CONCURRENT MULTI-TERMINAL
OPERATION OF THE 'CRIME' SYSTEM

A Thesis

Presented to

The Faculty of the Department of Electrical Engineering

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Electrical Engineering

---

by

Asif Karachiwala

July 1976

ACKNOWLEDGEMENT

The author would like to express sincere gratitude to his major advisor Dr. Batten, for his helpful suggestions and patience during this entire project. Additional gratitude is due to Dr. Bargainer for his comments and interest in this project, and to Dr. Johnson for serving on the committee.

The author gratefully acknowledges the financial support of the Law Enforcement Assistance Agency and the Electrical Engineering Department at University of Houston.

CONCURRENT MULTI-TERMINAL

OPERATION OF THE 'CRIME' SYSTEM

An Abstract of A Thesis

Presented to

The Faculty of the Department of Electrical Engineering

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Electrical Engineering

---

by

Asif Karachiwala

July 1976

ABSTRACT

CRIME File System is a computerised data base system being used by the Oakland Police Department to aid investigators in identifying suspects on basis of known characteristics. The system can accomodate upto six investigating terminals, and the object of this thesis project is to study the extent to which time sharing can be introduced in the system with respect to these terminals, and incorporate the same.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 CRIME File System.

1.1.1 Purpose.

'Computerised Retrieval of Identifiers and Modus Operandi Elements' is the approach taken by the Oakland Police Department in utilising advanced technology for criminal investigation. Before the advent of computer random search systems, many of law enforcement's traditional crime fighting resources broke down under the sheer volume of crime. In police department such as Oakland, where 40,218 persons were arrested in 1970, the information necessary to identify the perpetrator of a crime was very often 'hidden' within a document storage facility. Hence the department wanted a fast, reliable and convenient means of searching sizable storage of identification elements of people, fingerprints and vehicles. Also, the system had to provide a means of updating information, and of presenting visual records of persons matching the observed characteristics.

1.1.2 System Description.

The CRIME File System was designed and built by Hewlett Packard Company and has been in operation since September 4, 1972. The system employs a computerised data base as a source of investigative leads to help in identifying suspects whose characteristics are partially known. There are four major characteristics contained in the data base; they are physical description, type of crime, address of mug shots and fingerprint cards, and description of vehicles with occupants of recent police interest. By entering known characteristics into the system, a search on the data base can yield a summary list and/or display of mug shots or fingerprints of all listed persons who match the information entered.

1.1.3 Hardware Configuration.

Hardware configuration of the system consists of :

- HP2100 Computer with 24K of memory.
- Two HP7900 Cartridge Disc Drives (12960-010).
- HP2748A Papertape Reader.
- HP2752A Teleprinter.
- HP2761A Mark Sense Card Reader.
- Two terminals each consisting of one IMAGE SYSTEMS microfilm unit with 100 address buffer and one KSR 33

teletype.

Note: The system is designed to handle expansion upto six terminals.

- HP12563A - Five spare disc cartridges.

Note: The total system contains seven removable disc platters (cartridges) and two fixed disc platters (one in each HP7900A).

1.1.4 Software Architecture.

General software architecture of the system consists of special application programs in HP's Moving Head Disc Operating System (DOS-M) with Extended File Management Package (EFMP). These programs are under the control of the operator at the system console. There are four programs each consisting of several segments, they are :

- UPDAT for updating data base.
- QUERY for implementing search on data base.
- INTIL for initializing data base.
- VERUP for verifying the file structure of data base.

UPDAT, VERUP and INTIL can be executed only from the system console, and INITL and VERUP are activated only when a new data base is to be initialized. While QUERY is initiated and terminated from the system console but accepts search commands and outputs results to the query terminals;

however only one of the six can be active at any one time.

The data base is divide into two functional files, the Subject File (SF) and Vehicle File (VF). Actual file structure is designed to provide fast retrieval for frequently used parameters, by maintaining reference files sorted on different characteristics. All files associated with SF and VF data bases are contained on three disc packs, and the fourth disc pack is used for DOS-M and CRIME programs.

1.2 Project Problem.

System experience with CRIME File System revealed appreciable success, as indicated by the Project Report of Oakland Police Department. However their evaluation also pointed out certain problems. One of their major complaints was that, if one terminal is executing the QUERY program, all other terminals are locked out until that terminal is released. Query from one terminal involves appreciable amount of system - investigator conversation before a search can be initiated. And since the system is locked to one terminal at a time, considerable time is wasted while the system is waiting for slow human response to its inquiry. Extreme is, of course, the case when the investigator forgets to give the terminal release command at the end of his query job. Hence there was a competetion for the use of the system, and many times very promising jobs had to be aborted due to delayed access to the system. Hence it was highly recommended to incorporate in the system a sort of time sharing capability to enable multiple inquiry from terminals.

The object of this thesis project is to study the extent and the manner in which the problem outlined above can be tackled; and incorporate necessary changes in the system.

1.3 System Implementation.

Project work was to be carried out at the Image Analysis Labratory of Electrical Engineering Department, University of Houston; and hence the CRIME File System had to be implemented on the available resources. Though these were quite compatible to the requirements, some minor differences warrented certain changes in the original system.

- The CRIME File System was designed to operate under HP's DOS-M system, while we needed to implement it under the DOS-III system. These two DOS systems are quite similar except for certain minor differences in the EFMP routines. Basically, the DEFINE statement, that defines the number of words to be used by EFMP for its internal tables and buffers, had to modified in all program segments.

- At the Oakland Police Department, hardware configuration included a two-drive disc unit, giving a total of four disc cartridges. At the Image Analysis Labratory we have a single-drive unit. Hence to be able to implement the CRIME File System on a two-disc system instead of a four-disc system, the size of the data base had to be reduced by a factor of ten. Thus the maximum number of entries in SF was reduced from 25,200 to 2,520, and VF was reduced from 31,500 to 3,150. This was achieved by making appropriate changes in the data base initialization program INITL.
- Oakland Police Department had obvious reservations in releasing their data base, hence a synthetic data base was generated. And since we did not have a card reader at the Image Analysis Labratory, card images were generated and stored on magnetic tape, and then read-off the tape by the UPDAT program. A Synthetic Data Base Generation Program was developed by this author in conjunction with Ha Nguyen, and it is documented in his thesis.

In this thesis the author has tried to explain all the aspects of the CRIME File System that are relevant to this project. However, for documentation of the complete original system, the reader is referred to :

- Oakland Police Department CRIME File System Project Report.
- Oakland Police Department CRIME System Internal Maintenance Specification.
- Source listing of the CRIME programs.

CHAPTER 2

SYSTEM DESIGN

2.1 Design Concepts.

The idea of incorporating time sharing capability in the CRIME system, is based on the reasoning that because human thinking processes and responses are slow relative to the logical and arithmetic capabilities of the computer, it should be possible to switch the computing resources from one user to another in such a way that each user could interact with a terminal online to the computer and think he had sole access to the computer. Now, I/O devices also have a speed disadvantage compared to the Central Processing Unit. Hence if the user is considered as much a part of the system as I/O devices, the idea of time sharing would be to share system resousces sequentially in time.

Time sharing systems can be classified as :

- Online File Maintenance and Retrieval Systems.
  These systems are characterized by the limited range of queries or additions which can be made to a common information base.
- Special Purpose Time Sharing Systems.

These systems allow the user to prepare and execute programs in a very limited number of languages.

- General purpose time sharing systems.

CRIME File System is data base system, and hence after time sharing is incorporated, it could be included in the first classification. However, since updating of the data base is done only from the system console, time sharing need only be incorporated in the QUERY section. Thus the object is to design an Online Retrieval System with time sharing capability.

There may be several users wanting to use the QUERY program at the same time, if time sharing is introduced. One way would be to provide seperate copies of the program for each user. However, considering the limited memory space of 24K words and the sizable QUERY program, an alternative way obviously need be sought. Hence let there be just one program being shared, but each user using it does so as a seperate process; and the processes run concurrently. By concurrent, we mean that two or more processes are in a 'state of execution'. A process is in a 'state of execution' if it has been started but not completed or terminated. Such a concurrent execution of two or more processes is called 'multiprogramming', and is employed in this project.

A single copy of the program which can be used concurrently by several processes is called a 'pure procedure' or is said to be 'reentrant'. For a program to be reentrant it must not modify itself, hence it was necessary to avoid any instruction modifying programming techniques in the QUERY program. Secondly, the program should not store data local to itself, hence separate data and temporary storage areas must be provided for each user of the program. This was taken care of in the 'context block' as explained in the next paragraph.

In order to switch the physical processor from one process to another, some information must be saved when a process is removed from control, and restored again when a process returns to control. This information is often called the 'context block'. The following is the type of information that must be saved and restored:

- The process must know what instruction to execute next when it assumes control of the physical processor.
- The address space of the process must be saved. This also ensures separate data and temporary storage areas as required for reentrant programs, mentioned in previous paragraph.
- The state of the I/O devices affecting the process must be saved.

There may be additional information required in other systems, but for QUERY this seems sufficient.

Assignment of the physical processor to processes is scheduled by 'processor management'. General description of the processor management employed in QUERY is illustrated in Figure 2.1. 'Process scheduler' and 'traffic controller' are two modules that control and keep track of state transitions of defferent processes. Process scheduler decides which of the processes receives the processor and at what time. Traffic controller keeps track of the status of each process. When a terminal user signs-on, his process is assigned READY state. Next, the process controller in conjunction with the traffic controller assigns it to the physical processor and labels its state as RUNNING. While it is running the traffic controller continuously updates the status information on other concurrent processes. When the process requests an I/O, it is put into WAIT state until the I/O request is complete, and then it is assigned READY state again. Each process has an identical state diagram. It is worthwhile to note here, that all the processes are identical since they all execute the same program. Secondly, as will be discussed later, no time slice allotment was employed, and hence in the state diagram, there is no direct path from RUNNING state to READY state.

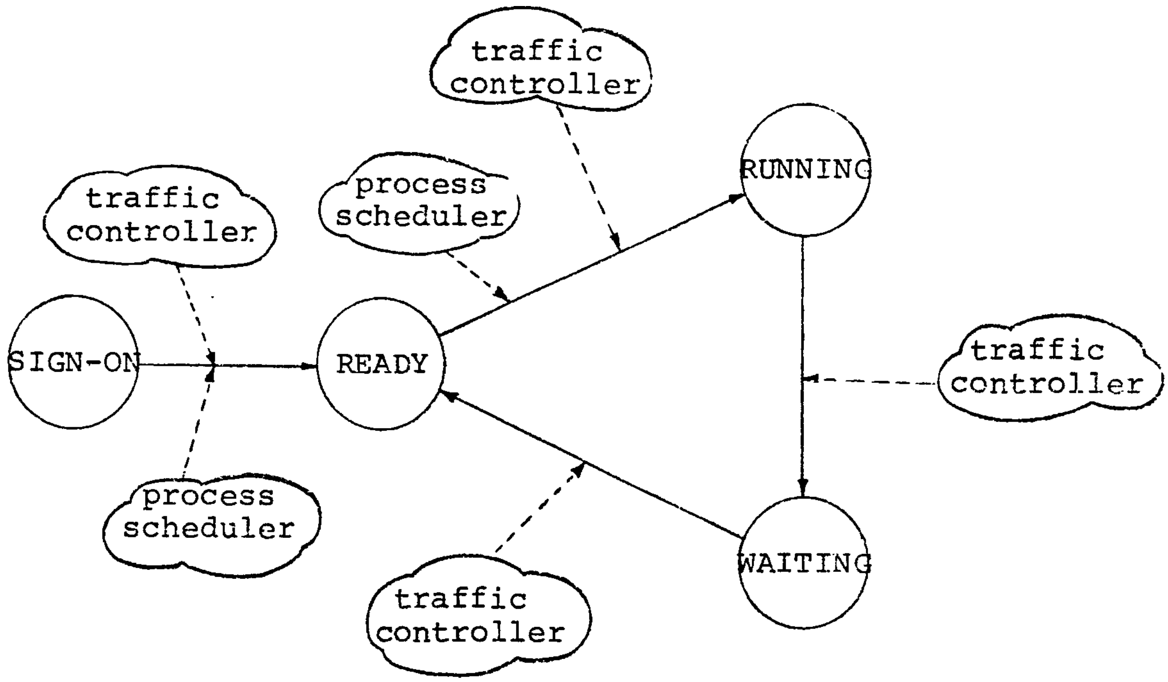

traffic
controller
process
scheduler
RUNNING
traffic
controller
SIGN-ON
READY
traffic
controller
process
scheduler
WAITING
traffic
controller

Figure 2.1

2.2 Design Implementation

Now that general design concepts have been presented, let us probe deeper into how these were implemented in the QUERY system.

The overall flow chart for the segments of the original QUERY program is shown in Figure 2.2. In brief, the functions of each of these segments can be described as :

- QYERY : This is the main program to which control is given by DOS directive from system console to initiate the QUERY program. It is a dummy main program for loading purposes and establishes a common block of 128 words for use throughout QUERY.
- QSEG1 : This overlay segment initiates operator communication through the system console, checks the validity of disc packs to be used, and verifies the active terminals by L.U.N. (logical unit number) and initializes certain common buffers and flags.
- QSG1A : This overlay segment polls all terminals for attention to sign on, and once a terminal has signed on, it transfers control to QSG1B for query commands.
- QSG1B : This overlay segment is responsible for controlling the operation of the query functions and does all user communications. It accepts and performs the

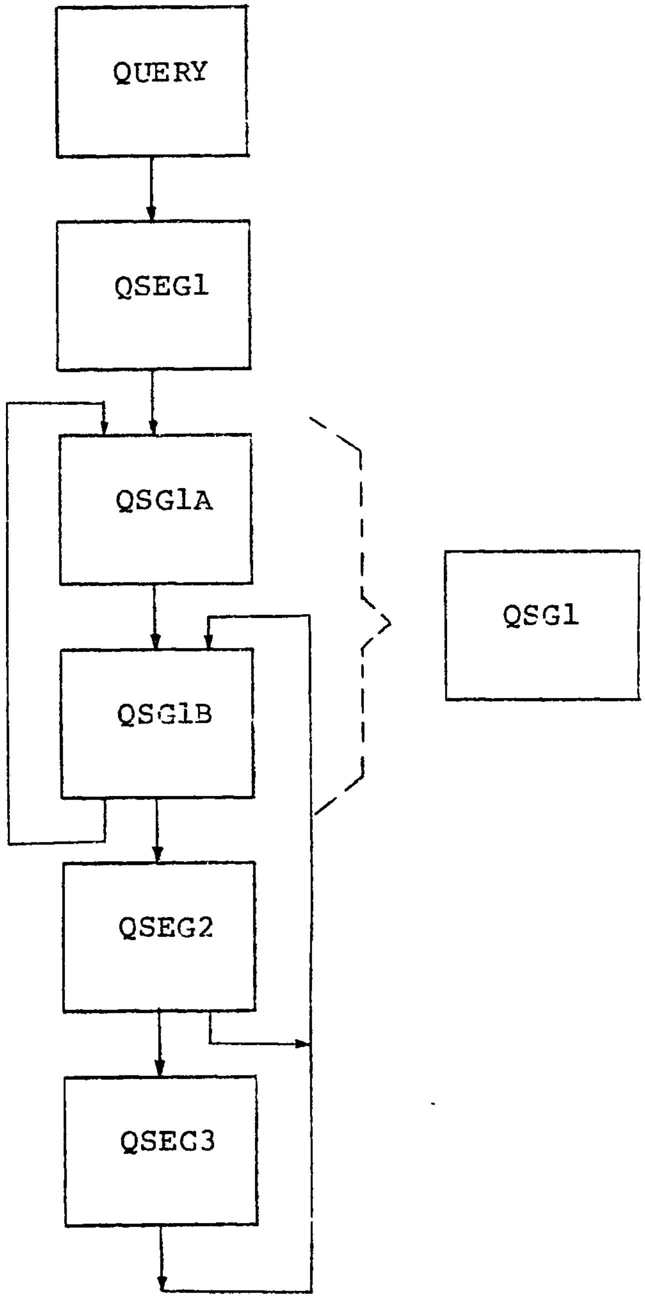


Figure 2.2

various query commands entered at the terminal. And prior to doing any I/O or EXEC calls, it checks to see if other terminals are requesting attention.

- QSEG2 & QSEG3 : These two overlay segments actually handle the search on data base based on entered characteristics. In the time sharing system these run to completion before releasing control, therefore these do not have appreciable effect on the object of incorporating time sharing in the system and they need not be further elaborated.

It can be seen from the above description that since segments QSG1A and QSG1B handle the terminal sign on procedure and user communications, these will have to be modified in order to implement time sharing. Minor modifications can also be envisioned in QUERY and QSEG1.

Segment QSG1A polls the terminals for attention to sign on, and transfers control to QSG1B when a terminal signs on. It is very possible that after a process for one terminal has reached QSG1B, some other terminal would want to sign on. In this case, switching control from one terminal to another would require jumping back to segment QSG1A, and hence would require swapping of overlays between main memory and disc storage. This would also require files to be opened and reset, and certain buffers to be initialised in each segment each time they were swapped. Hence to prevent this unnecessary overhead, it was pertinent to combine QSG1A and QSG1B into one overlay segment QSG1. And since QSG1A was comparatively small segment, the combined segment QSG1 did not overflow the memory.

Now let us discuss the information management required for multiprogramming QUERY, ie. the context block for each process. In the original version of QUERY, all the variables which were modified during the execution of QUERY, were assigned a common block of 128 words. However there were some unused locations in this block. These came in handy for assignment to certain variables local to QSG1, such as loop variables and flag variables. The idea was to maintain a copy of this common block for each process and swap them whenever control was switched from one process to another. Hence it was necessary to store all the variables that were manipulated during QSG1, in this common block.

A second common block was inserted to store certain status variables for the terminals and for process scheduling queue, and buffers in which information read from differnt terminals could be stored. Also included in this common block was a 128 × 6 word array , in which a copy of the first common block for each terminal could be stored. Detailed description of all these variables and arrays is given in Chapter 3, but the object behind this brief mention here is to indicate the division of information storage area into two blocks, one of which needs to be swapped each time the processes are switched, and the other which maintains the status of each process and can be modified or tested during execution of any process.

The file structure of CRIME File System was not of much consequence in this project, hence is not discussed in detail. However there are certain files whose mention is very pertinent here. The characteristics that are entered during execution of QSG1 are stored in a master interface file 'MIF', and when control is passed on to the search routines, search is performed based on parameters in this file. The hits encountered in the search are passed back to QSG1 in a master hit file 'MHF'. There are also six hit files 'HIT1' to 'HIT6', one for each terminal, for saving the hits for future referance once that terminal user gives the 'END' command to end his query job. With the inclusion of multiprogramming, search parameters can be concurrently entered from many terminals, and hence a separate interface file was created for each terminal - 'MIF1' to 'MIF6'. Each of these is 96 words long. Since hits obtained during search routines also need to be saved for

each terminal, information in MHF on return to QSG1 from search routines, is immidiately transferred to appropriate hit file 'HIT#'. In the original system, the size of file HIT1 was 8192 words while HIT2 to HIT6 were 256 words long. These same sizes are maintained, however if the number of hits in MHF is greater than the size of file HIT#, a warning to that effect is printed on the appropriate terminal so that the investigator can rerun the search with more parameters, thereby reducing the number of hits.
At this stage the above mentioned sizes of hit files was considered suitable, however if need be, it would not be too difficult to increase the sizes of these files.

Now let us discuss the general approach employed in scheduling and switching control from one process to another. In general, conversation of the system with the terminal user follows a sequence. The system asks a question of the user or it just gives a prompt requesting further information, and the user responds with the information. The system processes this input and outputs a prompt for signaling next input or comments on the previous entry and then outputs a prompt. It was observed, as could be expected, that system response to user's input was instataneous from user's standpoint, therefore it was considered unnecessary to assign fixed time slices to the process steps. Hence here, alloted time slice was the time interval, beginning when the user input was accepted by the system for processing, to the time the system response to this input was output to the user's terminal. Thus time slices were defferent for each process step, but could not be recognised by the user. The strategy was to utilise the time, while the system is awaiting a terminal user's input, in polling other terminals and employing round robin scheduling for switching control to another process.

As mentioned before, there are two important routines, process scheduler READA and traffic controller POLL. Now let us consider the significance of these.

QSG1 is divided into 29 program steps ie. there are 29 entry points. These entry points are given statement numbers 5201, 5202, ..., 5229. Whenever the program reaches a point where it needs to read from the terminal, READA is called. The general pattern of coding at this point is :

```
              NSTMT ← ##
              CALL READA
              GO TO 5200
        52##  ..........
```

Here NSTMT is the next statement number for the current process, that is passed on to READA. On exit from READA NSTMT holds the next statement number for the new process. Statement numbered 5200 is:

```
5200 GO TO (5201,5202,.........,5229)NSTMT
```

Hence on exit from READA, control will be transferred to appropriate entry point for the new process. A similar coding pattern is also employed where PRINT directive from the terminal user is processed, to interleave printing of the entries in 'hit files'. This will be clear when we consider an example later in the Chapter. Traffic controller is called from within READA and also at several places in the QSG1 program. It updates the status information of various terminals and the scheduling queue.

Each of the terminals has two status flags assigned to it, they are IOCMND and IOSTAT. IOCMND when set, indicates that I/O READ EXEC routine is to be called for that terminal when that I/O device becomes free. While IOSTAT when set, indicates that the process is ready to be scheduled for running ie. ready to be included in the scheduling queue, when the I/O device becomes free. These two flag variables are employed to provide concurrent I/O on the terminals. The object was to be able to replace all Fortran READ and WRITE statements by I/O EXEC calls which would initiate the I/O and then without waiting for completion, transfer control to he next instruction in the program. The flow charts for READA and POLL are shown in Figures 2.3 and 2.4, and in conjunction with the above discussion they become quite self explanatory. There are however couple of things that may need elaboration. The KOMON area mentioned in Figure 2.3 is the common block which is individual to each process and needs to be swapped each time processes are switched. Secondly each terminal is assigned a buffer of 36 words into which information from that terminal is read. And before exiting from READA, contents of the appropriate buffer is transferred to a local buffer of 36 words in the KOMON area.

At this point let us digress to an important aspect of the extent to which time sharing is to be provided in QUERY. During the time that search is being executed (ie. control is being transferred to QSEG2 and QSEG3 as in Figure 2.2), there is no system - user conversation. Further, it was observed that for most cases the time required for searching was not intolerably long. Hence it was decided to allow search to be performed uninterrupted. However consider a case where one user has given the PRINT command and while the hit list is being printed on his terminal, some other user enters the search segments. Since printing is interleaved, it is possible that in between printing of

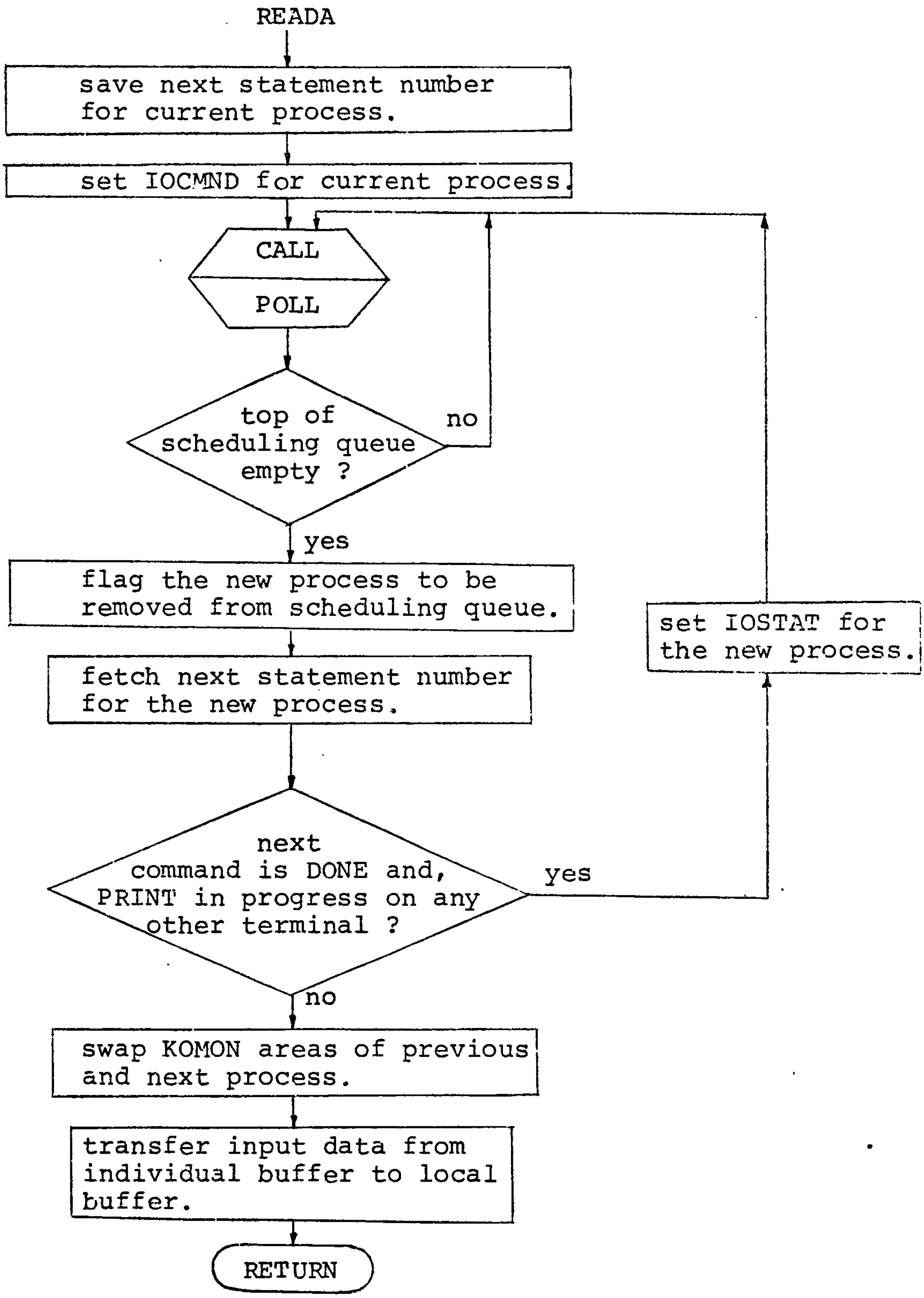

READA
save next statement number for current process.
set IOCMND for current process.
CALL
POLL
top of scheduling queue empty ?
no
yes
flag the new process to be removed from scheduling queue.
fetch next statement number for the new process.
next command is DONE and, PRINT in progress on any other terminal ?
yes
set IOSTAT for the new process.
no
swap KOMON areas of previous and next process.
transfer input data from individual buffer to local buffer.
RETURN

Figure 2.3

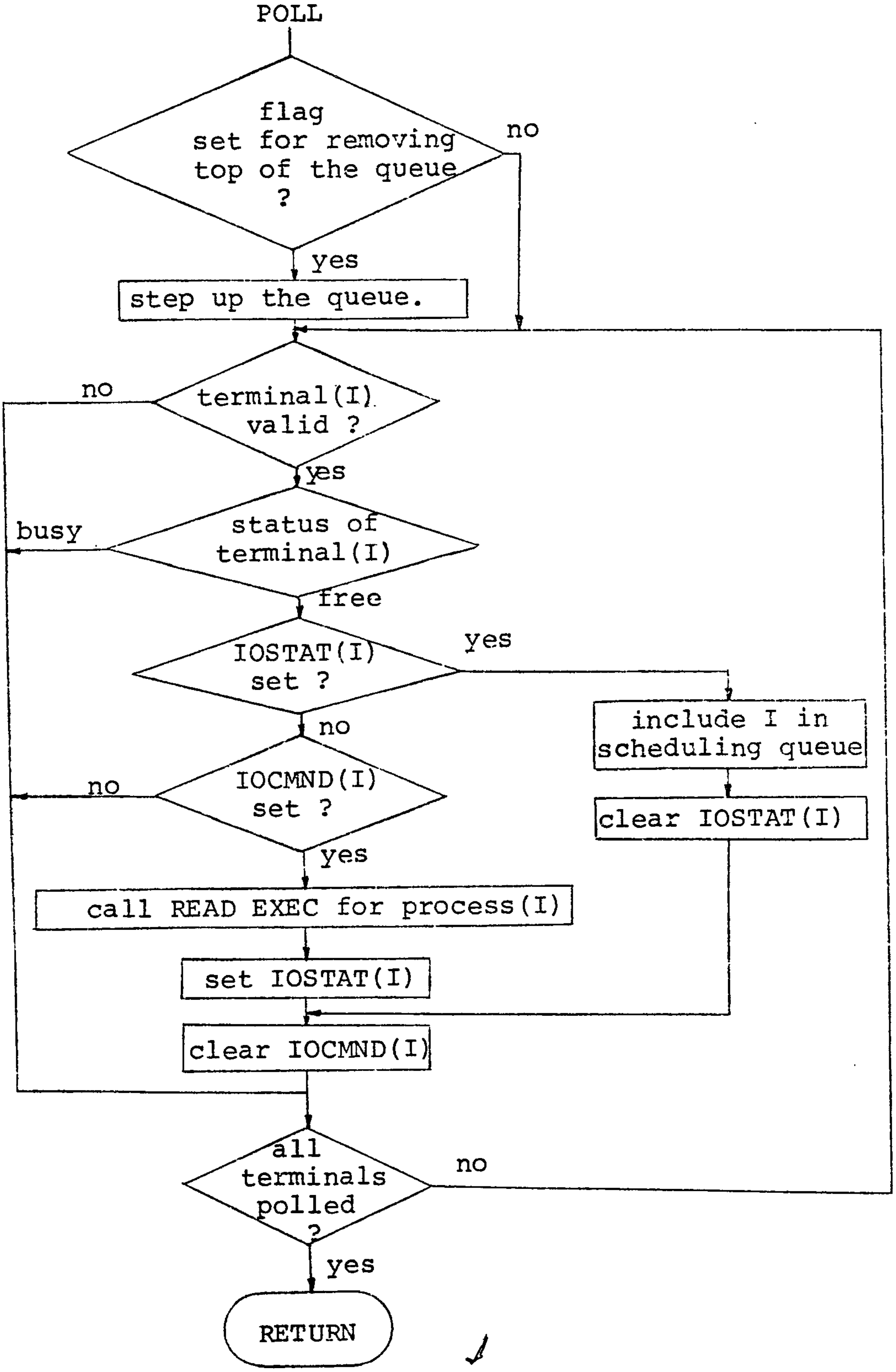

POLL
flag set for removing top of the queue ?
no
yes
step up the queue.
terminal(I) valid ?
no
yes
status of terminal(I)
busy
free
IOSTAT(I) set ?
yes
no
include I in scheduling queue
clear IOSTAT(I)
IOCMND(I) set ?
no
yes
call READ EXEC for process(I)
set IOSTAT(I)
clear IOCMND(I)
all terminals polled ?
no
yes
RETURN

Figure 2.4

two lines, the other terminal may enter search implementing routine, causing noticable time lag between two lines. This was considered undesirable and hence PRINT was given priority by introducing the feedback loop in READA, as shown in Figure 2.3.

2.3 Example.

Now let us consider an example to get a clear picture of how process switching and concurrent I/O is achieved. For simplicity, let us assume time sharing between just two users. As shown in Figure 2.5, say process 1 has reached statement 15, and next transfers control to READA, where immediately NSTMT(1) is set to 3. At this instant say, next statement for process 2 is 5202 ie. NSTMT(2) is 2, and there is a READ EXEC call pending on terminal 2. Let IOSTAT(1), IOCMND(2), IOCMND(1) be reset and IOSTAT(2) be set to 1, and the scheduling queue be empty. READA will set IOCMND(1) and call POLL. In POLL, since there is no I/O pending on terminal 1 and since IOSTAT(1) is 0 & IOCMND(1) is 1, a READ EXEC call will be given on terminal 1, and now IOSTAT(1) will be set and IOCMND(1) reset. If terminal 2 has still not completed its input, now both terminals are busy, hence queue remains empty. Thus READA will keep on calling POLL till one of the two

PROCESS 1

```
   15 NSTMT=3
      CALL READA
      GO TO 5200
 5203 .
      .
      .
      CALL POLL
      .
      .
C I/O WRITE
      CALL EXEC(2,KONWD,OUT16,17)
      GO TO 15
```

PROCESS 2

```
      NSTMT=2
      CALL READA
      GO TO 5200
 5202 .
      .
      .
      CALL POLL
      .
      .
  338 IOSTAT(IORDN)=1
C I/O WRITE
      CALL EXEC(2,KONWD,OUT24,8)
      NSTMT=16
      CALL READA
      GO TO 5200
 5216 .
      .
      CALL POLL
      .
      .
      GO TO 338
      .
```

Figure 2.5

terminals completes input.

Assume that terminal 2 completes input first. Since IOSTAT(2) is 1, it will be included in the queue, and READA will swap the appropriate KOMON blocks before transfering control to process 2. Process 2 now say has to process the PRINT directive, hence after certain amount of processing and calling POLL to check if terminal 1 has completed its input, it reaches an I/O WRITE EXEC call. At statement 338 IOSTAT(2) is set, before the WRITE EXEC call. This EXEC call starts write operation on terminal 2 and immidiately proceeds to next statement. Now process 2 transfers control to READA which in turn calls POLL. It should be noted that the queue is stepped up the first time POLL is called after switching processes. Hence once again the queue is empty. Now terminal 1 has an incomplete READ call on it while terminal 2 has an incomplete WRITE call on it, and until one of them completes, control will switch back and forth between READA and POLL.

Next, say input on terminal 1 is completed, and since IOSTAT(1) is set, terminal 1 is included in the queue and both IOSTAT(1) and IOCMND(1) are cleared. READA swaps the appropriate KOMON blocks, and process 2 starts executing and reaches an I/O WRITE EXEC call. It starts WRITE operation and continues executing till it again transfers control to READA. In READA, IOCMND(2) is set and control transferred to POLL.

Now terminal 2 completes its WRITE operation first, and since IOSTAT(2) is set, it is put on the queue. Both IOSTAT(2) and IOCMND(2) are cleared and after swapping KOMON areas, control is switched to process 2. Process 2 continues to execute, initiates another WRITE EXEC call and again transfers control to READA after setting IOSTAT(2). READA in turn calls POLL.

Now if terminal 1 is the first to complete the WRITE operation, POLL issues a READ EXEC call on terminal 1 since IOSTAT(1) is cleared and IOCMND(1) is set. Then IOSTAT(1) is set, IOCMND(1) cleared, and control returned to READA. Still the queue is empty, hence till terminal 2 finishes its WRITE operation or terminal 1 completes READ operation, control oscillates between READA and POLL.

This example is typical of the process swapping and concurrent I/O that would be expected in normal execution of QUERY. It is important to note here the difference between coding for issuing an I/O WRITE EXEC call in the two processes. This is so, because process 2 is shown to be responding to a PRINT directive from the user. In all places except in PRINT processing, a WRITE EXEC call is issued without either setting IOSTAT or calling READA. In all these cases there is a definite pattern of alternating READ and WRITE EXEC calls. Hence if WRITE is initiated and then if READ needs to be initiated on the same terminal, IOCMND is set, and routine POLL on detection of completion of WRITE, immidiately issues a READ EXEC call. Hence knowing the pattern of WRITE being followed by READ, there is no need to wait for completion of WRITE until a READ EXEC call is to be issued. On the other hand during processing of PRINT directive, two WRITEs can occur consecutively or a WRITE may be followed by READ; hence it is necessary for that process to wait till each WRITE is complete. The waiting time is utilised in polling other terminals for attention, by calling READA.

CHAPTER 3

DOCUMENTATION

This chapter gives the documentation of the modified QUERY program segments. Segments QUERY, QSEG1 and QSG1 are documented in their entirity, and since the remaining segments have very few modifications, the reader is referred to 'Oakland Police Department CRIME System Internal Maintenance Specifications', for their documentation.

3.1 QUERY

3.1.1 Program Objectives : A dummy main program written in Fortran IV for loading purposes. It establishes two common blocks, ICOMM of 128 words and IKOMM of 1017 words.

3.1.2 Flow of Control : Receives control from DOS-M executive and immediately transfers control to segment QSEG1.

3.1.3 Internal Arrays : INAME is a 3 word integer array that contains the name of the segment to which control is to be transferred - QSEG1.

3.2 QSEG1

3.2.1 Program Objectives : An overlay segment written in Fortran IV to which control is given by an EXEC call from QUERY. It initiates operator communication through the system console and checks the validity of disc pack labels for use in the CRIME system. It sets up and verifies the active terminals by LUN and initializes certain common buffers and flags. It also initializes 6 context blocks, one for each terminal.

3.2.2 Flow of control : Receives control from QUERY. After performing its function, control is transferred to QSG1.

3.2.3 Interface Parameters in First Common Block :

1. ITYPE is a simple integer variable that contains the code for the data file to be queried 'MS', 'FP' or 'VF' in ASCII.
2. ICMND is a simple integer variable that contains the current command (first two characters only) in ASCII.
3. IERRNO is a simple integer variable that is set to the EFMP error return code.
4. IYEAR is a simple integer variable that contains the last two digits of the current year.
5. ILUN is a simple integer variable that contains the L.U.N. of the currently active terminal.

6. NICKNM is a 5 word integer array that contains a subject nickname in ASCII or zero in the first word.
7. LUN is a 6 word integer array that contains the active terminals by Logical Unit Number (L.U.N.).
8. IBUFFR is a 6 word integer array. It is really 6 one word input buffers, one for each possible terminals.
9. IHITN is a 3 word integer array that contains the name of the current 'hit' file in ASCII, such as HIT1, HIT2, etc.
10. IFILEN is a 3 word integer array that contains the name of the current 'hit' file or file 'MHF' in ASCII.
11. IORDN is a simple integer variable that contains the ordinal number of the terminal (1-6) with respect to array LUN.
12. IDISPN is a 6 word integer array that contains a pointer (or displacement from the beginning) to the next entry in the 'hit' list (file HIT#) for use with the DISPLAY function for each of the 6 possible terminals.

3.2.4 Interface Parameters in Second Common STATEMENT :

1. LINET is a 36x6 word integer array that provides 36 word input buffer space for each of the 6 terminals.
2. LSTAT is a 128x6 word integer array that constitutes address areas for the 6 terminals where the first common block of 128 words for each terminal is stored.

3.2.5 Internal Simple Varibles :

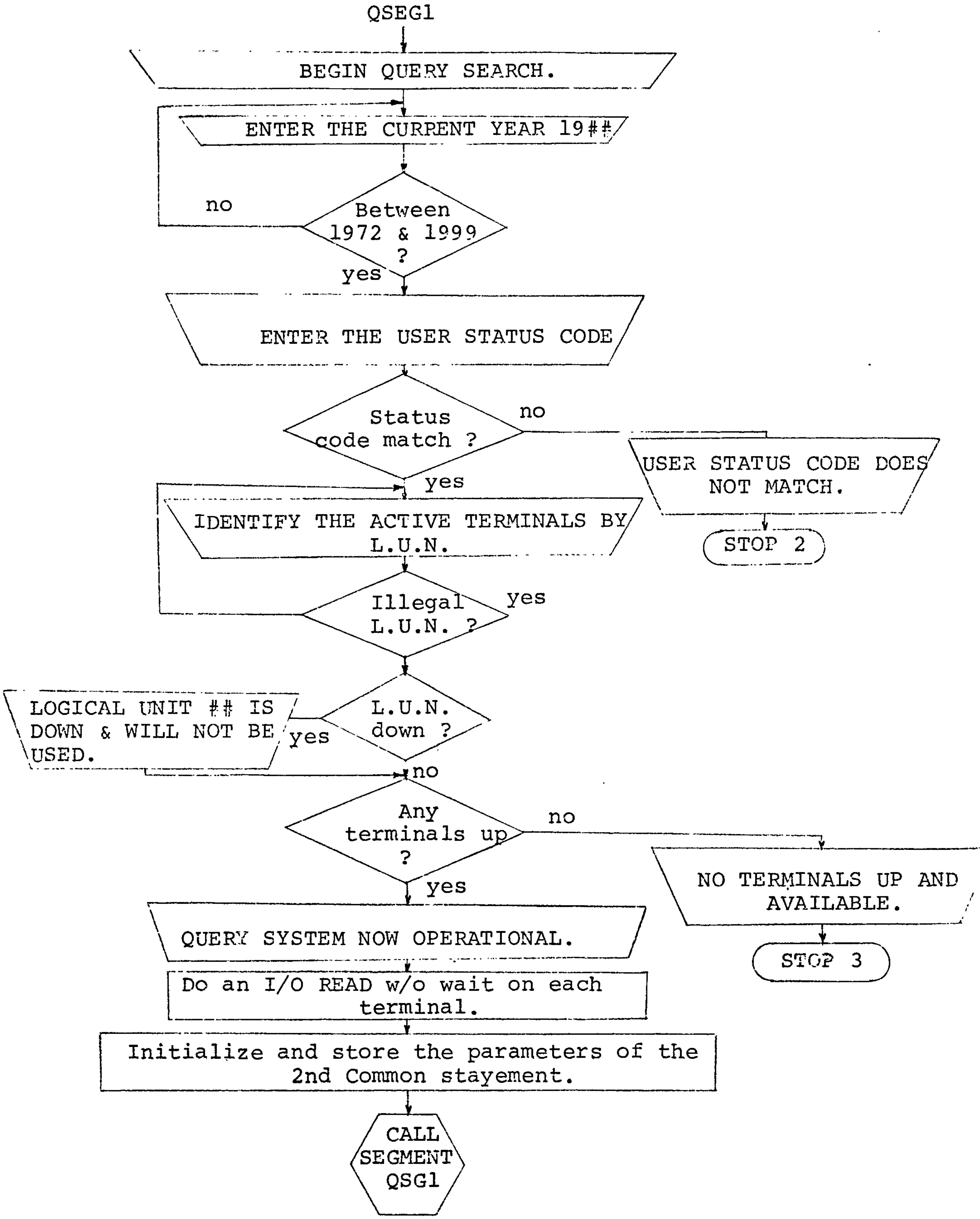
1. IOLDUS is a simple integer variable that contains the old user status code.

3.2.6 Internal Arrays :

1. IOPNTB is a 128 word integer array that contains the EFMP Opened - File Table.
2. ITRBUF is a 256 word integer array that contains the EFMP Temporary Record Buffers.
3. INAME is a 3 word integer array that contains the name of segment QSG1 in ASCII.
4. ISTATB is a 10 word integer array that contains the EFMP status of disc pack.
5. IFNAME is a 3 word integer array that temporarily contains the name of an EFMP file.
6. KOMON is a 128 word dummy array, equivalenced with the first common block for the purpose of swapping this block on switching terminals.
7. NOTRB is a two word integer array for specifying number of temporary buffers in DEFINE EXEC call.

3.2.7 Diagnostics Produced :

1. EFMP ERROR NUMBER ##
2. LOGICAL UNIT ### IS DOWN AND SO WILL NOT BE USED.



NOTE : The symbols used in the flow charts are the same as those used in the Oakland Police Department CRIME System Internal Maintenance Specification (Form 19601A).

3. THERE ARE NO TERMINALS UP AND AVAILABLE.

   QUERY TERMINATED.

   STOP 3

3.3 QSG1

3.3.1 Program Objectives :

a. QSG1 : An overlay segment written in Fortran IV, and is 'reentrant'. It accepts and performs the various query commands entered at the terminals. It is responsible for controlling the operation of the query functions and does all user communication.

b. READA : A subroutine written in Fortran IV. This is the process scheduler. It is responsible for switching control from one process to another, and swapping appropriate address areas.

c. POLL :A subroutine written in Fortran IV. It serves as the traffic controller, continuously polling the terminals for attention and consequently updating a scheduling queue.

3.3.2 Flow of control :

a. QSG1 : It receives initial control from QSEG1 . Then on, flow is controlled by the process scheduler and traffic controller routines READA and POLL. It transfers control to QSEG2 when any terminal enters the DONE command, to terminate the query codes for a SEARCH. Control is always returned from segment QSEG2 or QSEG3, following a search on the data base for 'hits'. Whether control is transferred from QSEG1 or either of QSEG2 or QSEG3, is decided by variable IERRNO.

b. READA : It is called by QSG1. Control is returned to QSG1.

c. POLL : It is called by segment QSG1 and subroutine READA. Control is returned to the caller.

3.3.3 Interface Parameters for First Common Statement :

a. QSG1 :

1. ITYPE is an integer variable that contains the code for the data file to be queried: 'MS', 'FP' or 'VF' in ASCII.
2. ICMND is a simple integer variable that contains the current command (first two characters only) in ASCII.
3. IERRNO is a simple integer variable that is set to the EFMP error return code. It is also used to determine whether the current call to segment QSG1 is from segment QSEG1 or is from completed search procedure (QSEG1 or QSEG3). The initiation call from segment QSEG1 sets IERRNO to -1. If IERRNO contains a number greater than -1, then it is the number of 'hits' found as the result of a search.
4. KFILEF is a 28 word integer array. This array is a set of

Key File flag words which correspond directly with the 28 MIF records in the Master Interface File.

5. NICKNAM is a 5 word integer array that contains a subject nicknam in ASCII or a zero in the first word, if there is none.
6. LUN is a 6 word integer array that contains the active terminals by L.U.N.
7. I,J,K,L,M,N are local integer varibles . These are included in this common block since they need to be saved for each process, and swapped when processes are switched.
8. IBUFFR is a 6 word integer array . It is really 6 one word input buffers, one for each possible terminal.
9. IYEAR is a simple integer variable that contains the last two digits of the current year.
10. ILUN is a simple integer variable that contains the L.U.N. of the currently active device.
11. IHITN is a 3 word integer array that contains the name of the current 'hit' file in ASCII, such as HIT1, HIT2, etc.
12. IFILEN is a 3 word integer array that contains the name of the current 'hit' file or file 'MHF' in ASCII.
13. IORDN is a dimple integer variable that contains the ordinal number of the terminal (1-6) with respect to array LUN.
14. IDISPN is a 6 word integer array that contains a pointer to the next entry in the 'hit' list (file HIT#) for use

with the DISPLAY function for each of the 6 terminals.

15.ITHV is a simple integer variable. It contains the limit of threshold value to be searched by the QSEG2 procedure. That is, if the number of 'hits' on the first pass of a query search is greater than the threshold value (ITHV) then the search is to continue by the QSEG2 method for the next set of characteristics, otherwise the QSEG3 method is to be performed for the remainder of the search parameters.

16.LINE is a 36 word array where the most recent input data for the current terminal is stored.

b. READA :

Same as that for segment QSG1.

c. POLL :

Same as that for segment QSG1.

3.3.4 Interface Parameters for the Second Common Statement :

a. QSG1 :

1. LINET is a 36×6 word integer array that provides 36 word input buffer space for each of the 6 terminals.
2. LSTAT is 128×6 word integer array that constitutes address areas for the 6 terminals to store the first common block of 128 words, one for each terminal.
3. STMT is a 6 word integer array that saves the next statement number for each terminal.
4. KUEUE is a 6 word integer array that constitutes the

scheduling queue.

5. NEXT is an integer variable that holds the number of the next terminal that is to take over control.
6. NSTMT is an integer variable that contains the statement number to be executed next, for the process currently in control.
7. LAST is an integer variable that specifies the last valid entry in the queue KUEUE.
8. NBYTES is an array of 6 words where the number of characters read in from the terminals are stored.
9. IOSTAT is a 6 word integer array which when set, indicates that the corresponding process is ready to be included in scheduling queue.
10. IOCMND is a 6 word integer array which when set, indicates that I/O READ EXEC call is to be issued when the device becomes free.

b. READA :

Same as that for QSG1.

c. POLL :

Same as that for QSG1.

3.3.5 Internal Simple Variables :

a. QSG1 :

1. CII contains the subject I.D. number in C.I.I. format (seven digits). This variable is double precesion floating

point.

2. IFINGR is an integer array. It contains an offset value used in computing the subscript value for referance in array ICOMM (a scratch area). In this application it is associated with the 10 finger codes.
3. OPD is a real variable that contains the subject I.D. number in O.P.D. format (six digits). It is equivalenced to CII to conserve space.

b. READA :

None

c. POLL :

None

3.3.6 Internal Arrays :

a. QSG1 :

1. IOPNTB is a 310 word integer array that contains the EFMP Opened - File Table.
2. ITRBUF is a 768 word integer array that contains the EFMP Temporary Record Buffers.
3. NOTRB is a 2 word integer array for specifying number of temporary record buffers in the DEFINE statement.
4. IMIF is a 3 word integer array that contains the name of file MIF1 or MIF2 .... orMIF6 in ASCII.
5. NAME is a 3 word integer array that contains the name of segment QSEG2 in ASCII.

6. IMHF is a 3 word integer array that contains the name of file MHF in ASCII.
7. ISUBJ is a 3 word integer array that contains the name of file SUBJF in ASCII.
8. IMVEHF is a 3 word integer array that contains the name of file MVEHF in ASCII.
9. IKEYSB is a 3 word integer array that contains the name of file KEYSB in ASCII.
10. IMAKEM is a 15 word integer array. Each entry contains a number that corresponds to those automobile make numbers that can have optional models. The numbers are arranged in ascending order and are taken directly from the VFI form. For example, the entry 3 is for American Motors, 12 is for Buick, etc.
11. IMAKEM is a 15 word integer array. This array corresponds directly with array IMAKEM. Each entry indicates the number of models possible for the corresponding make of automobole. For example, the make American Motors (#3) can have six possible models.
12. ISFM is a 48 word integer array. This buffer is used to read the records from master Subject or Vehicle EFMP files. The masters files are retrieved for the OPD, CII, PRINT and DISPLAY functions.
13. IMIFD is a 96 word integer array. It is the buffer used for processing the query entry data and transferring it to

EFMP file MIF for processing by segments QSEG2 or QSEG3.

14. Arrays I49, I50 and I53 are scratch areas. They are equivalenced together in common block array ICOMM to conserve space.

15. IKEYMV is a 3 word integer array that contains the name of the file KEYMV in ASCII.

16. KOMON is a 128 word integer array that is equivalenced to the first common block for the purpose of swapping the address areas when processes are switched.

17. Arrays OUT1, OUT2, OUT3, ..., OUT32 serve as output buffers.

b. READA :

1. NFLAG is 6 word integer array that indicates whether the contents of the individual input buffer are to be transferred to the local buffer LINE, or not. This is necessary since READA is also called during processing of PRINT command, at which time there is no input from the terminals.

c. POLL :

None.

3.3.7 Diagnostics Produced :

a. QSG1 :

1. EFMP ERROR NUMBER ##

STOP 5.

2. CURRENT 'HIT' LIST IS LONGER THAN THE ENTRIES SAVED.

3. I.D. NUMBER NOT IN THE SUBJECT FILE.

b. READA :

None.

c. POLL :

None.

3.3.8 Special Features :

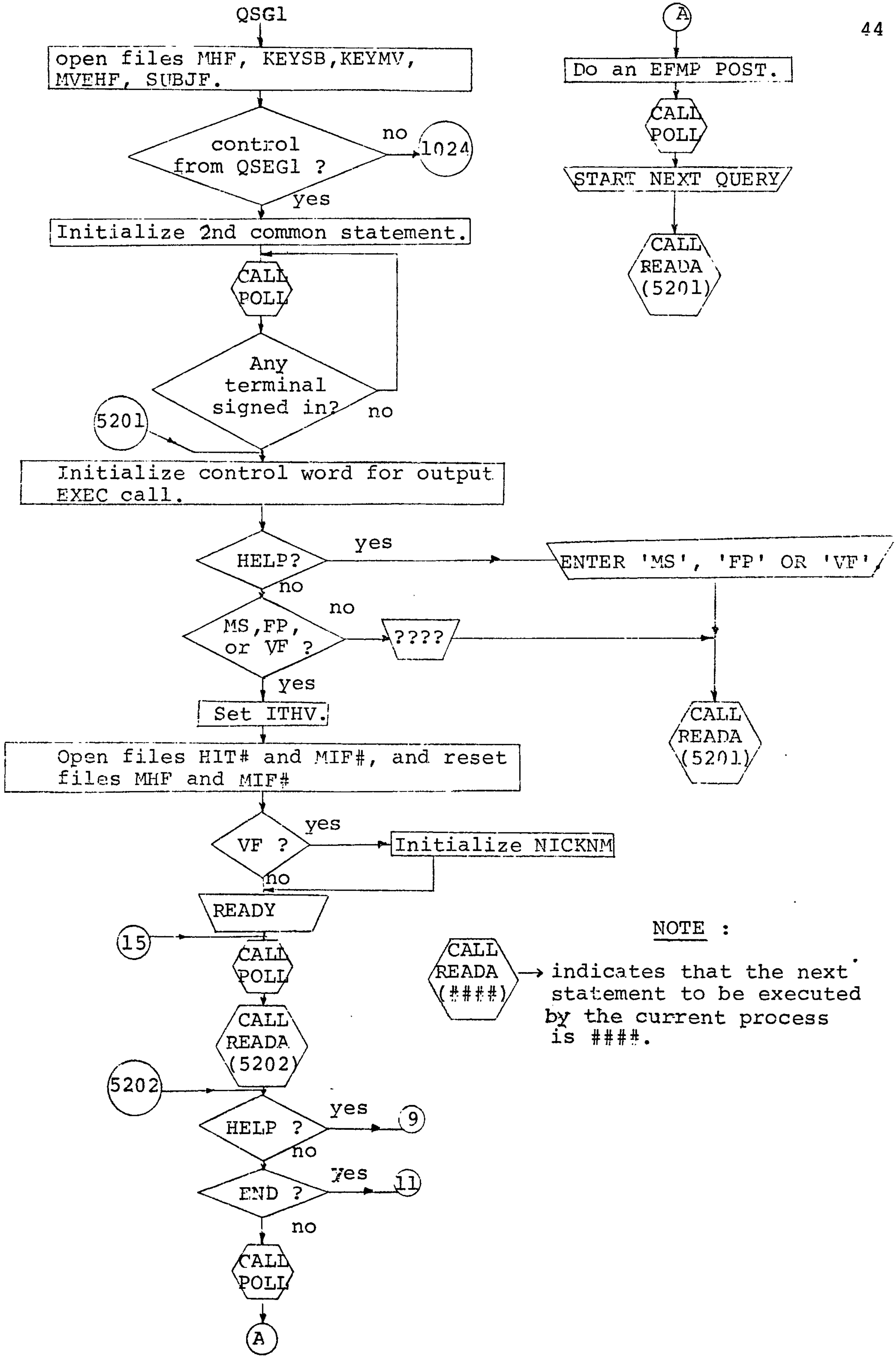
a. QSG1 : The operator can cause the termination of the printing initiated by the PRINT command, by changing the state of the switch register bit 15. This feature is used to terminate the listing of the 'hit' list in the event the user does not wish to continue after having begun. It does not matter whether switch register bit 15 is turned from off-to-on or on-to-off.

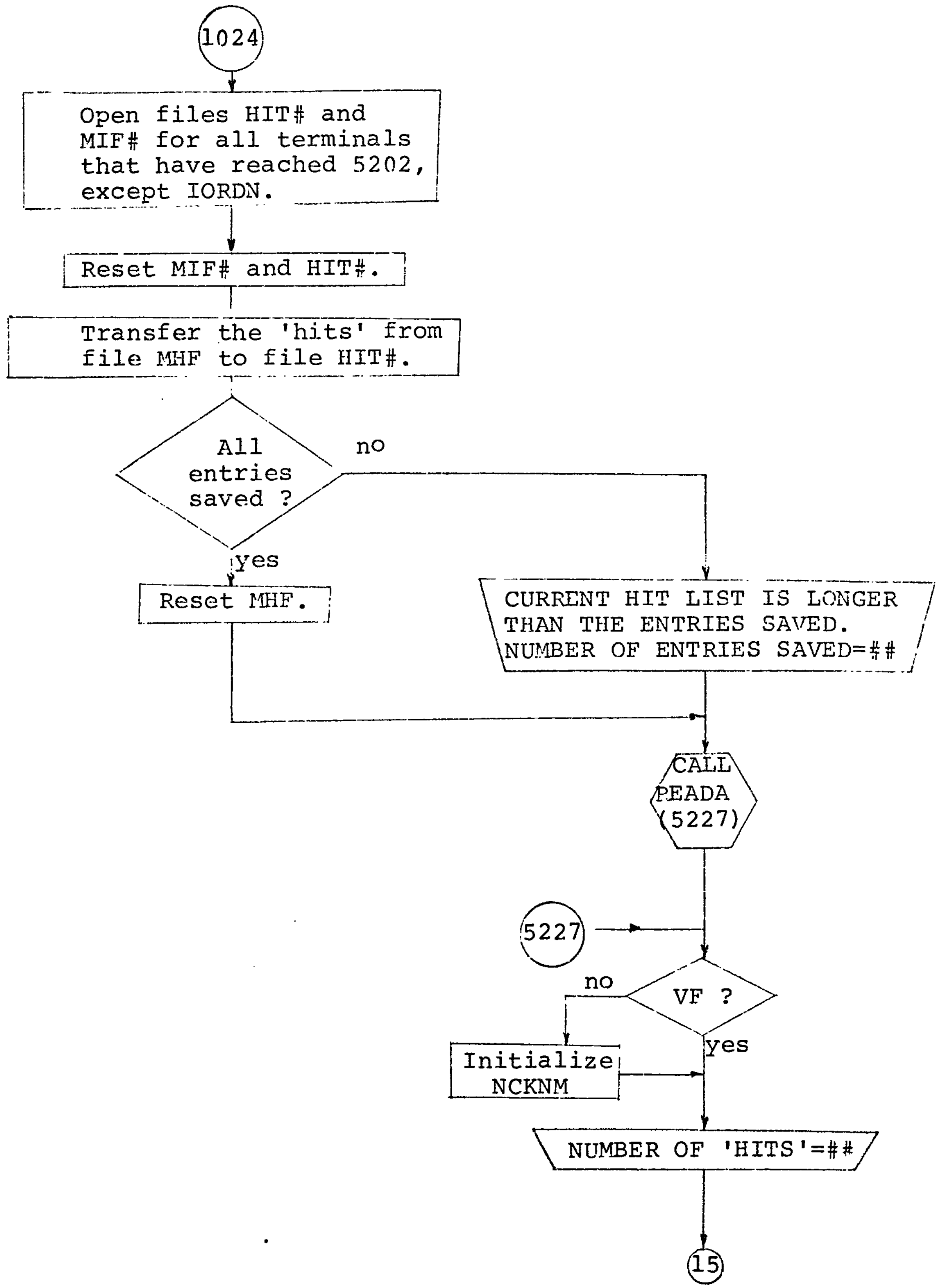
b. READA : None.

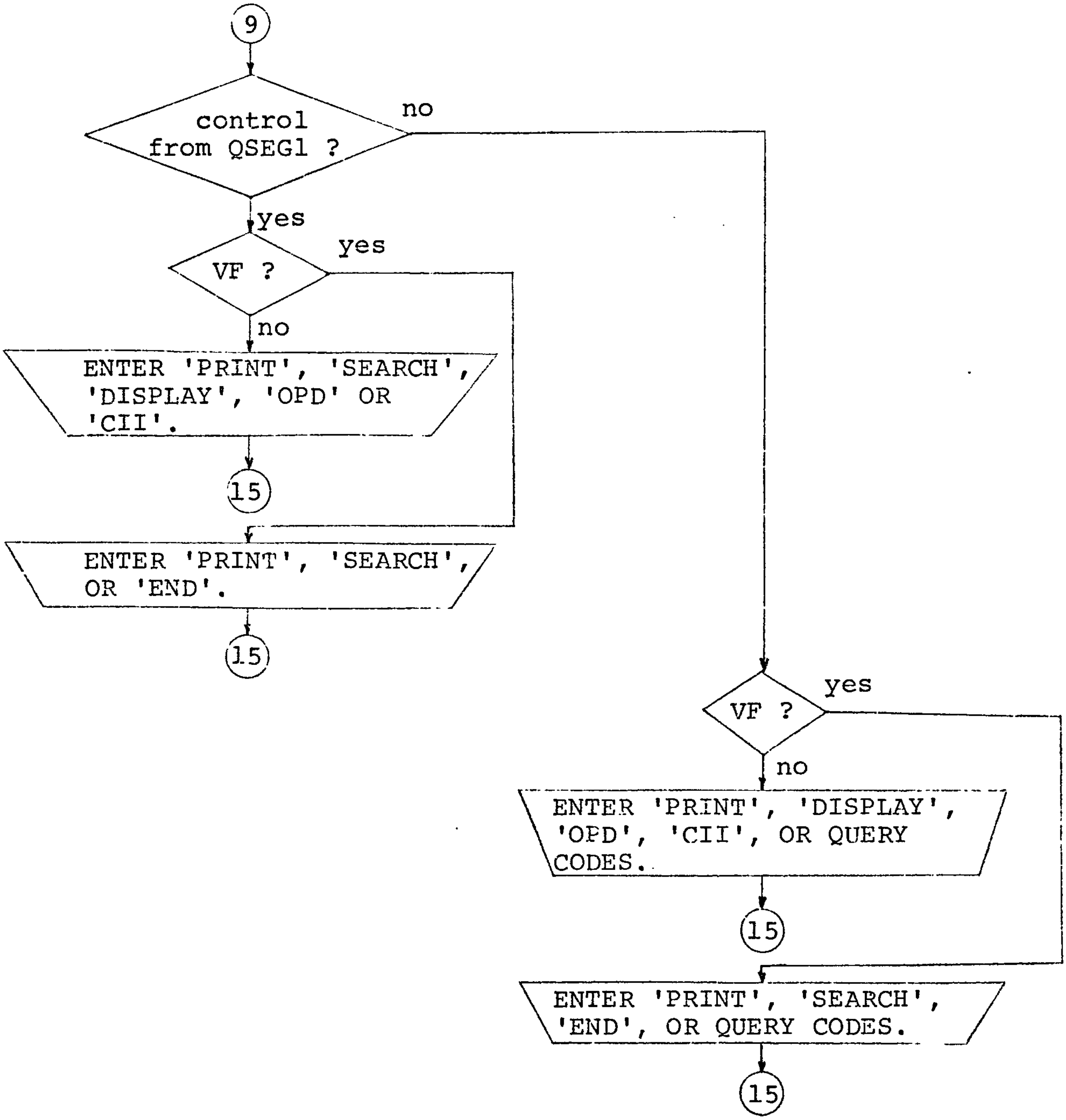
c. POLL : None.

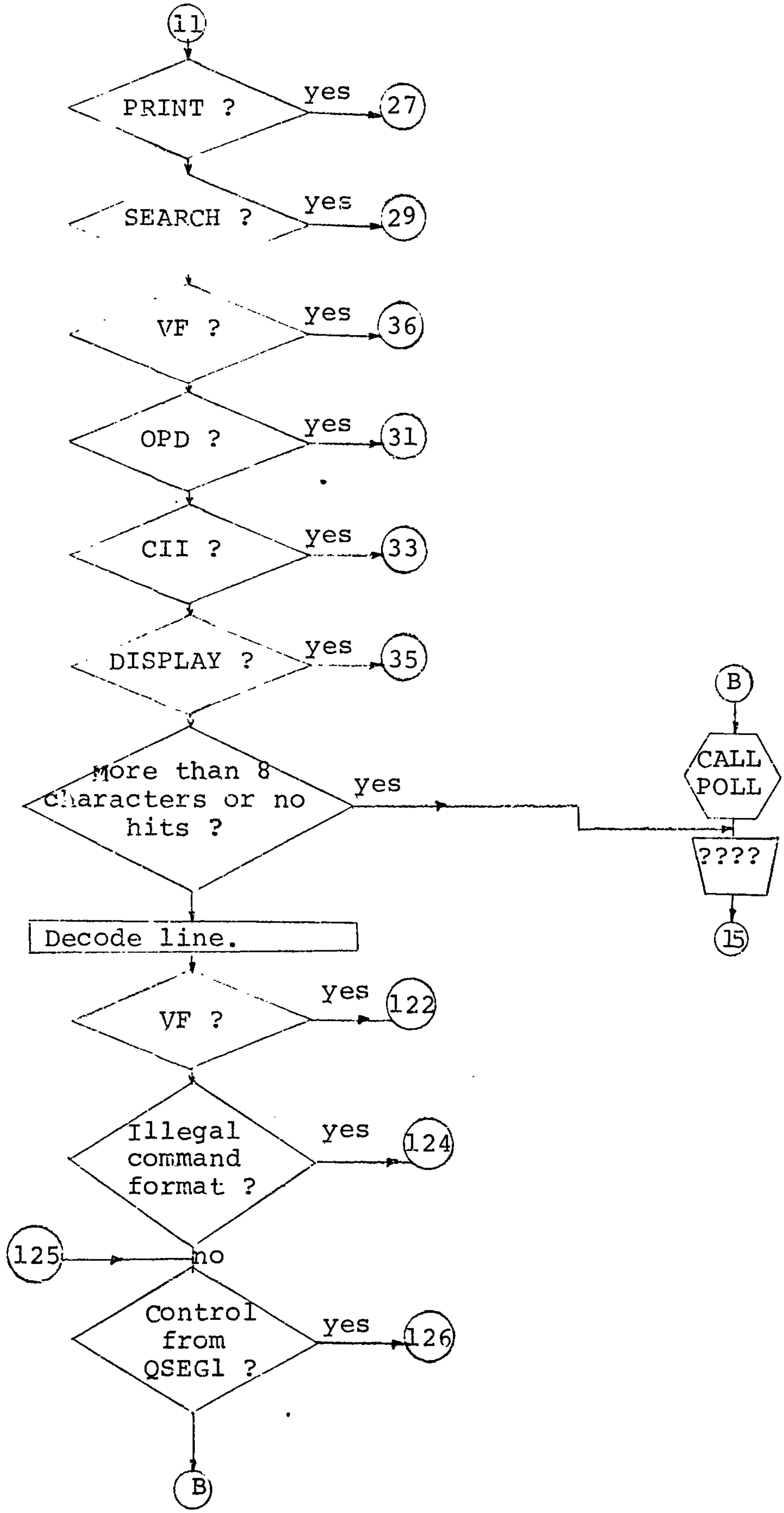
FLOW CHART FOR QSG1

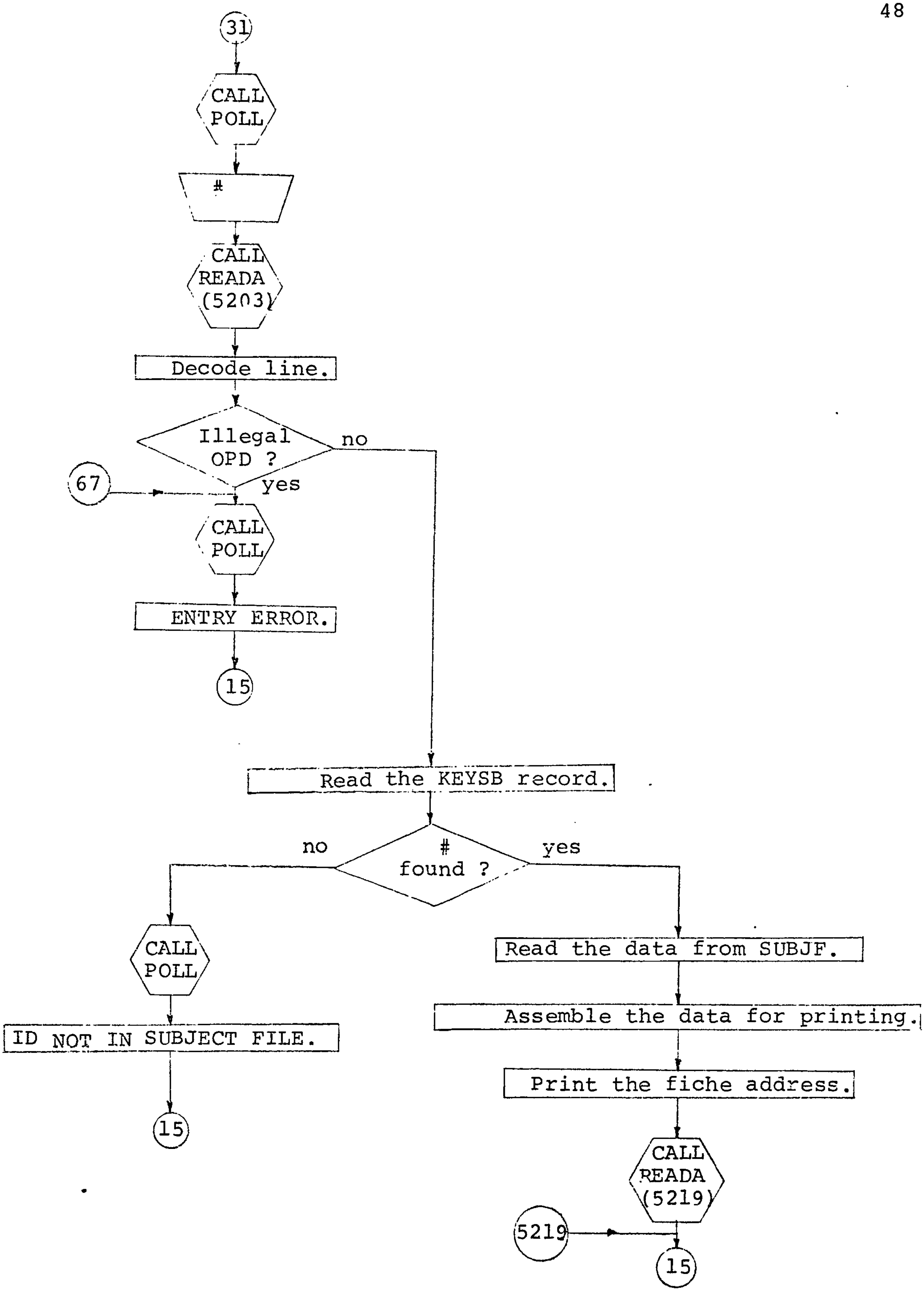
(Includes pages 44 to 60)

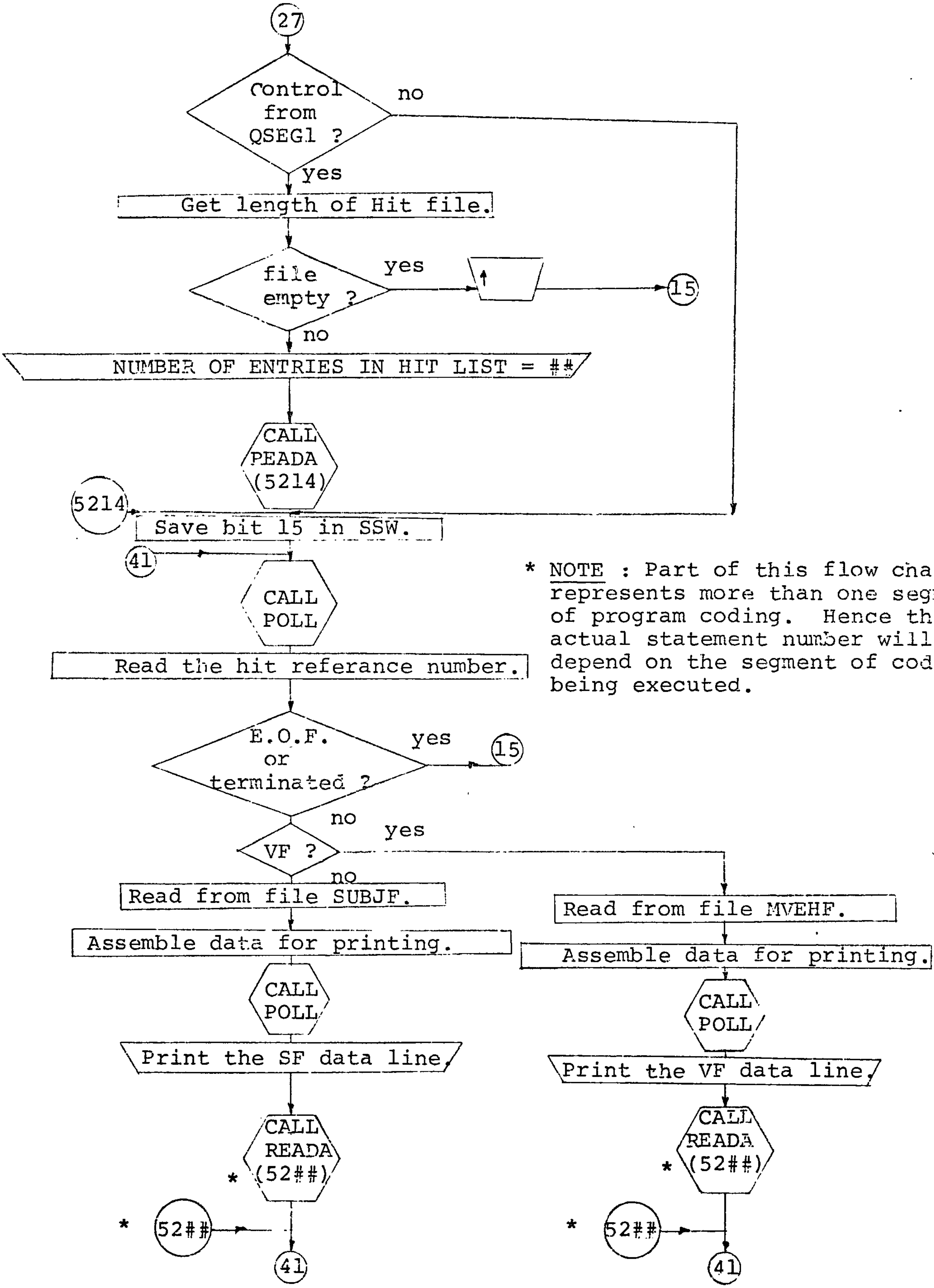

QSG1
open files MHF, KEYSB, KEYMV, MVEHF, SUBJF.
control from QSEG1 ?
no
1024
yes
Initialize 2nd common statement.
CALL POLL
Any terminal signed in?
no
5201
Initialize control word for output EXEC call.
HELP?
yes
ENTER 'MS', 'FP' OR 'VF'
no
MS, FP, or VF ?
no
????
CALL READA (5201)
yes
Set ITHV.
Open files HIT# and MIF#, and reset files MHF and MIF#
VF ?
yes
Initialize NICKNM
no
READY
15
CALL POLL
CALL READA (5202)
5202
HELP ?
yes
9
no
END ?
yes
11
no
CALL POLL
A
A
Do an EFMP POST.
CALL POLL
START NEXT QUERY
CALL READA (5201)
NOTE :
CALL READA (####)
indicates that the next statement to be executed by the current process is ####.


1024
Open files HIT# and MIF# for all terminals that have reached 5202, except IORDN.
Reset MIF# and HIT#.
Transfer the 'hits' from file MHF to file HIT#.
All entries saved ?
no
yes
Reset MHF.
CURRENT HIT LIST IS LONGER THAN THE ENTRIES SAVED. NUMBER OF ENTRIES SAVED=##
CALL PEADA (5227)
5227
VF ?
no
yes
Initialize NCKNM
NUMBER OF 'HITS'=##
15

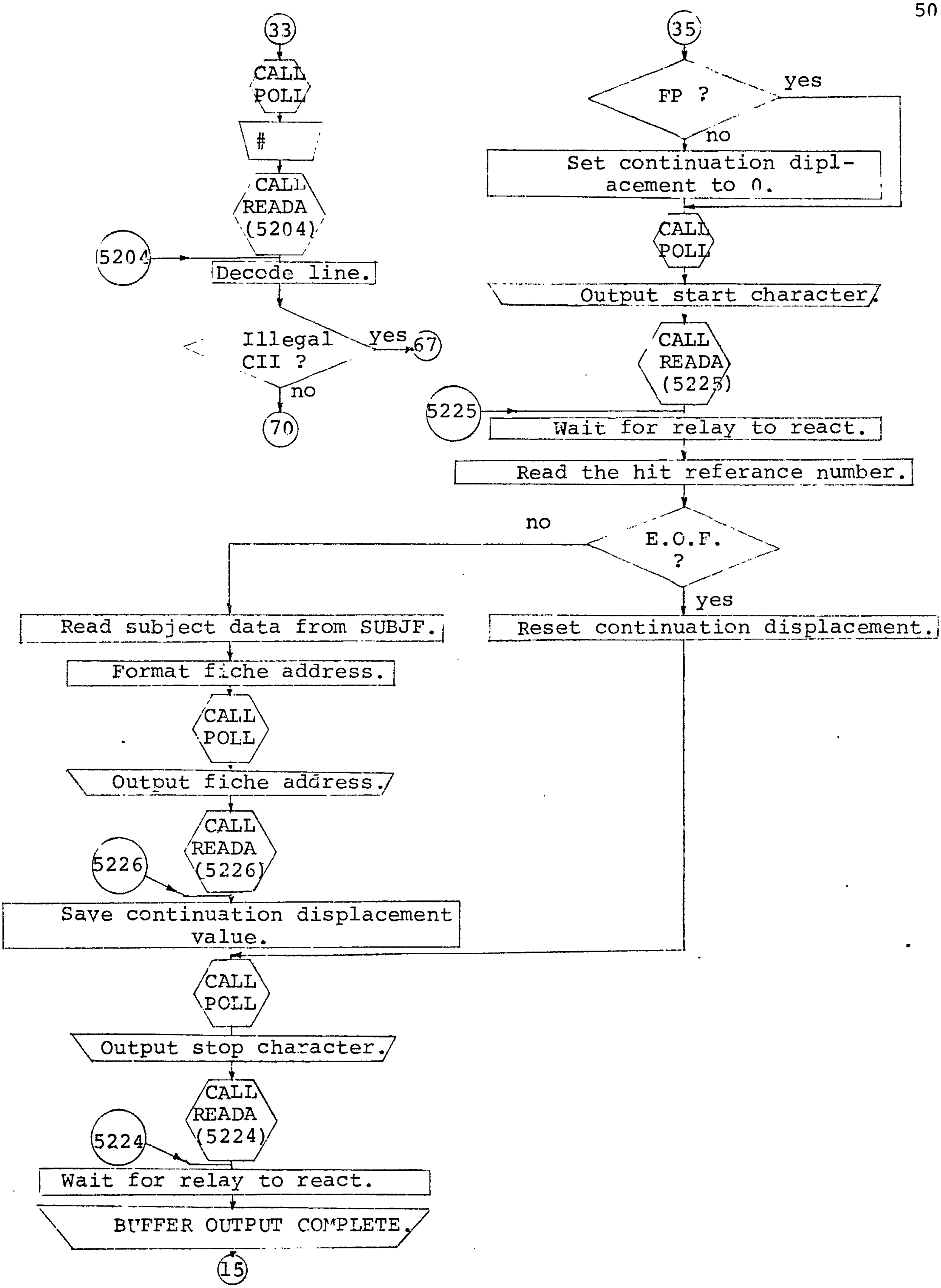

9
control from QSEG1 ?
no
yes
VF ?
yes
no
ENTER 'PRINT', 'SEARCH', 'DISPLAY', 'OPD' OR 'CII'.
15
ENTER 'PRINT', 'SEARCH', OR 'END'.
15
VF ?
yes
no
ENTER 'PRINT', 'DISPLAY', 'OPD', 'CII', OR QUERY CODES.
15
ENTER 'PRINT', 'SEARCH', 'END', OR QUERY CODES.
15

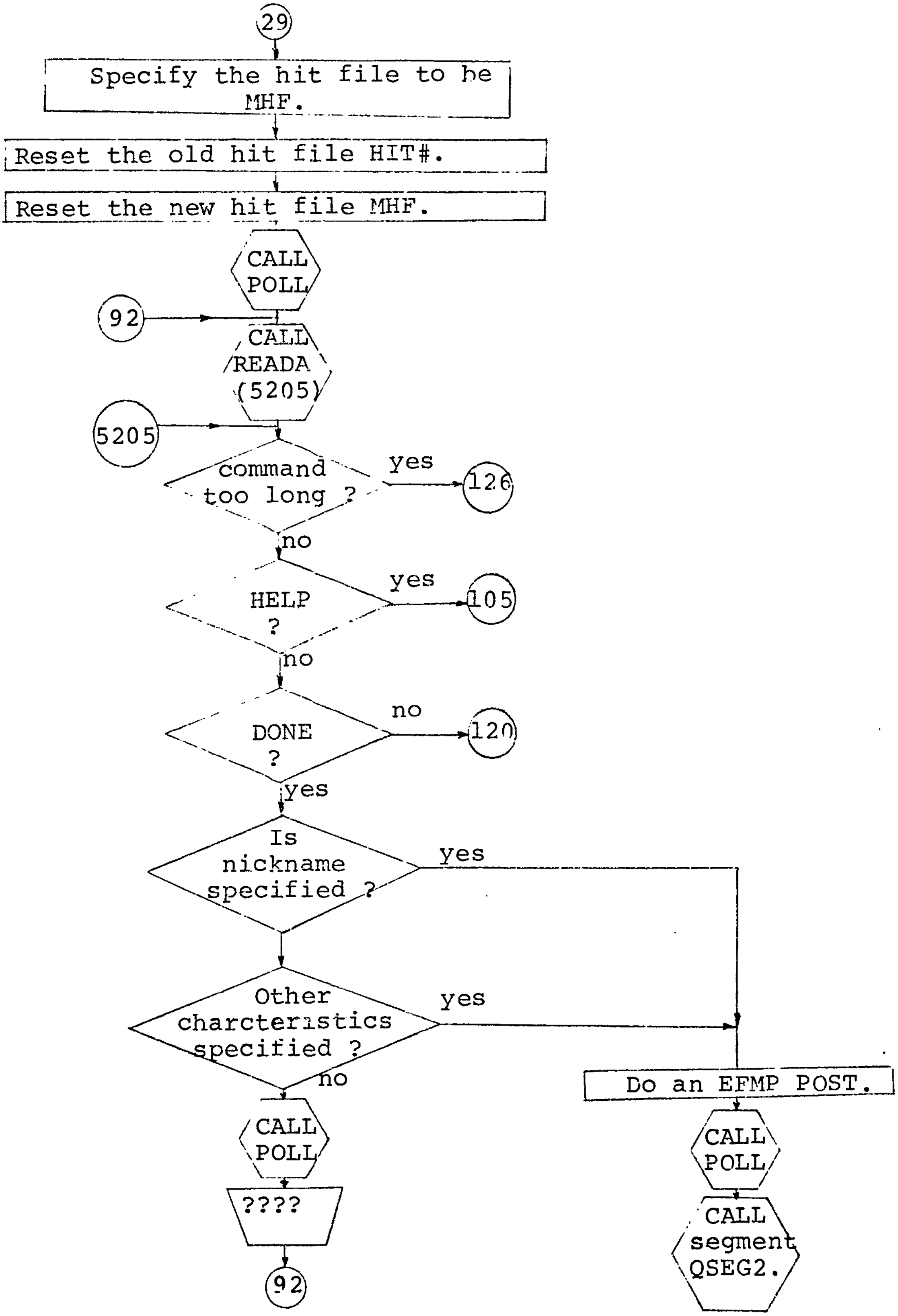

11
PRINT ?
yes
27
SEARCH ?
yes
29
VF ?
yes
36
OPD ?
yes
31
CII ?
yes
33
DISPLAY ?
yes
35
More than 8 characters or no hits ?
yes
B
CALL POLL
????
15
Decode line.
VF ?
yes
122
Illegal command format ?
yes
124
125
no
Control from QSEG1 ?
yes
126
B


31
CALL POLL
#
CALL READA (5203)
Decode line.
Illegal OPD ?
no
yes
67
CALL POLL
ENTRY ERROR.
15
Read the KEYSB record.
found ?
no
yes
CALL POLL
ID NOT IN SUBJECT FILE.
15
Read the data from SUBJF.
Assemble the data for printing.
Print the fiche address.
CALL READA (5219)
5219
15

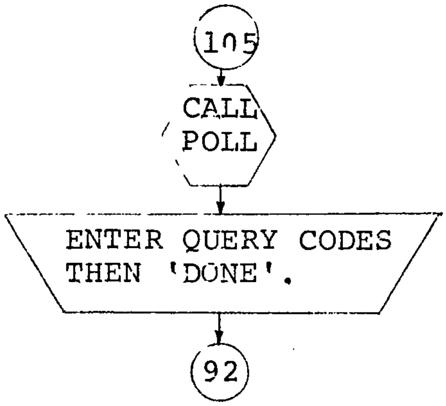


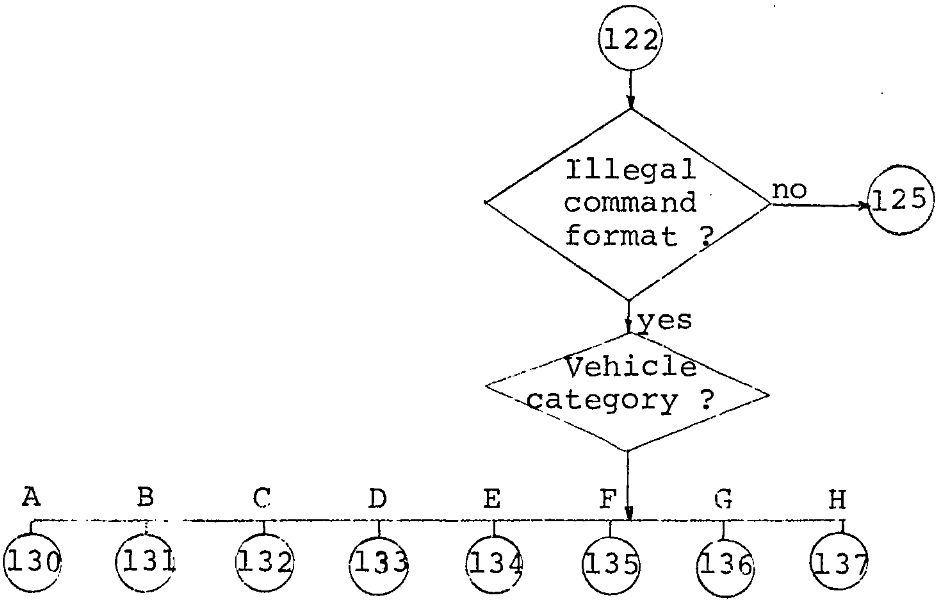
* NOTE : Part of this flow chart represents more than one segment of program coding. Hence the actual statement number will depend on the segment of coding being executed.

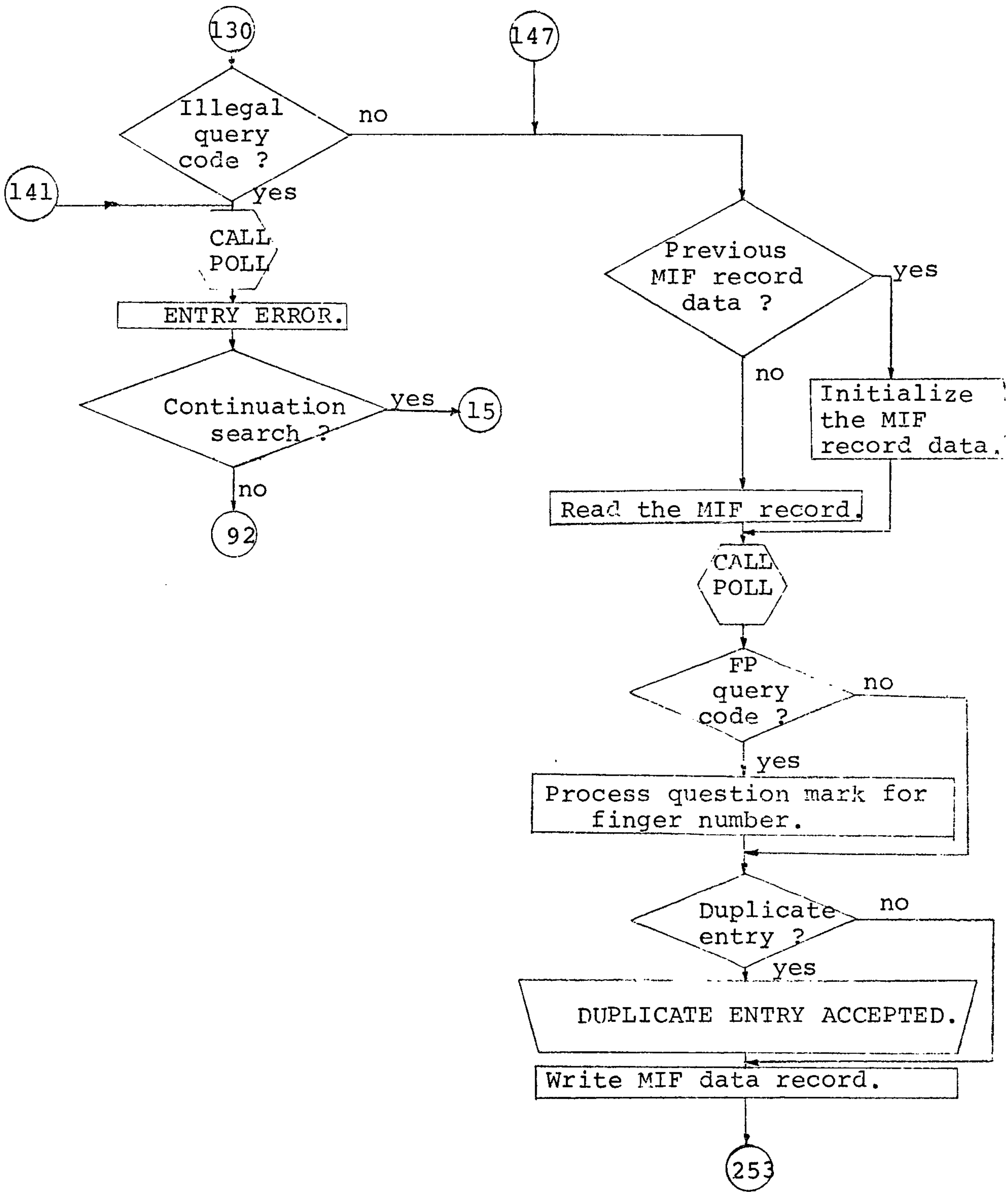

33
CALL POLL
#
CALL READA (5204)
5204
Decode line.
Illegal CII ?
yes
67
no
70
35
FP ?
yes
no
Set continuation diplacement to 0.
CALL POLL
Output start character.
CALL READA (5225)
5225
Wait for relay to react.
Read the hit referance number.
E.O.F. ?
no
yes
Read subject data from SUBJF.
Reset continuation displacement.
Format fiche address.
CALL POLL
Output fiche address.
CALL READA (5226)
5226
Save continuation displacement value.
CALL POLL
Output stop character.
CALL READA (5224)
5224
Wait for relay to react.
BUFFER OUTPUT COMPLETE.
15

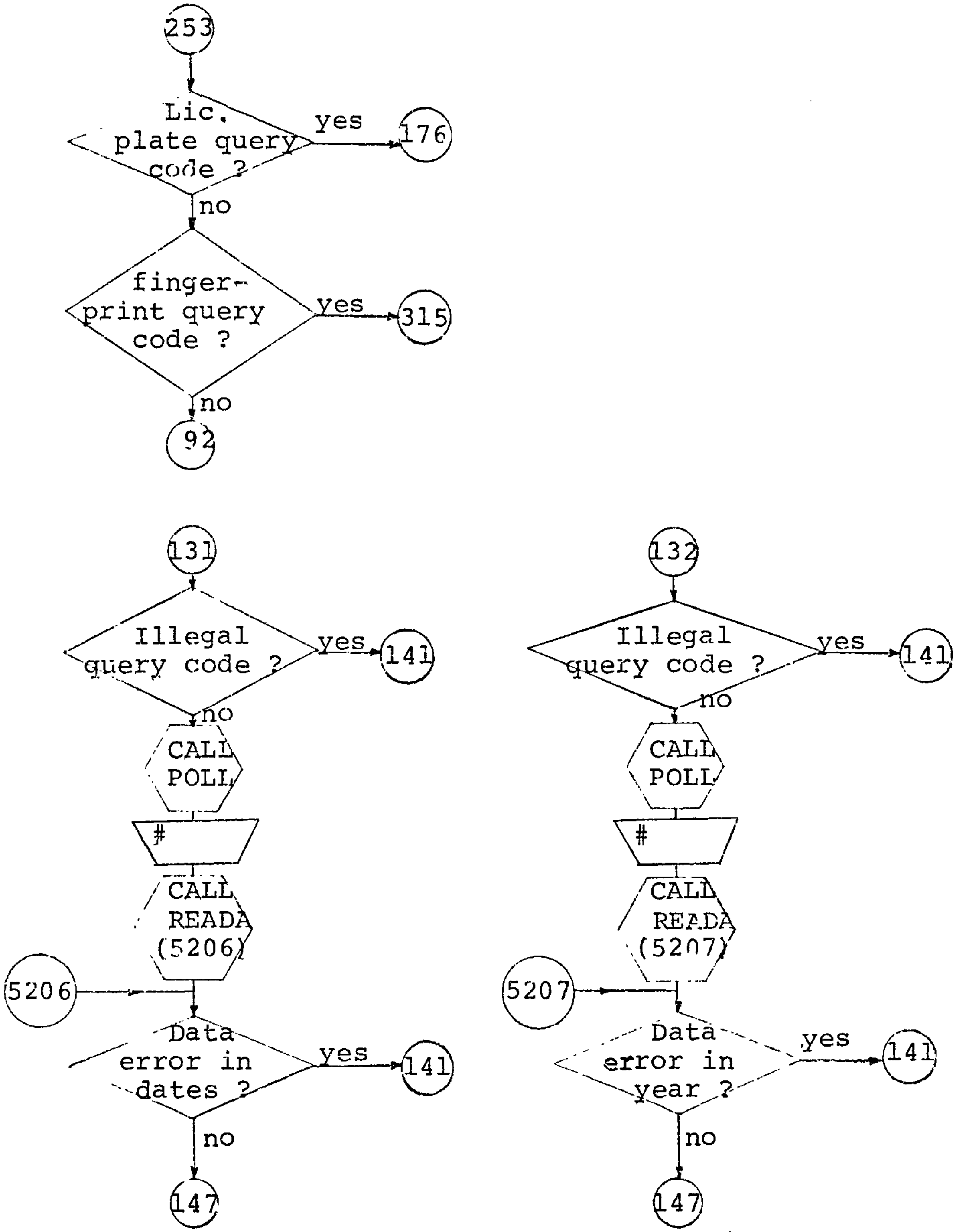


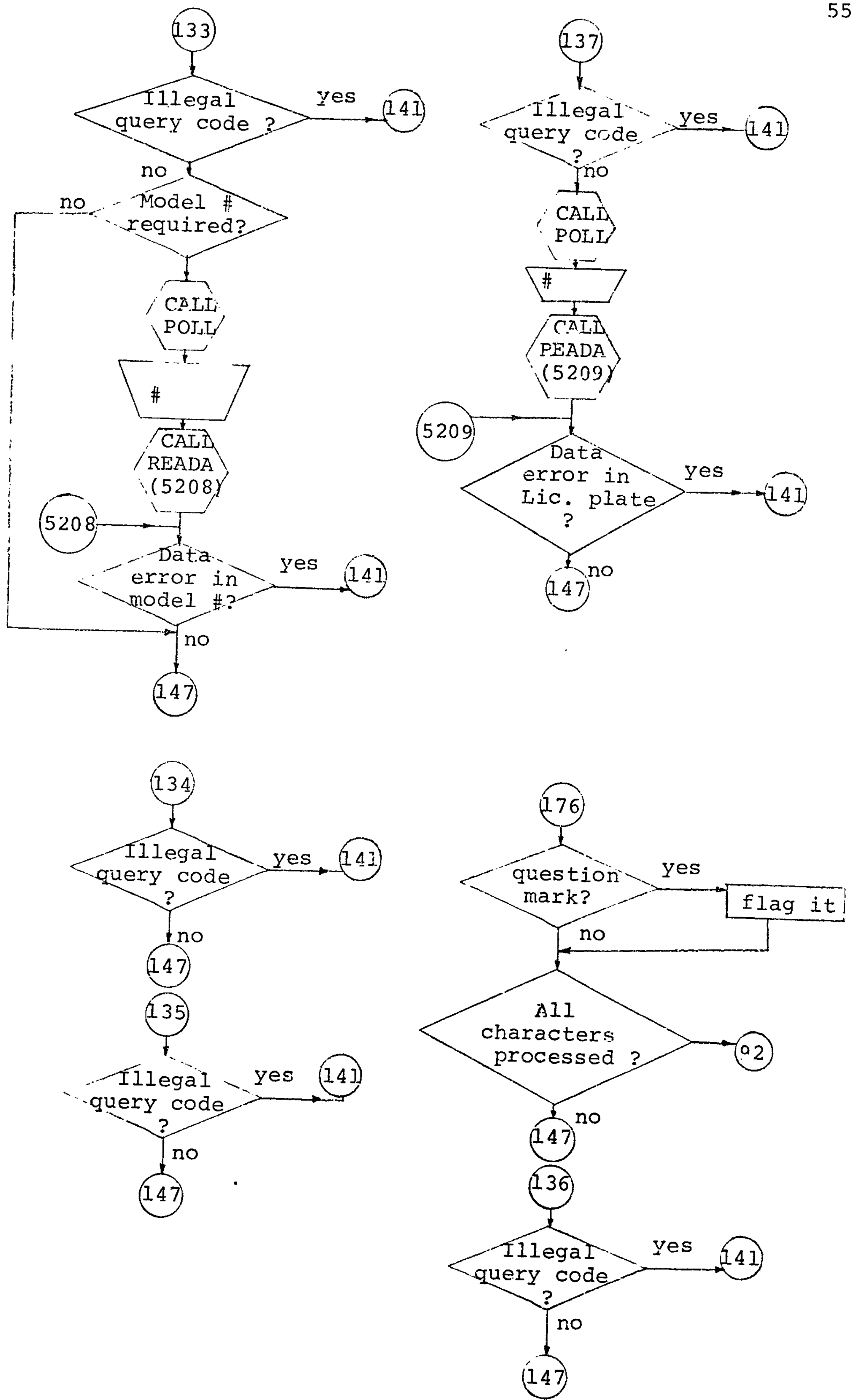
29
Specify the hit file to be MHF.
Reset the old hit file HIT#.
Reset the new hit file MHF.
CALL POLL
92
CALL READA (5205)
5205
command too long ?
yes
126
no
HELP ?
yes
105
no
DONE ?
no
120
yes
Is nickname specified ?
yes
Other charcteristics specified ?
yes
no
Do an EFMP POST.
CALL POLL
????
92
CALL POLL
CALL segment QSEG2.

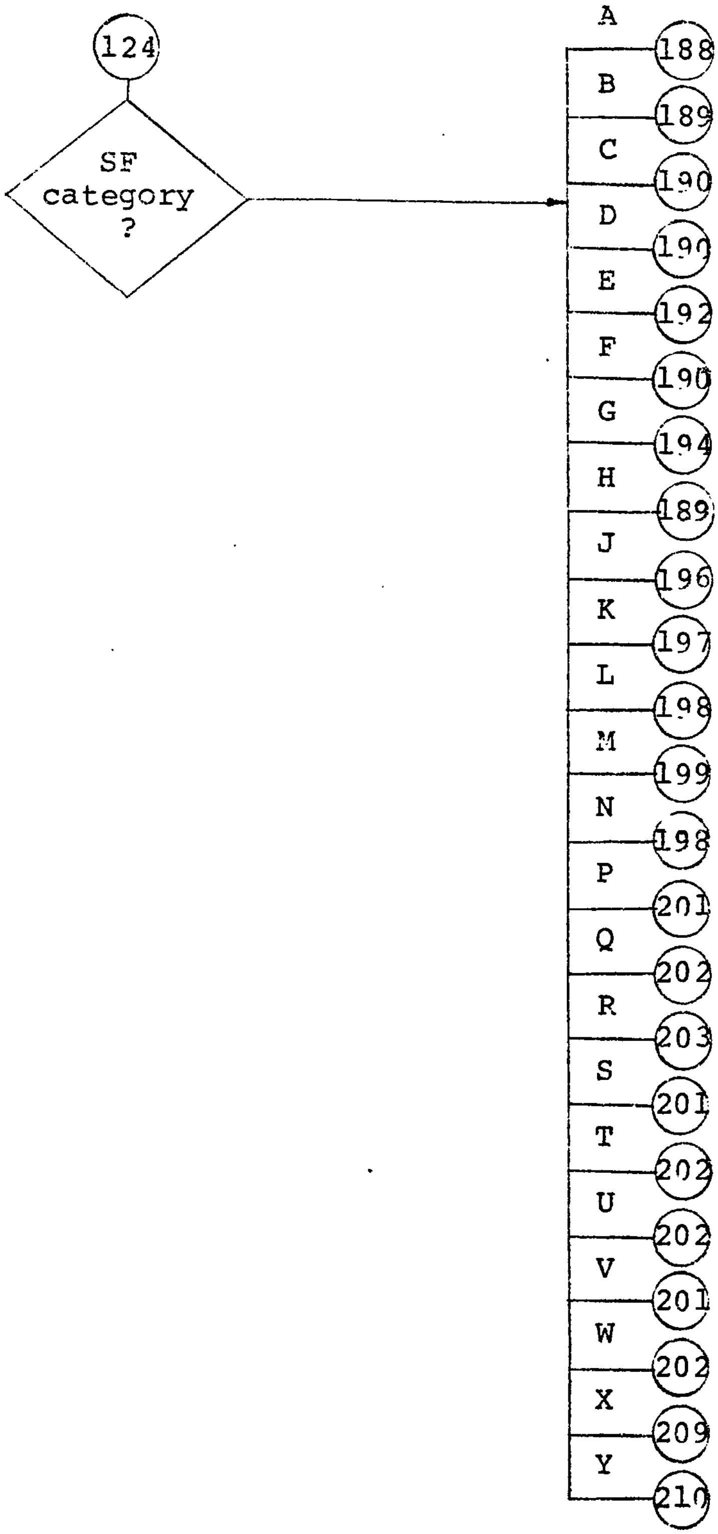

105
CALL POLL
ENTER QUERY CODES THEN 'DONE'.
92

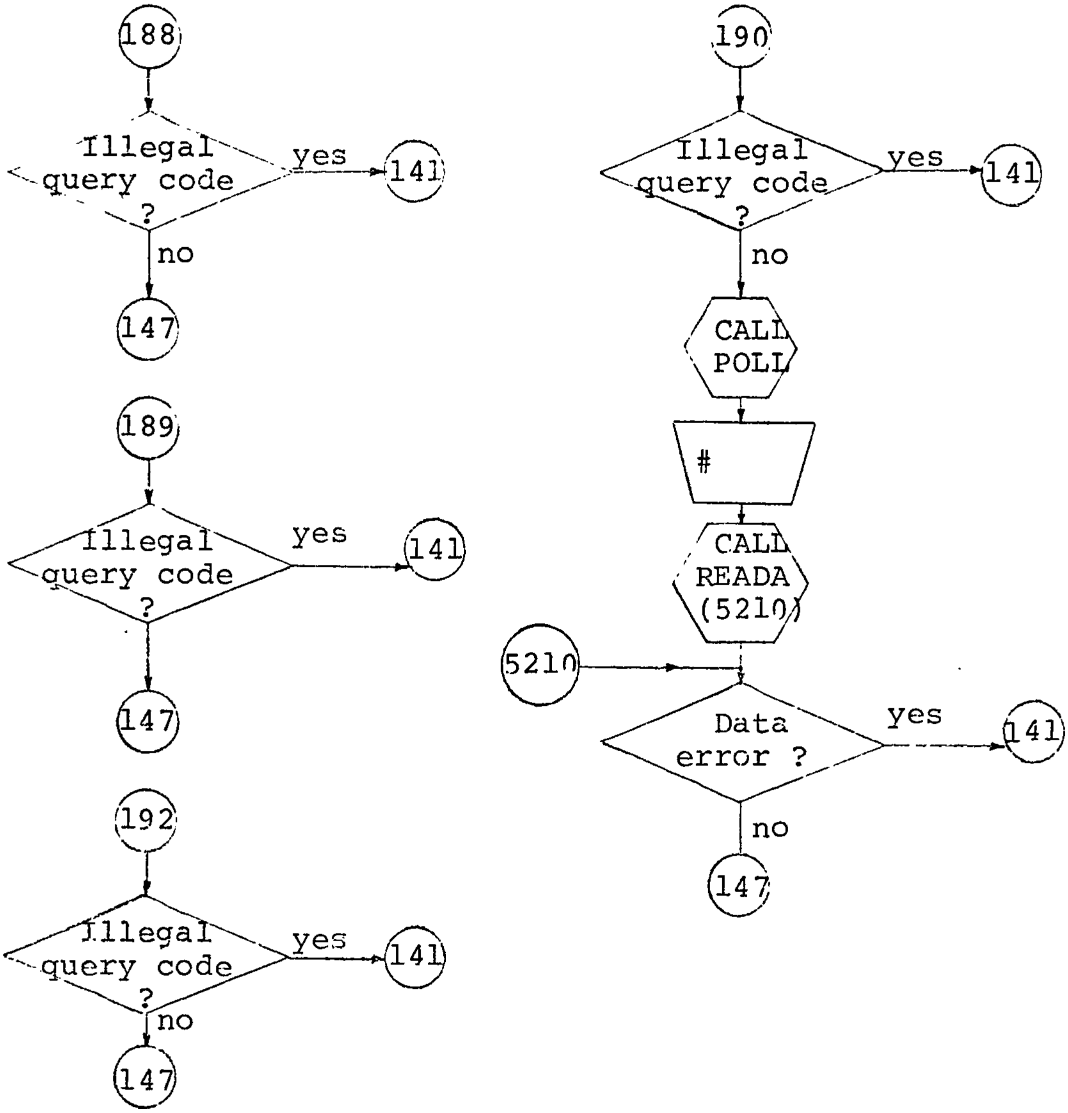

122
Illegal command format ?
no
125
yes
Vehicle category ?
A
B
C
D
E
F
G
H
130
131
132
133
134
135
136
137

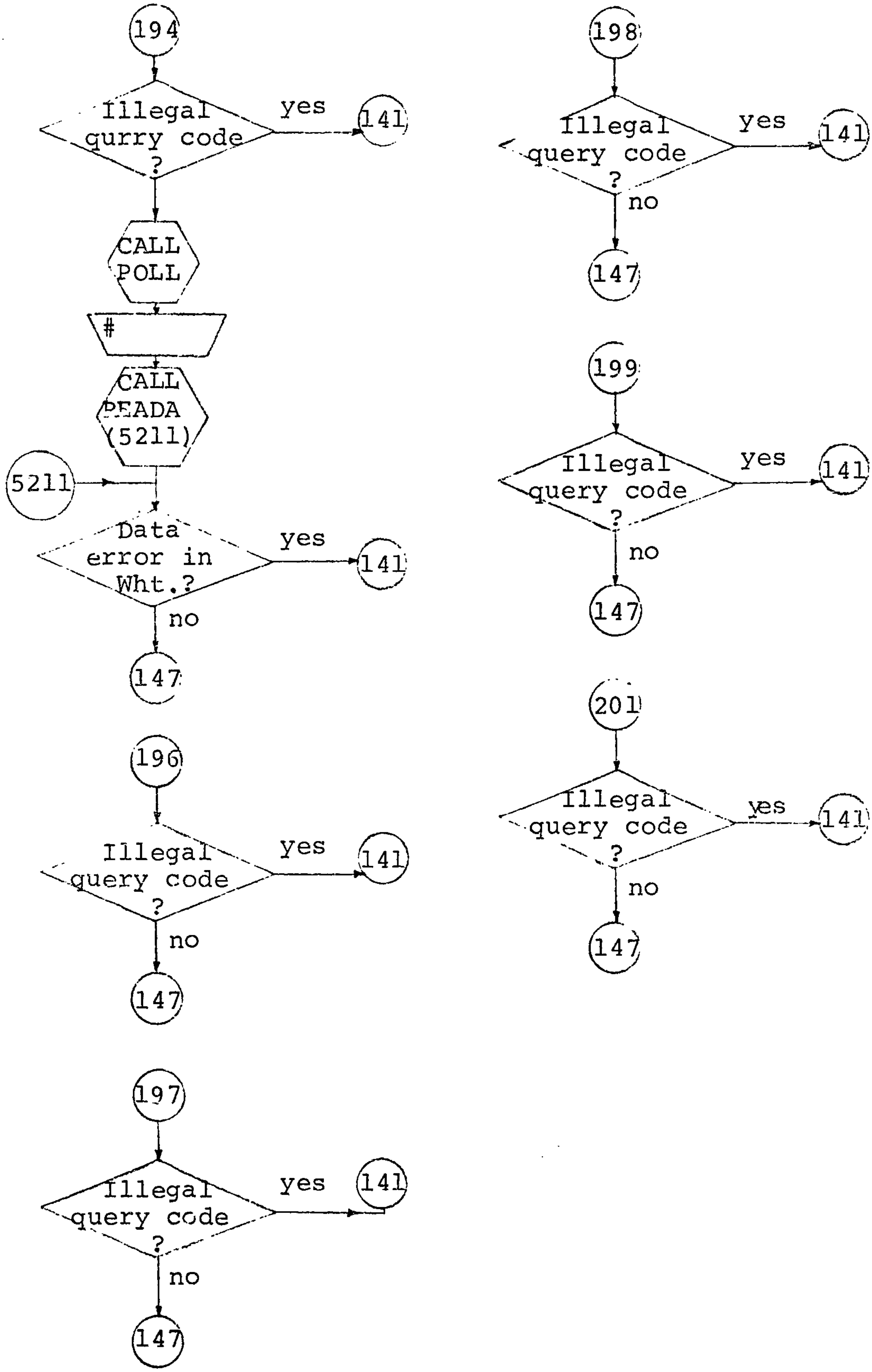

130
147
Illegal query code ?
no
yes
141
CALL POLL
ENTRY ERROR.
Continuation search ?
yes
15
no
92
Previous MIF record data ?
yes
no
Initialize the MIF record data.
Read the MIF record.
CALL POLL
FP query code ?
no
yes
Process question mark for finger number.
Duplicate entry ?
no
yes
DUPLICATE ENTRY ACCEPTED.
Write MIF data record.
253

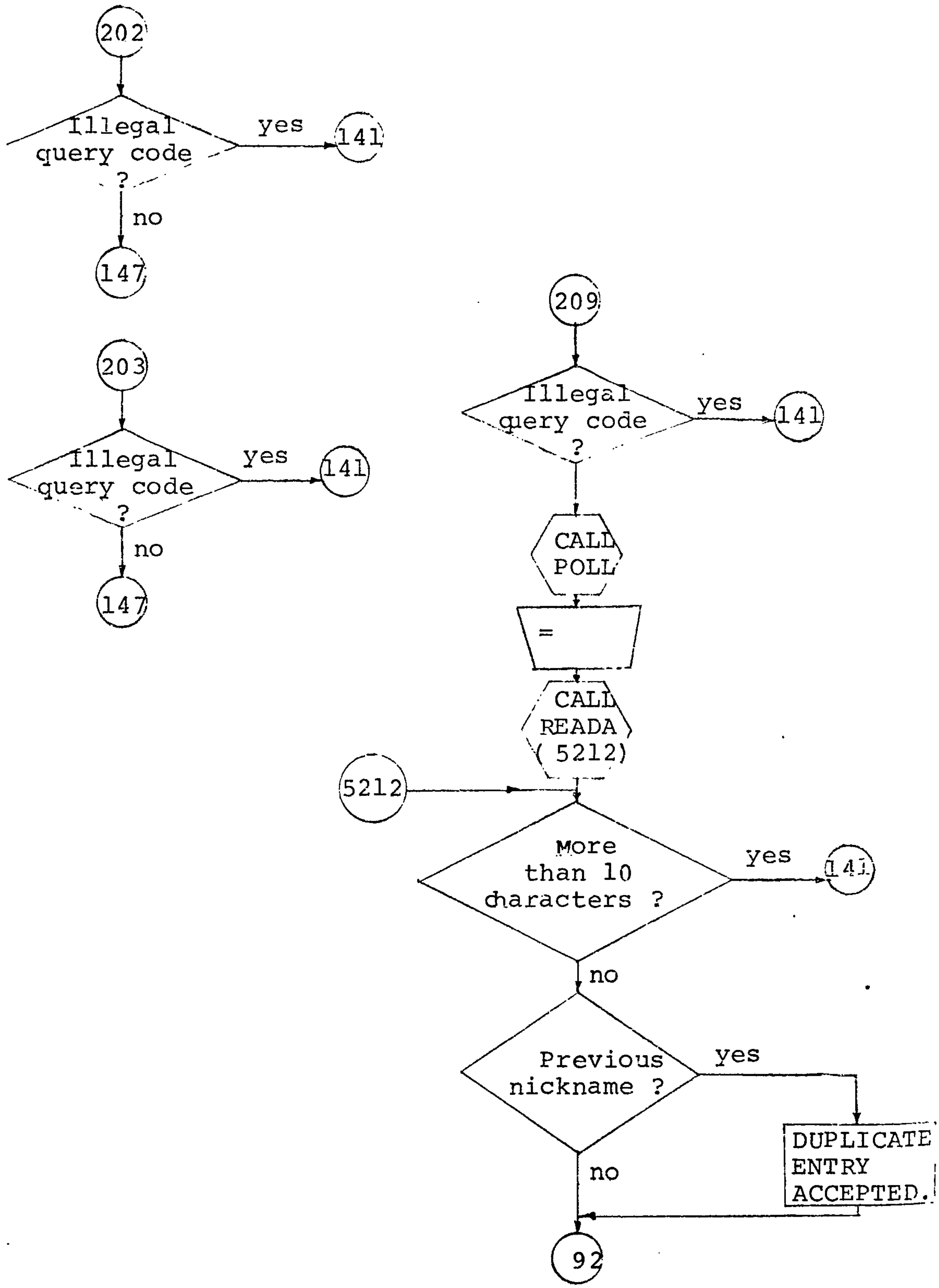

253
Lic. plate query code ?
yes
176
no
finger-print query code ?
yes
315
no
92
131
Illegal query code ?
yes
141
no
CALL POLL
#
CALL READA (5206)
5206
Data error in dates ?
yes
141
no
147
132
Illegal query code ?
yes
141
no
CALL POLL
#
CALL READA (5207)
5207
Data error in year ?
yes
141
no
147

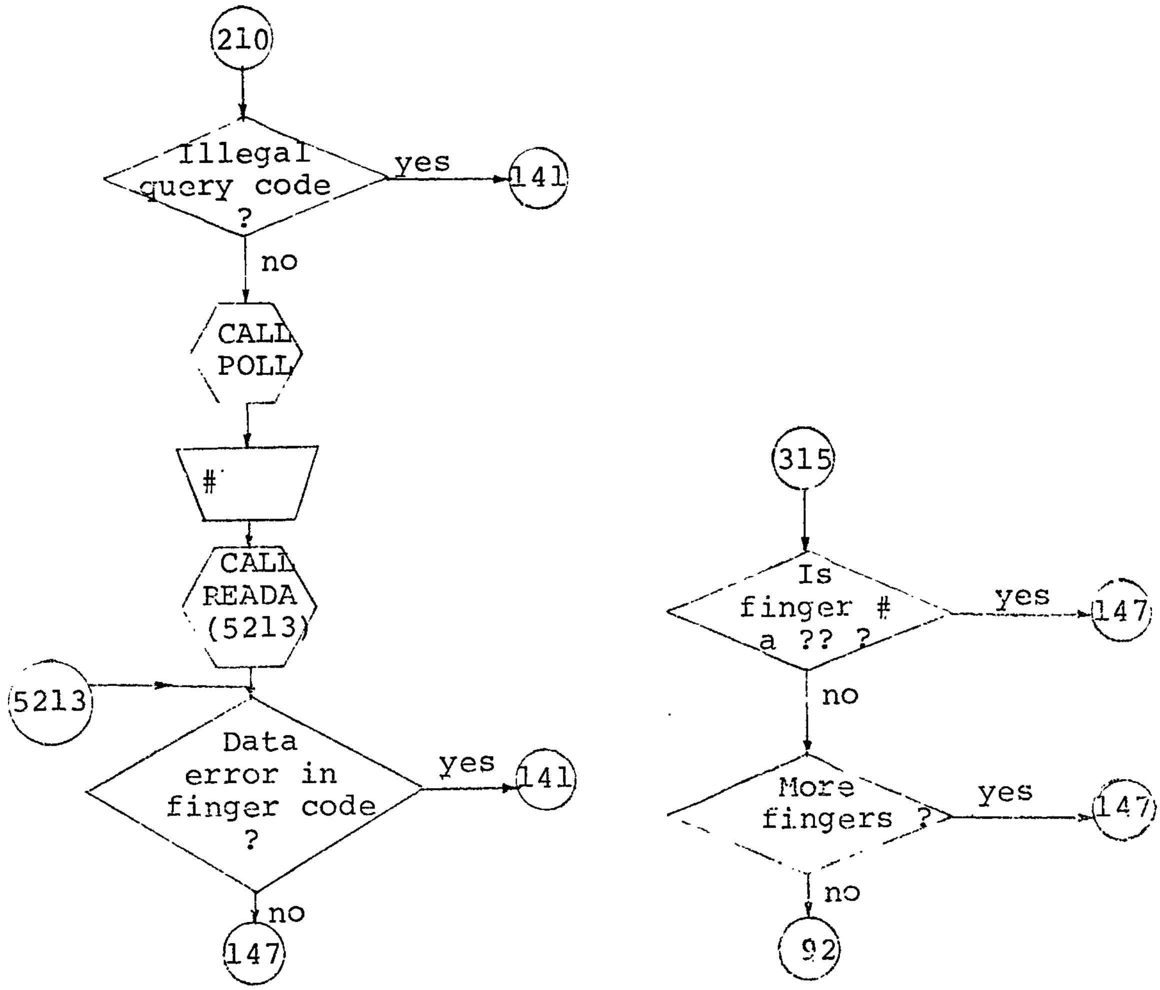

133
Illegal query code ?
yes
141
no
Model # required?
no
CALL POLL
#
CALL READA (5208)
5208
Data error in model #?
yes
141
no
147
137
Illegal query code ?
yes
141
no
CALL POLL
#
CALL PEADA (5209)
5209
Data error in Lic. plate ?
yes
141
no
147
134
Illegal query code ?
yes
141
no
147
135
Illegal query code ?
yes
141
no
147
176
question mark?
yes
flag it
no
All characters processed ?
92
no
147
136
Illegal query code ?
yes
141
no
147


124
SF category ?
A
188
B
189
C
190
D
190
E
192
F
190
G
194
H
189
J
196
K
197
L
198
M
199
N
198
P
201
Q
202
R
203
S
201
T
202
U
202
V
201
W
202
X
209
Y
210


188
Illegal query code ?
yes
141
no
147
189
Illegal query code ?
yes
141
147
192
Illegal query code ?
yes
141
no
147
190
Illegal query code ?
yes
141
no
CALL POLL
#
CALL READA (5210)
5210
Data error ?
yes
141
no
147


194
Illegal qurry code ?
yes
141
CALL POLL
#
CALL READA (5211)
5211
Data error in Wht.?
yes
141
no
147
196
Illegal query code ?
yes
141
no
147
197
Illegal query code ?
yes
141
no
147
198
Illegal query code ?
yes
141
no
147
199
Illegal query code ?
yes
141
no
147
201
Illegal query code ?
yes
141
no
147


202
Illegal query code ?
yes
141
no
147
203
Illegal query code ?
yes
141
no
147
209
Illegal query code ?
yes
141
CALL POLL
=
CALL READA (5212)
5212
More than 10 characters ?
yes
141
no
Previous nickname ?
yes
DUPLICATE ENTRY ACCEPTED.
no
92



NOTE : Flow charts for Subroutines READA and POLL are shown in Figures 2.3 and 2.4 .

CHAPTER 4

CONCLUSION

The design objective of incorporating time sharing capability in the QUERY portion of the CRIME File System was achieved. System - investigator conversation was made concurrent for all the query terminals, however actual database search was allowed to execute uninterrupted.

The system was implemented at the Image Analysis Labratory. A synthetic data base was generated for test purpose, and the system was successfully demonstrated using this data base.

The author would like to make a closing remark here that the system was designed and implemented using FORTRAN IV programming language, and employing HP's DOS-III system which is not particularly designed for concurrent multiterminal operation.

REFERANCES

1. CRIME File System Project Report, Research & Development Division, Oakland Police Department, 1972 (Federal Discretionary Grant # 71-DF-1067).

2. Oakland Police Department CRIME System Internal Maintenance Specification (Form 19601A), Hewlett Packard Co., Cupertino, California, 1973.

3. Source listing of CRIME programs.

4. Hewlett Packard 2100 Series Computers DOS III Disc Operating System, Cupertino, California, 1973.

5. <u>Timesharing System Design Concepts</u> - Richard W. Watson, New York (McGraw-Hill Book Co.),1970.

6. <u>Operating Systems</u> - Stuart E. Madnick and John D. Donovan, New York (McGraw-Hill Book Co.), 1974.

7. Nguyen Ha, A Computer System For The Mugfile Problem, MSEE Thesis, University of Houston, August 1976.

APPENDIX A

(PROGRAM LISTING)

```
C2Ø1M   JULY 23, 1976
FTN,L,T
      PROGRAM QUERY(3)
      DIMENSION INAME(3)
      COMMON ICOMM(128)
      COMMON IKOMM(1Ø17)
      DATA INAME/2HQS,2HEG,1H1/
C  CALL SEGMENT QSEG1
      CALL EXEC (8,INAME)
      END
       ENDS
```

```
[illegible] 2M   JULY 2[illegible], 197[illegible]
FT[illegible]
      PROGRAM [illegible]SE[illegible]1( )
      DIMENSION I[illegible]T[illegible](12[illegible]),ITRBUF(2[illegible]),I[illegible](3),[illegible](3),LU[illegible](6),
     LIMIT[illegible](3),IFIL[illegible](3),I[illegible]ISP[illegible](5),ISTAT[illegible](1[illegible]),IFNA[illegible](3),N[illegible]T[illegible]R(2),
     IK[illegible](12[illegible])
      COMMON ITYP[illegible],I[illegible],I[illegible],I[illegible](123)
      COM[illegible] LI[illegible]T([illegible],[illegible]),L[illegible]T[illegible](12[illegible],[illegible]),IK[illegible](3[illegible])
      EQUIVALENCE (I[illegible](1),ISTAT (1)), (I[illegible](11),IFNAME(1)), (ICOMM(14
     1),I[illegible]L[illegible]US), (I[illegible](2[illegible]),[illegible](1)), (I[illegible](34),LU[illegible](1)), (I[illegible](4[illegible]),I
     [illegible]),(ICOM[illegible](46),IY[illegible]R),(IC[illegible](47),ILU[illegible]),(ICOMM(4[illegible]),I[illegible]IT[illegible](1
     )), (IC[illegible]( 1),IFIL[illegible] (1)), (IC[illegible](5[illegible]),I[illegible]ISP[illegible](1))
     +,(K[illegible](1),ITYP[illegible]),(I[illegible](54),I[illegible]),(IC[illegible](47),ILUN)
      [illegible]AT  I[illegible]/[illegible]2H[illegible]2H /,
     1I[illegible] /[illegible] 1 [illegible]/
      WRITE (1,1)
    1 FORMAT(/"[illegible]IN [illegible]UERY [illegible]ARCH.")
    4 WRITE (1,2)
    2 FORMAT(/"[illegible]T[illegible] THE CURRENT YEAR 19[illegible].",/"[illegible] ")
      READ (1,[illegible]) IY[illegible]
      IF (IYEAR.LT.7[illegible].OR.IYEAR.GT.[illegible])4,3
C   [illegible] (T[illegible] OF [illegible] TO [illegible] USE[illegible] BY THE [illegible] FOR ITS
C                 I[illegible]T[illegible]AL [illegible]FFERS [illegible] TABLES.)
    3 NOTR(1)=1
      NOTR(2)=3
      CALL EXEC(-2[illegible],1,I[illegible]TR,128,ITRBUF,NOTR(1),2,IERRNO)
      IF (IERRNO.LT.1)5,6
    6 WRITE (1,7) IERRNO
    7 FORMAT(/"[illegible] NUMBER ",I2)
      PAUSE
      CALL [illegible]ERY
    5 I[illegible]L[illegible]US=1[illegible]
   16 WRITE (1,17)
   17 FORMAT(/"[illegible] THE USER STATUS CODE: 4 [illegible]IGITS MAXIMUM.",/"[illegible] ")
      READ (1,[illegible]) I[illegible]L[illegible]US
      IF (IA[illegible](I[illegible]L[illegible]US).GT.999[illegible])16,18
   18 ISTAT[illegible](1)=1
      DO 19 I=1,3,1
C   [illegible] STATU[illegible] (T[illegible] THE [illegible] OF THE 1ST. FILE ON A PACK.)
      CALL EX[illegible] (2[illegible],1 ,6,IF[illegible],I,J,ISTAT[illegible],IERRNO)
      IF (IERRNO.[illegible]T.[illegible])[illegible],2
C   [illegible] STATUS 1 ([illegible] ALL INFORMATION [illegible]TAIN[illegible] IN THE DIRECTORY FOR
C                    [illegible] FILE.)
    2 CALL EXEC (2[illegible],1 ,1,IFNAME,I,J,ISTAT ,IERRNO)
      IF (IERR[illegible]O.[illegible]T.[illegible])[illegible],21
   21 IF (ISTAT[illegible]([illegible]).[illegible] .I[illegible]US)1[illegible],[illegible]
   23 WRITE (1,24) I,ISTAT[illegible]([illegible])
   24 FORMAT(/"[illegible] STATU[illegible] CODE [illegible]OES NOT [illegible]ATCH ",/2X,"CURRENT USER STATU
     1S CODE [illegible] ",I1," IS ",I[illegible],".",2/"QUERY TERMINATE[illegible].")
      STOP [illegible]
   19 CONTINUE
   27 WRITE (1,22)
   22 FORMAT(/"ID[illegible]TIFY THE ACTIVE T[illegible]MINALS [illegible]Y LU[illegible] (UP TO [illegible]),",/"[illegible] ")
      DO 2[illegible] I=1,[illegible],1
   25 LUN(I)=7
      READ (1,*) LUN
      DO 2[illegible] I=1,[illegible],1
      IF (LUN(I).LT.7.OR.LUN(I).GT.[illegible])27,2[illegible]
   26 IF(LUN(I).[illegible].7)LUN(I)=[illegible]
      DO 2[illegible] I=1,[illegible],1
```

```
      DO 29 J=I+1,6,1
      IF(LUN(I).EQ.LUN(J))LUN(J)=0
   29 CONTINUE
      ILUN=0
      ITYPE=2H
      DO 30 I=1,6,1
      IF (LUN(I).LT.7)30,31
C   I/O STATUS EXEC CALL
   31 CALL EXEC (13,LUN(I),J,K)
      IF (IAND(J,[illegible]B).LT.[illegible]40000B)30,32
   32 WRITE (1,33) LUN(I)
   33 FORMAT(/"LOGICAL UNIT #",I2," IS DOWN AND SO WILL NOT BE USED.")
      LUN(I)=0
   30 CONTINUE
      DO 34 I=1,6,1
      IF (LUN(I).GT.0)35,34
   34 CONTINUE
      WRITE (1,36)
   36 FORMAT(/"THERE ARE NO TERMINALS UP AND AVAILABLE.",2/"QUERY TERMIN
     1ATED.")
      STOP 3
   35 WRITE (1,28)
   28 FORMAT(/"QUERY SYSTEM IS NOW OPERATIONAL.")
      DO 37 I=1,6,1
      LINET(1,I)=2H
      IDISPN(I)=1
      J=LUN(I)
      IF (J.LT.7)37,38
   38 WRITE (J,39) 003537B
   39 FORMAT(/" ",A2)
C   READ EXEC CALL
      CALL EXEC(1,J+020400B,LINET(1,I),-2)
   37 CONTINUE
      NICKNM(1)=[illegible]
      DO 40 I=2,5,1
   40 NICKNM(I)=2H
      IHITN(1)=2HHI
      IHITN(2)=2HTO
      IHITN(3)=1H
      IFILEN(1)=2HMH
      IFILEN(2)=2HF
      IFILEN(3)=1H
C STORE 2ND COMMON BLOCK INTO 6 ADDRESS AREAS.
      DO 60 I=1,6,1
      IF(LUN(I).LT.7)GO TO 60
      IORDN=I
      ILUN=LUN(I)
      DO 55 J=1,128
      LSTAT(J,I)=KOMON(J)
   55 CONTINUE
   60 CONTINUE
      IERRNO=-1
C   CALL SEGMENT QSG1
      CALL EXEC (8,INAME)
      END
       END$
```

```
C203M    JULY 23,1976
FTN
      PROGRAM QSG1(5)
      DOUBLE PRECISION CII
      INTEGER STMT,OUT1(7),OUT2(18),OUT3(7),OUT4(16),OUT5(3),
     *         OUT6(44),OUT6C(3),OUT7(14),OUT8(34),OUT9(21),OUT10(40),
     *         OUT11(27),OUT13(24),OU13C(3),OUT15(1),OUT16(17),
     *         OUT17(1),OUT18(1 ),OUT19(1),OUT2 (15),OUT21(2 ),
     *         OUT22(1),OUT4C(3),OUT4D(3),OUT6D(3),OU13D(3),
     *         OUT23(13),OUT24(16),OUT25(4),OUT26(2),OUT27(2),OUT28(14),
     *         OUT29(17),OUT30(4),OUT31(3),OUT32(32),OUT33(4)
      DIMENSION ICPNTS(31 ),ITRBUF(768),NOTRP(2),IMIF(3),IMHF(3),
     1IKEYSB(3),IKEYMV(3),ISUBJF(3),IMVEHF(3),IMAKEM(15),IMAKEN(15),
     1LINE(36),I49(4),I50(3),I53(8),IMIFD(96),KOMON(128),NAME(3),
     1ISEM(48),ISTAT(2)
      COMMON ITYPE,ICMND,IERRNO,KFILEF(28),NICKNM(5),LUN(6),I,J,K,L,M,N,
     1IYEAR,ILUN,INITT (3),IFILEN(3),ICRDN,IDISPN(3),ITHV,ICOMM(64)
      COMMON LINET(36,6),LSTAT(128,6),STMT(6),KUEUE(6),NEXT,NSTMT,LAST
     1,NBYTES(6),IOSTAT(6),IOCMND(6)
      EQUIVALENCE (ICOMM(1),ISEM(1),LINE(1)),(ICOMM(49),CII,OPD,I49(1)),
     1(ICOMM(50),I49(2),I50(1),IFINGR),(ICOMM(53),I53(1)),
     1(ICOMM(64),KONW ),(KOMON(1),ITYPE),
     1(OUT4C(1),OUT4(11)),(OUT6C(1),OUT6(39)),(OU13C(1),OUT13(19)),
     1(OUT4D(1),OUT4(14)),(OUT6D(1),OUT6(42)),(OU13D(1),OUT13(22))
      DATA IMIF(1)/2HMI/,IMIF(3)/2H  /,
     1IMHF/2HMH,1HF,1H /,ISUBJF/2HSU,2HBJ,1HF/,
     1IMVEHF/2HMV,2HEH,1HF/,IKEYSB/2HKE,2HYS,1HB/,IMAKEM/3,12,13,15,
     116,19,25,32,36,38,39,43,44,46,51/,IMAKEN/6,2,1,12,1,4,11,3*2,
     11,8,6,2*1/,NOTRP/1,4/,IKEYMV/2HKE,2HYM,1HV/,NAME/2HQS,2HEG,1H2/
      DATA OUT1 /0064128,2H??,2H??,0064128,0064128,0570408,0035378/
      DATA OUT2 / 064128,2HEN,2HTE,2HR ,2H'M,2HS',2H, ,2H'F,2HP',2H, ,
     1            2HOR,2H ',2H/F,2H',,0064128,0064128,0570408,0035378/
      DATA OUT3/0064128,2HRE,2HAD,2HY ,0064128,0570408,0035378/
      DATA OUT4 / 064128,2HNU,2HMB,2HER,2H O,2HF ,2H'H,2HIT,2HS',2H =/
      DATA OUT5 /0064128,0570408,0035378/
      DATA OUT6 /0064128,2HCU,2HRR,2HEN,2HT ,2H'H,2HIT,2H-L,2HIS,2HT',
     1            2H I,2HS ,2HLO,2HNG,2HER,2H T,2HHA,2HN ,2HEN,2HTR,
     2            2HIE,2HS ,2HSA,2HVE,2HD,,0064128,2HNU,2HMB,2HER,2H O,
     3            2HF ,2H'H,2HIT,2HS',2H S,2HAV,2HED,2H =/
      DATA OUT7 / 064128,2HST,2HAR,2HT ,2HNE,2HXT,2H Q,2HUE,2HRY,2H. ,
     1            64128, 64128, 57 4 8, 35378/
      DATA OUT8 /0064128,2HEN,2HTE,2HR ,2H'P,2HRI,2HNT,2H',,2H ',2HSE,
     1            2HAR,2HCH,2H',,2H ',2HDI,2HSP,2HLA,2HY',2H, ,2H'C,
     2            2HPD,2H',,2H ',2HCI,2HI',2H, ,2HOR,2H ',2HEN,2HD',
     3            2H. ,0064128,0570408,0035378/
      DATA OUT9 /0064128,2HEN,2HTE,2HR ,2H'P,2HRI,2HNT,2H',,2H ',2HSE,
     1            2HAR,2HCH,2H',,2H O,2HR ,2H'E,2HND,2H'., 064128,
     2            0570408,0035378/
      DATA OUT10/0064128,2HEN,2HTE,2HR ,2H'P,2HRI,2HNT,2H',,2H ',2HSE,
     1            2HAR,2HCH,2H',,2H ',2HDI,2HSP,2HLA,2HY',2H, ,2H'C,
     2            2HPD,2H',,2H ',2HCI,2HI',2H, ,2H'E,2HND,2H',,2H O,
     3            2HR ,2HOU,2HER,2HY ,2HCO,2HDE,2H. ,0064128,0570408,
     4            0035378/
      DATA OUT11/ 64128,2HEN,2HTE,2HR ,2H'P,2HRI,2HNT,2H',,2H ',2HSE,
     1            2HAR,2HCH,2H',,2H ',2HEN,2HD',2H, ,2HOR,2HOU,2HER,
     2            2HY ,2HCO,2HDE,2H. ,0064128,0570408,0035378/
      DATA OUT13/ 64128,2HNU,2HMB,2HER,2H O,2HF ,2HEN,2HTR,2HIE,2HS ,
     1            2HIN,2H ',2HHI,2HT-,2HLI,2HST,2H' ,2H= /
      DATA OUT15/0215378/
      DATA OUT16/0064128,2HID,2H 4,2H N,2HOT,2H I,2HN ,2HSU,2HBJ,2HEC,
```

```
      1              2HT  ,2HFI,2HLE,2H.  ,006412B,057040B,003537B/
       DATA OUT17/007137B/
       DATA OUT18/ 6412B,2HEN,2HTR,2HY  ,2HER,2HRO,2HR./
       DATA OUT19/007537B/
       DATA OUT20/2HPU,2HFF,2HER,2H C,2HUT,2HPU,2HT  ,2HCO,2HMP,2HLE,
      1              2HTE,2H.  ,006412B,057040B,003537B/
       DATA OUT21/  6412B,2HEN,2HTE,2HR  ,2HJU,2HER,2HY  ,2HCO,2HDE,2H(S,
      1              2H)  ,2HTH,2HEN,2H ',2HDO,2HNE,2H'.,006412B,057040B,
      2              003537B/
       DATA OUT22/ 65537B/
       DATA OUT23(1)/006412B/,OUT23(6)/020040B/
       DATA OUT25(1)/020040B/
       DATA OUT26/006412B,020041B/
       DATA OUT27(2)/037401B/
       DATA OUT28(3)/020040B/,OUT28(5)/020057B/,OUT28(7)/020057B/,
      *      OUT28(9)/020040B/,OUT28(11)/020057B/,OUT28(13)/020057B/
       DATA OUT29(1)/  6412B/
       DATA OUT30(1)/006412B/
       DATA OUT40/006412B,057040B,003537B/
       DATA OUT60/ 6412B, 57040B, 03537B/
       DATA OUT130/006412B,057040B,003537B/
       DATA OUT32/2HDU,2HPL,2HIC,2HAT,2HE  ,2HEN,2HTR,2HY  ,2HAC,
      *              2HCE,2HPT,2HED,2H.  /
C
       N=IERRNO
C EFMP DEFINE
       CALL EXEC(-24,1,IOPNTP,310,ITRBUF,NOTRB(1),2,IERRNO)
       IF(IERRNO.GE.1)GO TO 2
C EFMP OPEN (MAKE FILE ACCESSIBLE)
       CALL EXEC(24,4,IMHF,2,1,0,1,IERRNO)
       IF(IERRNO.GT.0)GO TO 2
C EFMP OPEN (MAKE FILE ACCESSIBLE)
       CALL EXEC(24,4,IKEYSB,1,1,0,1,IERRNO)
       IF(IERRNO.GT. )GO TO 2
C EFMP OPEN (MAKE FILE ACCESSIBLE)
       CALL EXEC(24,4,IKEYMV,3,1,0,1,IERRNO)
       IF(IERRNO.GT.0)GO TO 2
C EFMP OPEN (MAKE FILE ACCESSIBLE)
       CALL EXEC(24,4,IMVEHF,3,1,0,3,IERRNO)
       IF(IERRNO.GT.0)GO TO 2
C EFMP OPEN (MAKE FILE ACCESSIBLE)
       CALL EXEC(24,4,ISUBJF,1,1,0,3,IERRNO)
       IF(IERRNO.GT.0)GO TO 2
       IF(N.GE. )GO TO 1024
C INITIALIZE SECOND COMMON STATEMENT
       DO 1000 I=1,6
       STMT(I)=1
       IOSTAT(I)=1
       IOCAND(I)=0
1000   KUEUE(I)=0
       LAST=1
       NEXT=0
C
       ILUN=0
1001   CALL POLL
       IF(NEXT.EQ.0)GO TO 1001
       I=NEXT
       IORDN=I
       ILUN=LUN(I)
       KUEUE(I)=0
```

```
      LAST=1
      NEXT=0
      GO TO 1003
C ENTRY POINT FOR CHANGE OF TERMINAL.
5201  I=IORDN
 1003 KONWD=0202002+ILUN
      IF(LINET(1,I).EQ.2HHE)1005,1004
 1004 ITYPE=LINET(1,I)
      IF(ITYPE.NE.2HMS.AND.ITYPE.NE.2HFP.AND.ITYPE.NE.2HVF)1007,1008
 1008 IHITN(2)=I+2HT0
      DO 1013 J=1,3,1
1013  IFILEN(J)=IHITN(J)
      IF(ITYPE.EQ.2HVF)1012,1016
C EFMP READ
1016  CALL EXEC(24,6,IKEYSB,1,J,IERRNO)
1020  IF(IERRNO.GT.0)GO TO 2
      ITHV=IFIX(FLOAT(IAND(ISSW(15),000037B))/32.*FLOAT(J))
      IF(ITHV.LT.1)ITHV=J/2
      N=-1
      GO TO 1023
C EFMP READ
1012  CALL EXEC(24,6,IKEYMV,1,J,IERRNO)
      GO TO 1020
1007  ITYPE=2H
C I/O WRITE
      CALL EXEC(2,KONWD,OUT1,7)
      GO TO 1010
C I/O WRITE
1005  CALL EXEC(2,KONWD,OUT2,18)
1010  LINET(1,I)=2H
      NSTMT=1
      CALL READA
      GO TO 5200
 1024 DO 1022 I=1,6,1
      IF(I.EQ.IORDN.OR.STMT(I).EQ.1)GO TO 1022
      IMIF(2)=2HF0+I
      IHITN(2)=2HT0+I
C EFMP OPEN (MAKE FILE ACCESSIBLE)
      CALL EXEC(24,4,IMIF,2,1,0,1,IERRNO)
      IF(IERRNO.GT.0)GO TO 2
C EFMP OPEN (MAKE FILE ACCESSIBLE)
      CALL EXEC(24,4,IHITN,2,1,0,2,IERRNO)
      IF(IERRNO.GT.0)GO TO 2
 1022 CONTINUE
1023  IHITN(2)=2HT0+IORDN
      IMIF(2)=2HF0+IORDN
C EFMP STATUS CALL
      CALL EXEC(24,10,2,IHITN,2,J,ISTAT,IERRNO)
      IF(IERRNO.GT. )GO TO 2
      IF(ISTAT(2).EQ.0)GO TO 16
C EFMP OPEN (MAKE FILE ACCESSIBLE)
      CALL EXEC(24,4,IHITN,2,1, ,2,IERRNO)
C EFMP OPEN (MAKE FILE ACCESSIBLE)
      CALL EXEC(24,4,IMIF,2,1,0,1,IERRNO)
      IF(IERRNO.GT.0)GO TO 2
C EFMP RESET (TO RESET THE HIGHEST RECORD POINTER FOR A FILE
C                  TO A LOWER VALUE)
   16 CALL EXEC(24,9,IMIF,2,0,IERRNO)
      IF(IERRNO.GT. )GO TO 2
      IMIF(2)=2HF
```

```
      GO TO 1
    2 WRITE(1,3)IERRNO
    3 FORMAT(/"EFMP ERROR NUMBER ",I2)
      PAUSE 5
      CALL QUERY
    1 IF(J.LT.0)GO TO 1027
      DO 1028 I=1,3,1
 1028 IFILEN(I)=IHITN(I)
      I=0
C EFMP RESET (TO RESET THE HIGHEST RECORD ACCESSED POINTERFOR A FILE
C                TO A LOWER VALUE)
      CALL EXEC(24,9,IHITN,2,0,IERRNO)
      IF(IERRNO.GT.0)2,1029
 1029 I=I+1
      CALL POLL(NEXT)
C EFMP READ (TO RETRIEVE THE NEXT RECORD FROM A FILE)
      CALL EXEC(24,6,IMHF,I,J,IERRNO)
      IF(IERRNO.EQ.21)1027,1030
 1030 IF(IERRNO.GT.0)2,1031
C EFMP WRITE (TO WRITE INTO THE NEXT RECORD OF A FILE)
 1031 CALL EXEC(24,8,IHITN,I,J,IERRNO)
      IF(IERRNO.EQ.21)1032,1033
 1033 IF(IERRNO.GT.0)2,1029
 1032 I=I-1
      CALL POLL
      CALL CODE
      WRITE(OUT6C,26)I
   26 FORMAT(I6)
      IOSTAT(IORDN)=1
C I/O WRITE
      CALL EXEC(2,KONWD,OUT6,41)
      NSTMT=2
      CALL READA
      GO TO 5200
C I/O WRITE
 5202 CONTINUE
C EFMP RESET (TO RESET THE HIGHEST RECORD ACCESSED POINTER FOR A FILE
C                TO A LOWER VALUE)
 1027 CALL EXEC(24,9,IMHF,2,0,IERRNO)
      IF(IERRNO.GT.0)2,5
    5 DO 6 I=1,28,1
    6 KFILEF(I)=0
      IF(ITYPE.EQ.2HVF)111,48
   48 NICKNM(1)=0
      DO 49 I=2,5,1
   49 NICKNM(I)=2H
  111 IF (N.GT.-1)329,330
C I/O WRITE
  330 CALL EXEC(2,KONWD,OUT3,7)
      GO TO 15
  329 CALL CODE
      WRITE(OUT4C,112)N
  112 FORMAT(I6)
      KONWD=020200B+ILUN
C I/O WRITE
      CALL EXEC(2,KONWD,OUT4,16)
   15 CALL POLL
      DO 139 I=1,5,1
  139 LINET(I,IORDN)=2H
      NSTMT=3
```

```
      CALL READA
      GO TO 5200
C ENTRY POINT FOR CHANGE OF TERMINAL.
 5203 ICMND=LINE(1)
      IF (ICMND.EQ.2HHE)9,10
   10 IF (ICMND.NE.2HEN)11,20
   20 CALL POLL
C   EFMP POST (PHYSICALLY WRITE ON THE DISC.)
      CALL EXEC (24,14,IERRNO)
      IF (IERRNO.GT.0)2,19
   19 CALL POLL
C I/O WRITE
      CALL EXEC(2,KONWD,OUT7,14)
      CALL POLL
      LINET(1,IORDN)=2H
      NSTMT=1
      CALL READA
      GO TO 5200
    9 CALL POLL
      IF (N.GT.0)113,114
  114 IF (ITYPE.EQ.2HVF)8,17
C I/O WRITE
   17 CALL EXEC(2,KONWD,OUT8,34)
      GO TO 15
C I/O WRITE
    8 CALL EXEC(2,KONWD,OUT9,21)
      GO TO 15
  113 IF (ITYPE.EQ.2HVF)115,116
C I/O WRITE
  116 CALL EXEC(2,KONWD,OUT10,40)
      GO TO 15
C I/O WRITE
  115 CALL EXEC(2,KONWD,OUT11,27)
      GO TO 15
   11 IF (ICMND.EQ.2HPR)27,28
   28 IF (ICMND.EQ.2HSE)29,30
   30 IF (ITYPE.EQ.2HVF)36,38
   38 IF (ICMND.EQ.2HLP)31,32
   32 IF (ICMND.EQ.2HCI)33,34
   34 IF (ICMND.EQ.2HDI)35,36
   36 IF (IERRNO.GT.8.OR.N.LT.1)121,120
  120 CALL CODE
      READ (LINE,119) I53
  119 FORMAT(8R1)
      K=-1
      L=-1
      M=-1
      IF (ITYPE.EQ.2HVF)122,123
  123 IF (I53(1).GT.000101B.AND.I53(1).LT.000132B.AND.I53(1).NE.000111B.
     1AND.I53(1).NE.    117B.AND.I53(2).GT.000060B.AND.I53(2).LT.000072B)
     2124,125
  125 IF (N.LT.0)126,121
  121 CALL POLL
C I/O WRITE
      CALL EXEC(2,KCNWD,OUT1,7)
      GO TO 15
   27 IERRNO=N
      N=-1
      IF (IERRNO.GT.0)276,277
C   EFMP STATUS 1
```

```
  277 CALL EXEC (24,10,1,IFILEN,2,I,LINE,IERRNO)
      IF(IERRNO.GT.0)2,254
  254 IF(LINE(1 ).LT.1)1110,256
C I/O WRITE
 1110 CALL EXEC(2,KONWD,OUT4D,3)
      GO TO 15
  256 CALL CODE
      WRITE(OU13C,255)LINE(10)
  255 FORMAT(I6)
      IOSTAT(IORDN)=1
C I/O WRITE
      CALL EXEC(2,KONWD,OUT13,21)
      NSTMT=4
      CALL READA
C ENTRY POINT FOR CHANGE OF TERMINAL.
 5204 CONTINUE
  276 I=0
      M=IAND(ISSW(15),100000B)
   41 I=I+1
      CALL POLL
C   EFMP READ (TO RETRIEVE THE NEXT RECORD FROM A FILE.)
      CALL EXEC (24,6,IFILEN,I,J,IERRNO)
      IF (IERRNO.EQ.21.OR.M.NE.IAND(ISSW(15),100000B).AND.IERRNO.LT.1)15
     1,42
   42 IF (IERRNO.GT.0)2,43
   43 IF (ITYPE.EQ.2HVE)44,45
C   EFMP READ (TO RETRIEVE THE NEXT RECORD FROM A FILE.)
   45 CALL EXEC (24,6,ISUBJF,J,ISFM,IERRNO)
      IF (IERRNO.GT.0)2,46
   46 IF (ISFM(1).GT.-1.AND.ISFM(2).GT.-1)50,51
   51 J=IABS(ISFM(2))+10 0
      K=-ISFM(1)
      L=2HC0
   52 CALL CODE
      WRITE (I50,53) J
   53 FORMAT(1X,I5)
      K=K+1000
      CALL CODE
      WRITE (I49,54) K
   54 FORMAT(I4)
      ICOMM(49)=IOR(IAND(ICOMM(49),0 357B),L)
      DO 55 J=14,3,-1
      IF (ISFM(J).NE.2H  )56,55
   55 CONTINUE
      STOP 1
   56 CALL POLL
      DO 1111 K=1,4,1
      K1=K+1
      OUT23(K1)=I49(K)
 1111 CONTINUE
      DO 1112 K=3,J,1
      K1=K+4
      OUT23(K1)=ISFM(K)
 1112 CONTINUE
      IOSTAT(IORDN)=1
C I/O WRITE
      CALL EXEC(2,KONWD,OUT23,18)
      NSTMT=5
      CALL READA
      GO TO 5200
```

```
C ENTRY POINT FOR CHANGE OF TERMINAL.
 5205 GO TO 8
  338 I1=0
      DO 1115 J=22,28,2
      I1=I1+1
      OUT24(I1)=020040B
      I1=I1+1
      K=J+31
      OUT24(I1)=IAND(ICOMM(K),000377B)
      OUT24(I1)=IOR(OUT24(I1),020000B)
      K=J+32
      I1=I1+1
      OUT24(I1)=ICOMM(K)
      I1=I1+1
      OUT24(I1)=ISFM(J)
 1115 CONTINUE
      IOSTAT(IORDN)=1
C I/O WRITE
      CALL EXEC(2,KONWD,OUT24,16)
      NSTMT=6
      CALL READA
      GO TO 5200
C ENTRY POINT FOR CHANGE OF TERMINAL.
 5206 GO TO 41
  340 OUT25(2)=IAND(ICOMM(49),000377B)
      OUT25(2)=IOR(OUT25(2),020000B)
      OUT25(3)=ICOMM(50)
      OUT25(4)=ISFM(30)
      IOSTAT(IORDN)=1
C I/O WRITE
      CALL EXEC(2,KONWD,OUT25,4)
      NSTMT=7
      CALL READA
      GO TO 5200
C ENTRY POINT FOR CHANGE OF TERMINAL.
 5207 GO TO 41
   56 J=ISFM(2)+10000
      K=ISFM(1)
      L=2HP
      GO TO 52
C   EFMP READ (TO RETRIEVE THE NEXT RECORD FROM A FILE.)
   44 CALL EXEC (24,6,IMVEHF,J,ISFM,IERRNO)
      IF (IERRNO.GT.0)2,58
   58 ICOMM(49)=1HM
      IF (ISFM(10).LT.0) ICOMM(49)=1HF
      ICOMM(50)=IAND(ISFM(10), 074000B)/256
      IF (ICOMM(50).LT.2) ICOMM(50)=1HW
      IF (ICOMM(50).LT.3) ICOMM(50)=1HN
      IF (ICOMM(50).LT.4) ICOMM(50)=1HM
      IF (ICOMM(50).LT.5) ICOMM(50)=1HI
      IF (ICOMM(50).LT.6) ICOMM(50)=1HC
      IF (ICOMM(50).LT.7) ICOMM(50)=1HJ
      IF (ICOMM(50).LT.8) ICOMM(50)=1HO
      ICOMM(51)=IAND(ISFM(11),000037B)
      ICOMM(52)=IAND(ISFM(11),000740B)/32
      ICOMM(53)=IAND(ISFM(11), 077000B)/512
      IF (ISFM(11).LT.0) ICOMM(53)=ICOMM(53)+000100B
      ICOMM(54)=IAND(ISFM(12),003700B)/64
      ICOMM(55)=IAND(ISFM(12), 074000B)/2048
      ICOMM(56)=IAND(ISFM(12),000077B)+72
```

```
      DO 60 J=9,1,-1
      IF (ISFM(J).NE.2H  )61,60
   60 CONTINUE
      J = 1
   61 CALL POLL
      IOSTAT(IORDN)=1
C I/O WRITE
      CALL EXEC(2,KONWD,OUT26,2)
      NSTMT=18
      CALL READA
      GO TO 5200
C ENTRY POINT FOR CHANGE OF TERMINAL.
 5208 K=1
   63 OUT27(1)=ISFM(K)
      IOSTAT(IORDN)=1
C I/O WRITE
      CALL EXEC(2,KONWD,OUT27,-3)
      NSTMT=9
      CALL READA
      GO TO 5200
C ENTRY POINT FOR CHANGE OF TERMINAL.
 5209 K=K+1
      IF(K.GT.J)62,63
   62 DO 1113 J=1,2,1
      J1=J+48
      OUT28(J)=IAND(ICOMM(J1),177400B)
      OUT28(J)=OUT28(J)/256
      OUT28(J)=IOR(OUT28(J),020000B)
 1113 CONTINUE
      J1=50
      DO 1114 J=4,14,2
      J1=J1+1
      CALL CODE
      WRITE(NTEMP,59)ICOMM(J1)
      OUT28(J)=NTEMP
 1114 CONTINUE
   59 FORMAT(I2)
      IOSTAT(IORDN)=1
      NSTMT=10
      CALL EXEC(2,KONWD,OUT28,14)
      CALL READA
      GO TO 5200
C ENTRY POINT FOR CHANGE OF TERMINAL.
 5210 GO TO 41
   31 N=-1
      CALL POLL
C I/O WRITE
      CALL EXEC(2,KONWD,OUT15,1)
      DO 65 I=1,3,1
   65 LINET(I,IORDN)=2H
      NSTMT=11
      CALL READA
      GO TO 5200
C ENTRY POINT FOR CHANGE OF TERMINAL.
 5211 OPD=-1
      CALL OLDIO
      CALL CODE
      READ (LINE,66) OPD
   66 FORMAT(4PF6.0)
      CALL NEWIO
```

```
      IF (OPD.LT.0..OR.OPD.GT.99.9999.OR.IERRNO.GT.6)67,68
   68 L=IFIX(OPD)
      K=L+1001
      M=IFIX((OPD-FLOAT(L))*10000.+.5)
C     EFMP READ (TO RETRIEVE THE NEXT RECORD FROM A FILE.)
   70 CALL EXEC (24,6,IKEYSB,1,I,IERRNO)
      IF (IERRNO.GT.0)2,71
   71 IF (K.EQ.1)72,73
C     EFMP READ (TO RETRIEVE THE NEXT RECORD FROM A FILE.)
   73 CALL EXEC (24,6,IKEYSF,K,J,IERRNO)
      IF (IERRNO.GT.0)2,74
   74 IF (J.NE. )72,75
   75 CALL POLL
C I/O WRITE
      CALL EXEC(2,KONWD,OUT16,17)
      GO TO 15
   72 IF (J.GT.I)75,77
C     EFMP READ (TO RETRIEVE THE NEXT RECORD FROM A FILE.)
   77 CALL EXEC (24,6,ISUBJF,J,ISFM,IERRNO)
      IF (IERRNO.GT.0)2,78
   78 J=J+1
      IF (L.NE.ISFM(1))75,79
   79 IF (M.NE,ISFM(2))72,80
   80 IF (ITYPE.EQ.2HFP)81,82
   82 K=49
      DO 83 J=21,27,2
      IF (ISFM(J).LT.0)84,85
   85 ICOMM(K)=ISFM(J)+1000
      GO TO 83
   84 ICOMM(K)=0
   83 K=K+1
      CALL CODE
      WRITE (I53,86) (ICOMM(J),J=49,52,1)
   86 FORMAT(4I4)
      CALL POLL
      IF (ICMND.EQ.2HPR)338,339
  339 I1=1
      DO 1116 J=22,28,2
      I1=I1+1
      OUT29(I1)=020400B
      I1=I1+1
      K=J+31
      OUT29(I1)=ICOMM(K)
      I1=I1+1
      OUT29(I1)=ISFM(J)
 1116 CONTINUE
      IOSTAT(IORDN)=1
C I/O WRITE
      CALL EXEC(2,KONWD,OUT29,17)
      NSTMT=12
      CALL READA
      GO TO 5200
C ENTRY POINT FOR CHANGE OF TERMINAL.
 5212 GO TO 15
   81 J=ISFM(29)+1000
      CALL CODE
      WRITE (I49,54) J
      CALL POLL
      IF (ICMND.EQ.2HPR)340,98
   98 IF (ICMND.EQ.2HDI)99,341
```

```
  341 OUT30(2)=IAND(ICOMM(40),000377B)
      OUT30(2)=IOR(OUT30(2),020000B)
      OUT30(3)=ICOMM(50)
      OUT30(4)=ISFM(30)
      IOSTAT(IORDN)=1
C I/O WRITE
      CALL EXEC(2,KONWD,OUT30,4)
      NSTMT=13
      CALL READA
      GO TO 5200
C ENTRY POINT FOR CHANGE OF TERMINAL.
 5213 GO TO 15
   33 N=-1
      CALL POLL
C I/O WRITE
      CALL EXEC(2,KONWD,OUT15,1)
      DO 89 I=1,4,1
   89 LINET(I,IORDN)=2H
      NSTMT=14
      CALL READA
      GO TO 5200
C ENTRY POINT FOR CHANGE OF TERMINAL.
 5214 CII=-1.D00
      CALL OLDIO
      CALL CODE
      READ (LINE,90) CII
   90 FORMAT(4PD7.0)
      CALL NEWIO
      IF (CII.LT.0.D00.OR.CII.GT.999.9999D00.OR.IERRNO.GT.7)67,91
   91 L=-IDINT(CII)
      K=L+100
      M=IDINT((CII+DBLE(FLOAT(L)))*10000.D00+.5D00)
      IF (L.EQ.0) M=-M
      J=1
      GO TO 7
   67 CALL POLL
C I/O WRITE
      CALL EXEC(2,KONWD,OUT18,10)
      GO TO 15
   35 L=IDISPN(IORDN)
      IF (ITYPE.NE.2HFP) IDISPN(IORDN)=0
      N=-1
      CALL POLL
      IOSTAT(IORDN)=1
C I/O WRITE
      CALL EXEC(2,KONWD,OUT17,1)
      NSTMT=15
      CALL READA
      GO TO 5200
C ENTRY POINT FOR CHANGE OF TERMINAL.
 5215 DO 243 I=1,37,1
  243 CALL POLL
      DO 39 K=L,L+99,1
C   EFMP READ (TO RETRIEVE THE NEXT RECORD FROM A FILE.)
      CALL EXEC (24,6,IFILEN,K,J,IERRNO)
      IF (IERRNO.EQ.21)93,94
   94 IF (IERRNO.GT.0)2,95
C   EFMP READ (TO RETRIEVE THE NEXT RECORD FROM A FILE.)
   95 CALL EXEC (24,6,ISUBJF,J,ISFM,IERRNO)
      IF (IERRNO.GT.0)2,227
```

```
  227 IF (ITYPE.EQ.2HFP)81,97
   97 M=28
      J=0
      DO 245 I=21,27,2
      IF (ISFM(I).LT.0)246,96
   96 ICOMM(I+M)=ISFM(I)+1000
      M=M-1
      IF (J.GT.23) ISFM(J)=ISFM(I+1)
      J=J+2
      GO TO 245
  246 M=M-2
      IF (J.LT.24) J=I+1
  245 CONTINUE
      CALL CODE
      WRITE (I53,86) (ICOMM(J),J=49,M+28,1)
      J=M-20
      IDISPN(IORDN)=IDISPN(IORDN)+J
      IF (IDISPN(IORDN).GT.100) J=J+100-IDISPN(IORDN)
      CALL POLL
      IF(J.LE.0)GO TO 100
      M=22
  999 I=M+31
      ITEST=2*J+20
      OUT33(1)=IAND(ICOMM(I),000377B)
      OUT33(1)=OUT33(1)*256
      I=I+1
      NTEMP=IAND(ICOMM(I),177400B)
      NTEMP=NTEMP/256
      OUT33(1)=IOR(OUT33(1),NTEMP)
      NTEMP=IOR(ICOMM(I),000377B)
      OUT33(2)=NTEMP*256
      NTEMP=IAND(ISFM(M),177400B)
      NTEMP=NTEMP/256
      OUT33(2)=IOR(OUT33(2),NTEMP)
      OUT33(3)=IAND(ISFM(M),000377B)
      OUT33(3)=OUT33(3)*256
      OUT33(3)=IOR(OUT33(3),000040B)
      OUT33(4)=020137B
      M=M+2
      IOSTAT(IORDN)=1
C I/O WRITE
      CALL EXEC(2,KCNWD,OUT33,4)
      NSTMT=16
      CALL READA
      GO TO 5200
 5216 IF(M.GT.ITEST)100,999
  100 IF(IDISPN(IORDN).GT.100)101,99
   99 OUT31(1)=IAND(ICOMM(49),000377B)
      OUT31(1)=OUT31(1)*256
      NTEMP=IAND(ICOMM(50),177400B)
      NTEMP=NTEMP/256
      OUT31(1)=IOR(OUT31(1),NTEMP)
      NTEMP=IOR(ICOMM(50),000377B)
      OUT31(2)=NTEMP*256
      NTEMP=IAND(ISFM(30),177400B)
      NTEMP=NTEMP/256
      OUT31(2)=IOR(OUT31(2),NTEMP)
      OUT31(3)=IAND(ISFM(30),000377B)
      OUT31(3)=OUT31(3)*256
      OUT31(3)=IOR(OUT31(3),000137B)
```

```
      IOSTAT(IORDN)=1
C I/O WRITE
      CALL EXEC(2,KONWD,OUT31,3)
      NSTMT=17
      CALL READA
      GO TO 52
C ENTRY POINT FOR CHANGE OF TERMINAL.
 5217 CONTINUE
   39 CALL POLL
      IDISPN(IORDN)=L+100
      GO TO 103
   93 IDISPN(IORDN)=1
      GO TO 103
  101 IDISPN(IORDN)=K
  103 CALL POLL
      IOSTAT(IORDN)=1
C I/O WRITE
      CALL EXEC(2,KONWD,OUT19,1)
      NSTMT=19
      CALL READA
      GO TO 5200
C ENTRY POINT FOR CHANGE OF TERMINAL.
 5218 DO 104 I=1,37,1
  104 CALL POLL
C I/O WRITE
      CALL EXEC(2,KONWD,OUT20,15)
      GO TO 15
   29 IDISPN(IORDN)=1
      DO 184 I=1,3,1
  184 IFILEN(I)=IMHF(I)
C  EFMP RESET (TO RESET THE HIGHEST RECORD ACCESSED POINTER FOR A FILE
C                 TO A LOWER VALUE.)
      CALL EXEC (24,9,IFITN,2,0,IERRNO)
      IF (IERRNO.GT.0)2,139
C  EFMP RESET (TO RESET THE HIGHEST RECORD ACCESSED POINTER FOR A FILE
C                 TO A LOWER VALUE.)
  139 CALL EXEC (24,9,IMHF,2,0,IERRNO)
      IF(IERRNO.GT.0)2,142
  142 CALL POLL
C I/O WRITE
      CALL EXEC(2,KONWD,OUT5,3)
   92 N=-1
      DO 138 I=1,5,1
  138 LINET(I,IORDN)=2H
      NSTMT=19
      CALL READA
      GO TO 5200
C ENTRY POINT FOR CHANGE OF TERMINAL.
 5219 ICMND=LINE(1)
      IF (IERRNO.GT.6)126,1 7
  107 IF (ICMND.EQ.2HFE)105,106
  106 IF (ICMND.NE.2HLO)120,108
  108 IF (NICKNM(1).NE.0)280,304
  304 DO 279 I=1,20,1
      IF (KFILEF(I).GT.0)280,279
  279 CONTINUE
  126 CALL POLL
C I/O WRITE
      CALL EXEC(2,KONWD,OUT1,7)
      CALL POLL
```

```
      GO TO 92
C   EFMP POST
  280 CALL EXEC (24,14,IERRNO)
      IF (IERRNO.GT.0)2,244
  244 CALL POLL
C CALL SEGMENT OSEG2
      CALL EXEC(8,NAME)
      STOP 12
  105 CALL POLL
C I/O WRITE
      CALL EXEC(2,KONWD,OUT21,20)
      CALL POLL
      GO TO 92
  122 IF (I53(1).LT.000101B.OR.I53(1).GT.000110B.OR.I53(2).LT.000061B.OR
     1.I53(2).GT.000071B)125,127
  127 GO TO (130,131,132,133,134,135,136,137), I53(1)-000100B
  130 I=22
  160 IF (IERRNO.NE.2.OR.I53(2).GT.000062B)141,140
  140 K=I53(2)-000061B
      IF (I.LT.22)163,334
  334 K=K+1
      IF (K.GT.1) K=0
  163 J=1
  147 IF (KFILEF(I).LT.1)143,144
  144 IMIF(2)=2HF0+IORDN
C   EFMP READ (TO RETRIEVE THE NEXT RECORD FROM A FILE.)
      CALL EXEC(24,6,IMIF,I,IMIFD,IERRNO)
      IF (IERRNO.GT.0)2,145
  143 KFILEF(I)=1
      DO 128 IERRNO=1,96,1
  128 IMIFD(IERRNO)=-1
  145 CALL POLL
      IF (ICMND.NE.2HY1)317,324
  324 IERRNO=I+   47B
      IF (ICOMM(IFINGR).GT.000071B.AND.(IMIFD(1).GT.-1.AND.IMIFD(4).LT.0
     1.OR.IERRNO.EQ.ICOMM(IFINGR+6).OR.IERRNO.EQ.ICOMM(IFINGR+12)))315,3
     226
  326 IF (ICOMM(IFINGR).GT.000071B)327,328
  328 IMIFD(4)=-1
      GO TO 317
  327 IMIFD(4)=
  317 IF(IMIFD(J).EQ.-1.OR.I.GE.24)GO TO 5220
      IOSTAT(IORDN)=1
C I/O WRITE
      CALL EXEC(2,KONWD,OUT32,13)
      NSTMT=20
      CALL READA
      GO TO 520
C I/O WRITE
 5220 IF(I.NE.2.OR.J.GT.1)273,332
  332 DO 333 K=2,9,1
  333 IMIFD(K)=-1
  274 IMIFD(J)=LINE(J)
      J=J+1
      IF (LINE(J).GT.-1)274,275
  273 IMIFD(J)=K
      IF (L.NE.-1) IMIFD(J+1)=L
      IF (M.NE.-1) IMIFD(J+2)=M
  275 IMIF(2)=2HF0+IORDN
C   EFMP WRITE (TO WRITE INTO THE NEXT RECORD OF A FILE.)
```

```
      CALL EXEC(24,8,IMIF,I,IMIFD,IERRNO)
      IF (IERRNO.GT.0)2,253
  253 IF (I.GT.22)176,173
  173 IF (ICMND.EQ.2HY1)315,142
  131 IF (IERRNO.NE.2.OR.I53(2).NE.000061B)141,146
  146 CALL POLL
C I/O WRITE
      CALL EXEC(2,KONWD,OUT15,1)
      DO 148 I=1,7,1
  148 LINET(I,IORDN)=2H
      NSTMT=21
      CALL READA
      GO TO 5200
C ENTRY POINT FOR CHANGE OF TERMINAL.
 5221 IF(IERRNO.NE.6.AND.IERRNO.NE.13)141,149
  149 DO 150 I=49,55,1
  150 ICOMM(I)=-1
      CALL CODE
      READ (LINE,151) (ICOMM(I),I=49,55,1)
  151 FORMAT(3I2,A1,3I2)
      I=49
      J=50
      K=51
  155 IF (ICOMM(I).LT.1.OR.ICOMM(I).GT.31.OR.ICOMM(J).LT.1.OR.ICOMM(J).G
     1T.12,OR.ICOMM(K).LT.72.OR.ICOMM(K).GT.IYEAR)141,152
  152 IF (IERRNO.LT.13.OR.I.GT.49)153,154
  154 IF (ICOMM(52).NE.1H,)141,156
  156 I=53
      J=54
      K=55
      IF (ICOMM(51).GT.ICOMM(55).OR.ICOMM(51).EQ.ICOMM(55).AND.(ICOMM(50
     1).GT.ICOMM(54).OR.ICOMM(50).EQ.ICOMM(54).AND.ICOMM(49).GE.ICOMM(53
     2)))141,155
  153 I=22
      K=IOR(IOR(2048*ICOMM(50),64*ICOMM(49)),ICOMM(51)-72)
      L=K
      IF (IERRNO.GT.6)99997,287
99997 L=IOR(IOR(2048*ICOMM(54),64*ICOMM(53)),ICOMM(55)-72)
  287 J=2
      GO TO 147
  132 IF (IERRNO.NE.2.OR.I53(2).NE.000061B)141,157
  157 CALL POLL
C I/O WRITE
      CALL EXEC(2,KONWD,OUT15,1)
      LINET(1,IORDN)=2H
      NSTMT=22
      CALL READA
      GO TO 5200
C ENTRY POINT FOR CHANGE OF TERMINAL.
 5222 IF(IERRNO.NE.2)141,158
  158 I=21
      CALL CODE
      READ (LINE,159) K
  159 FORMAT(I2)
      IF (K.GT.-1.AND.K.LT.IYEAR+2)178,141
  141 CALL POLL
C I/O WRITE
      CALL EXEC(2,KONWD,OUT18,10)
      CALL POLL
      IF(N.LT.0)92,15
```

```
  133 IF (IERRNO.GT.3)141,177
  177 CALL OLDIO
      CALL CODE
      READ (LINE,186) K
  186 FORMAT(1X,I2)
      CALL NEWIO
      IF (K.LT.1.OR.K.GT.67)141,179
  179 DO 185 J=1,15,1
      IF (K.EQ.IMAKEN(J))181,180
  18? CONTINUE
  183 I=21
      IF (L.LT.1) L=-1
      GO TO 163
  181 CALL POLL
C I/O WRITE
      CALL EXEC(2,KONWD,OUT15,1)
      LINET(1,IORDN)=2H
      NSTMT=23
      CALL READA
      GO TO 5200
C ENTRY POINT FOR CHANGE OF TERMINAL.
 5223 IF(IERRNO.GT.2)141,182
  182 CALL OLDIO
      CALL CODE
      READ (LINE,159) L
      CALL NEWIO
      IF (L.LT.?.OR.L.GT.IMAKEN(J))141,183
  134 IF (IERRNO.GT.3)141,162
  162 CALL OLDIO
      CALL CODE
      READ (LINE,186) K
      CALL NEWIO
      I=20
      IF (K.LT.1.OR.K.GT.1?)141,178
  135 IF (IERRNO.GT.7.OR.IERRNO.EQ.4)141,164
  164 IF (IERRNO.GT.3)165,166
  166 CALL OLDIO
      CALL CODE
      READ (LINE,186) K
      CALL NEWIO
      L=0
  167 I=2?
      IF (K.LT.1.OR.K.GT.10)141,163
  165 CALL CODE
      READ (LINE,*) K,L
      IF (L.LT.1.OR.L.GT.10)141,167
  136 I=19
      GO TO 160
  137 IF (IERRNO.NE.2.OR.IE3(2).NE.000061B)141,168
  168 CALL POLL
C I/O WRITE
      CALL EXEC(2,KONWD,OUT15,1)
      DO 169 I=1,3,1
  169 LINET(I,IORDN)=2H
      NSTMT=24
      CALL READA
      GO TO 5200
C ENTRY POINT FOR CHANGE OF TERMINAL.
 5224 IF(IERRNO.GT.6)141,170
  170 CALL CODE
```

```
      READ (LINE,171) (ICOMM(I),I=49,54,1)
  171 FORMAT(6R1)
      DO 174 I=49,54,1
      IF (ICOMM(I).NE.000077B)175,174
  174 CONTINUE
      GO TO 141
  175 DO 331 I=49,IERRNO+48,1
      IF (ICOMM(I).LT.000060B.OR.ICOMM(I).GT.000071B.AND.ICOMM(I).LT.000
     1101B.AND.ICOMM(I).NE.000077B.OR.ICOMM(I).GT.000132B)141,331
  331 CONTINUE
      I=23
  172 K=ICOMM(I+26)
      GO TO 163
  176 IF (K.EQ.000077B) KFILEF(I)=0
      I=I+1
      IF (I.GT.28)142,172
  124 IF (I53(1).LT.000111B) GO TO (188,189,190,190,192,190,194,189), I5
     13(1)-000120B
99999 IF (I53(1).GT.000117B) GO TO (2 1,2 2,2 3,2 1,2 2,2 2,2 1,2 2,209,
     1210), I53(1)-000117B
99998 GO TO (196,197,198,199,198), I53(1)-000111B
  188 I=1
      GO TO 160
  189 IF (IERRNO.NE.2.AND.IERRNO.NE.5)141,211
  211 K=I53(2)-000060B
      IF (IERRNO.GT.2)212,187
  187 L=-2
  215 IF (K.LT.1.OR.K.GT.5)141,195
  195 IF (I53(1).LT.000111 B)214,242
  242 I=3
  178 J=3
      GO TO 147
  214 J=2
  222 I=1
      GO TO 147
  212 L=I53(5)-000060B
      IF (L.LT.1.OR.L.GT.5.OR.K.EQ.L.OR.I53(3).NE.000054B.OR.I53(1).NE.I
     153(4))141,215
  190 IF (IERRNO.NE.2.OR.I53(2).NE.000061B)141,216
  216 CALL POLL
C I/O WRITE
      CALL EXEC(2,KONWD,OUT15,1)
      DO 217 I=1,5,1
  217 LINET(I,IGRUN)=2H
      NSTMT=25
      CALL READA
      GO TO 52
C ENTRY POINT FOR CHANGE OF TERMINAL,.
 5225 IF(I53(1).EQ.000124B)191,226
  226 IF (IERRNO.NE.2.AND.IERRNO.NE.5)141,218
  218 IF (IERRNO.GT.2)219,220
  220 CALL CODE
      READ (LINE,159) K
      L=K
  224 IF (K.LT.1)141,233
  233 IF (I53(1).LT.000106B)221,257
  257 J=1
  235 I=2
      GO TO 147
  221 IF (K.LE.IYEAR.AND.L.GT.IYEAR)141,335
```

```
335 IF (K.GT.IYEAR)336,225
225 K=IYEAR-K
    L=IYEAR-L
337 J=4
    GO TO 222
336 K=IYEAR-K+100
    L=IYEAR-L+10
    GO TO 337
219 CALL CODE
    READ (LINE,223) K,I,L
223 FORMAT(I2,A1,I2)
    IF (I.NE.1H,.OR.L.LT.10.OR.K.GE.L)141,224
191 IF (IERRNO.NE.1.AND.IERRNO.NE.3.AND.IERRNO.NE.5)141,228
228 CALL CODE
    READ (LINE,229) (ICOMM(I),I=49,IERRNO+48,1)
229 FORMAT(5R1)
    L=-2
    M=-3
    IF (IERRNO-3)230,231,232
232 M=ICOMM(53)-0000060B
    IF (ICOMM(52).NE.0000543.OR.M.LT.0.OR.M.GT.5)141,231
231 L=ICOMM(51)-0000060B
    IF (ICOMM(50).NE.0000543.OR.L.LT.0.OR.L.GT.5.OR.L.EQ.M)141,230
230 K=ICOMM(49)-0000060B
    IF (K.LT.0.OR.K.GT.5.OR.K.EQ.L.OR.K.EQ.M)141,129
129 I=8
    GO TO 163
192 IF (IERRNO.GT.7.OR.IERRNO.EQ.4)141,248
248 IF (IERRNO.GT.3)249,250
250 CALL OLDIO
    CALL CODE
    READ (LINE,186) K
    CALL NEWIO
251 IF (K.LT.1.OR.K.GT.33)141,252
252 J=1
260 IF (K.GT.5.AND.K.NE.8.AND.K.NE.14.AND.K.NE.19)258,259
259 IF (K.GT.4)262,278
278 LINE(J)=5
    J=J+1
    GO TO 258
262 IF (K.GT.5)263,265
265 DO 264 I=1,4,1
    LINE(J)=I
264 J=J+1
    GO TO 258
263 IF (K.GT.14)266,267
267 LINE(J)=K-2
    J=J+1
    GO TO 269
266 DO 268 I=15,17,1
    LINE(J)=I
268 J=J+1
269 K=K-1
258 LINE(J)=K
    K=L
    L=-1
    J=J+1
    IF (K.GT.-1)260,261
261 LINE(J)=-1
    IF (J.LT.4)271,272
```

```
  272 DO 270 I=1,J-2,1
      DO 270 K=I+1,J-1,1
      IF (LINE(K).EQ.LINE(I).AND.LINE(K).GT.-1) LINE(K)=LINE(K+1)
  270 CONTINUE
  271 J=1
      GO TO 235
  249 CALL CODE
      READ (LINE,*) K,L
      IF (L.LT.1.OR.L.GT.33.OR.K.EQ.L)141,251
  194 IF (IERRNO.NE.2.OR.I53(2).NE.000061B)141,236
  236 CALL POLL
C I/O WRITE
      CALL EXEC(2,KONWD,OUT15,1)
      DO 237 I=1,4,1
  237 LINET(I,IORDN)=2H
      NSTMT=26
      CALL READA
      GO TO 5200
C ENTRY POINT FOR CHANGE OF TERMINAL.
 5226 IF(IERRNO.LT.2.OR.IERRNO.GT.7.OR.IERRNO.EQ.4)141,238
  238 IF (IERRNO.GT.3)239,240
  240 CALL CODE
      READ (LINE,*) K
      L=K
  193 IF (K.LT.1)141,241
  241 I=3
      GO TO 163
  239 CALL CODE
      READ (LINE,*) K,L
      IF (L.GT.K)193,141
  196 IF (IERRNO.NE.2.AND.IERRNO.NE.5.AND.IERRNO.NE.8)141,213
  213 K=I53(2)-000060B
      IF (IERRNO.GT.2)281,314
  314 L=-2
      M=-3
  282 IF (K.LT.1.OR.K.GT.9.OR.K.EQ.L.OR.K.EQ.M)141,283
  283 I=4
      GO TO 163
  281 L=I53(5)-000060B
      IF (IERRNO.GT.5)284,316
  316 M=-3
  285 IF (L.LT.1.OR.L.GT.9.OR.L.EQ.M.OR.I53(3).NE.000054B.OR.I53(4).NE.0
     100112B)141,282
  284 M=I53(8)-000060B
      IF (M.LT.1.OR.M.GT.9.OR.I53(6).NE.000054B.OR.I53(7).NE.000112B)141
     1,285
  197 IF (IERRNO.NE.2.OR.I53(2).GT.000062B)141,286
  286 I=5
      K=I53(2)-000060B
      IF (I53(1)-000114B)163,287,288
  288 IF (I53(1).LT.000116B)178,200
  200 J=4
      GO TO 147
  198 IF (IERRNO.NE.2.OR.I53(2).GT.000063B)141,286
  199 IF (IERRNO.NE.2.OR.I53(2).GT.000064B)141,286
  201 IF (IERRNO.NE.2.AND.IERRNO.NE.5)141,289
  289 K=I53(2)-000060B
      IF (IERRNO.GT.2)290,323
  323 L=-2
  291 IF (K.LT.1.OR.K.GT.4)141,292
```

```
  292 IF (I53(1)-0000123B)293,294,295
  295 I=3
  296 J=5
      GO TO 147
  294 I=6
      GO TO 296
  293 I=6
      GO TO 163
  290 L=I53(5)-0000060B
      IF (L.LT.1.OR.L.GT.4.OR.K.EQ.L.OR.I53(3).NE.000054B.OR.I53(1).NE.I
     153(4))141,291
  202 IF (IERRNO.NE.2.OR.I53(2).GT.000064B)141,204
  204 K=I53(2)-0000060B
      IF (I53(1).GT.0000121B)207,205
  205 I=6
      GO TO 178
  207 I=7
      IF (I53(1)-0000125B)163,287,178
  203 IF (IERRNO.GT.2.OR.I53(2).NE.0000061B)141,298
  298 K=1
      I=6
      GO TO 200
  209 IF (IERRNO.GT.2.OR.I53(2).NE.0000061B)141,299
  299 CALL POLL
C I/O WRITE
      CALL EXEC(2,KONWD,OUT22,1)
      DO 301 I=1,9,1
  301 LINET(I,IORDN)=2H
      NSTMT=27
      CALL READA
      GO TO 5200
C ENTRY POINT FOR CHANGE OF TERMINAL.
 5227 IF(IERRNO.GT.10)141,302
  302 IF(NICKNM(1).EQ.0)GO TO 5228
      IOSTAT(IORDN)=1
C I/O WRITE
      CALL EXEC(2,KONWD,OUT32,13)
      NSTMT=28
      CALL READA
      GO TO 5200
C I/O WRITE
 5228 DO 303 I=1,5,1
  303 NICKNM(I)=LINE(I)
      GO TO 142
  210 IF (IERRNO.GT,2,OR.I53(2).NE.0000061B)141,206
  206 CALL POLL
C I/O WRITE
      CALL EXEC(2,KONWD,OUT15,1)
      DO 208 I=1,9,1
  208 LINET(I,IORDN)=2H
      NSTMT=29
      CALL READA
      GO TO 5200
C ENTRY POINT FOR CHANGE OF TERMINAL.
 5229 ICOMM(55)=-1
      ICOMM(61)=-2
      ICOMM(67)=-1
      ICOMM(73)=-1
      IF (IERRNO.NE.5.AND.IERRNO.NE.11.AND.IERRNO.NE.17)141,234
  234 IF (IERRNO-11)297,305,306
```

```
  306 CALL CODE
      READ (LINE,3 7) (ICOMM(I),I=49,65,1)
  307 FORMAT(17R1)
      DO 321 I=65,65,1
      IF (ICOMM(I).LT.000 6 B.OR.ICOMM(I).GT.000071B.AND.ICOMM(I).NE.000
     1 77B)141,321
  321 CONTINUE
      IF (ICOMM(6 ).NE.000 73B.OR.ICOMM(62).NE.000054B.OR.ICOMM(61).LT.0
     1000060B.OR.ICOMM(61).GT.000071B.AND.ICOMM(61).NE.000077B.OR.ICOMM(6
     23).EQ.000077B.AND.ICOMM(64).EQ.000077B.AND.ICOMM(65).EQ.000077B)14
     31,308
  305 CALL CODE
      READ (LINE,309) (ICOMM(I),I=49,59,1)
  309 FORMAT(11R1)
  308 DO 325 I=57,59,1
      IF (ICOMM(I).LT.000060B.OR.ICOMM(I).GT.000071B.AND.ICOMM(I).NE.000
     1077B)141,325
  325 CONTINUE
      IF (ICOMM(54).NE.00 073B.OR.ICOMM(56).NE.000054B.OR.ICOMM(55).LT.0
     1000600.OR.ICOMM(55).GT.000071B.AND.ICOMM(55).NE.000077B.OR.ICOMM(5
     27).EQ.000077B.AND.ICOMM(58).EQ.000077B.AND.ICOMM(59).EQ.000077B)14
     31,310
  297 CALL CODE
      READ (LINE,311) (ICOMM(I),I=49,53,1)
  311 FORMAT(5R1)
  310 DO 322 I=51,53,1
      IF (ICOMM(I).LT.000060B.OR.ICOMM(I).GT.000071B.AND.ICOMM(I).NE.000
     1077B)141,322
  322 CONTINUE
      IF (ICOMM(50).NE.000054B.OR.ICOMM(49).EQ.ICOMM(55).OR.ICOMM(49).EQ
     1.ICOMM(61).OR.ICOMM(55).EQ.ICOMM(61).OR.ICOMM(49).LT.000060B.OR.IC
     2OMM(49).GT.   71B.AND.ICOMM(49).NE.000 77B.OR.ICOMM(51).EQ.00 077
     3B.AND.ICOMM(52).EQ.000077B.AND.ICOMM(53).EQ.000077B)141,312
  312 IFINGR=49
  320 I=ICOMM(IFINGR)-000 47B
      IF (I.GT.18) I=9
      K=ICOMM(IFINGR+2)-000060B
      L=ICOMM(IFINGR+3)-000060B
      M=ICOMM(IFINGR+4)-000060B
      GO TO 163
  315 IF (ICOMM(IFINGR).GT.000071B.AND.I.LT.18)318,319
  319 IFINGR=IFINGR+6
      IF (IFINGR.GT.61.OR.ICOMM(IFINGR).LT.0)142,320
  318 I=I+1
      GO TO 147
C THIS STATEMENT DIRECTS PROGRAM EXECUTION TO APPROPRIATE ENTRY POINT
C AFTER CONTROL HAS BEEN SWITCHED FROM ONE TERMINAL TO ANOTHER.
 5200 GO TO (5201,5202,5203,5204,5205,5206,52 7,
     15208,5209,5210,5211,5212,5213,5214,5215,5216,5217,
     15218,5219,5220,5221,5222,5223,5224,5225,5226,5227,5228,
     15229)NSTMT
      END
C
C ***********************************************
      SUBROUTINE READA
C ***********************************************
C PROCESS SCHEDULER ROUTINE.
      INTEGER STMT,LINE(36),KOMON(128),NFLAG(6)
      COMMON ICOMM(128)
      COMMON LINET(36,6),LSTAT(128,6),STMT(6),KUEUE(6),NEXT,NSTMT,LAST
```

```
       1,NBYTES(6),IOSTAT(6),ICCMND(6)
        EQUIVALENCE (ICOMM(1),KOMON(1)),(ICOMM(57),IORDN),
       1(ICOMM(65),LINE(1)),(ICOMM(3),IERRNO)
        NFLAG(IORDN)=IOSTAT(IORDN)
        STMT(IORDN)=NSTMT
        IOCMND(IORDN)=1
      1 CALL POLL
        IF(NEXT.EQ.0)1,2
      2 N1=NEXT
        NEXT=-1
        NSTMT=STMT(N1)
        IF(NSTMT.NE.19.OR.LINET(1,N1).NE.2HDD)5,6
      6 DO 7 I1=1,6
        IF(STMT(I1).GE.4.AND.STMT(I1).LE.10.OR.STMT(I1).EQ.12.OR.
       *STMT(I1).EQ.13)8,7
      8 IOSTAT(N1)=1
        GO TO 1
      7 CONTINUE
C SWAP KOMON AREA
      5 J1=IORDN
        DO 3 I1=1,128
        LSTAT(I1,J1)=KOMON(I1)
        KOMON(I1)=LSTAT(I1,N1)
      3 CONTINUE
C
        IERRNO=NBYTES(N1)
        IF(NFLAG(IORDN).EQ.1)GO TO 10
        DO 4 I1=1,36,1
        LINE(I1)=LINET(I1,IORDN)
      4 CONTINUE
     10 RETURN
        END
C
C ****************************************************************
        SUBROUTINE POLL
C ****************************************************************
C TRAFFIC CONTROLLER ROUTINE.
        INTEGER STMT,LUN(6)
        COMMON ICOMM(128)
        COMMON LINET(36,6),LSTAT(128,6),STMT(6),KUEUE(6),NEXT,NSTMT,LAST
       1,NBYTES(6),IOSTAT(6),IOCMND(6)
        EQUIVALENCE (ICOMM(37),LUN(1)),(ICOMM(50),ILUN),(ICOMM(57),
       1IORDN)
        IF(NEXT.LT.0)1,3
      1 DO 2 I1=1,5,1
        J1=I1+1
        KUEUE(I1)=KUEUE(J1)
      2 CONTINUE
        KUEUE(6)=0
        LAST=LAST-1
      3 DO 100 I1=1,6,1
        IF(LUN(I1).LT.7)100,4
C I/O STATUS CALL
      4 CALL EXEC(13,LUN(I1),J1,NBYTES(I1))
        IF(J1.LT.0)100,5
      5 IF(IOSTAT(I1).NE.1)6,8
      6 IF(IOCMND(I1).EQ.1)7,10
C I/O READ CALL
      7 CALL EXEC(1,LUN(I1)+020400B,LINET(1,I1),-72)
        IOSTAT(I1)=1
```

```
      GO TO 10
    8 DO 9 J1=1,LAST,1
      IF(KUEUE(J1).EQ.I1)10,9
    9 CONTINUE
      KUEUE(LAST)=I1
      LAST=LAST+1
      IOSTAT(I1)=0
   10 IOCMND(I1)=0
  100 CONTINUE
      NEXT=KUEUE(1)
      RETURN
      END
       END$
```