFF9),
+C (BUFF(10),BUFF10)

EQUIVALENCE (BUFF(41),KEDT),(BUFF(8),KDAT)
EQUIVALENCE (KEDT,BUFF(4)), KID,BUFF 1
EQUIVALENCE (KEDT,BUFF(5)),(KEDT,BUFF(6))
DATA FIELD/16.16.36.9.8.8.8.8.8.4.7.16.16.13/
DATA KCASH/+=/ DATA KAST/++/

1 CONTINUE READ A DEPICTCR CARD
1 READ(KARD,900)(BUFF(I),I=4,8),(BUFF(I),I=11,15),(BUFF(I),I=16,36)
+ +KEDT
900 FORMAT(I28,I12,T35,I13,A1,I2,T55,I13,T41,I1,T53,I2,T75,A2,A2,T27,
+ I1,T7,T8,A2,T8,T42,5A2,T80,A1)
10 IF(KEST=KAST)10,39,10
11 IF(LV=KCRV)11,20,11
12 CALL ERR(1)
13 CONTINUE CALL STACK
GO TO 1
20 CONTINUE BUFF2=0
BUFF3=0
BUFF9=0
25 KDAT=KDAT-150
GO TO 30
27 KDAT=KDAT-470
30 KEDT=IPV8(KS,0.0,5)+KN*KS
BUFF10=200
C...NCW PACK FIELDS
C...BUFF(1) & (2) ALREADY SET UP.
KCNT=32
IFLD=3
110 IFRS=KEDT
KS=IPV8(KS,0.7),0.8.15)
ILAST=KEDT+FIELD(IFLD)-1
KEDT=ILAST+1
IFWD=IFRS+16+1
ILWD=ILAST+16+1
IFBT=IFRS-(IFWD-1)*16
ILBT=IL-1-(ILWD-1)*16
IFL=IFWD-LWD)130,120,120
120 BUFFER(IFWD)=IPV8(KK+BUFF(IFWD),IFBT,ILBT)
121 IF(IFLD=15)125,1100,1100
125 IFL=IFLC+1
GO TO 110
130 K1=ILBT-IFBT+17
K2=K1+18
BUFF(ILWD)=IPV8(KK+K2.15),BUFF(ILWD),0,ILBT)
K1=K1-1
K2=K2-1
BUFF(IFWD)=IPV8(KK+K1+K2),BUFF(IFWD),IFBT,15
GO TO 121
C...PUT ALPHA FIELDS IN
110 IFD=11,1=16.36
1100 BUFF(1-7)=BUFF(1)
GO TC(39,12),IERFL
39 RETURN
END
** RDWDC

SUBROUTINE RDWDC
INTEGER CCW(165),BUFF(41),STURC(340),FLPTR(4)
COMMON CCW,FLPTR,STURC,IERFL,BUFF
EQUIVALENCE (BUFF(1),I1),(BUFF(2),I2),(BUFF(3),I3)

C...READ A MODIFICATION CARD
II=0
I2=0
I3=15
K=I3+19+6
READ(*,900)(BUFF(I),I=7,K)

900 FORMAT(2I3,T76,11,T30,11,T28,211,T68,211,T31,212,T62,212,T38,213,
+ 214,213,11,12,T76,11,T61,11,T76,11)
BUFF(9)=0
DO 10 I=10,16
IF(EUFF(I))10,10,5
5 IM1=I-1
11=IPVE(I,IM1,IM1,IM1)
10 CONTINUE
DO 20 I=17,K
IF(EUFF(I))20,20,15
15 IM1=I-17
I2=IPVE(I,12,IM1,IM1)
20 CONTINUE
END
** RECKV

SUBROUTINE RECKV(ISCW0,K)
C... ISCWO IS RECORD NO. OF INDEX ENTRY
C... K IS RECORD NO. OF INDEX ENTRY
C... ON RETURN ISCVO WILL BE NEW POINTER
INTEGER C0W(N),FLPTR(4),S(10),BUFF(41),DOLSL,SZMAS,
+ LSTCL(10),MPT(30),CSPTR(4),STURC(340)
COMMON C0MP,FLPTR,STURC,IERFL,BUFF
EQUIVALENCE (CIV,COW(24))
EQUIVALENCE (CSPTR(1),ICP)
EQUIVALENCE (CSPTR(1),MPT(1))
STORE FREE (COW(28),ICP(28))
EQUIVALENCE (STURC(8),IST8),(STURC(2),IST9)
EQUIVALENCE (SZMAS,COMM(3)),(IDSCC,COMM(6)),(MFXN,COMM(19)),
+ (NCFLD,COMM(21)),(IGPF,COMM(20)),(IDAT,COMM(36)),
+ (INCF,COMM(22)),(INCRH,COMM(11)),(INXF,COMM(2)),(INCRN,COMM(10))
DATA DOLSL/*'*$/*
IERFL=1
C...SAVE OLD LENGTH. CALCULATE NEW LENGTH
LN=GFLD(3)
NN=LN+INCRM
C...MAKE SURE LENGTH NOT TOO GREAT
IF(AN-320),10,10.5
5 CALL ERG(38)
IERFL=2
GO TO 999
C...SAVE 10 WORDS CF STURC. FREE OLD AREA.
10 DO 11 I=1,10
   S(I)=STURC(I)
11 STURC(I)=DOLSL
C...SAVE POINTERS FOR OLD AND NEW LOCATIONS
NEWPT=IENO
C...CALC POINTERS
   IENDS=IGVB(IEND,0,11)
   IENDS=IGVE(IEND,12,15)
   IST8=LN
   IST5=INXF
   CALL WMASR(ISCW0,LN)
C...RESTORE FIRST 10 WORDS
   DO 15 I=1,10
      STURC(I)=S(I)
15 15 STURC(I)=S(I)
C...PUT IN NEW LENGTH
   CALL IPFLD NN,X,3
C...ZERO CUT THE NEW WORDS.
   N1=LN+1
   DO 20 I=N1,NN
      STURC(I)=0
20 20 STURC(I)=0
C...SET FREE AREA WORDS
   N1=NN+1
   N2=NN+7
   DO 30 I=N1,N2
      STURC(I)=DOLSL
      IF(IENDS-SZMAS+100),35,37,35
      STURC(AN+1)=32000
      GO TO 40
37 STURC(AN+1)=0
40 STURC(AN+2)=0
C...PUT RECORD ON DISK WITH 10 WORDS OF FREE AREA
   CALL WMASR(IEND,1)
   GO TO(50,45),IERFL
45 IERFL=2
   GO TO 999
50 CONTINUE
C...SAVE NEW POINTER
   ISW=IEND
C...UPDATE INDEX
   READ(S'K')ID1,ID2,ID3
   WRITE(S'K')ID1,ID2,ID3,IERFL
C...UPDATE FREE AREA POINTERS IN COMM
IENDW=IENDW+H/N/20
I=IENDW-16
IF(I)>00,SS,55
55 IENDS=IENDS+1
IEND=1
60 IEND=IPVB(IENCS,IEND,0.11)
INX=15&0
C..UPDATE CATEGORY CHAIN
KADV=GFLD(ICP28)
READ(2,KADV)(IPT
CALL INVRT(2,IPN,12,ISCW0,NEWPT,KPT)
GO TC(125.80),IERFL
80 WRITE(1,901)KADV
901 FORMAT(*CATEG. *,14)
IERFL=1
125 CONTINUE
C..SAVE COURSES AND CHAINS
C..CALCULATE POINTER TO LAST FIELD
MFC=IFXF+ICPF+1
IG=GFLC(4)
IFIELD=MFC+IG*KFLD-1
C..IFIELD SHOULD HAVE NC+OF CATE FIELD IN LAST GROUP
118 ISWCh=1
ICNT=0
119 I=IAS-GFLD(IFIELD)
IF(I)=IDAT-KDAT)1125,120,1125
120 ICNT=ICNT+1
1=IAS-GFLD(IFIELD-5)
2=GFLD(IFIELD-4)
3=GFLD(IFIELD-3)
3=IPV8(I3,64,0,7)
LSTCL(ICNT)=IPV8(11,12,0,5)+13
IFIELD(IFIELD-NCFLD)
IF(ICNT-10)1119,119,122
122 ISWCh=2
1125 CONTINUE
DD 180 I=1,ICNT
KID=LSTCL(I)
CALL HSHFI(KID,IR,IDIV,2,CSPTR)
IF(IR)130,130,135
135 CONTINUE
IPT=ICP
CALL INVRT(3,IPN,30,ISCW0,NEWPT,KPT)
GO TC(160,130),IERFL
130 CONTINUE
WRITE(1,902)KID
902 FORMAT(*DESC. *,16)
180 CONTINUE
GO TC(185,118),ISWCh
185 CONTINUE
990 ISCW0=ISV
999 RETURN
END
** RWASC **

```
SUBROUTINE RWASC
INTEGER BUFF(41), COMM(165), FLPTR(4), STURC(340), STUD(41)
INTEGER CHAR(40)
COMM CN, COMM, FLPTR, STURC, IERFL, BUFF
EQUIVALENCE (COMM(19), ICM19)
EQUIVALENCE (NCRD, COMM(34)), (KHSN, BUFF(39)), (CHAR(1), COMM(126))
DATA KAST, **/ , KCASH, **/ ,
C*** THIS READS A MASTER CARD WITH DATA FOR MAND. BLOCK
C*** AND FIRST VARIABLE SUBFIELD. THIS IS A LOCAL ROUTINE
K = ICM19 + 1
DO 5 I = K, 41
  5 BUFF(I) = 0
CALL SAVE
10 READ NCRD = 90, 179, 180
90 FORMAT(T79, 2A1)
  IF(180 - KAST) = 26, 450, 20
  IF(179 - KCASH) = 40, 30
  30 CALL ERR(15)
  CALL STACK
  GO TO 10
40 CALL READ
READ(NCRD, 900) STUD
C*** NEED TO MAKE FCRMAT MATCH LOCAL CARDS
900 FORMAT(2I3.21A1, T28, 3I1, 2I2, 11, 12, 2I3, 214, 213, 2(11, 12), 11)
  BUFF(41) = STUD(41)
  BUFF(5) = STUD(1)
  BUFF(6) = STUD(2)
  BUFF(7) = 0
C*** Buff(5) will be set = STUD ENTRY FOR PLACES WITH 3-FIELD ID NO.
C*** Buff(39) used for NAME HASH
KHSN = 0
DO 50 I = 3, 15, 2
50 KHSN = KHSN + STUD(I)
DO 60 I = 31, 3660
60 Buff(1-I4) = STUD(I)
  Buff(8) = Stud(26)
  Buff(9) = Stud(24)
  Buff(10) = Stud(25)
  Buff(11) = 0
  Buff(12) = 0
  Buff(13) = Stud(27)
  Buff(14) = Stud(28)
  Buff(15) = Stud(40)
  Buff(16) = 0
  Buff(25) = 0
  Buff(26) = Stud(39)
  Buff(27) = 0
  Buff(28) = 0
C*** New put NAME IN A3 FORM INTO Buff(31) THRU Buff(37)
C*** Use Buff(38) for WORD COUNT FOR NAME
CALL A1A3(STUD, 3, 23, Buff, 31, CHAR)
DO 70 I = 1, 21
  II = 24 - I
  IF(STUD(II) = 16448) 80, 70, 80
70 CONTINUE
80 Buff(38) = (II - 1) / 3
RETURN
450 IERFL = 2
RETURN
END
```
** RMVEC

SUBROUTINE RMVEC
  INTEGER CMYV(165),FLPTR(4),STURC(340),BUFF(41)
  INTEGER SG(18),BUFF7,BUFF9,EFF40,BUFF8
  COMMON CMYV,FLPTR,STURC,IERFL,BUFF
  EQUIVALENCE (CMYV(6),ICMV6)(KARD,COMM(34)),(BUFF(7),BUFF7)
    +
  DATA IS/1/,KAST/**/
C****A LCC-V ROUTINE TO READ CARDS WITH
C****CATA TO MODIFY GROUPS AT END OF RECORD
  GO TO TC (10,50),IS
  10 IS=2
  ICMV6=1
C****READ GRADE SYMBOLS
  DO 800 I=1,18
    II=I+32
  800 READ(9*II)SG(I)
    BUFF9=0
  50 READ(KARC,900)(BUFF(I),I=1,7),KECD
    EFF40=2
  900 FORMAT(2I3,T77,11,T28,12,T35,13,A1,T42,A2,T80,A1)
    IF(KECD-KAST)52,99,52
  52 DO 60 I=1,18
    IF(SG(I)-BUFF7)60,70,60
    CONTINUE
    CALL ERR(20)
    CALL STACK
    GO TO 50
  70 BUFF8=I-9
    BUFF7=5
  99 RETURN
END
** RVBLC

SUBROUTINE RVBLC
INTEGER BUFF(41),STURC(340),FLPTR(4),COMM(165),SG(18)
INTEGER BUFF7,BUFF9,BUFF4,BUFF5,BUFF6,BUFF10
COMMON COMM,FLPTR,STURC,IERFL,BUFF
EQUIVALENCE (BUFF(41),KC),(BUFF(4),BUFF4),(BUFF(10),BFF10)
EQUIVALENCE (CRCD,COMM(34)),(BUFF(20),SG(1)),(COMM(36),ICM36),
BUFF(9),BUFF9),(BUFF(7),BUFF7),(COMM(6),ICM6),
BUFF(5),BUFF5),(BUFF(6),BUFF6)
DATA KAST/****/
DATA KS/1/
    *.*.READS A CARD WITH GROUP DATA
GO TO TC(10,100),KS
10 CALL ERR(S)
   READ(6,902)IDT
902 FORMAT(13)
   IF(IDT=400)1S,15,20
15 IDT=IDT-150
   GO TO 25
20 IDT=IDT-50
25 DO 30 I=33,50
30 REAC(9*I)SG(I-32)
KS=2
ICM6=2
BUFF10=IDT
ICM36=IDT
100 CONTINUE
    *.*.TESTS SHOULD BE MADE TO BE SURE CARD IS RIGHT TYPE
    *.*.DO NOT LET THIS PROGRAM BE IN AN OVERLAY
READ(CRCD,900)(BUFF(I),I=1,3),(BUFF(I),I=5,9),KS
900 FORMAT(213,T7,T11,T28,T12,T35,T13,A1,T2,T42,A2,T80,A1)
   IF(KC-KAST)12,99,9
   DO 120 I=1,17
120 CONTINUE
   IF(SG(I)-BUFF9)120,125,120
125 CONTINUE
BUFF9=1
BUFF4=IPVB(BUFFS,BUFF6,0.5)+BUFF7
    *.*.SECTION CCCE MUST BE MOVED OVER TO BE ABLE TO FIT IN
    *.*.8 BITS FOR IPFLD.
BUFF7=IGVB(BUFF7,0.7)
    *.*.PUT IN ANY OTHER SPECIAL PROCESSING NEEDED
99 RETURN END
** SHCVE

SUBROUTINE SHCVE(ICL,T,ICISP)
INTEGER CCMM(165),FLPTR(4),STURC(340)
CCMM,CCMM,FLPTR,STURC,IERFL
EQUIVALENCE(STURC(1),IST)
EQUIVALENCE(CCM(15),ICM(15),(CON(14),ICM(14),(CM(16),ICM(16))

ICLT IS POINT AT WHICH SPACE IS TO BE PROVIDED. IDISP IS SIZE OF
SPACE. CURRENT LENGTH IS L. LA IS CURRENT ACTIVE LENGTH.
L=GFLD(3)
LA=ICM14+ICM15*ICVE(IST,0,0)+GFLC(2)+ICM16*GFLD(4)
IF(L-LA-ICISP)20,50,50

10 IERFL=2
RETURN
50 LA=ICL+IDISP
K=LA-ICLT+1
IF(IDISP)45,460,50
40 K=1
GO TO 55
45 K=-1
LA=ICLT
LAB=LAI+IDISP
CD 60 I=1,N
STURC(LAI)=STURC(LA)
LAB=LAI+K
LA=LA+K
60 CONTINUE
460 RETURN
END
** STATS

SUBROUTINE STATS
INTEGER CMN(165),FLPTR(4),STURC(340),OP(4),CRIT(12),IVAL(50),
+ SPTR,LINE(15)
REAL SUM(3),SSUM(3),MAX(3),MIN(3)
COMMON CMN,FLPTR,STURC,ERFL,OP,CRIT,IVAL,SPTR
DATA KELNK,KCMMA/*"*/

GATHERS STATISTICS ON RECORDS POINTED TO BY WORKING SET.

CALL ERR(61)
READ(6,901)LINE
901 FORMAT(15A1)
J=0
DO 10 I=1,15
L=LINE(I)
4 IF (L=KELNK) 5,10,4
5 CALL FIELD(LINE,I,NF)
30 CONTINUE
40 IF (L=KCMMA) 5,10,5
50 IF (SPTR) 25,25,5
90 IF (NF) 15,15,7
J=J+1
LINE(J)=NF
IF (J=3) 10,20,20
10 CONTINUE
15 IF (J=99,59,20
20 IF (SPTR) 25,25,30
30 CONTINUE
50 CALL ERR(57)
GO TO 99
35 DO 15 I=1,3
SUM(I)=0.0
SSUM(I)=0.0
MIN(I)=99999.0
MAX(I)=-99999.0
55 KNT=0
DO 100 I=1,SPTR
90 CALL ISYRT(ISWID,1,i+2)
100 CALL RASR(ISWID)
KNT=KNT+1
DO 50 K=1,J
X=FLDLINE(K)
SUM(K)=SUM(K)+X
SSUM(K)=SSUM(K)+X*X
50 IF (MIN(K)=X) 45,45,45
45 IF (MAX(K)=X) 47,50,50
47 MAX(K)=X
50 CONTINUE
100 CONTINUE
XNT=KNT
DO 110 I=1,J
A=SUM(I)/XNT
SD=SSUM(I)/XNT-A*A
WRITE(1,903)LINE(I)
903 FORMAT(*FIELD*,12,*)
904 FORMAT(*SUM='*,E13.6,2X,*AVE='*,E13.6,2X,*SD='*,E13.6,2X,*MAX='*,
+E13.6,2X,*MIN='*,E13.6,2X)
110 CONTINUE
99 RETURN
END
SUBROUTINE WASR(ISW,KEY)
INTEGER CCVV(165),FLPTR(4),STURC(340)
INTEGER GLPTR(4),S(320)
COMMON CCVV,FLPTR,STURC,IERFL
EQUIVALENCE (IF1,GLPTR(1)),(KSC1,GLPTR(2)),(IF2,GLPTR(3))
(KSC2,GLPTR(4)),(S(1),STURC(1))
C***THIS WRITES THE MASTER RECORD CURRENTLY IN STURC BACK ON THE DISK.
C***IF KEY=0 EXACTLY STURC(1)-(LENGTH OF REC) IS WRITTEN. IF KEY=1 THE
C***THE 10 NEXT WCRS ARE WRITTEN. IF KEY GT 1 THE NO. WORDS WRITTEN=KEY
IS=I GV B(ISW(:,:.1))
IW=I GV B(ISW(:2+12,:.15))
CALL DRIVE(ISW(IW))
DO 5 I=1,4
5 GLPTR(I)=FLPTR(I)
GO TO (200,210,210),IERFL
200 NWDS=GFLD(3)+10*KEY
IF (KEY=1)201,201,220
201 IRN=320-IW*20
ISWCH=1
IF (NWDS)212,210,211
211 IF (NWDS=340)213,213,212
212 WRITE(IF1,KSC1)
GO TO 210
213 IF (NWDS=IRN)202,202,206
202 IRN=NWDS
ISWCH=2
206 WRITE(IF1,KSC1)(STURC(I),I=1,IRN)
GO TO (207,210),ISWCH
207 IRN=IRN+1
WRITE(IF2,KSC2)(STURC(I),I=IRN,NWDS)
210 RETURN
END
APPENDIX E

PROGRAM LISTINGS FOR SCRIPT

The main programs for initializing files, adding new lessons to the disk file, and polling the terminals are given first followed by the subprograms.

The listing for IPVB/IGUB is given in the listings for STRIPE. The interrupt level subroutine, ILS03, and the interrupt response and utility routine, LMCC, are not listed here since they come from Logicon, Inc.
**INIT**

```
INTEGER MESS(64),BUFF(64),ZB(20),Z(S)
EQUIVALENCE (ZB(1),Z(1))
DATA LF,KR/ZG00A,Z0000/
DEFINE FILE 2(6400,I1,42),I(100,32,UN1),6(32,10,UN6)
DEFINE FILE 3(64,5,UN3),4(48,20,UN4),5(32,10,UN5)
C GET CHAR CUES
100 READ(2,902)(MESS(I),I=2,63)
902 FORMAT(63A1)
MESS(1)=62
CALL EETTY(MESS,BUFF(63))
DO 110 I=1,30
   J=63-I
110 MESS(I)=BUFF(J)
MESS(29)=4
MESS(28)=LF
MESS(27)=KR
WRITE(6,3)(MESS(I),I=1,30)
DO 150 K=7,30
   READ(2,902)(MESS(I),I=2,63)
   IF(MESS(2)-23616)125,200,125
125 MESS(1)=20
   CALL EETTY(MESS,BUFF(64))
   DO 128 I=2,63
      IF(MESS(I)-16448)128,126,128
126 IF(MESS(I+1)-16448)128,127,128
127 KNT=I
      GO TO 129
128 CONTINUE
129 CONTINUE
   IP=65-KNT
   BUFF(IP)=LF
   BUFF(IP-1)=KR
   BUFF(IP-2)=4
   DO 130 I=1,10
      II=11-I
      III=65-II
   130 WRITE(6,9)(MESS(I),I=1,10)
200 DO 220 I=1,20
   220 ZB(I)=0
   DO 230 I=1,16
   230 IF(ZB(I)+16)230,231,230
   231 WRITE(5,1)ZB
   250 WRITE(5,1)Z
   CALL EXIT
END
```

FILE LIMIT RESPONSE NOT IMPLEMENTED
TIME LIMIT
RETYPE RESPONSE
READY, TYPE ID
LOGGED OFF
TYPE 1 TC BEGIN
OUT OF RANGE
ID NO. IN USE
CAN'T INTERPRET
**ADLSN**

```
INTEGER MSSNW(20), MESS(64), BUFF(64)
DATA L, KR/(2000, 2000, 0),
DEFINE FILE 5(64.1, U, KN2), 1(100,32, U, N1), 6(32, 10, U, N6)
DEFINE FILE 3(64, 5, U, N3), 4(48, 20, U, N4), 5(32, 10, U, N5)
READ(*', 1) TL, INDXP, KPDTE, KPMSG
INDXP = INXP + 1
K1 = KPDTE + 1
K2 = KPMSG + 1
C...PUT IN MESSAGE VECTOR
15 READ(2', 922) MSSNM
922 FORMAT(204)
DO 20 I=1, 20
K=MSSNW(I)
IF(K)30, 30, 25
25 CONTINUE
KPDTE=KPDTE+1
WRITE(2'KPDTE') MSSNM(I)
20 CONTINUE
GO TO 15
30 LMV=KPDTE-K1+1
C...PUT IN DECISION TABLE
READ(2', 900) IR, IC
900 FORMAT(214)
KREC=IR*IC
DO 10 I=1, KREC
READ(2', 901) NS, NCM, IPTR
901 FORMAT(314)
K=IPV(NS, 0, 0, 5) + IPV(NCM, 0, 6, 8) + IPTR-1
KPDTE=KPDTE+1
WRITE(2'KPDTE') K
10 CONTINUE
C...PUT IN CODE MESSAGES
51 READ(2', 902)(MESS(I), I=2, 63)
902 FORMAT(63A1)
IF(MESS(2)='23616') 52, 100, 52
52 DO 60 I=2, 63
IF(MESS(I)='16448') 60, 65, 60
55 IF(MESS(I+1)='16448') 60, 65, 60
60 CONTINUE
I=61
65 MESS(I)=I+1
CALL EBTY(MESS, BUFF(64))
K=65-L
BUFF(K)=LF
BUFF(K-1)=KR
BUFF(K-2)=4
I=(I+2)/2
DO 75 L=1, 1
K=64-L
KK=65-2*L
KPP=KK-1
BUFF(K)=IPV(BUFF(KK), BUFF(KK), 0, 7)
KPMGS=KPMGS+1
WRITE(1'KPMGS') (BUFF(I), I=33, 64)
GO TO 51
100 CONTINUE
WRITE(3'INDXP') IR, IC, K2, K1, LMV
WRITE(6'1) ITL, INDXP, KPDTE, KPMGS
CALL EXIT
END
```
ZWEI UND ZWEI IST FUNF. NICHT WIEHR
1) JA 2) NEIN
DU HAST RECHT. DO YOU WANT THE NEXT LESSON
DIESE IST VIER. XXXX
DIESE IST FUNF. XXXX
VERSUCHE BITTE WIEDER
ACH, DU LIEBER. LEHR ZAHLEN. VERSUCHE WIEDER.
DAS IST ALLES.
DO YOU WANT THE NEXT LESSON
THEN REQUEST MODULE 2.
NO HELP NEEDED
*
INTEGER Te(64,e),TP(15,e),ISTS(2C),M0C( ECC).STAT.TIME,CEV.

EXTERNAL USER
CCMNC TE,T,F,ISYS,WCC,STAT,INTR,KNT,DEV,CHAR,TIME

EQUIVALENCE (ISYS(I),ISYS1)

FILE 1 FOR DECISION TABLES
FILE 2 FOR INDEX
FILE 3 FOR SAVING PARTITIONS
FILE 4 FOR TEACHER FILES
FILE 5 FOR SYSTEM PARAMETER

define file a (ie0.2.L.NA).£(32 .XC.L.NS).

CALL FIRST

READ(1,5C)

FILE ENTER MCLLE NC IN I2(*)

FILE 2 CECISICN TACLES
FILE 3 FCR INCeX
FILE»- FCR SAVING PARTITIONS
..FILE S  FCR TEACHER FILES
..FILE e  FCR SYSTEM FARAMETERS

CALL LIMT(LSER)
CALL LTIMc(TIME)
DC 12C 1=1,6
CALL LMAS

IF(IGVE(TF(I,I),0,C))5C.SC.eC

IF(ISENS(I))1 10,110,60

CCNTINUE

IF(IGVE(TF(I,I),A,A))6S.6£.1CC es

t p (i, i) = ifv 6(1. t f (i, i)).a.
a)

TF(9,I)=1
CALL LRtAC(I)

IT=TIME-TF(2,1)

IF(I)1CC,1CC,EE

IF(IT-ISYS1)5E,9S,9C

CCNTINUE

CALL #ESC(5,1)

CALL CECISICN TACLES

TP(E,I)=3
TP(5,1)=1

TP(1,1)=C

GC TC 11C

CCNTINUE

IF(IGVE(TF(I,I),3,3))96,96,1CC

IF(IGVE(TF(I,I),7,7))1CC,1CC,97

CALL #ESC(1,1)

TF(I,I)=IPVE(1,TF(I,I),3,2)

CCNTINUE

II=I-1

IF(IGVE(INTR,II,II))11C,11C,1GC

CCNTINUE

CALL Lfffffff

CCNTINUE

CALL LLNMK

GC TC 30

END
** BREAK **

SUBROUTINE BREAK
INTEGER TB(64,8), TP(15,8), ISYS(20), MOD(500), STAT, TIME, DEV+
CHAR(30)
COMMON TB, TP, ISYS, MOD, STAT, INTR, KNT, DEV, CHAR, TIME
EQUIVALENCE(ISYS(1), ISYS1)
TP(2, STAT) = TIME + ISYS1
IFL = TP(1, STAT)
IF (IGVB(IFL, 0, 0)) 10, 10, 20
CONTINUE
I = STAT - 1
INTR = IPVE(I, INTR, I, I)
TP(15, STAT) = 2
TP(1, STAT) = IPVE(I, IFL, 0, 0)
I = 0
RETURN
20 TB(64, STAT) = 0
I = 1
RETURN
END
** CALC **

SUBROUTINE CALC
REAL IVL1,IVL2
INTEGER TO(64,8),TP(15,8),SYS(20),MOD(500),STAT,TIME,DEV,
+ CHAR(20)
INTEGER DLNR,ELNK
COMMON TE,TP,ISYS,MOD,STAT,INTK,DEV,CHAR,TIME
EQUIVALENCE (ELNK,CHAR(30)),(DCLR,CHAR(17)),(KE,CHAR(13))
EQUIVALENCE (CHAR(20),K20)
IP=63
IB=1
CALL NELNK(IP,IB)
KF=(IB-1)*8+1
IOP=IGVE(TB(IP,DEV),KF,KF+6)
DO 10 I=13,16
IF(ICP-CHAR(I))10,5,10
5 IOP=1-7
IP=IP-2
GO TO 64
10 CONTINUE
30 CALL KNUM(IP,IE,IVL1,IERR)
GO TO (40,55),IERR
40 CALL NELNK(IP,IB)
KF=(IE-1)*8
IOP=IGVE(TB(IP,DEV),KF,KF+6)
DO 50 I=18,21
IF(IOP=CHAR(I))50,45,50
45 IOP=I-17
GO TO 60
50 CONTINUE
55 CALL MEGG(9,DEV)
RETURN
60 IF(IE-2)62,61,61
61 KF=1
KL=7
IB=1
GO TO 66
62 KF=5
KL=15
IB=2
66 IOP=IGVE(TB(IP,DEV),KF,KL)
IF(IOP=K20)64,63,64
63 IOP=5
GO TO 60
64 CALL NELNK(IP,IB)
CALL KNUM(IP,IE,IVL2,IERR)
GO TO (65,55),IERR
65 GO TO (70,75,80,85,90,95,96,97,98),IOP
70 RSLT=IVL1+IVL2
GO TO 100
75 RSLT=IVL1-IVL2
GO TO 100
80 RSLT=IVL1*IVL2
GO TO 100
85 RSLT=IVL1/IVL2
GO TO 100
90 RSLT=IVL1**IVL2
GO TO 100
95 RSLT=EXP(IVL2)
GO TO 100
96 RSLT=ALCG(IVL2)
GO TO 100
97 RSLT=SIN(IVL2)
GO TO 100
98 RSLT=CCS(IVL2)
100 CALL TRANS(RSLT,DEV)
CALL LWRI5(DEV)
RETURN
END
**FIRST**

SUBROUTINE FIRST

INTEGER TB(64,8), TP(15,8), ISYS(20), MOD(500), STAT, TIME, DEV,
        + CHAR(30)

COMMON TB, TP, ISYS, MOD, STAT, INTR, KNT, DEV, CHAR, TIME

C...GENERATE SYSTEM, SET MODULE PTRS TO NULL, ETC.

INTR=0
READ(6*1) ISYS
READ(6*3) CHAR
DO 20 J=1,8
   DO 10 I=1,14
   10  TP(I,J)=0
        TP(15,J)=1
        TP(6,J)=1
   20  TB(I,J)=0
RETURN
END
** ISENS

ISENS DC **-
LDX I I ISENS
LD I I 0
S K1
STO DEV
A SR
STO SHIFT
LD DEV
SLA 3
OR MASK
STO ICC+1
XOR ICC
AND MASK1

SHIFT NOP
MDX L ISENS+1
BSC I ISENS
BSS E 0
IOCC DC 0
DC **-
MASK DC /7701
MASK1 DC /1111
DEV DC **-
SR SRA 0
K1 DC 1
END
** KALM

SUBCUTINE KALM(IF,IE,VAL,IERR)

INTEGER K(I)

INTEGER IE(64,8),IF(15,8),ISYS(2C),MCD(SEC),STAT,TIME,DEV,

* CHAR(2C)

* CC*KCN TE,TP,ISYS,KCC,STAT,INTR,KAT,DEV,CHAR,TIME

EQUIVALENCE (CHAR(3C),KELNK),(CHAR(1C),K1C),(CHAR(23),IFER)

C...LEE IF FCR FTR TC WCRE, IE FCR EYTE

IE=1
IFLG=1
IERR=1
IF(IE-2)1C=2C,2C
1C KNT=IGVE(IE(IF,DEV)),1,7)
IE=1
GC TC 06
15 I=1+1
K(I)=KNT
2C KNT=IGVE(IE(IF,DEV)),9,15)
IE=2
GC TC 06
25 I=1+1
K(I)=KNT
3C IF=IE-1
GC TC 06
5C IF(IGVE(KNT,E,11)-3)6C,65,6C
5E KNT=IGVE(KNT,12,15)
GC TC (15,2E),IE
6C CONTINUE
IF(KNT-IFER)62,61,62
61 IE=1
IFLG=2
GC TC (2C,3C),IE
62 IF(IE)6E,6E,7C
6E IERR=2
RETURN
7C VAL=C
GC TC (76,78),IFLG
7E IE=1
7E CC EC J=1,11
8C VAL=VAL+1C+K(J)
GC TC (1CC,6E),IFLG
8E IE=IE+1
K=1
EC EC J=II+1
VAL=VAL+K(J)*100*C4K
9C K=K-1
1CC RETURN
** LDMOD

*ONE* WORD INTEGERS
SUBROUTINE LDMOD(ID)
  INTEGER TD(64,8), TP(15,8), ISYS(20), MOD(500), STAT, TIME, DEV,
  COMMON TB, TP, ISYS, MOD, STAT, INTR, KNT, DEV, CHAR, TIME
  EQUIVALENCE (MOD(1), MOD1), (MOD(2), MOD2), (MOD(3), MOD3),
  *(MOD(4), MOD4)*
  READ(3*ID) IR, IC, NMSG, IPTR, LMV
  K=IR*IC+LMV+4
  MOD1=IR
  MOD2=IC
  MOD3=NMSG-1
  MOD4=LMV+4
  READ(2*IPTR)(MOD(1), I=5, K)
  RETURN
END
**MCCLL**

```
SLEFCTIME, MCCLL(J)
INTEGER MESS(32)
INTEGER IC(64, 8), ISYS(2), MCC(200), STAT, TIME, CEV
4 IFifr(3)
CC NCC TE, TF, ISYS, MCCC, STAT, INTR, KN, CEV, CFAR, TIME
EQUIVALENCE (MESS(32), MESS(8))
EQUIVALENCE (MCC(1) , 1R), (MCC(2) , IC), (MCC(2) , WPT), (MCC(4) , IPT)
C**..USE IF (5, CEV) FOR CLR. STATE, J IS INPLT
C**..THIS VERSION ASSUMES JUST ONE MCCLL IN CCR
ACK = TF(J, CEV)
1C K = TF(5, CEV)
1C J = 12, 16, 16
1C CALL MESS(7, CEV) RETURN
C**..CALC. TXT TC (J, K) POSITION. ALLCW FOR FIRST 4 MESSAGES.
IE M = (K - 1) * J + 1 + IPT
1C K = TF(4, CEV)
1C MESS = MESS(K + 1, E)
1C TF(5, CEV) = K
1C ACK = TF(4, CEV) RETURN
C**..MESSAGING IN MCC AND OVERLAYING PREVIOUS ECT
1C TF(J, CEV) = K
1C CALL MESS(K) RETURN
C**..CONTINUE CALLING
C**..WRITE (CEV) TF(2, CEV) = NS = TF(4, CEV)
1C TE(J, CEV) = K
1C ENTER(1, CEV) = IE + 1 + IPT
1C MESS(K) = K
1C TF(4, CEV) = K
C**..CONTINUE CALLING
C**..WRITE
```
** NBLNK

SUBROUTINE NBLNK(IP,IB)
INTEGER ELNK
INTEGER TO(64,8),TP(15,8),ISYS(20),MOD(500),STAT,TIME,DEV,
+ CHAR(30)
COMMON TB,TP,ISYS,MOD,STAT,INTR,KNT,DEV,CHAR,TIME
EQUIVALENCE (CHAR(30),BLNK)
C***THIS ROUTINE IS TO RETURN IP AND IB AS THE WORD AND BYTE
C***CF THE NEXT NON-BLANK CHARACTER
5 GO TC(21,16),IB
   K=IGVB(TB(IP,DEV),KF,KF+6)
   IF(K=BLNK)30,10,30
10 GO TC(15,20),IB
   IB=2
16 KF=5
   GO TO 5
20 IB=1
   IP=IP-1
21 KF=1
   GO TO 5
30 RETURN
END
** PSAVE

```
SUBROUTINE PSAVE
INTEGER TB(64,8),TP(15,8),ISYS(20),MOD(500),STAT,TIME,DEV,
      CHAR(30)
COMMON TB,TP,ISYS,MOD,STAT,INTR,KNT,DEV,CHAR,TIME
ID=TP(5,DEV)
WRITE(4,ID)TP(9,DEV)
RETURN
END
```

** PLOAD

```
SUBROUTINE PLOAD
INTEGER TB(64,8),TP(15,8),ISYS(20),MOD(500),STAT,TIME,DEV,
      CHAR(30)
COMMON TB,TP,ISYS,MOD,STAT,INTR,KNT,DEV,CHAR,TIME
ID=TP(5,DEV)
READ(4,ID)TP(9,DEV)
RETURN
END
```

** MSEG

```
SUBROUTINE MSEG(IM,IDEV)
INTEGER IM(10),
       TB(64,8),TP(15,8),ISYS(20),MOD(500),STAT,TIME,DEV,
       CHAR(30)
COMMON TB,TP,ISYS,MOD,STAT,INTR,KNT,DEV,CHAR,TIME
READ(6*IM+6)*
DO 20 I=1,10
   J=53+I
   TB(J,IDEV)=M(I)
   CALL WRIT(IDEV)
   RETURN
20 RETURN
END
```
** SERV

SUBROUTINE SERV
INTEGER TB(64,8),TP(15,8),ISIS(20),MOD(500),STAT,TIME,DEV.
+ CHAR(30)
INTEGER DLCLK,ELNK
COMMON TB,TP,ISIS,MOD,STAT,INTR,KNT,DEV,CHAR,TIME
EQUIVALENCE (BLNK,CHAR(30))((DLCLK,CHAR(17)),(KE,CHAR(13)))
EQUIVALENCE (ISIS(1),ISIS),(CHAR(22),ITIC)
C.*** ALSO USED FOR BUFFER OVERFLOW INTERRUPT
C.*** DETERMINE IF TERMINAL CAME ON REC OR XMIT MODES
C.*** MUST PRECEDE ALL CONTROL RESPONSES
C.*** RESPONSE FOR HELP
C.*** C RESPONSE FOR CALCULATE MODE
C.*** E RESPONSE FOR EXIT
C.*** R RESPONSE FOR RETURN TO MODULE FROM CALCULATE MODE
C.*** S RESPONSE TO SAVE STATE
C.**** RESPONSE TO LOAD SAVED STATE
C.**** FIND FIRST NON-BLANK AND EVALUATE
CALL LTST(DEV,IBUS)
IF(IBUS).NE.40.1010
4 I=TP(15,DEV)
GO TO (3,1,2),I
1 CALL MEGS(4,DEV)
TP(15,DEV)=3
GO TO 1000
2 TP(15,DEV)=1
GO TO 112
3 IF(IGVB(TB(64,DEV),7,7)))110,110,10
10 CONTINUE
GO 1000 I=1,5
II=64-I
KCHAR=IGVB(TB(II,DEV),1,7)
IF(KCHAR-BLNK)=120,50,120
50 KCHAR=IGVB(TB(II,DEV),9,15)
IF(KCHAR-BLNK)=120,100,120
100 CONTINUE
105 CALL MEGS(2,DEV)
GO TO 1000
110 CONTINUE
IF(TP(3,DEV))112,112,150
112 CONTINUE
TB(64,DEV)=0
CALL LREAD(DEV)
GO TO 1000
120 IF(KCHAR-ITIC)=129,121,129
C.*** DETERMINE ID NO.
121 IF(IB-2)=122,123,123
122 IB=2
GO TO 124
123 I=II-1
IB=1
124 CALL KNUM(II,IB,D,IERR)
ID=C
C.
IF(IERR-2)=128,127,127
127 CALL MEGS(9,DEV)
GO TO 112
C.
128 IF(I-1)=1,8
IF(TP(5,1)-I)=126,125,126
125 CALL MEGS(0,DEV)
GO TO 1000
126 CONTINUE
TP(5,DEV)=ID
CALL MEGS(6,DEV)
GO TO 1000
129 IF(KCHAR-CLLR)=130,200,130
130 IF(IGVB(TP(1,DEV),1,2)))135,135,131
131 CALL CALC
GO TO 1000
135 DO 140 J=1,10
I=J-1
IF(KHAR-CHAR(I))140,150,140
140 CONTINUE
GO TO 105
C...MODULE IS TO FILL KEUF(.DEV) WITH APPROP. MESSAGE. LEAVES ACC.=2
150 CONTINUE
CALL MCDLL(1)
GO TO 1000
C...CTRL REQUEST
200 GO TO(210,220)*IB
210 KHAR=IGVE(TB(II,.DEV),9,15)
GO TO 230
220 KHAR=IGVE(TB(II-1,.DEV),1,7)
230 DO 237 I=11,16
J=I-10
IF(KHAR-CHAR(I))237,238,237
237 CONTINUE
GO TO 105
238 GO TO(400,600,250,800,700,500,J,400)
400 I=0
GO TO 150
C...EXIT ROUTINE NEEDS MORE WORK
C...SET DEVICE INACTIVE, ETC.
250 TP(1,.DEV)=0
CALL LMAN(D,DEV)
GO TO 1000
500 TP(1,.DEV)=IPVE(3,TP(1,.DEV),1,2)
GO TO 112
600 TP(1,.DEV)=IPVE(0,TP(1,.DEV),1,2)
GO TO 112
700 CALL PSAVE
GO TO 112
800 CALL PLOAD
GO TO 112
1000 DEV=DEV-1
INTR=IPVB(0,INTR,DEV,DEV)
1010 RETURN
END
** TERMIN**

SUBROUTINE TERM
INTEGER TD(64,8), TP(15,8), ISYS(20), MOD(500), STAT, TIME, DEV,
+ CHAR(30)
COMMON TP, ISYS, MOD, STAT, INTR, KNT, DEV, CHAR, TIME
EQUIVALENCE (ISYS(1), ISYS(1))
I = STAT - 1
INTR = IPV8(1, INTR, 1, 1)
IF (IGVB(TD(64, STAT), 7)) 10, 10, 20
20 TP(2, STAT) = TIME + ISYS(1)
10 CONTINUE
I = 0
RETURN
END
SUBROUTINE TRNSL(RR, I)
INTEGER TB(64,8), TP(15,8), ISYS(20), MOD(500), STAT, TIME, DEV, CHAR(30)
COMMON TB, TP, ISYS, MOD, STAT, INTR, KNT, DEV, CHAR, TIME
EQUIVALENCE (CHAR(29), KE), (CHAR(28), LF), (CHAR(27), KR)
EQUIVALENCE (CHAR(30), KE), (CHAR(13), KE)
EQUIVALENCE (CHAR(18), K18), (CHAR(19), K19), (CHAR(23), K23)
R=RR
ISIGN=K18
IF(R)2,4,4
2 ISIGN=K19
R=-R
4 ILX=-32
4 IMX=32
5 IMX={(ILX+IMX)/2
10 IF(R-10.0**IMX)20,20,30
20 IMX=IMX
GOTO 35
30 ILX=IMX
35 IF((IMX-ILX-1)40,40,5
40 R=R/10.0**ILX
R=R+0.000001
K1=R
R=(R-K1)*10.0
K2=R
R=(R-K2)*10.0
K3=R
R=(R-K3)*10.0
K4=R
R=(R-K4)*10.0
K5=R
R=(R-K5)*10.0
K6=R
TB(63,1)=IPVE(ISIGN,K23,0,7)
TB(62,1)=IPVE(CHAR(K1+1),CHAR(K2+1),0,7)
TB(61,1)=IPVE(CHAR(K3+1),CHAR(K4+1),0,7)
TB(60,1)=IPVE(CHAR(K5+1),CHAR(K6+1),0,7)
ISIGN=K18
IF((IHX)45,50,50
45 ISIGN=K15
IHX=IHX
50 IH1=IHX/10
IH2=IHX-10*IH1+1
IH1=IH1+1
TB(59,1)=IPVE(K6,ISIGN,0,7)
TB(58,1)=IPVE(CHAR(IH1),CHAR(IH2),0,7)
TB(57,1)=IPVE(K6,LF,0,7)
TB(56,1)=IPVE(K6,LT,0,7)
RETURN
END