A MICROCOMPUTER BASED GENERATOR OF RECURRING OPERATIONAL REPORTS

John Bartlett Godley

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A Microcomputer Based Generator of
Recurring Operational Reports

by

John Bartlett Godley

June 1977

Thesis Advisor: S. L. Holl

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): A Microcomputer Based Generator of Recurring Operational Reports	5. TYPE OF REPORT & PERIOD COVERED: Master's Thesis; June 1977
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): John Bartlett Godley	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Naval Postgraduate School, Monterey, California 93940	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Naval Postgraduate School, Monterey, California 93940	12. REPORT DATE: June 1977
	13. NUMBER OF PAGES: 125
14. MONITORING AGENCY NAME & ADDRESS (If different from Controlling Office): Naval Postgraduate School, Monterey, California 93940	15. SECURITY CLASS. (of this report): Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Microcomputer
Report Generator

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This thesis proposes a Report Originating System to provide afloat and small commands with the capability of automatic data processing assistance in report generation. The discussed system is completely implementable in small, inexpensive general purpose microcomputer hardware. The principal benefit of the system lies in its ability to prompt the user to solicit the

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

information required to be submitted in the report and to partially analyze the user's responses for correctness of form and content. Such computerized assistance should result in higher report quality and the concomitant reduction of correcting message traffic. The Report Originating System incorporates a line editing capability which lends itself to any text editing process. Thus, frequently modified locally prepared documents such as unit instructions and directives can be originated and updated with this system.

Approved for public release; distribution unlimited.

A MICROCOMPUTER BASED GENERATOR OF RECURRING OPERATIONAL REPORTS

by

John Bartlett Godley
Lieutenant-Commander, United States Naval Reserve
B. S., New Mexico State University, 1964

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL

June 1977

ABSTRACT

This thesis proposes a Report Originating System to provide afloat and small commands with the capability of automatic data processing assistance in report generation. The discussed system is completely implementable in small, inexpensive general purpose microcomputer hardware. The principal benefit of the system lies in its ability to prompt the user to solicit the information required to be submitted in the report and to partially analyze the user's responses for correctness of form and content. Such computerized assistance should result in higher report quality and the concomitant reduction of correcting message traffic. The Report Originating System incorporates a line editing capability which lends itself to any text editing process. Thus, frequently modified locally prepared documents such as unit instructions and directives can be originated and updated with this system.

TABLE OF CONTENTS

I. INTRODUCTION

With the increased use of computers in military communications, command and control, a trend towards highly formatted message reports has occurred. A high degree of formatting in a report is conducive to machine processing of the report without the need for human intervention. This, in turn, allows processing a greater amount of data than was feasible in a manually oriented environment.

Although the advent of machine processable reports has been a boon to report recipients it has generated problems for the report originator. To be useful, a formatted report must be precise; there is no room for either syntax or content errors. While in manually processed reports, the originator was granted some degree of compositional freedom, the great variety exhibited by ordinary prose is incompatible with contemporary automated report processing. Reports intended for direct computer consumption must be strictly formatted and the computer is intollerent of errors in syntax or content. Satisfying the computer's demand for precise formatting is a difficult and time consuming task. With the advent of machine readable formatted reports this odious burden has been exported from the staff or the command center to the report writer in the field.

To reduce the submission of erroneous formatted reports, a great deal of attention is given to the preparation of these messages. For example, the Department of Defense provides a mobile instructional team to present a two week course to middle management officers on Force Status Reporting. This emphasis has increased the burden of the

shipboard administrator and has detracted from his other professional duties. Needless to say, computer acceptable report formats have not been popularly received in the fleet.

The computer can be used to help the shipboard manager write error free reports more quickly. This thesis proposes the implementation of a Report Originating System to provide such a function. The Report Originating System was initially developed by LCDR J. G. Holyoak [4]. That system was tailored specifically to the Naval Force Status Reort. The goals of the present work were to broaden the applicability of the Report Originating System to enccmpass all formatted and partially formatted reports, to strengthen the system's error detecting capabilities, and to simplify and fully document the user's command repertoire. A great portion of the original code remains intact and the original data structures are largely unchanged. The major coding modifications dealt with improving the editing facilities, providing the ability to handle multiple lines of text, extending the error analysis capability and expanding the system to handle larger size documents. The software documentation was expanded considerably, not only to facilitate maintenance but also to serve as a design document in the event that the Report Originating System is rewritten in another language for another computer.

In the next chapter, the events leading up to formatted reports are discussed. The Report Originating System concepts and constructs are examined in Chapter III while the programmatic details of the system are discussed in Chapter IV. Recommendations for further development and conclusions are considered in Chapters V and VI respectively. The three appendices provide detailed guidance to users of the system.

II. BACKGROUND

The information reporting requirements levied upon the commanding officer of a naval ship have increased markedly in recent years. This phenomenon is primarily due to two factors: the ability to communicate reliably to almost any point on the earth and the ability to handle and condense an enormous volume of information to a succinct and comprehensible quantity. Reliable communications stem from the technological strides in communications electronics made during the last three decades. The ability to process large quantities of information is a result of the digital computer. Knowing that it is possible to receive and correlate data from units deployed around the world, the top echelons of command have found it very easy to demand new reports from subordinate units for areas of high level interest.

In recent years, there have been attempts to reduce the amount of paperwork generated within the Department of Defense. In 1972, Secretary of Defense Melvin E. Laird directed that all DOD activities review and eliminate paperwork that was counterproductive to efficient management efforts. A year earlier, the Vice Chief of Naval Operations solicited the aid of the Naval Audit Service to perform an audit on fleet reporting requirements. The Naval Audit Service was in the midst of its audit when the Laird policy was decreed. Although there was considerable motivation to reduce the problem, the Naval Audit Service's findings showed that the situation had worsened rather than improved. Of 147 specific reporting requirements analyzed, 24 percent were redundant. By 1973 the 147 required reports had grown

to 160 [9]. The best that can be said about our progress in reducing reports is that our efforts may have slowed the rate of growth.

When discussing the enormous reporting problems within the Department of Defense, one often loses sight of the origin of the information. While the plethora of information readily available to the highest echelons of command may often allow top management to respond quickly and accurately to impending crises, this information is not gathered without cost. In today's Navy, the originators of this information, the units of the fleet, usually must operate without the benefit of automated information processing. This means that each bit of data processed in the Pentagon's information system was introduced into the data base by laborious means. Time spent gathering, collating and preparing data for submission was not spent on improving operational readiness. Although we have not reduced the operational responsibilities of commanding officers, we have greatly increased their administrative responsibilities. This additional administrative burden has been added usually without a commensurate strengthening of the capability to deal with increased paperwork.

One attempt by the Navy to reduce reporting requirements was the Composite Reporting (COMPREP) study concluded in June 1975. The goal of this study was to devise one composite report that would combine the information requirements of the Naval Force Status Report (NAVFORSTAT), Movement Report (MOVREP), Casualty Report (CASREPT) and Emergency MILSTRIP (Military Standard Recquisitioning and Issue Proceedures). The study developed the reporting structure, training and software necessary to accomplish this goal. Based on the results of this study, the Naval Electronics System Command recommended that the COMPREP

system be implemented using a dedicated microprocessor to support the system [6].

At the shipboard level, an originator or "drafter" of a report usually works in the following manner. The last example of the submitted report is reviewed to determine what in that report remains pertinent. The information that is unchanged since the last submission is copied into the new report along with other data reflecting changes in the current status. This process of selective plagiarism is almost universal. Selection of a pragmatic model upon which to base the report assists the drafter in spending the least amount of his time in report generation while affording the greatest confidence that the report will be accepted. The goal of acceptability frequently competes with the goal of accuracy. This method of report generation is particularly adaptable to data processing assistance.

Since efforts to reduce have not been successful, alternate approaches to reducing the reporting burden on fleet units is indicated. If we can not eliminate, at least we can assist. The preceding paragraph inferred that the computer, whose appetite for data partially created the report problem, may also be a key to a solution. By transferring the reported information from paper to a medium that can be accessed by computer, modern data processing methods can be applied to the problem. The information that remains unchanged between individual reports could be rapidly included in the new report; other information could be edited to reflect the modified situation. The commercial world is well on its way towards automated reporting and information processing. The computer industry has responded with the development of hardware and software appropriate to the task. These capabilities can now be provided to the small military command at a moderate cost.

III. REPORT ORIGINATING SYSTEM CONCEPTS AND CONSTRUCTS

The Report Originating System is composed of two computer programs, CREATE and ROS, written in the PL/M programming language for an 8080 based microcomputer. The equipment used to implement the Report Originating System consisted of an Intellec-8 MCS microcomputer with 16,000 bytes of random access memory, a Shuggart dual floppy disk drive and a Datamedia Elite 2500 CRT display and entry terminal. Many host computer configurations were feasible; this hardware was selected because it was inexpensive and was available in the U. S. Naval Postgraduate School's Computer Laboratory. The CREATE program consists of approximately 4500 bytes of machine code while ROS consists of approximately 7200 bytes. These program sizes resulted in 7000 and 4000 bytes of memory available as a work area after the program and the operating system were loaded.

A. CONCEPTS.

Reports are typically subdivided into data elements which correspond to a paragraph or formatted line. Each data element generally has associated with it a data label which may be a paragraph heading or an acronym used to describe the information that follows. The data element has been chosen to be the building block upon which the Report Originating System is based. Originally, [4] the Report Originating System was intended to assist in generating highly formatted recurring operational reports such as the Naval Forces Status Report (NAVFORSTAT), the Movement Report

(MOVREP), the Casualty Report (CASREPT) and RAINFORM report. These reports are very common, very cryptic and highly prone to format errors. Inclusion of a "format free" formatting convention broadened the range of application of the Report Originating System to include generation of any report or document whether primarily formatted or textual.

Every report, whether formatted or textual, is considered to be composed of data elements. This is easy to envision for highly formatted reports but it also can be applied to unformatted reports or documents. These usually adhere to the pattern of the naval letter or directive - that is, they are divided into various sections, each with an identifying paragraph heading followed by text. These headings make natural data labels in the Report Originating System. The concept of the basic report information is expanded from the single line of the NAVFORSTAT or MOVREP to the section or paragraph of a unit instruction. This expansion allows a broader range of applicable documents.

An individual data element is composed of a data label, the basic report information, error analysis information, and prompting information. These segments of the data element always appear in this order. A collection of one or more data elements, preceded by a directory to the data element location and terminated by an end of file marker, forms a "DAT" (for data base) file. A DAT file is the end product of the CREATE program. Figure 1 shows a graphical representation of the contents of a DAT file.

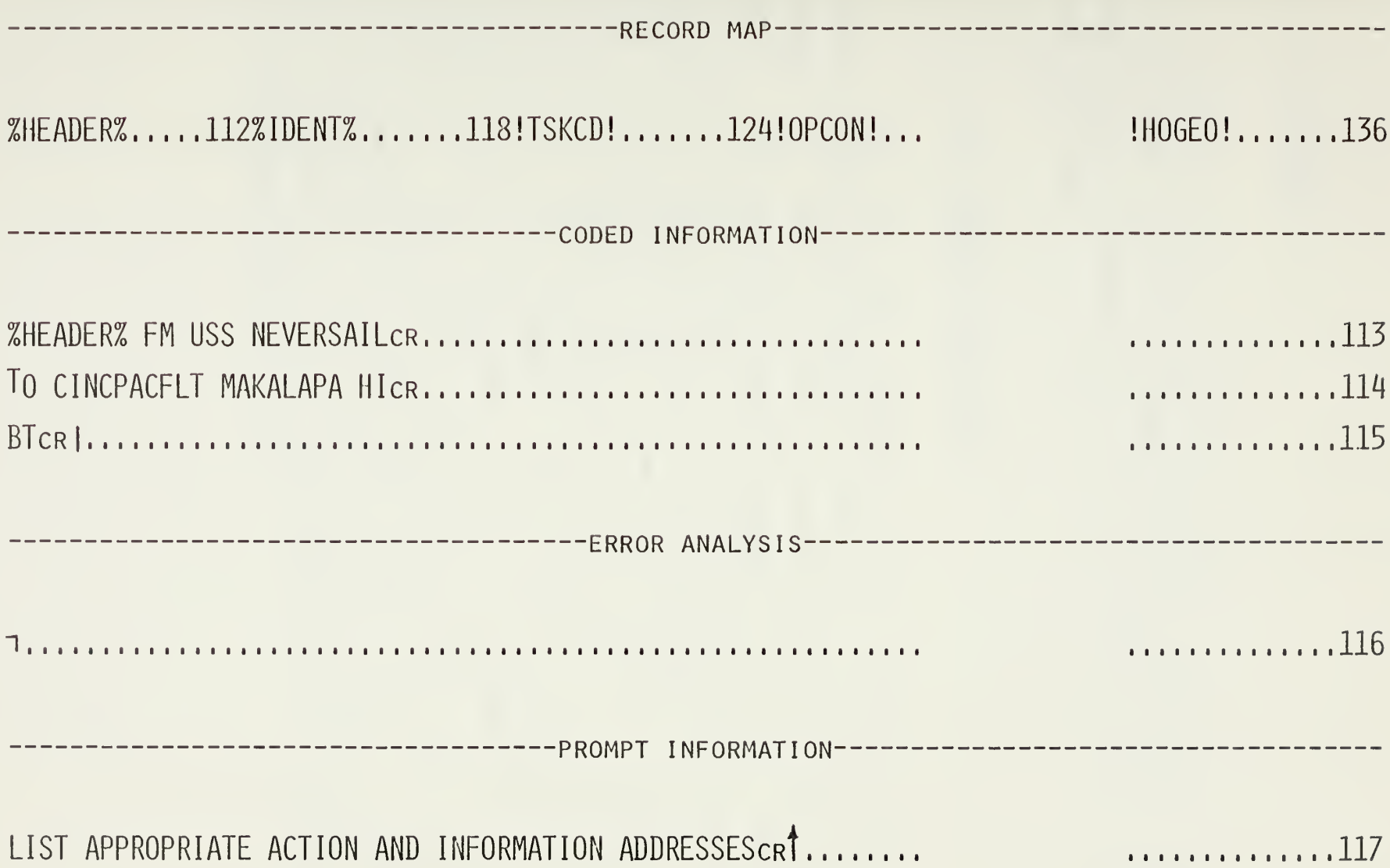

Figure 1 - A GRAPHICAL REPRESENTATION OF A DAT FILE

B. CONSTRUCTS.

A DAT file contains all the necessary information used by the ROS program to generate a specific report. The composition and construction of the DAT file sections are described below.

1. Data Element Directory.

Each DAT file commences with a directory section called a RECORD MAP. This map indicates the initial disk address of each of the data elements contained within that report. The RECORD MAP section occupies the first twelve 128 byte records within the DAT file and provides enough space for 96 separate entries. Each RECORD MAP entry is 16 bytes (characters) long. The first 14 bytes contain the data label while the last two bytes point to the disk location of the described data element. To provide for variable data label lengths, a data label is preceded by and followed by a delimiter. Thus, the RECORD MAP can accommodate data labels containing twelve or fewer characters. Bytes not required to store characters or disk location pointers are filled with binary zeros. A byte containing binary zeros would look like 00000000 while a byte containing the character zero would look like 00110000, the ASCII "zero" character code.

2. Data Elements.

The primary information content of a DAT file is included in the data element section of the file. This section contains one or more data elements, up to a maximum of 96. A data element should be included for each possible data label in a report. Each data element occupies at least three 128 byte records; one for the basic report information, one for error analysis information and one for prompting information.

a. Basic Report Information.

The basic report information section of a data element is also referred to as the "code area" or "code information area". The initial entry in the coded area is the data label bracketed by the delimiters "%" or "!". The percentage symbol indicates that the data label is not to be included in the final report while the exclamation symbol indicates inclusion. The remainder of the coded area contains the actual information to be included in the report. If this information requires more space than initially provided, additional 128 byte records are allocated. The coded area may be subdivided into subfields by using virgules or braces to facilitate error analysis. This function is described more completely in Appendix C. The end of the code area is represented by the end of code delimiter (|).

b. Error Analysis Information.

The next logically sequential area of the DAT file is occupied by records used for error analysis. This area contains error analysis commands separated by virgules or braces to correspond to the subfields of the coded area. The error analysis area is terminated by the end of error delimiter (¬).

c. Prompting Information.

The third area of a data element is used to store prompting information. Each data element should have associated with it a prompting question which indicates to the ultimate user the information required by that data element. Prompting questions should explicitly ask for the sought information and may contain exemplary or amplifying information as well. The prompting information contains as many records as required and is terminated by the end of prompt character (↑).

The sequence of coded area, error analysis information and prompting information is repeated for each data element included in the RECORD MAP.

3. End Of File.

After all information has been entered regarding the specific data elements of the report, the end of file character, a back slash, is used to denote the end of the report. When encountered, this symbol will cause any report information resident in memory to be written to the disk and

will close the DAT file. The DAT file may be reopened and modified using the ROS program.

The basic repeating structure of the DAT file allows initial creation of a report through the CREATE program and subsequent definition and modification of the report by the ROS program. The DAT file is composed of logically sequential 128 byte records used to store the information. Only 90 bytes of each of these records are used to store the actual information; this space is adequate to store a single line of text. Most of the remaining bytes are unused except for the bytes used to point to the storage location of the next logically sequential record. These pointers allow insertion of data in the middle of an existing file.

C. IMPLEMENTATION.

Since both the CREATE program and the ROS program are based on the data structure of the DAT file, both contain similar data handling structures. The structures in the ROS program are more powerful since it is intended that the ROS program be used to modify the DAT files. In a typical application, the CREATE program would be used to build an initial "draft" or DAT file. This initial file is composed of sequential records analagous to the sequential lines of a rough draft. Limited editing facilities are provided in the CREATE program since most changes during the CREATE execution would correct input errors. The ROS program provides more elaborate editing facilities to allow insertion and deletion of text, rapid accessing of specific records within the DAT file as well as modifying the current line. This capability requires the use of the record pointer system incorporated at the end of each record.

Amplification of the functioning of each subroutine in ROS and CREATE is provided in the next chapter. This information is provided to assist in further program development and maintenance. The three appendices ennumerate the use of ROS and CREATE and of the line editor incorporated in each.

IV. SOFTWARE DESIGN FEATURES

The Report Originating System is composed of two independent software programs. The principal program, ROS, is the nucleus of the system and generates the desired report using previously stored data as a basis of information. The CREATE program produces the data base for a specific report from which the ROS program may generate reports.

The CREATE program is a general software tool used to produce a data base named a DAT file. Each DAT file is specifically tailored to the requirements of the ultimate user and to the format of a specific report. Therefore, each user will have several DAT files available on individual diskettes to meet his varying reporting needs. A specific DAT file is composed by inputting information concerning base-line information for the report, error analysis criteria and prompting questions. The details of the CREATE program concepts and operation are explained in detail in Section B and in Appendix C.

The ROS program uses the information contained within the individual DAT files as input. The ROS program contains the necessary data handling structures to allow the system user to make appropriate modifications to this initial data so that it will reflect the currently reported status. The ROS program will output a new file named a Message (MSG) file as well as modifying the composition of data within the DAT file. This MSG file contains all the required report information in a format suitable for transmission to the

requesting authority. ROS program details may be found in the next section and in Appendix A.

The software as described in the following pages significantly reflects the microcomputer system in which these programs were implemented. Again, the specific microcomputer system used was chosen primarily because it was available. Many suitable new commercial microcomputer systems have appeared since the inception of this project. The following discussion of program structure is intended to assist any further development or maintenance of the Report Originating System on the Intellec-8 system or the development of a comparable software package on another system.

A. ROS PROGRAM ORGANIZATION.

The ROS program is the instrument used by report drafters to originate reports required by higher authority. A copy of the program appears at the end of this thesis. The Report Originating System is written in the PL/M high level programming language, compiled on an IBM-360/67 computer and executed on an Intellec-8 (8080 based) microcomputer using the CP/M monitor control program. The program is divided into five modules: Operating System Interface; Initialization; Editing; Error Analysis and Output. Each of these modules will be discussed in turn.

1. Operating System Interface Module.

The Operating System Interface Module provides the interface between the ROS program and the CP/M monitor control program. Principally, this module provides the

Input/Output (I/O) operations necessary for the program. The module consists of 18 subroutines most of which crypticly pass a "function number" and an "information address" from the ROS program to the CP/M via a system entry point named "ENTRY". Varying the function number provides various capabilities such as reading from and writing to peripheral devices, opening, closing and searching files and transferring data between memory and diskette storage. Altogether there are 23 distinct I/O operations that may be performed by the CP/M system. These are covered in detail in the CP/M INTERFACE GUIDE [2]. The subroutines comprising the Operating System Interface Module fall into three functional groups discussed below.

a. Console Input/Output.

(1) CRTIN - Defines the input port number for the console and extracts ASCII coded information from that port representing characters displayed on the console.

(2) READC - Calls for another character to be read from the console and then translates lower case ASCII characters to upper case ASCII for internal representation.

(3) PRINTCHAR - Transmits ASCII character from memory location, through the computer output port, to the console for display.

(4) PRINTCHARI - Determines when a control character is encountered and transmits the character from memory to the console. Character will be preceeded by an up-arrow.

(5) CRLF - Displays a carriage return and a line feed (i.e. goes to the beginning of the next line).

(6) PRINT - Displays the string of ASCII characters that commences at the given location in memory and is terminated with a "$".

b. Diskette Input/Output.

(1) DISKREAD - Reads the next 128 byte record from the file that is being accessed. The contents are placed in the 128 byte area of memory as specified by the Direct Memory Access (DMA) pointer.

(2) DISKWRITE - Writes the 128 bytes following the address specified to the next available record in the file being accessed.

(3) OPEN - Opens the file specified by the address passed to this procedure. If the file is not located in the directory, a diagnostic message is returned. Opening the file allows further transactions with that file to occur.

(4) CLOSE - Closes the file specified by the address passed to this procedure. If the file is not located in the directory, a diagnostic message is returned. Closing the file updates the directory entry for the file and terminates further processing of that file.

(5) SEARCH - Searches the directory to match the <filename> <filetype> contained in the File Control Block (FCB) passed to this procedure. If no match is made, a diagnostic message is returned; otherwise the address of the first occurrence of <filename> <filetype> is returned.

(6) MAKE - Creates a directory entry for <filename> <filetype> as represented by the FCB passed to this procedure. The file created is initialized to empty.

(7) MOVE - Moves the number of bytes (characters) from the source location to the destination location. Source address, destination address and number of bytes (length) are passed as parameters.

c. Utility Functions.

(1) MON1 - A monitor procedure which passes the function number and the address of the information to the CP/M. This procedure does not return a value. It is called by many of the previously mentioned subroutines.

(2) MON2 - A monitor procedure providing the same function as MON1 except that a return value is expected.

(3) MON3 - A monitor procedure providing the same function as MON2 except that the returned value is an "address" type rather than a "byte" type.

(4) SET$DMA - Sets the Direct Memory Access (DMA) address to the address of the 128 byte DMA buffer specified. Subsequent disk I/O operations will take place starting with the memory location specified.

(5) LIFTHEAD - Lifts the read/write head from the diskette drive currently in use.

(6) ERROR - This procedure is called from other procedures and prints diagnostic messages to the operator.

2. Initialization Module.

The initialization module consists of 14 subprograms which open the appropriate DAT file, create a MSG file in which to store the output and create working maps to include those data elements selected for modification and those data elements which are to appear in the resultant MSG file. These working maps are derived from the responses to data element queries initiated by this module. The module subroutines may be classified in three functional groups.

a. File Operations.

(1) MAKEMSGFILE - This procedure creates a MSG file control block (FCB) and creates and opens a MSG file. First the procedure inserts the <filetype>, "MSG" into the MSG file control block. Next the "extent" and the "next record" fields of the DAT FCB are zeroed. A search of the directory is made to determine if a MSG file already exists; if so, an error message is returned. Next, a directory entry for <filename>.MSG is made and finally, the file is opened.

(2) INIT - This procedure opens the DAT file and MSG file. First <filename> is moved into the FCB and then the procedure calls MAKEMSGFILE to create and open a MSG file. Next, the <filetype>, "DAT", is moved to the DAT FCB and the "extent" and "next record" fields of this FCB are zeroed. The file <filename>.DAT is opened and then checked to see if the file was successfully opened.

(3) INC$RM - Increments the record map pointer.

(4) INC$WM - Increments the working map pointer.

(5) PRINT$DATA$E - This procedure displays the data element labels associated with a DAT file in order that selection of the data elements may be accomplished. This procedure walks sequentially through the record map area, strips the data label delimiters (% or !) from the data label and displays them. The process is repeated until either the last data label has been displayed or a flag has been set which ceases the display.

(6) SAVEEXRN$NR - This procedure calculates the location of a data element and also calculates the number of records associated with the data element. The location is comprised of its "extent" and "record number". The extent is the first byte in the directory entry, the record number is the next byte and the length is calculated by computing the difference between the record number of the next data element and of the present data element.

(7) PRINT$PACE - Prints blanks equal to the length of the data label.

(8) CHK$RESPONSE - Goes sequentially through all data elements checking for responses to queries. If a "Y" response is given, SAVEEXRN$NR is called, if "N" is given the next data element is processed and if "S" is given in response, the interrogation process is stopped for the remaining data labels and the program continues execution.

(9) SEL$WE - This procedure locates the WORK$MAP area in memory, defines the number of data labels

to be printed on one CRT line, prints out the query, "SELECT DATA ELEMENTS TO WORK WITH", displays data labels and checks the responses. If the data elements to be modified have been selected, the procedure prints out the query, "SELECT DATA ELEMENTS TO BE INCLUDED IN THE REPORT". The selection process is repeated with a continuation of the working map area being constructed.

(10) RE-SELECT - This procedure sets the logical flags which allow SEL$WE to create the working map area for the data elements to be included in the report.

(11) SET$MEM - The limits of available memory are defined and the working map area is set up. The next record number to read is entered in the DAT FCB, the number of records occupied is determined and a counter for the number of records read is set to zero.

(12) OPEN$EXT - Sets the "extent" field in the DAT FCB and if the DAT FCB is not found in the directory, a diagnostic error message is returned.

(13) READDREC - Checks to see if the system is in the proper extent and if not, opens the proper extent, finds the appropriate record to start with. Reads in the 128 byte record and checks to see if it was read in properly. The number of records read counter is incremented and the data area pointer is moved to the beginning of the next record. The next record to be read is set to the beginning of the TBUFF area which contains the names of the files loaded into the transient program area of memory.

(14) READ$DAT - Checks to see if the available memory has been consumed and if the number of records read is equal to the number of records in that data element. If so, the reading of records is completed. If a new data

element is being accessed, the location and number of records information is reset. If all records have not been read, the next record is read. Finally, the extent and record number fields of the DAT FCB are set to the current values.

(15) READ$MORE - The current location of the working map is updated and the extent and the next record fields of the DAT FCB are reset. The pointer to the data area currently used is saved and the subsequent record is read. The location of the working map is saved and the next location of data is updated. This procedure continues to read records of multiple record data elements.

(16) ALLOCATE - Allocates an empty record from the freelist area of the diskette to allow insertion of a line in an existing file. Bookkeeping operations of maintaining freelist pointers and of identifying the appropriate extent and record for reading and writing operations are accomplished.

(17) FREE - Releases a record after an entire line has been deleted from an existing file. The released record is placed on the front of the freelist queue.

(18) DIGTONUM - Converts a string of ASCII digits to a numerical value so that they may be used in arithmetic operations.

(19) LONG - Determines the number of characters in a specific data label.

3. Editing Module.

The Editing Module provides the capability to modify the information contained within the DAT file. There are two modes of operation for this module, the prompted mode and the non-prompted mode. The prompted mode performs the editing operations without the user being directly aware that they are being accomplished; the non-prompted mode requires that the user explicitly invoke each of the desired editing commands. The Editing Module is composed of 27 subroutines which manipulate characters in the old and new buffer areas and implements updating the information stored in the DAT file. The subroutines are divided into two functional groups; those associated with the line editing functions and those associated with updating the information contained in the DAT file. The use of the line editor in manipulating information between the old buffer and the new buffer is discussed in Appendix B.

a. Line-Editing.

(1) BACK$UP - This procedure affects the new line buffer. The new buffer pointer is backspaced one space and a blank is printed followed by a backspace to allow insertion of a new character. If the line is backspaced to the beginning position, a bell is sounded.

(2) MOVETOOLD - The contents of the new buffer area are moved to the old buffer area and the old and new buffer pointers are reset to zero.

(3) OLDTONEW - The next character in old buffer is moved into the new buffer and inserted after the

current position of the new buffer pointer. Both old and new buffer pointers are incremented.

(4) ECHO$ON - Moves the next character from the old buffer to the next position in the new buffer. The character is displayed on the console CRT. Both the new and old buffer pointers are incremented.

(5) COPY$ONE - If the end of the old buffer has not been reached, then the procedure ECHO$ON is called. Otherwise, a bell is sounded.

(6) P$MOVE$ON - Moves the contents of the old buffer to the new buffer up to the character indicated in the old buffer.

(7) ENTER - This procedure displays either the "<" or ">" character and resets the flag "INSERT".

(8) PRINT$OLD - Displays the contents of the old buffer, including control characters, followed by a carriage return and a line feed.

(9) PRINT$NEW - Displays the contents of the new buffer including control characters.

(10) PRINT$BOTH - Displays the contents of the old buffer followed by the contents of the new buffer.

(11) COPYRMO$N - Copies the remaining characters from the old buffer to the new buffer. Displays the "+" character when completed.

(12) BSON - Backspaces both the old and new buffer pointers by one character position. If either pointer is in the initial buffer position then a bell rings.

(13) COPY$ON - Searches the old buffer from the current position of the old buffer pointer until the first occurrence of the character passed to this procedure. If no match occurs, a bell is sounded. Otherwise, the characters from the current position of the old buffer pointer up to the identified character are copied to the new buffer and displayed on the screen.

(14) DELETE - This procedure searches the old buffer for the first occurrence of the identified character. If this character is not found, a bell is sounded. Otherwise, all characters are deleted up until the first occurrence of the specified character and a "%" character is displayed in the position of each deleted character. The old buffer is finally compressed to remove the empty spaces resulting from the deletions.

(15) DEL$N - Resets the old buffer pointer and the new buffer pointer to zero and displays the end of file character (back slant) on the console CRT.

(16) DISPLAYRMO$N - This procedure will print blanks up until the current position of the old buffer pointer and then will display the remaining characters in the old buffer. The cursor moves to the beginning of the next line on the CRT and the contents of the new buffer are displayed.

(17) DEL$O - The characters from the current position of the old buffer pointer up to the end of the old buffer are deleted. Percentage signs are displayed in place of the deleted characters. If there are no characters in the old buffer, a bell is sounded.

(18) ESCAPE - This procedure turns off any special meaning of the character to follow and enters the character into the new buffer. This procedure may be used to override the action of extended line editor commands.

(19) PRINT$TAB - Enters the special tab character as the next character in the new buffer and displays the tab character on the console. If the tabulation will exceed the allowed size of the new buffer, a bell is sounded.

(20) BEGIN$WRD - This procedure backspaces to the first character of a word.

b. Updating Procedures.

(1) INC$DA - Increments the pointer in the current DAT file that points to the character accessed.

(2) INC$CA - Increments the pointer to the record containing the information base in the DAT file.

(3) INC$ER - Increments the pointer to the record containing the error analysis information in the DAT file.

(4) MOVEDLNEW - Moves the data label from the old buffer to the new buffer. If the fully prompted mode is in effect, the characters are merely moved to the new buffer; if the non-prompted mode is in use, the characters are moved to the new buffer and are also displayed on the console.

(5) MOVE$CODE$OLD - The data base information is moved from the DAT file to the old buffer for editing. Characters are moved singularly until the end of code character (|) is encountered. If the non-prompted mode has been selected, characters are echoed as they are moved. When all characters have been moved to the old buffer, MOVEDLNEW is called and the data label is moved to the new buffer.

(6) SET$PTR - This procedure sets pointers to the beginning of the error analysis section and the beginning of the prompting question area.

(7) NEXT$DE - This procedure sets up the system to edit a data element. First MOVE$CODE$OLD is called followed by INC$DA and SET$PTR.

(8) UPDATE$DAT - Updates the information to be included in the DAT file. Characters are transferred from the new buffer to a holding area called CODE$A, starting with the character following the data label up to the end of the new buffer. The end of code delimiter (|) is inserted at the end of the transferred data. This routine repetitively calls the subroutine INS$INC which transfers the data one character at a time.

4. Error Module.

The error module checks for possible error conditions and performs the error analysis operations on the edited data. Error commands contained in the error command records of the DAT file are interpreted and determine the comparative procedures to be executed. Procedures which accomplish the desired error checking process are

incorporated within this module. Utility functions, such as typing individual characters as alphabetic or numeric, are included and error warning diagnostic messages for display to the operator are available. The Error Module also contains the procedure LEDIT which provides the line editing capability. The module is divided into the Interpretation, Error Checking, Utility and Editing groups.

a. Utility Group.

(1) RE$ENTER - The contents of the new buffer are moved to the old buffer and the updated contents of the old buffer are displayed followed by the contents of the new buffer. This confirms that the updating process has been accomplished without error.

(2) WARNING - Displays various warning messages when an erroneous condition occurs.

(3) INC$NB - Increments a character pointer in the new buffer.

(4) SPPDCOM - Used to identify the punctuation characters, space, period and comma.

(5) ALPHA - Defines alphabetic characters to be letters A - Z, the characters CTRL A - CTRL Z or a period, space or comma.

(6) NUMERIC - Defines numeric characters as the digits 0 - 9, the minus sign (-) or space, period or comma.

(7) NEXT$SF - Walks through the DAT file to get the next subfield of a data element. Pointers to the information area and data analysis area are incremented appropriately.

b. Interpretation Group.

(1) DO$CMD - Checks for a null error command which returns control to the calling statement. Otherwise, error commands are identified by the first letter of the command and the appropriate error checking procedure is called.

c. Error Checking Group.

(1) CHK$ALPHA - Checks the contents of the new buffer up to the next error analysis delimiter (virgule or brace) for alphabetic characters. A warning message is returned if other characters are encountered.

(2) CHK$NUMERIC - Checks the contents of the new buffer up to the next delimiter for numeric characters. A warning message is displayed on an exception basis.

(3) CHK$PERCENT - Compares the revised numerical contents of a subfield against the previous contents. If the difference exceeds a specified amount, a warning message is displayed.

(4) CHK$SEQUENCE - Checks if the contents of the new subfield are sequential to the contents of the old subfield. May be used to check report serial numbers.

(5) CHK$LENGTH - Checks if the contents of the new subfield occupy the same number of character positions as the old subfield.

(6) CHK$ERR - Walks through the error analysis record of the DAT file one subfield at a time until the end of error character (¬) is encountered. Individual error commands are interpreted through calls on DO$CMD. The pointer in the error analysis record is incremented.

(7) ASK$QUESTION - Displays the prompting question applicable to the subfield under consideration.

(8) CHK$ANSWER - Checks the response given under the prompted mode for compliance with error commands. Performs the same function as CHK$ERR.

(9) END$IP - Checks if the contents of the new buffer will overwrite the error analysis section of the DAT file, places a virgule after the response to the prompted question and accomplishes error analysis for that response.

d. Editing.

(1) LEDIT - Acts as an input interpreter while in the edit mode. Input strings are first checked for basic line editor commands, then for extended line editor commands. The appropriate editing routines are called in these cases. If a rubout character is entered, the new buffer pointer is backspaced. If none of the above special characters are encountered, input characters are echoed until the new buffer is filled. When this occurs, a bell is sounded.

5. Output Module.

This module performs the actual updating of the edited information in the DAT file and also incorporates this information into the MSG file. Included are routines to perform the necessary file operations and an EDIT procedure which controls the editing process.

a. MSG File Operations.

(1) INC$MSG - Creates a buffer to be used to transfer data between the absolute addresses 80 - 100 hexadecimal. The diskette directory is checked to see if a MSG file exists and returns a warning if it does. A pointer in this buffer is incremented with each call.

(2) MOVE$MSG - Copies the edited information, character by character, into the DAT file. A check is made to see if the data label is to be included in the report. Characters are moved from the DAT file to the MSG file until an end of code delimiter (|) is encountered. The MSG file line is ended with a carriage return and a line feed.

(3) WRITE$MSG - Writes the record currently accessed to the MSG file.

(4) CLOSE$FILES - Closes the MSG file and saves the appropriate extent and record information.

b. DAT File and Utility Operations.

(1) BLANK$BUF - Blanks out the contents of the old and new buffers.

(2) BASE$NEXT$DE - Brings in the information concerning the next selected data element into the transfer buffer. Working map pointers are incremented to indicate the next data element.

(3) UPDATEDATFILE - Coordinates the actual updating of the DAT file by calling MOVE$MSG. DAT file extent and record entries in the DAT FCB are updated. This process is continued for all data elements.

c. Editing Coordination.

(1) EDIT - Coordinates the editing process for the prompted and non-prompted modes. If in the prompted mode, prompting questions are displayed and then LEDIT is called to handle the response and the response is checked. This sequence continues for all subfields for that data element. In the non-prompted mode, LEDIT is called to process the desired changes and the entered data undergoes error analysis. If a warning has resulted from error analysis, the same data element is reprocessed. Otherwise, the DAT file is updated, the old and new buffers are blanked out and the next data element is processed. This pattern is continued until the end of file character (back slant) is encountered and then the MSG file is created and closed.

B. CREATE PROGRAM ORGANIZATION.

The CREATE program is used by commands requiring reports from their subordinates to develop a DAT file with its data base of reporting information. The CREATE program is also available to the general user to provide a local capability of generating DAT files for unique reports or DAT files for interim use. In order to assure report uniformity and content reliability, it is imperative that standardized DAT files be produced and distributed by the highest possible level in the chain of command. A copy of the CREATE program is included at the end of this thesis. The program was written in the PL/M programming language, compiled on an IBM-360/67 computer and designed for execution on an Intellec-8 microcomputer using the CP/M operating system. The program is divided into three main modules: Initialization-Interface; Input-Editing and Finish. Many of the subroutines discussed in the following sections are also included in the ROS program software.

1. Initialization-Interface Module.

The Initialization-Interface Module includes subprograms to provide an interface between the CREATE program data structures and the CP/M control monitor program. The module opens a DAT file if one has not already been opened under that particular <filename> on the diskette in use.

a. Console Input/Output.

(1) CRTIN - Defines the input port number for the console and extracts ASCII coded information from that port representing characters displayed on the console.

(2) READC - Calls for another character to be read from the console and then translates lower case ASCII characters to upper case ASCII for internal representation.

(3) PRINTCHAR - Transmits ASCII character from memory location, through the computer output port, to the console for display.

(4) PRINTCHARI - Determines when a control character is encountered and transmits the character from memory to the console. Character will be preceeded by an up-arrow.

(5) CRLF - Displays a carriage return and a line feed (i.e. goes to the beginning of the next line).

(6) PRINT - Displays the string of ASCII characters that commences at the given location in memory and is terminated with a "$".

b. Diskette Input/Output.

(1) MOVE - Moves the number of bytes (characters) from the source location to the destination location. Source address, destination address and number of bytes (length) are passed as parameters.

(2) FILL - Used to fill the destination record with a particular character passed to this routine. FILL may be used to pad out a record with blanks or zeros.

(3) MON1 - A monitor procedure which passes the function number and the address of the information to the CP/M. This procedure does not return a value. It is called by many of the previously mentioned subroutines.

(4) MON2 - A monitor procedure providing the same function as MON1 except that a return value is expected.

(5) MON3 - A monitor procedure providing the same function as MON2 except that the returned value is an "address" type rather than a "byte" type.

(6) ERROR - This procedure is called from other procedures and prints diagnostic messages to the operator.

(7) DIGTONUM - Converts a string of ASCII digits to a numerical value so that they may be used in numeric operations.

2. Input-Editing Module.

The Input-Editing Module accepts entered information and builds the DAT file opened by the Initialization-Interface Module. The contents of a DAT file are a repetitive sequence of data base or "code information", "error commands" and "prompt information". These three categories of information are required for each data element included in the report. Although null or empty categories are allowed (but not recommended), at least one

128 byte record is allocated for each. If the input information exceeds the initial 128 byte allocation, additional sequential records are provided. Since the order of these three types of information is the keystone of the Report Originating System software, it is imperative that data entered during the creation of a DAT file conform to this sequence.

a. Input.

(1) PROMPT - Displays to the operator a prompting message to input coded information, error commands or prompt information.

(2) INC$RM - Increments the record map pointer.

(3) GO$NEXT$REC - Increments the storage pointer to the next even record boundary and increments the counter for the number of records utilized. For every 128 records used, the extent counter is incremented and the record number counter is reset to zero. The number of remaining records is decremented.

(4) MOVE$DL - Moves the data label into the record map area and moves the extent and record numbers into the two bytes following the data label.

b. Editing.

(1) BACK$UP - This procedure affects the new line buffer. The new buffer pointer is backspaced one space and a blank is printed followed by a backspace to allow insertion of a new character. If the line is backspaced to the beginning position, a bell is sounded.

(2) MOVETOOLD - The contents of the new buffer area are moved to the old buffer area and the old and new buffer pointers are reset to zero.

(3) ECHO$ON - Moves the next character from the old buffer to the next position in the new buffer. The character is displayed on the console CRT. Both the new and old buffer pointers are incremented.

(4) COPY$ONE - If the end of the old buffer has not been reached, then the procedure ECHO$ON is called. Otherwise, a bell is sounded.

(5) P$MOVE$ON - Moves the contents of the old buffer to the new buffer up to the character indicated in the old buffer.

(6) ENTER - This procedure displays either the "<" or ">" character and resets the flag "INSERT".

(7) PRINT$OLD - Displays the contents of the old buffer, including control characters, followed by a carriage return and a line feed.

(8) PRINT$NEW - Displays the contents of the new buffer including control characters.

(9) PRINT$BOTH - Displays the contents of the old buffer followed by the contents of the new buffer.

(10) COPYRMO$N - Copies the remaining characters from the old buffer to the new buffer. Displays the "+" character when completed.

(11) BSON - Backspaces both the old and new buffer pointers by one character position. If either pointer is in the initial buffer position then a bell is sounded.

(12) COPY$ON - Searches the old buffer from the current position of the old buffer pointer until the first occurrence of the character passed to this procedure. If no match occurs, a bell is sounded. Otherwise, the characters from the current position of the old buffer pointer until the identified character are copied to old buffer pointer up to the identified character are copied to the new buffer and displayed on the screen.

(13) DELETE - This procedure searches the old buffer for the first occurrence of the identified character. If this character is not found, a bell is sounded. Otherwise, all characters are deleted up until the first occurrence of the specified character and a "%" character is displayed in the position of each deleted character. The old buffer is finally compressed to remove the empty spaces resulting from the deletions.

(14) DEL$N - Resets the old buffer pointer and the new buffer pointer to zero and displays the end of file character (back slant) on the console CRT.

(15) DISPLAYRMO$N - This procedure will print blanks up until the current position of the old buffer pointer and then will display the remaining characters in the old buffer. The cursor moves to the beginning of the next line on the CRT and the contents of the new buffer are displayed.

(16) DEL$O - The characters from the current position of the old buffer pointer up to the end of the old buffer are deleted. Percentage signs are displayed in place of the deleted characters. If there are no characters in the old buffer, a bell is sounded.

(17) ESCAPE - This procedure turns off any special meaning of the character to follow and enters the character into the new buffer. This procedure may be used to override the action of extended line editor commands.

(18) PRINT$TAB - Enters the special tab character as the next character in the new buffer and displays the tab character on the console. If the tabulation will exceed the allowed size of the new buffer, a bell is sounded.

(19) LEDIT - Acts as an input interpreter while in the edit mode. Input strings are first checked for basic line editor commands, then for extended line editor commands. The appropriate editing routines are called in these cases. If a rubout character is entered, the new buffer pointer is backspaced. If none of the above special characters are encountered, input characters are echoed until the new buffer is filled. When this occurs, a bell is sounded.

(20) CONT$FILL - Stores the contents of the old buffer into the appropriate record in memory and moves the display cursor to the beginning of the next line.

3. Finish Module.

The Finish Module performs the necessary storing, writing and other file handling operations required when the data for a particular information category has been entered. If available memory has been consumed, the memory image of the edited data is written to the diskette and memory pointers are reset to allow further input. Upon encountering an end of code, end of error command or end of prompt delimiter, the contents of the old buffer are transferred to the appropriate record in memory. When an end of file delimiter is encountered, the necessary storage and file closing operations are accomplished.

a. Data Transfer.

(1) WRITE - Writes all records in memory out to the diskette, makes the appropriate directory entry, checks if the records were properly written to the diskette, resets the counter for the number of remaining records and resets the storage pointer to the appropriate memory address.

(2) INC$STORE - Increments the storage pointer and writes records in memory to the diskette if a memory overflow is possible.

(3) MOVE$STORE - Stores input information into the appropriate record in memory.

b. End of Segment.

(1) FILL$CODE$ZERO - Fills the remainder of the record currently in use with binary zeros.

(2) END$DL - When an end of code delimiter (!) is encountered in input, the next available record is allocated by a call to GO$NEXT$REC, the directory entry is made, the input information is stored in the record, the storage pointer is incremented and the end of code delimiter is appended to the input information. The remainder of the record is filled with binary zeros. The prompt message to input error commands is displayed.

(3) END$REC - When an end of prompt delimiter (↓) is enccuntered, the input prompt information is stored in the record in use, the storage pointer is incremented, the end of prompt character is written into the record and the prompt message for additional code information is displayed.

(4) END$ERR - When an end of error command delimiter (¬) is encountered, the error commands are stored in the record allocated, the storage pointer is incremented, the end of error command character is written into the record and the prompt message for prompting information is displayed.

(5) END$F - Called when the end of file character (back slant) is entered. Any input information is stored in the record in use followed by the end of file delimiter. The storage pointer is incremented, the records in memory are written to the diskette, the record map pointer is incremented and the end of code character (!) is

placed following the last entry in the record map. The extent and record number of the next record to use are stored and the DAT file is closed.

V. RECOMMENDATIONS

The goal of this thesis was to present a method by which computer assistance in report generation could be achieved. The project development was implemented using hardware and software assets at the Naval Postgraduate School. While that equipment was suitable, small computers supporting commercially-available text processing systems are now available. It is envisioned that virtually any general purpose microcomputer could form the nucleus for the implementation of the Report Originating System in the fleet. This chapter discusses some of the known limitations of the Intellec-8 based Report Originating System.

A. SINGLE LINE INSERTIONS.

The editing facilities of the ROS program require separate insertion and filing of each additional line of data. The line editor would be more convenient for preparation of lengthy textual material such as instructions and directives if it provided the capability to insert multiple lines of text prior to the filing operation. Such a feature would prove particularly beneficial in the revision of lengthy textual material.

B. OPERATING SYSTEM AND PROGRAMMING LANGUAGE.

When this project was started, the PL/M programming language and the CP/M operating system were one of the more powerful and higher level combinations available for the 8080 family of microcomputers. Since then, 8080 based systems have become more widely used for general purpose computing and have influenced the development of more advanced software support for this architecture. One aspect in the evolution of these microcomputers has been the implementation of self hosted compilers for a variety of high level programming languages. A similar evolutionary process has occurred in the area of microcomputer operating systems. State of the art commercial software could enhance the Report Originating System in two ways. The use of a higher level language (compared to PL/M) could broaden the capabilities of the system by improving file handling and data manipulation. Additionally, implementation of a more popular language would simplify software maintenance efforts because it would not be necessary to train system development programmers in a new language. One candidate for a programming language would be the Extended Basic language. This language provides a good balance of programming potential and ease of use. Whatever choice is made, it is recommended that the compiler be hosted in the microcomputer to facilitate development and maintenance.

C. EXPANSION OF THE CREATE PROGRAM EDITOR.

The CREATE program currently does not provide the data structures capable of supporting the editing facilities of

the ROS program. As a result, the basic line editor commands in the CREATE program are limited. Incorporation of expanded data structures in CREATE could support a larger set of basic line editor commands equivalent to those found in the ROS program. The addition of such structures would require major revision of the CREATE program; it may be determined that such revision is unfeasible.

D. IMPLEMENTATION IN ANOTHER MICROCOMPUTER ARCHITECTURE.

Although the 8080 family of microcomputers enjoys great popularity in the commercial world, this architecture is not common in the military. Implementation of the Report Originating System in another microcomputer architecture may enhance the system's appeal in the Navy. The system could be recoded to be implemented in existing military computers such as the AN/UYK-7, AN/UYK-14 and AN/UYK-20. New computer architectures offered by Digital Equipment Corporation (LSI-11) and by Texas Instruments (TI-9900) should be investigated as possible candidates. Another area of available hardware that should be investigated is the blossoming arena of word processing equipment [10]. This has become a very dynamic and competitive area in data processing development with new products being announced constantly. One of these off-the-shelf systems could accommodate the Report Originating System. One projection is that by 1986 word processing system hardware will be available for $150, plus the cost of a typewriter [1].

E. ERROR ANALYSIS EXPANSION.

Additicnal error checking capabilities could be easily developed and incorporated in the system. The ability to check special number and character sequences such as Date-Time Groups, Social Security Account Numbers and National Stock Numbers would open the areas of application of the system. For example, Date-Time Groups could be checked to see if the date is compatible with the month and that the time is a valid 24-hour clock time. Social Security Account Numbers could be checked for the appropriate number of digits irrespective of whether hyphens are incorporated in the format. A pattern for valid National Stock Numbers could assist in error detection in supply documents. The error analysis capability could be broadened to allow the user of the CREATE program to define his own error analysis commands for special applications. Statistical information concerning the most prevalent types of errors in formatted reports should be gathered as a guide to future development of standard error analysis.

F. TESTING AND DEBUGGING.

Before the Report Originating System could be implemented outside a laboratory environment, extensive testing and debugging efforts must be accomplished. It is inevitable that programming inconsistencies exist hidden in ROS and CREATE. These need to be detected by an extensive testing program.

VI. CONCLUSIONS

Although this study has focused on a single application of computer assistance to the shipboard manager, many management areas besides report generation could benefit from data processing assistance. In the area of training, a general purpose microcomputer system could be used to uniformly administer training courses and to automate the maintenance of individual training records. Such a system could also enhance record keeping functions in personnel administration and supply accounting. Computation of navigational equations to determine ship's position could be performed quickly and accurately by a general purpose computer. All of these potential applications point to the desirability for a non-tactical computer system aboard naval vessels.

There has been a trend in the military to automate report handling at the higher levels of command. Format free reports are being replaced by highly formatted, machine processable reports. To be useful, these reports must be checked for inconsistencies in content and form before being processed by the recipient. The optimal time for such error checking is before the report is released by the originator. The Report Originating System provides the framework that can reduce errors during report generation by incorporating precise prompting questions and can perform error analysis to identify format irregularities. This latter capability may be readily expanded to include a variety of analyses.

During the development of the Report Originating System, an attempt was made to make the system easy to use by an

operator with no prior computer experience. Simplicity of operation is the keystone to such a system's success; the most ingenious data processing system will fail if it is not easier to use than the current manual method. Development of any such system must adhere to this basic tenet.

The rapid technological progress in the area of microcomputers and the commensurate reduction in microcomputer hardware system cost has made Navy-wide implementation of such a system a realizable goal. Since the preponderance of computer hardware in the fleet is dedicated to specialized tactical systems and since the hardware associated with the Report Originating System is modest in cost, it is desirable that the system, if implemented, be based on a standard microcomputer architecture which could support a variety of other computational and information management systems. The use of standard architecture and of an easy to use high level programming language will facilitate program development and system software maintenance.

In summary, the Report Originating System advocated by this thesis demonstrates that computer assistance in information processing and management is readily supported by currently available computer systems and can be implemented at a modest cost. Such a tool could reduce the administrative burdens of the shipboard manager and free him to develop his nautical and naval warfare specialty skills.

APPENDIX A

ROS PROGRAM

User's Manual

1. Introduction.

The primary tool used by the message originator in the fleet to prepare the required report will be the ROS program. It is through the ROS program that the user interfaces with the data stored within the microcomputer system and the ROS program provides assistance to the user in the actual generation of the report text. In order to generate reports, there is little need for the user to understand the details of the program logic; instead he merely need know how to use this basic tool to format the desired report. Therefore, it is the purpose of this manual to provide the ultimate user with a step by step process by which he can use the ROS program. This manual describes the process implemented on the Intellec-8 microcomputer system.

2. Setting Up the System.

To use the ROS program to generate a report, the user must have available a DAT file for that particular report. Under normal circumstances, the authority requiring a specific report will provide all users with a diskette containing the necessary DAT file. It is incumbent on this authority to maintain the distributed DAT files so that current revisions to reporting procedures will always be available to the general user.

The fleet user will select the appropriate DAT file, insert the diskette into the drive mechanism and initiate the system by typing

ROS <filename> <cr>

where <filename> is the name of the report to be generated, such as FORSTAT or MOVREP, and <cr> denotes a carriage return. It is important that the <filename> be exactly the same as the name given to the DAT file by the originator. The system will now be ready to execute the report generation.

3. Execution.

After calling the desired DAT file by means of the above convention, the appropriate file will be opened, if in fact it exists on that diskette. If a file by the name of <filename> does not exist on that diskette, then an error message, "DAT FILE NOT PRESENT", will be displayed. This indicates to the user that either the wrong diskette is in use or that no DAT file for that specific report name exists and that the program CREATE should be used to create a DAT file. Creation of a DAT file is explained in the CREATE program user's manual, Appendix C.

When the appropriate DAT file has been opened, the system will respond by displaying the data labels contained within the file. Data labels are names given to a section of a report which allow the user to directly access that section. A data label usually represents an information element of the required report. For example, in a NAVFORSTAT, there are many data labels such as COMDR which represents information concerning the commanding officer and PERSN which represents information concerning personnel. The user will respond to the display of the data label by typing either a "Y" or "N" under the label; "Y" indicates

the user wishes to modify the information under that label, "N" indicates that data label should not be modified. Next the system will query which data labels are to be included in the report generated. Again the user will type a "Y" or "N" under the data label to indicate inclusion or exclusion, respectively. This feature is added since many formatted reports require inclusion of only changed data elements for most reports but require periodic submission of all data elements to verify the data base.

The next system query will concern whether prompting is desired. An affirmitive response (Y) by the user will cause entry into the fully instructional mode where the user is prompted by questions, the answers to which will be the required report information. A negative response (N) bypasses the prompting questions and allows the user to directly edit the information associated with the data label. This optional prompting provides instructional assistance to the unfamiliar user but does not encumber the experienced user with the slowness of the prompted mode.

After these preparatory questions have been answered, the ROS program is ready to formulate the desired report. As a basis, the program uses the information generated and modified in previous reports; this information is updated by the operator by using the facilities of either the basic line editor or the extended line editor, both explained in the Line Editor User's Manual, Appendix B. Both of the editors have the capability to perform the necessary data modification and the final choice of line editors is fundamentally operator preference.

If the user has opted for the instructional mode of operation, specific questions concerning each of the fields for the selected data elements will be written on the console's CRT screen. It is intended that the command

authority requiring the report (the originator of the DAT file) make individual questions unambiguous so the operator can respond with the correct answer in the required format. If form as well as content is critically important for the report's usefulness, then it must be made perfectly obvious that the response to a particular question must, for example, be four digits while the response to another question may require a two letter input. Careful wording of the prompting questions will allow the operator to respond rapidly and accurately.

While the operator enters the information sought, the ROS program analyses the entered data for errors. This analysis may be as simple as checking whether the response was alphabetic or numeric or may be so complex as to check for the reasonability of the answer. For example, it may suffice to check that the answer is "10" rather than "ten"; on the other hand, the operator may need to be cautioned after entering information that shows the number of assigned personnel increased 500 percent since the last report. If the error analysis indicates an error has occurred, a diagnostic message will be displayed to the operator who will then have the opportunity to make an appropriate correction. In the case of a question of reasonability, the second response will be considered authentic. When working in the prompted mode with the selected data element, most of the line editor's commands are implied. After receiving a response from the operator, the system will make the appropriate modifications to the data after the responses have been received for all queries concerning that particular data element. The newly submitted information is retained in a buffer until a command is given to update the old data. The operation of this process is explained more fully in the Line Editor User's Manual. When the information regarding a selected data element has been properly updated, the editing process will continue with the

next selected data element. The sequence of instructional questions, response, error analysis and information updating will continue for each data element selected. After all of these data elements have been processed, the DAT file will be updated to reflect the current information and a message file, which can be referenced by <filename>.MSG, will be created on the diskette. The message file will contain all the required information in the appropriate format for submission to the requesting organization.

The alternate method of ROS program execution is the non-instructional mode. This mode is intended for the experienced operator who understands the reporting requirements and who desires to rapidly edit the report information. The non-instructional mode is also prefaced by the system queries about data elements. During the editing process, the existing information for each data element will be sequentially displayed and directly edited with the line editor's facilities. The significant difference between this mode and the formerly described mode is the omission of prompting questions. This mode also provides the error analysis feature provided by the fully prompted mode. If errors are made, the same diagnostic messages will be displayed to the operator. It is assumed that the operator is intimately familiar with the proper format. In the non-instructional mode the editing process is explicit and not implied. Regardless of which line editor is in use, the operator must issue the necessary editing commands to modify and file the desired data and to quit the edit mode. After changes have been filed for a particular data element, the next selected element may be accessed by using the "NEXT" command (&NX). Again, input information will undergo the same degree of error analysis. As data element responses are filed, the next selected data element will be processed.

Upon completion of the updating process, the DAT file will reflect the latest changes of information and a message file will have been created.

A report origination session is shown below. Both the fully instructional and non-instructional modes are depicted to contrast their use. Only the basic line editor's facilities have been exercised in each case. Explanation of the action of the basic line editor's commands may be found in the Line Editor User's Manual.

4. Sample Report Originating Session.

The following sessions represent the origination of a typical Naval Force Status (NAVFORSTAT) report. The first example shows the updating process using the instructional mode of operation. The second example implements the same changes but uses the non-instructional mode available in the ROS program. In both examples, the basic line editor facility is used. Lines displayed by the ROS program are bracketed by asterisks (*); lines input by the operator are ended with a carriage return (<cr>). Amplifying comments are parenthesized and follow the line they pertain to.

A. Instructional Mode.

A> (System is ready)

ROS FORSTAT <cr> (FORSTAT is name of DAT file)

SELECT DATA ELEMENTS YOU DESIRE TO WORK WITH

*HEADER IDENT TSKCD OPCON ADCON *

Y Y Y N N<cr>

*HOGEO CCMDR ACTIV PERSN NE *

N N Y Y<cr>

(The system would continue to display the remainder of the elements in the system, expecting "Y" or "N" responses for each.)

SELECT DATA LABELS TO BE INCLUDED IN THE REPORT

*HEADER IDENT TSKCD OPCON ADCON *

Y Y Y N N <cr>

*HCGEO CCMDR ACTIV PERSN NE *

N N Y Y<cr>

(Again the system would continue to display the remaining elements)

DO YOU WISH TO BE PROMPTED?

Y <cr>

FM USS NEVERSAIL (First line of HEADER)

&NX<cr> (Go to next line.)

(Refer to Line Editor User's Manual for command meanings)

TO CINCPACFLT MAKALAPA HI

&NX<cr>

COMNAVSURFPAC SAN DIEGO CA

&NX<cr>

CTU 77.7.7

&IN CTU 76.1.2<cr> (Add another addressee)

<CTU 76.1.2>

&FI<cr> (Files the insertion)

CTU 76.1.2

&QU<cr> (Finished with changes to HEADER)

NAVFORSTAT 012 AS OF 012345Z MAY 77

(First line of IDENT section)

&CH /012/014/<cr>

ERROR: NUMBER IS NOT IN SEQUENCE. RE-ENTER CHANGE

&CH /012/013/ <cr> (Increment report number)

<NAVFORSTAT 013 AS OF 012345Z MAY 77>

&CH /012345/100200/<cr> (Now change the DTG)

<NAVFORSTAT 013 AS OF 100200Z MAY 77>

&FI <cr> (File the change)

NAVFORSTAT 013 AS OF 100200Z MAY 77

&QU<cr> (Finished with changes to IDENT)

WHO IS YOUR TASK ORGANIZATION COMMANDER?

RESPOND IN THE FORM CTA NNN.N.N

CTU 076.1.2<cr> (TSKCD data has been changed)

WHAT IS YOUR CURRENT EMPLOYMENT?

RESPOND WITH 2 LETTER CODE

OP<cr> (ACTIV data has been changed)

NAVY ENLISTED PERSONNEL-RESPOND WITH 4 DIGITS

STRUCTURED STRENGTH?

0240<cr>

AUTHORIZED STRENGTH?

0221<cr>

ASSIGNED STRENGTH?

0198<cr>

(This is the last of the selected data elements)

A>

(The above changes have been made, a message file has been created and the system is ready.)

B. Non-instructional Mode.

A> (System is ready)

ROS FORSTAT <cr> (FORSTAT is name of DAT file)

SELECT DATA ELEMENTS YOU DESIRE TO WORK WITH

*HEADER IDENT TSKCD OPCON ADCON *

Y Y Y N N<cr>

*HOGEO COMDR ACTIV PERSN NE *

N N Y Y<cr>

SELECT DATA LABELS TO BE INCLUDED IN THE REPORT

*HEADER IDENT TSKCD OPCON ADCON *

Y Y Y N N <cr>

*HOGEO COMDR ACTIV PERSN NE *

N N Y Y<cr>

(Again the system would continue to display the remaining elements)

DO YOU WISH TO BE PROMPTED?

N <cr>

FM USS NEVERSAIL (First line of HEADER)

&NX<cr>

TO CINCPACFLT MAKALAPA HI

&NX<cr>

COMNAVSURFPAC SAN DIEGO CA

&NX<cr>

CTU 77.7.7

&IN CTU 76.1.2<cr> (Add this addresse)

<CTU 76.1.2>

&FI<cr> (File the insertion)

CTU 76.1.2

&QU<cr> (Finished with changes to HEADER)

NAVFORSTAT 012 AS OF 012345Z MAY 77

(First line of IDENT section)

&CH /012/013/ <cr> (Increment the report number)

<NAVFORSTAT 013 AS OF 012345Z MAY 77>

&CH /012345/100200/<cr> (Change the DTG)

<NAVFORSTAT 013 AS OF 100200Z MAY 77>

&FI <cr> (File the change)

NAVFORSTAT 013 AS OF 100200Z MAY 77

&QU<cr> (Finished with changes to IDENT)

&NX<cr> (Want next data element)

TSKCD CTU C77.7.7 (First line of next data element)

&CH /077.7.7/076.1.2/<cr>

<TSKCD 076.1.2>

&FI<cr>

TSKCD CTU 076.1.2

&IN TSKCD CTU 077.7.7/DELETE<cr>

<TSKCD CTU 077.7.7/DELETE>

&FI<cr>

TSKCD CTU 077.7.7/DELETE

&NX<cr>

ACTIV IP

&CH /IP/OP/<cr>

<ACTIV OP>

&FI<cr>

ACTIV OP

&NX<cr>

PERSN NE/0240/0221/0195

&CH /0195/0198/<cr>

<PERSN NE/0240/0221/0198>

&FI<cr>

PERSN NE/0240/0221/0198

&QU<cr>

(Editing session is terminated and appropriate files created)

A> (System is ready for new task)

The resulting message file would look like this for both examples:

```
FM USS NEVERSAIL
TO CINCPACFLT MAKALAPA HI
COMNAVSURFPAC SAN DIEGO CA
CTU 77.7.7
CTU 76.1.2
BT
NAVFORSTAT 013 AS OF 100200Z MAY 77
FF 1234 NEVERSAIL/N09999
PART I
TSKCD CTU 076.1.2
TSKCD CTU 077.7.7/DELETE
ACTIV OP
PERSN NE/0240/0221/0198
BT
```

SUMMARY FOR ROS USERS

1. Turn on power to the microcomputer, diskette drive and terminal. (Diskette out!!)

2. Select proper diskette and insert into "A" slot.

3. Load the computer program by depressing the "RESET" switch. An "A>" should appear on the terminal screen.

4. Confirm that the appropriate DAT file is present on the diskette in use by typing "DIR <filename>.DAT <cr>". If so, "<filename>.DAT" will appear on the screen. If not, repeat Steps 2 through 4 with other available diskettes until the DAT file is found. If the DAT file does not exist, the CREATE program may be used to produce a DAT file.

5. Remove previcus report's MSG file by typing "ERA <filename>.MSG <cr>". Alternatively, if a back-up file is desired, type "ERA <filename>.BAK <cr>" to erase the previous back-up file followed by "REN <filename>.BAK=<filename>.MSG <cr>" to rename the MSG file as a BAK file.

6. Execute ROS program by typing "ROS <filename> <cr>". Answer system queries and select data elements. Make appropriate modifications to data elements as they appear using the line editor facilities.

7. When processing of the selected data elements is complete, "A>" will appear on the screen.

8. The MSG file may be viewed by typing "TYPE <filename>.MSG <cr>".

9. If additional corrections must be made, repeat Steps 5 through 8.

APPENDIX B

LINE EDITOR

User's Manual

1. Introduction.

ROS and CREATE users may create and change the contents of the ROS and CREATE files by using the features of the "line editor".

The LEDIT procedure functions with two lines of information, the "old line" containing the data as it exists in the file and the "new line" acting as a recepticle for additional or modified data. Data may be manipulated between the two lines by using a series of basic editing commands which are described below. In order to provide an editing capability which would prove useable by an operator of limited experience, the basic line editor's commands are succinct and limited in number. It is hoped that this philosophy will allow the ultimate user in the fleet to rapidly master the basic line editor's capabilities and to use the basic line editor without reference to a command list or user's manual. An expanded line editor is also available within the LEDIT framework which allows greater control over the line editing process. This expanded editor is intended for the more experienced user or programmer who requires this greater capability. Either the basic or the expanded line editor may be invoked by the user without any change to the ROS or CREATE programs. Discrimination between the two editors is by means of command format; each

editor has its distinct command set. These commands are structured in such a way that there should be no possibility of accidentally invoking a line editor command.

The line editor uses a series of commands, composed of an ampersand (&) followed by two characters, and possibly by amplifying information in an "argument list". The end of the line editor command sequence is denoted by a carriage return. The commands operate on the "old line" contents which are the last line displayed at the console. The resultant changes are held in the "new line" area until an explicit command is given to replace the "old line" contents with the "new line" contents. This requirement should prevent inadvertent modification of existing data. The following section enumerates the commands available to the user and defines the action they cause.

2. Basic Line Editor Commands.

The following commands are used with the basic line editor:

COMMAND	PARAMETERS	DESCRIPTION
&PR	<numeral>	"PRINT" Displays the number of lines indicated by numeral. After displaying lines, the contents of the "old line" are the last line displayed. If no parameter is present, one line will be displayed.

COMMAND	PARAMETERS	DESCRIPTION
&NX	<numeral>	"NEXT" Displays the ith line following the current line where i is equal to the numeral indicated. Only one line will be displayed which will be the contents of the "old line". If no numeral is specified, the next line will be displayed.
&UP	<numeral>	"UP" Displays the ith line before the current line where i is equal to the numeral indicated. Only one line will be displayed which will be the contents of the "old line". If no numeral is specified, the previous line will be displayed.
&CH	/<S1>/<S2>/	"CHANGE" Copies all characters from "old line" to "new line" until the first occurrence of string 1; string 2 is substituted in the "new line" in place of string 1 and the remaining characters of the "old line" are copied into the "new line". Contents of the "new line" are displayed between broken brackets (<>). String 2 may be a null string indicated by consecutive virgules (//).

COMMAND	PARAMETERS	DESCRIPTION
&DE	NONE	"DELETE" Deletes the contents of the "old line".
&IN	<string>	"INSERT" Inserts the string indicated after the current line but before the next line. Insert mode is terminated by a carriage return. Added lines are displayed between broken brackets (<>).
&FI	NONE	"FILE" Replaces the contents of "old line" with the contents of "new line" and clears "new line".
&QU	NONE	"QUIT" Terminates the edit session. Will not incorporate changes unless a previous file command has been given.
&PO	NONE	"PROMPT ON" Turns the prompt mode on.
&NP	NONE	"NO PROMPT" Turns prompt mode off.

3. Use of Basic Line Editor Commands.

In order for a command to be recognized it is necessary that the ampersand (&) be in the first input character position. The alphabetical characters of the command must immediately follow the ampersand with no spaces between them. A blank space between the command and any parameters

is necessary; a single carriage return denotes the end of a command and its associated string.

Whenever a command other than those listed above is used, an error condition will exist and the message "INVALID LINE EDITOR COMMAND" will be displayed. The LEDIT procedure will then await a corrected command input for a change or for termination of the input session.

An example of a line editing session is shown below to demonstrate use of the LEDIT commands. For the purpose of illustration, all lines displayed by the LEDIT procedure are bracketed by asterisks (*); for user input lines, a carriage return is shown as <cr>.

In this example, a textual file is used to demonstrate the basic line editor.

```
&PR 5<cr> (Prints 5 lines of the file)
*THIS FILE CONTAINS TEXTUAL MATERIAL TO*
*DEMONSTRATE THE USE OF THE LINE*
*EDITOR'S OPERATION.  THE OPERATOR CAN USE*
*THESE EDITORS TO MODIFY FILES ONE*
*LINE AT A TIME.  CHANGES MUST BE*
&NX<cr> (Prints the next line)
*FILED FOR EACH LINE SEPARATELY IF THEY*
&UP 3<cr> (Backs up three lines)
*THESE EDITORS TO MODIFY FILES ONE*
&CH /ONE/A SINGLE/<cr>
*<THESE EDITORS TO MODIFY FILES A SINGLE>*
```

&FI<cr> (Files the change)

&PR<cr> (Print out current line)

THESE EDITORS TO MODIFY FILES A SINGLE

&NX 4<cr> (Goes down four lines)

ARE TO BE INCLUDED IN THE FINAL FILE.

&CH /IN/ON/<cr> (Change the first occurrence of "IN")

<ARE TO BE CNCLUDED IN THE FINAL FILE.>

&NX<cr> (Gets next line, above change not filed)

THIS IS AN EXTRANEOUS LINE

&DE<cr> (Remove current line)

&FI<cr> (File the deletion)

&UP<cr> (Assumes up one line)

ARE TO BE INCLUDED IN THE FINAL FILE.

&IN THIS IS AN ADDITIONAL LINE OF TEXT<cr>

<THIS IS AN ADDITIONAL LINE OF TEXT>

&FI<cr> (Files the insertion)

&UP<cr>

ARE TO BE INCLUDED IN THE FINAL FILE.

&PR 2<cr>

ARE TO BE INCLUDED IN THE FINAL FILE.

THIS IS AN ADDITIONAL LINE OF TEXT

&QU<cr> (Ends the edit session)

It should be kept in mind that all modifications to lines affect only the "new line" area until a "FILE" command (&FI) is given; the file command causes modification of the "old line" area which represents the contents of the resulting file. This convention provides the user with the ability to make several temporary modifications to a line before making these modifications to the original file. By requiring the user to give an explicit "FILE" command, he has the opportunity to see the effect of the modification before it is entered; this will reduce erroneous modification of good files. The operator must also remember that the "QUIT" command (&QU) merely terminates the edit session and that any desired changes must be filed prior to ending the session.

4. The Extended Line Editor.

The extended line editor uses the same "old" and "new lines" as the basic line editor but allows more explicit control over the editing operation by invoking commands from a different command set. Extended line editor commands consist of a control character, i.e. simultaneously depressing the CTRL key and an alphabetical character key on the console keyboard. Since twenty-four different commands exist and the distinction between similar commands is very subtle, it is recommended that the basic line editor be used whenever possible. Table 1 summarizes the commands available in the extended line editor and describes the action these commands cause. Because of the cumulative complexity of these commands, an example session is not given. Potential users of the extended line editor are encouraged to practice using the various commands on a dummy file in order to gain an understanding of the ramifications of each command. Particularly useful members of this editor are the control characters A, B, D, E, F and W.

CONTROL CHARACTER	DEFINITION
A	Acts as a backspace and rub-out command on the new line only. (Same as the rub-out on many terminals).
B	Replace the "old line" with the contents of the "new line", empties the "new line". (Equivalent to &FI).
C	Copy one character from the "old line" to the "new line".
D	Copy the remaining characters from the "old line" to the "new line" echoing each character. Equivalent to &FI followed by &QU.
E	Toggle the insert mode. Begin insert prints "<", end insert prints ">". Position of the old pointer does not change during insert.
F	Delete the new line without updating the old line, then "quit".
G	Display contents of the "old" and "new lines" with control characters interpreted.
H	Copy remaining characters from the "old line" to "new line", echoing each. Wait for additional edit commands.
I	Tab. A tab stop is defined for every four characters. Same as TAB on many terminals.

CONTROL CHARACTER	DEFINITION
J	Line feed. Editing of that line is ceased.
K	(not used)
L	Copy the remaining characters of "old line" to "new line" without echoing. Wait for additional commands.
M	Carriage return. Editing of that line is ceased.
N	Backspace "old line" and "new line" one space.
O	Copy characters from the current position of old pointer to the next character typed.
P	Delete characters from the current position of old pointer to the next occurrence of the next character typed. Echoes a "%" for each character deleted. (The %'s do not become part of the new line).
Q	Delete the "new line" and reset the old pointer to the start of the "old line".
R	Display the remaining contents of the "old line" and all of the "new line".
S	Delete one character from the "old line". Echo a "%" for the deleted character.
T	Only used in CREATE to transmit information in "new line" to storage in memory. Used when input from the console exceeds one CRT line.

CONTROL CHARACTER	DEFINITION
U	Copy characters from the "old line" to the "new line" up to the next TAB character.
V	Escape character. Turns off any special meaning of the character which follows. enters the character into the "new line" and echoes the character (e.g. "CTRL V CTRL M" will echo ↑M).
W	Backspace up to the beginning of the last word.
X	Deletes characters from the current position of "old line" through next character typed.
Y	Copy the remaining characters from the "old line" to the "new line" echoing each, replace the old buffer with the "new line". Wait for additional commands.
Z	Copy characters from the "old line" to the "new line" through the next occurrence of the next character typed.

NOTE: The control character is entered by depressing the CTRL key and then simultaneously depressing the desired function key.

TABLE 1

EXTENDED LINE EDITOR FEATURES

APPENDIX C

CREATE PROGRAM

User's Manual

1. Introduction.

The CREATE program is designed to allow the operator to create a data base for a specific report. The CREATE program is the primary tool used by the report requiring authority to produce the DAT files that are supplied to the ultimate fleet user. The same CREATE program is also available to the shipboard user to produce DAT files for reports that are currently not supported by a distributed DAT file. This situation could arise if a new reporting requirement is levied on the fleet unit and the requiring authority has not distributed the appropriate DAT file or if some damage has been inflicted upon the diskette containing the required DAT file. It is the guiding intention of this report generating system that the ultimate user be provided with this capability as an interim measure only; the responsibility to provide and maintain DAT files properly lies with the authority requiring the report.

A DAT file is comprised of three different categories of information: the report data base as represented in the last submitted report; the prompting questions to which the operator responds in order to update the report data base; and the error analysis directives which are used by the ROS program to check input data. The report data base contains not only the information associated with each of the data labels but also incorporates the required message header

information and perhaps some trailing information that may be required for some message reports. Thus, using a NAVFORSTAT for an example, the data base would contain the addressees in the naval message format, the subject and Standard Subject Identification Classification (SSIC) lines, the PART I information, i.e. the data labels and their associated data, and PART II and PART III narrative information as required.

A DAT file can be subdivided into lines each of which can contain up to 128 characters or symbols and it may consist of as many lines as are necessary to represent the coded information, the error analysis directives and the prompting questions. Each of these categories will consist of at least one line. Therefore, as a minimum, a data element will consume at least three lines. If the report material is primarily textual and self-explanatory, the error analysis and prompting sections may be left empty. This is accomplished by entering the appropriate record delimeter as explained in Section 3.

2. Setting Up the System.

In order to originate a DAT file for a required report, the user locates a diskette with ample space remaining which contains a copy of the CREATE program. Auxillary diskettes, containing the necessary system programs and the ROS and CREATE programs but without specific DAT files will be provided as part of the initial equipment allowance or may be obtained from higher authority. Each diskette contains sufficient space to hold several DAT and MSG files for different reports. Since the shipboard user is creating a DAT file only as an interim measure until the appropriate diskette is provided by the report requiring authority, the interim DAT file may co-occupy a diskette with another DAT file and then may be erased after the officially distributed DAT file is received.

After the diskette is inserted in the drive mechanism, the CREATE program is invoked by typing

CREATE <filename> <cr>

where <filename> consists of eight characters, or less, and is the name of the DAT file to be created. This <filename> should have mnemonic value so that it can be easily recalled or identified by the user when originating a report. For example, the <filename>, "FORSTAT" would be appropriate for the Naval Forces Status Report (NAVFORSTAT). (Note that "NAVFORSTAT" was not used because it exceeds the eight character limitation on <filename>.)

3. Execution.

Upon receipt of the CREATE command, the system will be ready to commence building the DAT file and will respond with the prompting message, "EXPECTING CODE INFORMATION". This is a request that the user type in the appropriate data label followed by the information to be associated with that data label. To identify the data label, it should be bracketed by a pair of one of the following specific symbols; bracketing the data label with exclamation points (!) indicates that the label itself is to be included in the final report while bracketing with percent signs (%) indicates the label is not to be included in the report. Any information following the bracketed labels is considered to be part of the data base information for that data element. After all the appropriate coded information has been entered, the end of the coded information section is marked by inputting the bar character (|).

Now the system will respond by displaying the prompt, "EXPECTING ERROR COMMANDS". Error commands are applicable to individual fields within the coded information line. For example, fields in the NAVFORSTAT data element, COMDR, are

denoted by virgules (/) separating the fields. The data line "ACTIV IP" contains one field whose contents are "IP"; the data line "COMDR CDR/JONES J.P./000111-10" contains three fields containing "CDR", "JONES J.P" and "000111-10" respectively. Notice that the data label is not considered to be a field. Data labels may be analyzed in different ways by either checking their content or by comparing the revised information to what was previously reported. The specific error commands are:

ALPHA	Checks the field to ensure all characters are alphabetic.
NUM	Checks the field to ensure all characters are numerals.
PCT<number>	Checks the field to see if it is within <number> percent of the previously reported value. The <number> should be two digits between "01" and "99", inclusive.
SEQ	Checks the field to see if it is sequential to the last reported value.
LNG	Checks the field to see if it contains the same number of digits as the previously reported value.

Error commands are separated using the same delimiter as the coded information. If a report contains defined fields separated by virgules, the fields for error analysis will be defined using virgules. If no defined fields exist in the report then a field may be denoted by using braces ({}) as delimiters in both the coded information section and the error command section. These braces will not appear in the final report format. If a specific field of a data element is not to be checked, then two consecutive delimiters are entered, indicating no error command. If error analysis for one field of a data element is desired, all fields for that

particular data element must have error analysis, i.e. either one of the five commands above or the null command. The end of the error analysis section is marked by the tilde (¬) character. If no error analysis is desired for a data element, a solitary tilde should be the response to the prompt message. Thus, for the data element, "COMDR" previously shown, an error command response might be

//ALPHA/NUM¬

which would be interpreted as meaning no error analysis is desired for the field containing "CDR"; the characters in the name field should be alaphabetics; and the characters in the final field should be numerals.

The system will now respond by displaying, "EXPECTING PROMPT INFORMATION". Prompting information consists of questions and explanatory statements which are to be displayed to the user during report origination. This prompting information should be concise and unambiguous. By reading the prompting information, the user should know specifically what information is being sought and in what format it should be entered. Again there is a correspondance between data element fields and prompting information fields. During ROS program execution, prompting questions are displayed individually for each field of the data element with responses to each question being entered before continuing on to the next question. Unlike the error analysis section, each field of the data element should have an appropriate prompting question; if it is decided that a data element need no prompting questions, then a null prompting area should be entered. However, it is strongly reccmmended that every data element have associated prompting questions, even if the questions appear obvious. The end of the prompting area is denoted by entering an up-arrow (↑).

The process will continue to seek code information, error analysis information and prompting questions until the

end of file is entered. This is accomplished by typing a back slant character in response to the "EXPECTING CODE INFORMATION" query. At this time, execution of the CREATE program ceases and a DAT file exists. This file may be referenced by <filename>.DAT and can be utilized by the ROS program as a data base.

It is recommended that the DAT files produced be as comprehensive as possible to ensure that they will be useful for all possible situations. When working with a highly formatted report such as NAVFORSTAT, all data labels that could possibly apply to a unit should be included in the DAT file; if, at the time of creation, a particular data label is not pertinent, the label itself may be included in the file along with a null code area (the data label followed by a bar), the error analysis area and the prompting questions. If this is done, when that particular data element becomes applicable to the unit, the system need not be modified to include the information in the report. By using the report inclusion feature of the ROS program, as explained in Appendix A, the generated report content may be tailored by data element selection.

4. Sample CREATE Session.

An example of the beginning of a CREATE session for a NAVFORSTAT is shown below. Reiteration of the complete session would be excessively lengthy and redundent. It is hoped that this example will provide enough insight to DAT file creation to allow the ultimate user to produce a usable file after a short experimentation session. Lines displayed by the system are bracketed by asterisks (*); lines input by the operator are ended with a carriage return <cr> or do not start and end with asterisks. In these cases, it is not necessary for the operator to enter a carriage return.

A> (System is ready)

CREATE FORSTAT<cr> (FORSTAT is the name of the DAT file produced)

EXPECTING CCDE INFORMATION (Prompt message)

%HEADER% FM USS NEVERSAIL<cr>

TO CINCPACFLT MAKALAPA HI<cr>

COMNAVSURFPAC SAN DIEGO CA<cr>

CTU 77.7.7<cr>

CTU 76.1.2<cr>

BT<cr>

| (Indicates the end of the code area for HEADER)

EXPECTING ERROR COMMANDS

¬ (No errcr commands are appropriate for this data element)

EXPECTING PROMPT INFORMATION

LIST APPROPRIATE ACTION AND INFORMATION ADDRESSEES IN <cr> STANDARD NAVAL MESSAGE FORMAT ENDING WITH THE "BT" LINE.♠

EXPECTING CCDE INFORMATION

(Ready for the next data element)

%IDENT% NAVFORSTAT {000} AS OF {000000}Z JAN 00<cr>

FF 1234, NEVERSAIL/N09999 <cr>

PART I|

EXPECTING ERROR COMMANDS

{SEQ}{LNG}/¬

(Indicates that the report number field is tc be checked for sequence and the DTG field is to be checked for 6 digits.

All other fields have null commands)

EXPECTING PROMPT INFORMATION

MODIFY THE REPORT NUMBER AND "AS OF" DTG APPROPRIATELY↑

EXPECTING CODE INFORMATION

!TSKCD! CTU 077.7.7|

EXPECTING ERROR COMMANDS

¬ (No appropriate error commands)

EXPECTING PROMPT INFORMATION

WHO IS YOUR TASK ORGANIZATION COMMANDER?<cr>

RESPOND IN THE FORM CTA NNN.N.N↑

EXPECTING CODE INFORMATION

!OPCON! CTU 077.7.7|

EXPECTING ERROR COMMANDS

¬ (No appropriate error commands)

EXPECTING PROMPT INFORMATION

WHO IS YOUR OPERATIONAL COMMANDER?<cr>

RESPOND IN THE FORM CTA NNN.N.N↑

EXPECTING CODE INFORMATION

!HOGEO! SAN DIEGO CA|

EXPECTING ERROR COMMANDS

ALPHA¬ (Alphabetic characters only)

EXPECTING PROMPT INFORMATION

WHERE IS YOUR HOMEPORT?↑

EXPECTING CODE INFORMATION

!CCMDR! CDR/BATCH W.T./012345-60|

EXPECTING ERROR COMMANDS

ALPHA/ALPHA/NUM¬

(Alphabetic characters in the first

two fields, numeric in last)

EXPECTING PROMPT INFORMATION

WHAT IS YOUR COMMANDING OFFICER'S RANK?/HIS NAME? LAST NAME FOLLOWED BY INITIALS/HIS LINEAL NUMBER?↑

EXPECTING CODE INFORMATION

!ACTIV! IP|

EXPECTING ERROR COMMANDS

ALPHA¬

EXPECTING PROMPT INFORMATION

WHAT IS YOUR CURRENT EMPLOYMENT?↑

EXPECTING CODE INFORMATION

!XFERR! NC00C/000000/00-00.0N, 000-00.0E/000000|

EXPECTING ERRCR COMMANDS

LNG/LNG/LNG/LNG¬ (Checks all fields for proper number of characters)

EXPECTING PROMPT INFORMATION

TO WHAT ACTIVITY ARE YOU TRANSFERRING?/<cr>

WHEN DO YCU TRANSFER? YYMMDD/<cr>

WHERE DOES TRANSFER TAKE PLACE? XX-XX.XN, XXX-XX.XE/<cr>

WHEN DO YOU DEPART FOR TRANSFER POINT? YYMMDD↑

EXPECTING CODE INFORMATION

!ARRDT!| (Null code area)

EXPECTING ERROR COMMANDS

LNG¬

EXPECTING PROMPT INFORMATION

WHAT IS YOUR ARRIVAL DATE AFTER TRANSFER?<cr>

GIVE IN FORM YYMMDD↑

EXPECTING CODE INFORMATION

!FADDD! 1|

EXPECTING ERROR COMMANDS

NUM¬

EXPECTING PROMPT INFORMATION

WHAT IS YOUR FORCE ACTIVITY DIGIT?↑

EXPECTING CODE INFORMATION

!PERSN NE!/0240/0221/0195|

EXPECTING ERROR COMMANDS

/PCT10/PCT10/PCT10¬

(Check each field for 10% changes)

EXPECTING PROMPT INFORMATION

NAVY ENLISTED PERSONNEL-RESPOND WITH 4 DIGITS/<cr>

STRUCTURED STRENGTH?/AUTHORIZED STRENGTH?/<cr>

ASSIGNED STRENGTH?↑

EXPECTING CODE INFORMATION

(The CREATE program will continue to request the three categories of information. After all desired data labels have been included in the DAT file, a back slant should be entered in response to "EXPECTING CODE INFORMATION".)

Again, it is emphasized that prompting questions and error commands be as comprehensive and unambiguous as possible. If, before composing these questions, the drafter anticipates the questions of the ultimate user, a much better product will result. During execution of the CREATE program, the features of either line editor may be used. It should be remembered that modifications filed during the ROS program execution are incorporated not only in the MSG file but are also reflected in the code information area of the DAT file.

SUMMARY FOR CREATE USERS

1. Turn on power to the microcomputer, diskette drive and terminal. (Diskette out!!)

2. Select appropriate diskette and insert into the "A" slot.

3. Load the computer program by depressing the "RESET" switch. An "A>" should appear on the screen.

4. Confirm that the required DAT file is present on the diskette by typing "DIR *.DAT <cr>". This will list all DAT files present on that diskette. If necessary, repeat Steps 2 through 4 for all diskettes.

5. Execute CREATE program by typing "CREATE <filename> <cr>". Respond to the system queries until the last applicable data element has been entered, then close the file.

Delimiter symbols are:

| - End of data code area.
¬ - End of error command area.
↑ - End of prompt area.
\ - End of file.

6. After closing the file, "A>" will appear on the screen. Verify that the DAT file is present by typing "DIR <filename>.DAT <cr>".

7. Execute the ROS program if desired.

```
                    /* ****************
                    *                  *
                    *    ROS PROGRAM   *
                    *                  *
                    **************** */

  /* ************************************************

     A REPORT ORIGINATION SYSTEM DESIGNED FOR SHIPBOARD
    USE IN THE GENERATION OF REQUIRED RECURRING REPORTS.
    THE SYSTEM USES AS INPUT A DATA BASE (DAT) FILE AND
    PRODUCES AS OUTPUT A MESSAGE (MSG) FILE.  THE
    SOFTWARE SYSTEM CONSISTS OF TWO PROGRAMS: ROS AND
    CREATE.  CREATE IS USED TO CREATE A DAT FILE AND ROS
    IS USED TO UPDATE THE DAT FILE AND CREATE A MESSAGE.

       THE ROS PROGRAM IS MADE UP ON THE FOLLOWING MODULES:

             1.  OPERATING SYSTEM INTERFACE
             2.  INITIALIZE
             3.  EDITING
             4.  ERROR
             5.  OUTPUT

       THE CREATE PROGRAM IS MADE UP OF THE FOLLOWING
    MODULES:

             1.  INITIALIZE
             2.  INPUT-EDITING
             3.  FINISH

       BOTH PROGRAMS WERE DESIGNED FOR EXECUTION ON THE
    INTELLEC-8 MICROCOMPUTER SYSTEM, WITH CROSS COMPILATION
    BEING DONE ON AN IBM 360/65.

  ************************************************** */

100H:  /* PROGRAM TO BE LOADED INTO MEMORY STARTING HERE */

  /* ************************************************

        OPERATING SYSTEM INTERFACE DECLARATIONS.

  ************************************************** */

DECLARE
    LIT         LITERALLY       'LITERALLY',
    BOOT        LIT     '0',
    ENTRY       LIT     '0005H',  /* ENTRY POINT TO OS */
    TRUE        LIT     '1',
    FALSE       LIT     '0',
    FOREVER     LIT     'WHILE TRUE',
    CR          LIT     '0DH',
    LF          LIT     '0AH',
    CTI         LIT     '0',
    CTS         LIT     '1',
    DCNT            BYTE,
    BDOSA           ADDRESS INITIAL (0006H),
    SBDOS   BASED BDOSA  ADDRESS;

  /* ************************************************

    INITIALIZE DECLARATIONS

  ************************************************** */

DECLARE
    PROMPT  BYTE INITIAL(FALSE),
    RM          ADDRESS INITIAL(80H),
    RMPTR       BASED RM  BYTE,
    DLSLEN      BYTE INITIAL (0),
```

```
    NUM$REC    BYTE,
    WORK$MAP        ADDRESS,
    WMPTR      BASED WORK$MAP  BYTE,
    EXT$RN     BASED WORK$MAP  ADDRESS,
    NUM$ELEMENTS          BYTE,
    E$R$A                 ADDRESS,
    ER       BASED E$R$A   BYTE,
    NR$READ     BYTE,
    NR     BYTE,
    SAVE$EXT        BYTE,
    SAVE$RN         BYTE,
    DAT$AREA        ADDRESS,
    DAT BASED DAT$AREA  BYTE,
    BASE$DAT$AREA    ADDRESS,
    T$DAT$AREA       ADDRESS,
    CODE$A           ADDRESS,
    CODE BASED CODE$A    BYTE,
    B$CODE$A         ADDRESS,
    TOP$MEM          ADDRESS,
    MSG$AREA         ADDRESS INITIAL (80H),
    MSG  BASED MSG$AREA BYTE,
    SAV$EXT    BYTE,
    SAV$RN     BYTE,
    OLD$EXT    BYTE,
    CLD$RN     BYTE,
    MORE     BYTE INITIAL (FALSE),
    STOP BYTE INITIAL (FALSE),
    MODIFY BYTE INITIAL (TRUE),
    HOLD$WM     ADDRESS;

  /* ************************************************************

                    EDITING DECLARATIONS

  ************************************************************ */

DECLARE
    BUFFER (180)          BYTE,
    SIZE$NBUF LIT     '90',
    NEW$BUF               ADDRESS,
    NBUF   BASED NEW$BUF  BYTE,
    NPTR                  BYTE,
    OLD$BUF               ADDRESS,
    OBUF   BASED OLD$BUF  BYTE,
    OPTR                  BYTE,
    NB          ADDRESS,
    IN BASED NB BYTE,
    OB          ADDRESS,
    INSERT     BYTE INITIAL (FALSE),
    INSERTION BYTE INITIAL (FALSE),
    PERCENT    LIT          '25H',
    BS         LIT          '08H',    /* BACKSPACE  */
    BELL       LIT          '07H',
    TAB        LIT          '09H',
    EOP        LIT          '5EH', /* UP-ARROW; END OF PROMPT */
    EOC        LIT          '7CH', /*BAR; END OF CODE */
    ERR        LIT          '7EH', /* TILDE; END OF ERROR */
    CTLZ       LIT          '1AH',
    RUBOUT     LIT          '7FH',
    END$FILE   LIT          '5CH', /*BACK SLANT */
    BLANK      LIT   '20H',
    AMPERSAND  LIT   '26H',
    SLASH      LIT   '2FH',
    LBRACE     LIT   '7BH',
    RBRACE     LIT   '7DH',
    CHAR      BYTE,
    ED$BUF$PTR ADDRESS,
    (ED$BUF BASED ED$BUF$PTR) (100) BYTE,
    FREELIST$EXT BYTE INITIAL (0),
    FREELIST$RN  BYTE INITIAL (0),
    PROMPT$AREA      ADDRESS;
```

```
  /* ************************************************

                        ERROR DECLARATIONS

  ************************************************ */

DECLARE
    ERRA     ADDRESS,
    ECMD     BASED ERRA BYTE,
    BERRA    ADDRESS,
    WARN     BYTE INITIAL (FALSE);

  /* ************************************************

                        OUTPUT DECLARATIONS

  ************************************************ */

DECLARE
    DAT$FCB     ADDRESS INITIAL (5CH),
    DFCB    BASED DAT$FCB  BYTE,
    MSG$FCB(33)      BYTE,
    PRINT$LABEL    BYTE INITIAL (FALSE);


  /* ************************************************

                    OPERATING SYSTEM INTERFACE MODULE

       FUNCTION:  SERVES AS AN INTERFACE TO THE RESIDENT
    OPERATING SYSTEM.  IT ALLOWS INPUT/OUTPUT OPERATIONS
    TO BE HANDLED BY SYSTEM CALLS.

  ************************************************ */

CRTIN: PROCEDURE BYTE;
    DO WHILE INPUT(CTS);
    END;
    RETURN NOT INPUT(CTI) AND 07FH;
    END CRTIN;

READC: PROCEDURE BYTE;
    DECLARE C BYTE;
    IF (C:=CRTIN) >= 110$0001B  /* LOWER CASE A */
       AND C <= 0111$1010B /* LOWERCASE Z */ THEN
       C = C AND 101$1111B;  /* BECOMES UPPER CASE  */
    RETURN C;
    END READC;

MON1: PROCEDURE (FUNC, INFO);
    DECLARE FUNC BYTE, INFO ADDRESS;
    GO TO ENTRY;
    END MON1;

MON2: PROCEDURE (FUNC,INFO) BYTE;
    DECLARE FUNC BYTE, INFO ADDRESS;
    GO TO ENTRY;
    END MON2;

MON3: PROCEDURE (FUNC, INFO) ADDRESS;
     DECLARE FUNC BYTE, INFO ADDRESS;
    GO TO ENTRY;
    END MON3;

PRINTCHAR: PROCEDURE (B);
    DECLARE B BYTE;
    CALL MON1(2,B);
    END PRINTCHAR;

PRINTCHARI: PROCEDURE(C);
    DECLARE C BYTE;
    IF (C AND 0110$0000B) = 0 /* CONTROL CHAR */ THEN
```

```
        DO;
          CALL PRINTCHAR(EOP);
          CALL PRINTCHAR(C OR 40H);
        END;
      ELSE
      CALL PRINTCHAR(C);
      END PRINTCHARI;

CRLF: PROCEDURE;
    CALL PRINTCHAR(CR);
    CALL PRINTCHAR(LF);
     END CRLF;

PRINT: PROCEDURE (A);
     DECLARE A ADDRESS;
     CALL MON1(9,A);
     CALL CRLF;
     END PRINT;

SET$DMA: PROCEDURE(A);
     DECLARE A ADDRESS;
     CALL MON1(26,A);
     END SET$DMA;

DISKREAD: PROCEDURE(A) BYTE;
     DECLARE A ADDRESS;
     RETURN MON2(20,A);
     END DISKREAD;

DISKWRITE: PROCEDURE(A) BYTE;
     DECLARE A ADDRESS;
     RETURN MON2(21,A);
     END DISKWRITE;

OPEN: PROCEDURE(A) BYTE;
     DECLARE A ADDRESS;
     RETURN MON2(15,A);
     END OPEN;

CLOSE: PROCEDURE(A) BYTE;
     DECLARE A ADDRESS;
     RETURN MON2(16,A);
     END CLOSE;

SEARCH: PROCEDURE (FCB) BYTE;
     DECLARE FCB ADDRESS;
     RETURN MON2(17,FCB);
     END SEARCH;

MAKE: PROCEDURE (FCB) BYTE;
     DECLARE FCB ADDRESS;
     RETURN MON2(22,FCB);
     END MAKE;

LIFTHEAD: PROCEDURE;
     CALL MON1(12,0);
     END LIFTHEAD;

MOVE: PROCEDURE (SOURCE,DEST,N);
     DECLARE (SOURCE,DEST) ADDRESS,
       (S BASED SOURCE, D BASED DEST, N ) BYTE;
     DO WHILE (N:=N-1) <> 255;
        D=S; SOURCE=SOURCE+1; DEST=DEST+1;
     END;
     END MOVE;

ERROR: PROCEDURE(I);
     DECLARE I BYTE;
          DO CASE I;
           ; /* CASE 0 NULL ERROR STATEMENT */
          CALL PRINT(.'DISK READ ERROR $');
          CALL PRINT(.'ERROR COMMAND NOT DEFINED  $');
```

```
        CALL PRINT(.'A MESSAGE FILE EXISTS    $');
        CALL PRINT(.'DISK WRITE ERROR         $');
        CALL PRINT(.'OUT OF DIRECTORY SPACE   $');
        CALL PRINT(.'DAT FILE NOT PRESENT$');
        CALL PRINT(.'MSG FILE NOT PRESENT$');
    END;
    GO TO BOOT;
    END ERROR;

  /* ************************************************

                    INITIALIZE MODULE

       FUNCTIONS: TO OPEN THE APPROPRIATE DAT FILE, MAKE
    A MESSAGE FILE AND ALLOW THE USER TO SELECT A SET OF
    DATA ELEMENTS TO WORK WITH.  IT THEN INITIALIZES
    MEMORY WITH THE SELECTED DATA ELEMENTS.

  ************************************************ */


MAKE$MSG$FILE: PROCEDURE;
    CALL MOVE(.'MSG',.MSG$FCB + 9,3);
    MSG$FCB,MSG$FCB(12),MSG$FCB(32) = 0;
    IF SEARCH(.MSG$FCB) <> 255 THEN
        CALL ERROR(3);
    IF MAKE(.MSG$FCB) = 255 THEN
        CALL ERROR(5);
    IF OPEN(.MSG$FCB) = 255 THEN
        CALL ERROR(7);
    END MAKE$MSG$FILE;

INIT: PROCEDURE;
    CALL MOVE(5DH,.MSG$FCB+1,8);
    CALL MAKE$MSG$FILE;
    CALL MOVE(.'DAT',DAT$FCB+9,3);
    DFCB(12),DFCB(32)=0;
    IF OPEN(DAT$FCB)=255 THEN
        CALL ERROR(6);
    IF (DCNT:=DISKREAD(DAT$FCB)) <> 0 THEN
        CALL ERROR(1);
    CALL LIFTHEAD;
    END INIT;


INC$RM: PROCEDURE;
    RM = RM + 1;
    END INC$RM;

INC$WM: PROCEDURE;
    WORK$MAP = WORK$MAP + 1;
    END INC$WM;

PRINT$DATA$E: PROCEDURE;
    DECLARE (I,J) BYTE;
    DO I = 1 TO NUM$ELEMENTS;
       IF RMPTR = EOC THEN /* END OF RECORD MAP */
             DO;
             STOP = TRUE;
        RETURN;
        END;
        CALL INC$RM;
        DL$LEN = 1;
        DO WHILE (RMPTR <> '!') OR (RMPTR <> PERCENT);
             CALL PRINTCHAR(RMPTR);
             CALL INC$RM;
             DL$LEN = DL$LEN + 1;
        END;
    CALL INC$RM;
    CALL PRINTCHAR(' ');
        DO WHILE RMPTR = '0';
             CALL INC$RM;
```

```
          END;
          RM = RM + 2;
          END;
     END PRINT$DATA$E;

SAVE$EX$RN$NR: PROCEDURE;
     WMPTR = ER; /* EXTENT */
     CALL INC$WM;
     WMPTR = ER(1); /* RN */
     CALL INC$WM;
     WMPTR = ER(17) - ER(1); /* NUMBER OF RECORDS */
     CALL INC$WM;
     END SAVE$EX$RN$NR;

PRINTSPACE: PROCEDURE;
     DECLARE I BYTE;
     DO I = 1 TO DL$LEN;
          CALL PRINTCHAR(' ');
     END;
     END PRINTSPACE;

CHK$RESPONSE: PROCEDURE;
     DECLARE (I,C) BYTE;
     DO I = 1 TO NUM$ELEMENTS;
          CALL PRINTCHAR(C:=READC);
          IF C = 'Y' THEN
               CALL SAVE$EX$RN$NR;
          ELSE
          IF C = 'S' THEN
               DO;
               STOP = TRUE;
               RETURN;
               END;
          E$R$A = E$R$A + 16;
          CALL PRINTSPACE;
     END;
     END CHK$RESPONSE;

SEL$WE: PROCEDURE;
     WORK$MAP = .MEMORY + 1536;
     NUM$ELEMENTS=5;
     IF (MODIFY = TRUE) THEN
          CALL PRINT(.'SELECT DATA ELEMENTS TO WORK WITH$');
     ELSE
          DO;
          CALL PRINT(.'SELECT DATA ELEMENTS TO BE
               INCLUDED IN THE REPORT $');
          WORK$MAP = .MEMORY + 1792;
          END;
     E$R$A = RM + 14;
     DO WHILE (RMPTR <> EOC) AND (STOP = FALSE);
          DO WHILE RM < 100H;
          CALL PRINTDATA$E;
          CALL CRLF;
          CALL CHK$RESPONSE;
          END;
          IF (DCNT:=DISKREAD(DAT$FCB)) <> 0 THEN
               CALL ERROR(1);
     END;
     END SEL$WE;

RE$SELECT: PROCEDURE;
      STOP = FALSE;
      MODIFY = FALSE;
      CALL SEL$WE;
      END RE$SELECT;


SET$MEM: PROCEDURE;
     DAT$AREA,BASE$DAT$AREA = WORK$MAP;
     TOP$MEM = SBDOS - 1;
     WORK$MAP = .MEMORY;
```

```
     DFCB(32) = WMPTR(1); /* RN TO START READ */
     NR = WMPTR(2);
     NR$READ = 0;
     END SET$MEM;

OPEN$EXT: PROCEDURE;
     DFCB(12) = WMPTR;
     IF OPEN(DAT$FCB) = 255 THEN
          CALL ERROR(1);
     END OPEN$EXT;

READ$D$REC: PROCEDURE;
     IF DFCB(12) <> WMPTR THEN
     CALL OPEN$EXT;
     CALL SET$DMA(DAT$AREA);
     IF (DCNT:=DISKREAD(DAT$FCB)) <> 0 THEN
          CALL ERROR(1);
     NR$READ = NR$READ + 1;
     DAT$AREA = DAT$AREA + 128;
     CALL SET$DMA (80H);
     DFCB(12) = DAT(126);
     DFCB(32) = DAT(127);
     END READ$D$REC;

READ$DAT: PROCEDURE;
     DO WHILE DAT$AREA+128 < TOP$MEM;
        IF NR$READ = NR THEN
          IF (WORK$MAP:=WORK$MAP+3) >= BASE$DAT$AREA-1 THEN
             DO; /* FINISHED */ MORE = FALSE; RETURN; END;
          ELSE
             DO; DFCB(32) = WMPTR(1);  NR$READ = 0;
                    NR = WMPTR(2); END;
        CALL READ$D$REC;
     END;
     MORE = TRUE;
     SAVE$EXT = DFCB(12);
     SAVE$RN = DFCB(32);
     HOLD$WM = WORK$MAP;
     END READ$DAT;

READ$MORE: PROCEDURE;
     DECLARE HOLD ADDRESS;
     HOLD,WORK$MAP = HOLD$WM;
     DFCB(12) = SAVE$EXT;
     DFCB(32) = SAVE$RN;
     DAT$AREA = BASE$DAT$AREA;
     CALL READ$DAT;
     WORK$MAP = HOLD;
     T$DAT$AREA = DAT$AREA;
     DAT$AREA = BASE$DAT$AREA;
     END READ$MORE;

ALLOCATE: PROCEDURE;
     SAV$EXT = DFCB(12);
     SAV$RN = DFCB(32);
     DAT(126) = FREELIST$EXT;
     DAT(127) = FREELIST$RN;
     OLD$EXT = DAT(252);
     OLD$RN = DAT(253);
     DFCB(12) = FREELIST$EXT;
     DFCB(32) = FREELIST$RN;
     CALL READ$D$REC;
     FREELIST$EXT = DAT(126);
     FREELIST$RN = DAT(127);
     DAT(124) = OLD$EXT;
     DAT(125) = OLD$RN;
     DAT(126) = SAV$EXT;
     DAT(127) = SAV$RN;
     END ALLOCATE;

FREE: PROCEDURE;
     DFCB(12) = DAT(126);
```

```
      DFCB(32) = DAT(127);
      DAT(252) = DAT(124);
      DAT(253) = DAT(125);
      DAT(126) = FREELIST$EXT;
      DAT(127) = FREELIST$RN;
      DAT$AREA = DAT$AREA - 128;
      FREELIST$EXT = DAT(126);
      FREELIST$RN = DAT(127);
      DAT(126) = DFCB(12);
      DAT(127) = DFCB(32);
      CALL READ$D$REC;
      END FREE;

DIG$TO$NUM: PROCEDURE (STR) BYTE;
 /* CONVERTS A STRING OF ASCII DIGITS TO A DECIMAL NUMBER */
      DECLARE
      STR ADDRESS,
      STR1 BASED STR BYTE,
           I BYTE INITIAL (0),
           M BYTE INITIAL (0);
      DO WHILE STR1(I) = BLANK;
           I = I + 1;
      END;
      IF STR1(I) = CR THEN
           RETURN 1;
      DO WHILE (STR1(I+1) <> BLANK) OR (STR1(I+1) <> CR);
           M=(M*10) + (STR1(I)-48);
           I = I + 1;
      END;
      RETURN M;
      END DIG$TO$NUM;

LONG: PROCEDURE BYTE;
      DECLARE L BYTE INITIAL (0);

      IF (RM(0) = '!') OR (RM(0) = ' ') THEN
           DO;
           L = L + 1;
      DO WHILE (RM(L) <> '!') OR
                (RM(L) <> ' ');
                L = L + 1;
           END;
           RETURN L;
           END;
      ELSE
      RETURN DL$LEN;
      END LONG;

  /* ***************************************************

                              EDITING MODULE

        FUNCTION:  TO ALLOW ENTRY OF DATA AND EDITING OF
      ENTERED DATA BY USE OF LINE EDITING FUNCTIONS.  THE
      USER MAY SELECT TO ENTER DATA DIRECTLY INTO THE
      CODED AREA OR BE PROMPTED AS TO WHAT INFORMATION IS
      REQUIRED.

  *************************************************** */

               /*  PROCEDURES OF THE LINE EDITOR  */


BACK$UP: PROCEDURE;
      IF NPTR > 0 THEN
         DO;
           NPTR = NPTR - 1;
           CALL PRINTCHAR(BS);
           CALL PRINTCHAR(' ');
           CALL PRINTCHAR(BS);
         END;
      ELSE
```

```
        CALL PRINTCHAR(BELL);
    END BACK$UP;

MOVE$TO$OLD: PROCEDURE;
    CALL MOVE(NEW$BUF+1,OLD$BUF+1,(OBUF:=NPTR));
    OPTR = 0; NPTR = 0;
    END MOVE$TO$OLD;

OLD$TO$NEW: PROCEDURE;
    NBUF(NPTR:=NPTR+1) = OBUF(OPTR:=OPTR+1);
    END OLD$TO$NEW;

ECHO$ON: PROCEDURE;
   CALL PRINTCHAR(NBUF(NPTR:=NPTR+1):=(OBUF(OPTR:=OPTR+1)));
    END ECHO$ON;

COPY$ONE: PROCEDURE;
    IF OPTR <= OBUF THEN
        CALL ECHO$ON;
    ELSE CALL PRINTCHAR(BELL);
    END COPY$ONE;

P$MOVE$ON: PROCEDURE; /* PARTIAL MOVE OLD TO NEW */
    DO WHILE OPTR < OBUF;
    CALL ECHO$ON;
    END;
    END P$MOVE$ON;

ENTER: PROCEDURE;
    IF INSERT THEN
       CALL PRINTCHAR('>');
    ELSE
       CALL PRINTCHAR('<');
    INSERT = NOT(INSERT);
    END ENTER;

PRINT$OLD: PROCEDURE;
    DECLARE I BYTE;
    DO I= 1 TO OBUF;
    CALL PRINTCHARI(OBUF(I));
    END;
    CALL CRLF;
    END PRINT$OLD;

PRINT$NEW: PROCEDURE;
    DECLARE I BYTE;
    DO I = 1 TO NPTR;
    CALL PRINTCHARI(NBUF(I));
    END;
    END PRINT$NEW;

PRINT$BOTH: PROCEDURE;
    CALL PRINT$OLD;
    CALL PRINT$NEW;
    END PRINT$BOTH;

COPY$RM$O$N: PROCEDURE;
  /* COPIES REMAINING CHARACTERS FOR OLD TO NEW BUFFERS */

    DO WHILE OPTR <= OBUF;
        CALL OLD$TO$NEW;
    END;
    CALL PRINTCHAR('+');  /* INDICATES WHEN DONE  */
    END COPY$RM$O$N;

BS$O$N: PROCEDURE;
    /* BACKSPACE OLD PTR AND NEW PTR 1 CHAR  */
    IF (OPTR > 0) AND (NPTR > 0) THEN
       DO;
        OPTR = OPTR - 1;
        NPTR = NPTR - 1;
         OBUF = OBUF - 1;
```

```
        END;
    ELSE
        CALL PRINTCHAR(BELL);
    END BS$O$N;

COPY$ON: PROCEDURE (C);
    DECLARE (C,I) BYTE;
    I=OPTR;
    DO WHILE OBUF(I:=I+1) <> C;
       IF I > OBUF THEN /* NO MATCH */
         DO;
           CALL PRINTCHAR(BELL);
           RETURN;
         END;
    END; /* DO WHILE */
    DO WHILE OPTR < I;
         CALL ECHO$ON;
    END;
    END COPY$ON;

DELETE: PROCEDURE(ECHO);
    DECLARE (I,J,P1,CHAR1,ECHO) BYTE;
    P1=OPTR;
    CHAR1 = READC;
    DO WHILE (OBUF(P1:=P1+1) <> CHAR1);
       IF P1 > OBUF THEN /* NO MATCH */
         DO;
            CALL PRINTCHAR(BELL);
            RETURN;
         END;
    END;  /* DO WHILE  */
    IF ECHO THEN
        DO I = OPTR+1 TO P1;
             CALL PRINTCHAR(PERCENT);
        END;

     /*  NOW CONDENSE THE BUFFER   */
    J=OPTR;
    I=P1;
    DO WHILE I <= OBUF;
         OBUF(J:=J+1) = OBUF(I:=I+1);
    END;
    OBUF = OBUF - (P1-OPTR+1);
    END DELETE;

DEL$N: PROCEDURE;
    OPTR,NPTR = 0;
    OBUF = 0;
    CALL PRINTCHAR(END$FILE);
    CALL CRLF;
    END DEL$N;

DISPLAY$RM$O$N: PROCEDURE;
    DECLARE I BYTE;
    I = 0;
    CALL CRLF;
    DO WHILE (I:=I+1) <= OBUF;
       IF I <= OPTR THEN /* EVEN LINE */
         CALL PRINTCHAR(' ');
       ELSE CALL PRINTCHAR(OBUF(I));
    END;
    CALL CRLF;
    CALL PRINT$NEW;
    END DISPLAY$RM$O$N;

DEL$O: PROCEDURE;
    IF OPTR > 0 THEN
       DO;
        DECLARE I BYTE;
         I = OPTR-1;
         DO WHILE (I:=I+1) < OBUF;
            OBUF(I) = OBUF(I+1);
```

```
            END;
          CALL PRINTCHAR(PERCENT);
          OBUF = OBUF - 1;
        END;
    ELSE CALL PRINTCHAR(BELL);
    END DEL$O;

ESCAPE: PROCEDURE;
    /* TURNS OFF SPECIAL MEANING OF CHARACTER TO FOLLOW
         AND ENTERS CHARACTER IN NEW BUFFER   */

    CALL PRINTCHARI(CHAR:=READC);
    NBUF(NPTR:=NPTR+1) = CHAR;
    END ESCAPE;

PRINT$TAB: PROCEDURE;
    IF (NPTR + 5) > SIZE$NBUF THEN
       CALL PRINTCHAR(BELL);
    ELSE
       NBUF(NPTR:=NPTR+1) = TAB;
       CALL PRINTCHAR(TAB);
    END PRINT$TAB;

BEGIN$WRD: PROCEDURE;
    DO WHILE NEW$BUF(NPTR-1) <> ' ';
         CALL BACK$UP;
    END;
    END BEGIN$WRD;


       /* END OF PROCEDURES CALLED FROM THE LINE EDITOR */


INC$DA: PROCEDURE;
    DAT$AREA = DAT$AREA +1;
    END INC$DA;

INC$CA: PROCEDURE;
    CODE$A = CODE$A + 1;
    END INC$CA;

INC$ER: PROCEDURE;
    ERRA = ERRA + 1;
    END INC$ER;

MOVE$DL$NEW: PROCEDURE;
    DO WHILE NPTR <= DL$LEN;
         IF PROMPT THEN CALL OLD$TO$NEW;
         ELSE CALL ECHO$ON;
    END;
    NB = NEW$BUF + NPTR + 1;
    OPTR = NPTR;
    END MOVE$DL$NEW;

MOVE$CODE$OLD: PROCEDURE;
    DECLARE DEST ADDRESS, D BASED DEST BYTE;
    DEST = OLD$BUF+1;
    OPTR,NPTR,OBUF = 0;
    DO WHILE DAT <> EOC;
         D = DAT;
         IF NOT(PROMPT) THEN CALL PRINTCHAR(D);
         CALL INC$DA;
         DEST = DEST +1;
         OBUF = OBUF +1;
    END;
    CALL CRLF;
    CALL MOVE$DL$NEW;
    END MOVE$CODE$OLD;

SET$PTR: PROCEDURE;
    DO WHILE DAT <> ERR;
         CALL INC$DA;
```

```
    END;
    CALL INC$DA;
    BERRA,ERRA = DAT$AREA;
    CALL INC$DA;
    DO WHILE DAT <> ERR;
       CALL INC$DA;
    END;
    CALL INC$DA;
    PROMPT$AREA = DAT$AREA;
    END SET$PTR;

NEXT$DE: PROCEDURE;
    CALL MOVE$CODE$OLD;
    CALL INC$DA;
    CALL SET$PTR;
    END NEXT$DE;

UPDATE$DAT: PROCEDURE;
    DECLARE T ADDRESS, (I,A) BYTE;
    INS$INC: PROCEDURE;
         CODE = NBUF(I); I= I + 1;
         CALL INC$CA;
         END INS$INC;

    CODE$A = B$CODE$A + DL$LEN;
    I = DL$LEN+1;
    DO WHILE (A:=I <= NPTR) AND (I <= OBUF);
       CALL INS$INC;
    END;
    IF A THEN /* CODE LINE HAS GROWN */
       DO;
         DO WHILE I <= NPTR+1;
              IF CODE = ERR THEN /* AT ERROR CMDS */
              CALL ERROR(0);
            ELSE
              CALL INS$INC;
         END;
         CODE = EOC;
       END;
    ELSE
       DO;
         CODE = EOC; T = OLD$BUF+OBUF+1;
         DO WHILE (CODE$A :=CODE$A +1) <= T;
              CODE = 0;
         END;
       END;
    END UPDATE$DAT;

  /* ************************************************

                  ERROR MODULE
      FUNCTION:  TO CHECK FOR POSSIBLE ERROR CONDITIONS.
        ERROR COMMANDS ARE DEFINED IN DO$CMD PROCEDURE.

  ************************************************** */

RE$ENTER: PROCEDURE;
    CALL MOVE$TO$OLD;
    CALL PRINT$OLD;
    NPTR = NB - NEW$BUF;
    CALL PRINT$NEW;
    END RE$ENTER;


WARNING: PROCEDURE (I);
    DECLARE I BYTE;
    WARN = TRUE;
    DO CASE I;
         CALL PRINT(.'WILL DESTROY OLD INFORMATION $');
         CALL PRINT(.'EXPECTING ALPHABETIC CHARACTER $');
         CALL PRINT(.'EXPECTING NUMERIC CHARACTER $');
         CALL PRINT(.'CHANGE GREATER THAN SPECIFIED
```

```
                PERCENTAGE $');
          CALL PRINT(.'INPUT NUMBER OUT OF SEQUENCE $');
          CALL PRINT(.'NUMBER OF CHARACTERS NOT
                THE SAME AS ORIGINAL FIELD $');
      END;
     CALL CRLF;
     END WARNING;

INC$NB: PROCEDURE;
     NB = NB + 1;
     END INC$NB;

SP$PD$COM: PROCEDURE BYTE;
     DECLARE SPACE LIT '20H',
          PERIOD LIT '2EH',
          COMMA LIT '2CH';
     RETURN ((TN = SPACE) OR (TN = PERIOD) OR (TN = COMMA));
     END SP$PD$COM;

ALPHA: PROCEDURE BYTE;
     DECLARE LCA LIT '61H', LCZ LIT '7AH';
     RETURN (((TN >= 'A') AND (TN <= 'Z')) OR ((TN >= LCA)
        AND (TN <= LCZ)) OR SP$PD$COM);
     END ALPHA;

CHK$ALPHA: PROCEDURE;
     DO WHILE (TN<>SLASH) OR (TN<>LBRACE) OR (TN<>RBRACE);
          IF NOT(ALPHA) THEN
             DO;
               CALL WARNING(1);
               RETURN;
             END;
          CALL INC$NB;
     END;
     END CHK$ALPHA;

NUMERIC: PROCEDURE BYTE;
     RETURN (((TN - '0') <= 9) OR (TN = 2DH /* MINUS */)
        OR SP$PD$COM);
     END NUMERIC;

CHK$NUMERIC: PROCEDURE;
     DO WHILE (TN<>SLASH) OR (TN<>LBRACE) OR (TN<>RBRACE);
          IF NOT(NUMERIC) THEN
          DO; CALL  WARNING(2); RETURN; END;
          CALL INC$NB;
     END;
     END CHK$NUMERIC;

NEXT$SF: PROCEDURE;
     DO WHILE (CODE<>SLASH) OR (CODE<>LBRACE)
           OR (CODE<>RBRACE);
          CALL INC$CA;
     END;
     CALL INC$CA;
     DO WHILE (DAT<>LBRACE) OR (DAT<>SLASH)
           OR (DAT<>RBRACE);
          CALL INC$DA;
     END;
     CALL INC$DA;
     CALL INC$NB;
     CALL INC$ER;
     END NEXT$SF;


CHK$PERCENT: PROCEDURE;
     DECLARE (X, Y, Z, M, PCT) BYTE;
     CALL INC$ER;
     CALL INC$ER;
     M = DIG$TO$NUM(ECMD);
     X = DIG$TO$NUM(TN);
     Y = DIG$TO$NUM(DAT);
```

```
     IF X>Y THEN Z=X-Y;
     ELSE Z=Y-X;
     IF (Z/Y) > M THEN
          CALL WARNING(3);
     END CHK$PERCENT;

CHK$SEQUENCE: PROCEDURE;
     DECLARE (X,Y) BYTE;
     X = DIG$TO$NUM(TN);
     Y = DIG$TO$NUM(DAT);
     IF (X-Y) <> 1 THEN
     CALL WARNING(4);
     END CHK$SEQUENCE;

CHK$LENGTH: PROCEDURE;
     DECLARE (X, Y) BYTE INITIAL (1);
     DO WHILE (DAT<>LBRACE) OR (DAT<>SLASH)
          OR (DAT<>RBRACE);
          CALL INC$DA;
          X = X + 1;
     END;
     DO WHILE (TN<>SLASH) OR (TN<>LBRACE) OR (TN<>RBRACE);
          CALL INC$NB;
          Y = Y + 1;
     END;
     IF X <> Y THEN
          CALL WARNING(5);
     END CHK$LENGTH;

DO$CMD: PROCEDURE;
      IF ECMD = '0' THEN RETURN; ELSE
      IF ECMD = 'A' THEN
           DO;
           CALL CHK$ALPHA;
           DO WHILE (ECMD <> SLASH) OR (ECMD <> LBRACE)
                OR (ECMD <> RBRACE);
                CALL INC$ER;
           END; END; ELSE
      IF ECMD = 'N' THEN
           DO;
           CALL CHK$NUMERIC;
           DO WHILE (ECMD <> SLASH) OR (ECMD <> LBRACE)
                OR (ECMD <> RBRACE);
                CALL INC$ER;
           END; END; ELSE
      IF ECMD = 'P' THEN
           DO;
           CALL CHK$PERCENT;
           DO WHILE (ECMD <> SLASH) OR (ECMD <> LBRACE)
                OR (ECMD <> RBRACE);
                CALL INC$ER;
           END; END; ELSE
      IF ECMD = 'S' THEN
           DO;
           CALL CHK$SEQUENCE;
           DO WHILE (ECMD <> SLASH) OR (ECMD <> LBRACE)
                OR (ECMD <> RBRACE);
                CALL INC$ER;
           END; END; ELSE
      IF ECMD = 'L' THEN
           DO;
           CALL CHK$LENGTH;
           DO WHILE (ECMD <> SLASH) OR (ECMD <> LBRACE)
                OR (ECMD <> RBRACE);
                 CALL INC$ER;
           END; END; ELSE
      CALL ERROR(2);
      END DO$CMD;

CHK$ERR: PROCEDURE;
     WARN = FALSE;
     DO WHILE ECMD <> ERR;
```

```
          IF (ECMD = SLASH) OR (ECMD=LBRACE) OR (ECMD=RBRACE)
               THEN
               CALL NEXT$SF;
          CALL DO$CMD;
          IF WARN THEN RETURN;
          CALL INC$ER;
          END;
     END CHK$ERR;

ASK$QUESTION: PROCEDURE;
     DO WHILE (DAT <> SLASH) OR (DAT <> LBRACE) OR
          (DAT <> RBRACE);
          CALL PRINTCHAR(DAT);
          CALL INC$DA;
     END;
     END ASK$QUESTION;

CHK$ANSWER: PROCEDURE;
     WARN = FALSE;
     DO WHILE (ECMD <> SLASH) OR (ECMD <> LBRACE) OR
          (ECMD <> RBRACE);
          CALL DO$CMD;
          IF WARN THEN RETURN;
          CALL INC$ER;
     END;
     END CHK$ANSWER;

END$IP: PROCEDURE;
     OB = CLD$BUF + DL$LEN + 1;
     IF (NB:= NEW$BUF+DL$LEN+1) > NEW$BUF + NPTR THEN
        DO;
          CALL WARNING(0);
          RETURN;
        END;
     NBUF(NPTR+1) = '/';
     CALL CHK$ERR;
     END END$IP;

LEDIT: PROCEDURE;
DECLARE
     I BYTE,
     LTR BYTE,
     FOUND BYTE INITIAL (FALSE),
     M BYTE;

     DO WHILE NPTR < SIZE$NBUF;
     IF (CHAR:=READC) = AMPERSAND THEN
     /* BASIC LINE EDITOR COMMAND */
          DO;
           CALL PRINTCHAR(ED$BUF(0):=AMPERSAND);
          I = 1;
          DO WHILE (CHAR:=READC) <> CR;
           CALL PRINTCHAR(ED$BUF(I):=CHAR);
                I = I + 1;
                END;
     IF (ED$BUF(1) = 'C') AND (ED$BUF(2) = 'H') THEN
          /* CHANGE COMMAND */
                DO;
                I = 3;
                DO WHILE ED$BUF(I) = BLANK;
                     I = I + 1;
                END;
                IF ED$BUF(I:=I+1) <> SLASH THEN
                /* VIRGULE EXPECTED */
                     CALL ERROR(8);
                LTR = (I:=I+1);
                CALL COPY$ON(ED$BUF(LTR));
                DO WHILE FOUND = FALSE;
                     DO WHILE (ED$BUF(I) <> SLASH) AND
                          (ED$BUF(I) = OBUF(OPTR));
                          I = I + 1;
                          IF (OPTR:=OPTR+1) > OBUF THEN
```

```
                        CALL ERROR(9);
                    END;
                    IF (ED$BUF(I) <> OBUF(OPTR)) THEN
                    /* LOOK FOR NEXT OCCURRENCE */
                        DO;
                        I = LTR;
                        CALL COPY$ON(ED$BUF(I));
                        END;
                    ELSE
                        FOUND = TRUE;
                END;
                CALL ENTER;
                DO WHILE ED$BUF(I+1) <> SLASH;
                    NBUF(NPTR:=NPTR+1) = ED$BUF(I:=I+1);
                END;
                CALL ENTER;
                CALL P$MOVE$ON;
        END;
    IF(ED$BUF(1) = 'D') AND (ED$BUF(2) = 'E') THEN
    /* DELETE COMMAND */
        CALL DELETE(FALSE);
        /* DELETE OLD BUFFER, DO NOT ECHO PERCENT SIGN */
    IF(ED$BUF(1) = 'F') AND (ED$BUF(2) = 'I') THEN
    /* FILE COMMAND */
        CALL MOVE$TO$OLD;
    IF (ED$BUF(1) = 'Q') AND (ED$BUF(2) = 'U') THEN
    /* QUIT COMMAND */
        GO TO ENDEDIT2;
    IF (ED$BUF(1) = 'N') AND (ED$BUF(2) = 'X') THEN
    /* NEXT COMMAND */
        DO;
        M = DIG$TO$NUM(ED$BUF$PTR+3);
        DO I=1 TO M;
            CALL READ$MORE;
        END;
        CALL MOVE$CODE$OLD;
        CALL PRINT$OLD;
        OPTR, NPTR = 0;
        END;
    IF (ED$BUF(1) = 'U') AND (ED$BUF(2) = 'P') THEN
    /* UP COMMAND */
        DO;
        M = DIG$TO$NUM(ED$BUF$PTR+3);
        DO I=1 TO M;
        DAT$AREA = DAT$AREA - 128;
        END;
        CALL MOVE$CODE$OLD;
        CALL PRINT$OLD;
        OPTR, NPTR = 0;
        END;
    IF (ED$BUF(1) = 'P') AND (ED$BUF(2) = 'O') THEN
    /* PROMPT-ON COMMAND */
        PROMPT = TRUE;
    IF (ED$BUF(1) = 'N') AND (ED$BUF(2) = 'P') THEN
    /* NO-PROMPT COMMAND */
        PROMPT = FALSE;
    IF (ED$BUF(1) = 'I') AND (ED$BUF(2) = 'N') THEN
    /* INSERT COMMAND */
        DO;
        CALL ENTER;
        INSERT ION = TRUE;
        END;
END;
    IF (CHAR) <= CTLZ THEN /* CONTROL CHAR */
     DO CASE CHAR;
            /* CAS 0 NULL */
            ;

            /* CASE 1 CONTROL A  */
            CALL BACKUP;

            /* CASE 2 CONTROL B  */
```

```
 CALL MOVE$TO$OLD;

 /* CASE 3 CONTROL C  */
  CALL COPY$ONE;

 /* CASE 4 CONTROL D  */
 DO;
  CALL P$MOVE$ON;
  GO TO ENDEDIT1;
END;

 /* CASE 5 CONTROL E  */
 CALL ENTER;

 /*  CASE 6 CONTROL F  */
 GO TO ENDEDIT2;

 /* CASE 7 CONTROL G  */
 CALL PRINT$BOTH;

/* CASE 8 CONTROL H  */
 CALL P$MOVE$ON;

/* CASE 9 CONTROL I  */
 CALL PRINT$TAB;

 /* CASE 10 CONTROL J  */
 GO TO ENDEDIT1;

 /* CASE 11 CONTROL K  */
 ;

 /* CASE 12 CONTOL L  */
 CALL COPY$RM$O$N;

 /* CASE 13 CONTROL M  */
 GO TO CARRIAGE$RETURN;

 /* CASE 14 CONTROL N  */
 CALL BS$O$N;

 /* CASE 15 CONTROL O  */
  CALL COPY$ON(READC);

 /* CASE 16 CONTROL P  */
 CALL DELETE(TRUE);

 /* CASE 17 CONTROL Q  */
 CALL DEL$N;

 /* CASE 18 CONTROL R  */
 CALL DISPLAY$RM$O$N;

 /* CASE 19 CONTROL S  */
 CALL DEL$O;

 /* CASE 20 CONTROL T  */
 ;

 /* CASE 21 CONTROL U */
 CALL COPY$ON(TAB);

 /* CASE 22 CONTROL V  */
 CALL ESCAPE;

 /* CASE 23 CONTROL W  */
 CALL BEGIN$WRD;

 /* CASE 24 CONTROL X  */
 CALL DELETE(FALSE);

 /*CASE 25 CONTROL Y */
```

```
                DO;
                   CALL P$MOVE$ON;
                   CALL MOVE$TO$OLD;
                END;

                /* CASE 26 CONTROL Z  */
                 CALL COPY$ON(READC);


          END;
     ELSE  /* CHECK SPECIAL CASES */
CARRIAGE$RETURN:
     IF (CHAR = CR) AND INSERTION THEN
          DO;
          CALL PRINT(.'EXPECTING A FILE OR QUIT COMMAND $');
          CALL ENTER;
          INSERTION = FALSE;
          END;
     IF CHAR = RUBOUT THEN
        CALL BACKUP;
     ELSE
          DO;
             CALL PRINTCHAR(CHAR);
             NBUF(NPTR:= NPTR+1)=CHAR;
                IF NPTR = 72 THEN CALL PRINTCHAR(BELL);
          IF NOT(INSERT) THEN OPTR = OPTR + 1;
          END;
     END;  /* DO WHILE */

     /* ARRIVE HERE IF BUFFER FULL */

     CALL PRINTCHAR(BELL);
     ENDEDIT1:
     ENDEDIT2: CALL CRLF;
     END LEDIT;

  /* ************************************************

                              OUTPUT MODULE

                FUNCTION: TO UPDATE THE DAT FILE AND THE
          INFORMATION JUST EDITED TO THE MESSAGE FILE.

  ************************************************** */

INC$MSG: PROCEDURE;
     IF (MSG$AREA:= MSG$AREA + 1) < 100H THEN
          RETURN;
     IF DISKWRITE(.MSG$FCB) <> 0 THEN
          CALL ERROR(4);
     MSG$AREA = 80H;
     END INC$MSG;

MOVE$MSG: PROCEDURE;
     IF DAT = '!' THEN
          DO;
     CALL INC$DA;
          DO WHILE DAT <> '!';
                CALL INC$DA;
          END;
     CALL INC$DA;
          END;
     DO WHILE DAT <> EOC;
          IF (DAT = LBRACE) OR (DAT = RBRACE) THEN
                CALL INC$DA;
          ELSE
                IF DAT = PERCENT THEN
                     CALL INC$DA;
                ELSE
                     MSG = DAT;
          CALL INC$MSG;
          CALL INC$DA;
```

```
    END;
    MSG = CR;
    CALL INC$MSG;
    MSG = LF;
    CALL INC$MSG;
    END MOVE$MSG;

WRITE$MSG: PROCEDURE;
    MSG = CTLZ;
    IF DISKWRITE(.MSG$FCB) <> 0 THEN
        CALL ERROR(4);
    END WRITE$MSG;

CLOSE$FILES: PROCEDURE;
    IF CLOSE(.MSG$FCB) = 255 THEN
        CALL ERROR(7);
    DFCB(12) = SAVE$EXT;
    DFCB(32) = SAVE$RN;
    IF CLOSE(DAT$FCB) = 255 THEN
        CALL ERROR(6);
    END CLOSE$FILES;

BLANK$BUF: PROCEDURE;
    DECLARE A ADDRESS, (B BASED A,I) BYTE;
    A = .BUFFER;
    DO I = 1 TO 180;
        B = 0; A = A + 1;
    END;
    END BLANK$BUF;

BASE$NEXT$DE: PROCEDURE;
    DECLARE I BYTE;
    DO I = 1 TO WMPTR(2);
        B$CODE$A = B$CODE$A + 128;
    END;
    WORK$MAP = WORK$MAP + 3;
    END BASE$NEXT$DE;

UPDATE$DAT$FILE: PROCEDURE;
    WORK$MAP = .MEMORY;
    B$CODE$A = BASE$DAT$AREA;
    DO WHILE B$CODE$A < T$DAT$AREA;
        CALL MOVE$MSG;
        DFCB(12) = WMPTR;
        DFCB(32) = WMPTR(1);
        CALL SETDMA(B$CODE$A);
        IF DISKWRITE(DAT$FCB) <> 0 THEN
            CALL ERROR(4);
        CALL BASE$NEXT$DE;
    END;
    CALL SET$DMA(80H);
    END UPDATE$DAT$FILE;


EDIT: PROCEDURE;
    CONTINUE:
    DO WHILE (DAT$AREA < T$DAT$AREA);
    CALL NEXT$DE;
       IF PROMPT THEN
         DO WHILE DAT <> EOP;
              CALL ASK$QUESTION;
              WARN = TRUE;
              DO WHILE WARN;
                 CALL LEDIT;
                 NBUF(NPTR:= NPTR+1)='/';
                 CALL CHK$ANSWER;
              END;
              CALL NEXT$SF;
         END;
       ELSE
         DO;
         CALL LEDIT;
```

```
            CALL END$IP;
          END;
        IF WARN THEN CALL RE$ENTER;
        ELSE
        DO;
          CALL UPDATE$DAT;
          CALL BLANK$BUF;
          CALL BASE$NEXT$DE;
          DAT$AREA,CODE$A = B$CODE$A;
        END;
    END; /* DO WHILE */

    CALL UPDATE$DAT$FILE;
    IF MORE THEN
        DO;
          CALL READ$MORE;
          GO TO CONTINUE;
        END;
    CALL WRITE$MSG;
    CALL CLOSE$FILES;
    GO TO BOOT;
    END EDIT;


     /******* START MAIN PROGRAM HERE    ****** */

OLD$BUF = (NEW$BUF := .BUFFER)+90;
OBUF = 0;
CALL INIT;
CALL SEL$WE;
CALL RES$SELECT;
CALL SET$MEM;
CALL READ$DAT;
CALL LIFTHEAD;
T$DAT$AREA = DAT$AREA;
B$CODE$A,CODE$A,DAT$AREA = BASE$DAT$AREA;
WORK$MAP = .MEMORY;
CALL CRLF;
CALL PRINT(.'DO YOU WISH TO BE PROMPTED?$');
CALL PRINTCHAR(CHAR:= READC);
CALL CRLF;
IF CHAR = 'Y' THEN PROMPT = TRUE;
CALL EDIT;
EOF
```

```
                    /* ****************
                    *                  *
                    *  CREATE PROGRAM  *
                    *                  *
                    **************** */

  /* ************************************************

      PROGRAM DESIGNED TO CREATE DAT EXECUTABLE FILES USED
    IN CONJUNCTION WITH REPORT ORIGINATION SYSTEM (ROS).
       ROS IS DESIGNED TO GENERATE FORMATTED REPORTS.

  ************************************************ */
100H:

  /* ************************************************

                          INITIALIZE DECLARATIONS

  ************************************************ */
DECLARE
    LIT         LITERALLY       'LITERALLY',
    BOOT        LIT   '0',
    ENTRY       LIT   '0005H',
    TRUE        LIT   '1',
    FALSE       LIT   '0',
    FOREVER     LIT   'WHILE TRUE',
    CR          LIT   '0DH',
    LF          LIT   '0AH',
    DCNT        BYTE,
    CTI         LIT   '0',
    CTS         LIT   '1';

  /* ************************************************

                        INPUT AND EDITING DECLARATIONS

  ************************************************ */
DECLARE
    BS          LIT   '08H',    /* BACKSPACE  */
    PERCENT     LIT   '25H',
    BELL        LIT   '07H',
    TAB         LIT   '09H',
    EOP         LIT   '5EH', /* UP-ARROW; END OF PROMPT */
    END$FILE    LIT   '5CH', /*BACK SLANT */
    EOC         LIT   '7CH', /*BAR; END OF CODE */
    ERR         LIT   '7EH', /* TILDE; END OF ERROR */
    CTLZ        LIT   '1AH',
    BLANK       LIT   '20H',
    AMPERSAND   LIT   '26H',
    SLASH       LIT   '2FH',
    LBRACE      LIT   '7BH',
    RBRACE      LIT   '7DH',
    RUBOUT      LIT   '7FH',
    DAT$FCB             ADDRESS INITIAL (5CH),
    DFCB  BASED DAT$FCB (33)  BYTE,
    NUM$REC             BYTE,
    TMEM                ADDRESS,
    RECORD$MAP                  ADDRESS,
    RMPTR BASED RECORD$MAP      BYTE,
    EXT         BYTE  INITIAL(0),
    SAVE$EX     BYTE  INITIAL(0),
    SAVE$RN     BYTE  INITIAL(0),
    STORE                       ADDRESS,
    SP    BASED STORE           BYTE,
    BSTORE                      ADDRESS,
    SPTR                        ADDRESS,
    BUFFER   (180)              BYTE,
    SIZE$NBUF LIT               '90',
    NEW$BUF                     ADDRESS,
```

```
    NBUF   BASED NEW$BUF  BYTE,
    NPTR          BYTE,
    CLD$BUF               ADDRESS,
    OBUF   BASED OLD$BUF  BYTE,
    OPTR          BYTE,
    INSERT    BYTE INITIAL (FALSE),
    CHAR      BYTE,
    MOD$128$MASK   LIT  '0FF80H',   /* GIVES MEMORY SIZE
        IN MULIPLES OF 128 BYTE BLOCKS */
    BDOSA     ADDRESS INITIAL (0006H),
    SBDOS     BASED BDOSA  ADDRESS,
    (LABEL$MOVED, REC$MAP$WRITTEN) BYTE INITIAL (FALSE),
    ED$BUF$PTR ADDRESS,
    (ED$BUF BASED ED$BUF$PTR) (100) BYTE,
    REC$MAP$EXT BYTE INITIAL (0),
    REC$MAP$REC BYTE INITIAL (0);


CRTIN: PROCEDURE BYTE;
    DO WHILE INPUT(CTS);
    END;
    RETURN NOT INPUT(CTI) AND 07FH;
    END CRTIN;

READC: PROCEDURE BYTE;

    /* GET A CHARACTER FROM THE CONSOLE AND TRANSLATE TO
         UPPER CASE      */

    DECLARE C BYTE;
    IF (C:=CRTIN) >= 110$0001B  /* LOWER CASE A */
       AND C <= 011110010B  /* LOWER CASE Z */ THEN
       C = C AND 101$1111B;  /* BECOMES UPPER CASE */
    RETURN C;
    END READC;

MON1: PROCEDURE (FUNC, INFO);
    DECLARE FUNC BYTE, INFO ADDRESS;
    GO TO ENTRY;
    END MON1;

MON2: PROCEDURE (FUNC,INFO) BYTE;
    DECLARE FUNC BYTE, INFO ADDRESS;
    GO TO ENTRY;
    END MON2;

MON3: PROCEDURE (FUNC, INFO) ADDRESS;
    DECLARE FUNC BYTE, INFO ADDRESS;
    GO TO ENTRY;
    END MON3;

PRINTCHAR: PROCEDURE (B);
    DECLARE B BYTE;
    CALL MON1(2,B);
    END PRINTCHAR;

PRINTCHARI: PROCEDURE(C);
    DECLARE C BYTE;
    IF (C AND 110$0000B) = 0 /* CONTROL CHAR */ THEN
       DO;
         CALL PRINTCHAR(EOP);
         CALL PRINTCHAR(C OR 40H);
       END;
    ELSE CALL PRINTCHAR(C);
    END PRINTCHARI;

CRLF: PROCEDURE;
   CALL PRINTCHAR(CR);
   CALL PRINTCHAR(LF);
    END CRLF;

PRINT: PROCEDURE (A);
```

```
     DECLARE A ADDRESS;
     CALL MON1(9,A);
     CALL CRLF;
     END PRINT;

MOVE: PROCEDURE (SOURCE,DEST,N);
     DECLARE (SOURCE,DEST) ADDRESS,
       (S BASED SOURCE, D BASED DEST, N ) BYTE;
     DO WHILE (N:=N-1) <> 255;
        D=S; SOURCE=SOURCE+1; DEST=DEST+1;
     END;
     END MOVE;

FILL: PROCEDURE(START,DEST,CHAR);
     DECLARE (START,DEST) ADDRESS,
        (S BASED START, CHAR) BYTE;
     DO WHILE START < DEST;
        S = CHAR;
        START = START + 1;
     END;
     END FILL;

ERROR: PROCEDURE(I);
     DECLARE I BYTE;
          DO CASE I;
          CALL PRINT(.'LACK ERROR COMMAND SPACE  $');
          CALL PRINT(.'DISK WRITE ERROR    $');
          CALL PRINT(.'FILE NOT PRESENT $');
          END;
     CALL CRLF;
     GO TO BOOT;
     END ERROR;

DIG$TO$NUM: PROCEDURE (STR) BYTE;
 /* CONVERTS A STRING OF ASCII DIGITS TO A DECIMAL NUMBER */
DECLARE
     STR ADDRESS,
     STR1 BASED STR BYTE,
     I BYTE INITIAL (0),
     M BYTE INITIAL (0);
     DO WHILE STR1(I) = BLANK;
          I = I + 1;
     END;
     IF STR1(I) = CR THEN
          RETURN 1;
     DO WHILE STR1(I+1) <> (BLANK OR SLASH);
          M = (M*10) + (STR1(I)-48);
          I = I + 1;
     END;
     RETURN M;
END DIG$TO$NUM;

  /* ***************************************************

                          INPUT AND EDITING MODULE

  *************************************************** */

PROMPT: PROCEDURE(I);
     DECLARE I BYTE;
     CALL MON1(9,.'EXPECTING $');
     DO CASE I;
        CALL PRINT(.'CODE INFORMATION $');
        CALL PRINT(.'ERROR COMMANDS $');
        CALL PRINT(.'PROMPT INFORMATION $');
     END;
     END PROMPT;

INC$RM: PROCEDURE;
     RECORD$MAP = RECORD$MAP + 1;
     END INC$RM;
```

```
GO$NEXT$REC: PROCEDURE;
    /* INCREMENTS STORAGE POINTER TO NEXT EVEN
       RECORD SECTOR */
    DO WHILE SPTR < STORE;
       SPTR = SPTR + 128;
      NUM$REC=NUM$REC+1;
       IF NUM$REC = 128 THEN
         DO;
            EXT = EXT + 1;
            NUM$REC = 0;
         END;
    END;
    STORE = SPTR-1;
    END GO$NEXT$REC;

MOVE$DL: PROCEDURE;
    DECLARE A ADDRESS, I BYTE, DELIMIT BYTE;
    IF (LABEL$MOVED = TRUE) THEN
       RETURN;
    I = 1;
    A = RECORD$MAP + 14;
    RMPTR, DELIMIT = OBUF(I);
    CALL INC$RM;
    I = I + 1;
    DO WHILE (RMPTR:=OBUF(I)) <> DELIMIT;
       I = I + 1;
       CALL INC$RM;
    END;
    RMPTR = DELIMIT;
     DO WHILE (RECORD$MAP:=RECORD$MAP + 1) < A;
       RMPTR = '0';
    END;
    RMPTR = EXT;
    CALL INC$RM;
    RMPTR = NUM$REC;
    CALL INC$RM;
    LABEL$MOVED = TRUE;
    END MOVE$DL;

WRITE: PROCEDURE;
     DECLARE A ADDRESS;
     IF REC$MAP$WRITTEN = FALSE THEN
          DO;
          A = .MEMORY - 128;
          SAVE$EX  = DFCB(12);
          SAVE$RN = DFCB(32);
          DFCB(12) = REC$MAP$EXT;
          DFCB(32) = REC$MAP$REC;
          DO WHILE (A:=A+128) < BSTORE;
               CALL MOVE(A, 80H, 128);
               IF (DCNT:=MON2(21,DAT$FCB))<>0  THEN
                    CALL ERROR(0);
          END;
          CALL MON1(12,0);
          DFCB(12) = SAVE$EX;
          DFCB(32) = SAVE$RN;
          END;
     A = BSTORE - 128;
     DO WHILE (A:=A+128) < STORE;
          CALL MOVE(A, 80H, 128);
          IF(DCNT:=MON2(21,DAT$FCB)) <> 0  THEN
               CALL ERROR(0);
     END;
     STORE = BSTORE;
     CALL MON1(12,0);
     END WRITE;

INC$STORE: PROCEDURE;
    /* CHECKS FOR MEMORY OVERFLOW INCREMENTS STORAGE PTR */
    IF (STORE:=STORE+1) > TMEM THEN
          DO;
          IF (LABEL$MOVED = FALSE) THEN
```

```
            CALL MOVE$DL;
        CALL WRITE;
        REC$MAP$WRITTEN = TRUE;
        END;
    END INC$STORE;

MOVE$STORE: PROCEDURE;
    /* STORES INFORMATION FROM INPUT TO FILE MEMORY AREA */
    DECLARE I BYTE;
     DO I=1 TO OBUF;
        CALL INC$STORE;
        SP = OBUF(I);
    END;
    END MOVE$STORE;

FILL$CODE$ZERO: PROCEDURE;
    DECLARE (A,T) ADDRESS, B BASED A BYTE;
    A = STORE; T = SPTR + 128;
    DO WHILE (A:=A+1) < T;
       B = 0;
    END;
    END FILL$CODE$ZERO;

ZERO$BUF: PROCEDURE;
     DECLARE I BYTE;
     DO I = 0 TO 89;
     NBUF(I), OBUF(I) = '0';
     END;
END ZERO$BUF;

END$LINE: PROCEDURE;
    /* CHARACTER IS A CARRIAGE RETURN */
    CALL GO$NEXT$REC;
    CALL MOVE$DL;
    CALL MOVE$STORE;
     CALL INC$STORE;
    CALL ZERO$BUF;
    SP = CR;
    CALL CRLF;
    END END$LINE;

END$DL: PROCEDURE;
    /* CHARACTER IS A BAR (|) INDICATES END OF CODE AREA */
    CALL GO$NEXT$REC;
    CALL MOVE$DL;
    CALL MOVE$STORE;
    CALL INC$STORE;
    SP = EOC;
    CALL FILL$CODE$ZERO;
     CALL INC$STORE;
    CALL ZERO$BUF;
    CALL CRLF;
    CALL PROMPT(1);
    END END$DL;

END$REC: PROCEDURE;
    /* CALL WHEN A UP-ARROW  IS ENTERED FROM THE KEYBOARD.
          INDICATES END OF DECODED INFORMATION    */

    CALL GO$NEXT$REC;
    CALL MOVE$STORE;
    CALL INC$STORE;
    SP = EOP;
    LABEL$MOVED, REC$MAP$WRITTEN = FALSE;
     CALL FILL$CODE$ZERO;
     CALL INC$STORE;
    CALL ZERO$BUF;
    CALL CRLF;
    CALL PROMPT(0);
    END END$REC;

END$ERR: PROCEDURE;
```

```
    /* CALLED WHEN (TILDE) ENTERED AT KEYBOARD
       INDICATES END OF ERROR CHECKS    */
    CALL GO$NEXT$REC;
    CALL MOVE$STORE;
    CALL INC$STORE;
    SP = ERR;
    CALL FILL$CODE$ZERO;
     CALL INC$STORE;
    CALL ZERO$BUF;
    CALL CRLF;
    CALL PROMPT(2);
    END END$ERR;

END$F: PROCEDURE;
    /* CALLED WHEN END FILE ( ) BLACKSLASH ENTERED INDICATES
              END OF FILE    */
    DECLARE (EX,NR) BYTE;
    CALL MOVE$STORE;
    CALL INC$STORE;
    SP=END$FILE;
    CALL WRITE;
    CALL INC$RM;
    RMPTR = EOC; /* MARK END OF RECORD$MAP */
    RECORD$MAP = RECORD$MAP + 14; /* SAVE EXT AND RN OF
                NEXT RECORD TO BE WRITTEN    */
    RMPTR = DFCB(12);
    CALL INC$RM;
    RMPTR = DFCB(32);
    EX = DFCB(12); NR = DFCB(32);
    DFCB(32),DFCB(12) = 0;
    CALL MOVE(.MEMORY,80H,128);
    IF (DCNT := MON2(21,DAT$FCB)) <> 0 THEN
       CALL ERROR(1);
    DFCB(12) = EX; DFCB(32) = NR;
    IF MON2(16, DAT$FCB) = 255 THEN  /* CLOSE FILE */
       CALL ERROR(2);
    GO TO BOOT;
    END END$F;

BACK$UP: PROCEDURE;
    IF NPTR > 0 THEN
       DO;
         NPTR = NPTR -1;
         CALL PRINTCHAR(BS);
         CALL PRINTCHAR(' ');
         CALL PRINTCHAR(BS);
       END;
    ELSE
       CALL PRINTCHAR(BELL);
    END BACK$UP;

MOVE$TO$OLD: PROCEDURE;
    CALL MOVE(NEW$BUF+1,OLD$BUF+1,(OBUF:=NPTR));
    OPTR = 0; NPTR = 0;
    CALL CRLF;
    END MOVE$TO$OLD;

ECHO$ON: PROCEDURE;
   CALL PRINTCHAR(NBUF(NPTR:=NPTR+1):=(OBUF(OPTR:=OPTR+1)));
    END ECHO$ON;

COPY$ONE: PROCEDURE;
    IF CPTR < OBUF THEN
         CALL ECHO$ON;
    ELSE CALL PRINTCHAR(BELL);
    END COPY$ONE;

P$MOVE$ON: PROCEDURE; /* PARTIAL MOVE OLD TO NEW */
    DO WHILE OPTR <  OBUF;
    CALL ECHO$ON;
    END;
    END P$MOVE$ON;
```

```
ENTER: PROCEDURE;
    IF INSERT THEN
       CALL PRINTCHAR('>');
    ELSE
       CALL PRINTCHAR('<');
    INSERT = NOT(INSERT);
    END ENTER;

PRINT$OLD: PROCEDURE;
    DECLARE I BYTE;
    DO I= 1 TO OBUF;
       CALL PRINTCHARI(OBUF(I));
    END;
    CALL CRLF;
    END PRINT$OLD;

PRINT$NEW: PROCEDURE;
    DECLARE I BYTE;
    DO I = 1 TO NPTR;
       CALL PRINTCHARI(NBUF(I));
    END;
    END PRINT$NEW;

PRINT$BOTH: PROCEDURE;
    CALL PRINT$OLD;
    CALL PRINT$NEW;
    END PRINT$BOTH;

COPY$RM$O$N: PROCEDURE;
    /* COPIES REMAINING CHARACTERS FOR OLD TO NEW BUFFERS */

    DO WHILE OPTR <= OBUF;
       NBUF(NPTR:=NPTR+1) = OBUF(OPTR:=OPTR+1);
    END;
    CALL PRINTCHAR('+');  /* INDICATES WHEN DONE  */
    END COPY$RM$O$N;

BS$O$N: PROCEDURE;
    /* BACKSPACE OLD PTR AND NEW PTR 1 CHAR  */
    IF (OPTR > 0) AND (NPTR > 0) THEN
       DO;
        OPTR = OPTR - 1;
        NPTR = NPTR - 1;
         OBUF = OBUF - 1;
       END;
    ELSE
       CALL PRINTCHAR(BELL);
    END BS$O$N;

COPY$ON: PROCEDURE (C,NEXT);
    DECLARE (C,I,NEXT) BYTE;
    I=OPTR;
    DO WHILE OBUF(I:=I+1) <> C;
       IF I > OBUF THEN /* NO MATCH */
         DO;
           CALL PRINTCHAR(BELL);
           RETURN;
         END;
    END; /* DO WHILE */
    IF NOT(NEXT) THEN I=I-1;
    DO WHILE OPTR < I;
         CALL ECHO$ON;
    END;
    END COPY$ON;

DELETE: PROCEDURE(ECHO);

       /* ECHO TRUE INDICATES TO START FROM THE CURRENT
    POSITION OF OLD BUFFER AND ECHO A % (PERCENT) FOR THE
    DELETED CHARACTER.  ECHO FALSE INDICATES TO START AT
    THE BEGINNING OF THE OLD BUFFER AND DON'T ECHO FOR
```

```
     THE DELETED CHARACTERS. */

     DECLARE (I,J,P1,CHAR1,ECHO) BYTE;
     IF ECHO THEN P1 = 0;
     ELSE P1 = OPTR;
     CHAR1 = READC;
     DO WHILE (OBUF(P1:=P1+1) <> CHAR1);
        IF P1 > OBUF THEN /* NO MATCH */
          DO;
             CALL PRINTCHAR(BELL);
             RETURN;
          END;
     END;  /* DO WHILE  */
     IF ECHO THEN
         DO I = OPTR+1 TO P1;
              CALL PRINTCHAR(PERCENT);
         END;

      /*  NOW CONDENSE THE BUFFER   */
     J=OPTR;
     I=P1;
     DO WHILE I <= OBUF;
          OBUF(J:=J+1) = OBUF(I:=I+1);
     END;
     OBUF = OBUF - (P1-OPTR+1);
     END DELETE;

DEL$N: PROCEDURE;
     NPTR=0; OPTR=0;
     CALL PRINTCHAR(END$FILE);
     CALL CRLF;
     END DEL$N;

DISPLAY$RM$C$N: PROCEDURE;
     DECLARE I BYTE;
     I = 0;
     CALL CRLF;
     DO WHILE (I:=I+1) <= OBUF;
        IF I <= OPTR THEN /* EVEN LINE */
          CALL PRINTCHAR(' ');
        ELSE
          CALL PRINTCHAR(OBUF(I));
     END;
     CALL CRLF;
     CALL PRINT$NEW;
     END DISPLAY$RM$O$N;

DEL$O: PROCEDURE;
     IF OPTR > 0 THEN
        DO;
          DECLARE I BYTE;
          I = OPTR - 1;
          DO WHILE (I:=I+1) < OBUF;
               OBUF(I) = OBUF(I+1);
        END;
        CALL PRINTCHAR(PERCENT);
        OBUF = OBUF - 1;
        END;
     ELSE CALL PRINTCHAR(BELL);
     END DEL$O;

ESCAPE: PROCEDURE;
     /* TURNS OFF SPECIAL MEANING OF CHARACTER TO FOLLOW
          AND ENTERS CHARACTER IN NEW BUFFER   */

     CALL PRINTCHARI(CHAR:=READC);
     NBUF(NPTR:=NPTR+1) = CHAR;
     END ESCAPE;

CONT$FILL: PROCEDURE;
     CALL MOVE$STORE;
     CALL CRLF;
```

```
    END CONT$FILL;

PRINT$TAB: PROCEDURE;
    IF (NPTR + 5) > SIZE$NBUF THEN
       CALL PRINTCHAR(BELL);
    ELSE
       NBUF(NPTR:=NPTR+1) = TAB;
       CALL PRINTCHAR(TAB);
    END PRINT$TAB;

BEGIN$WRD: PROCEDURE;
    DO WHILE NEW$BUF(NPTR-1) <> ' ';
       CALL BACK$UP;
    END;
    END BEGIN$WRD;

LEDIT: PROCEDURE;
    DECLARE
    I BYTE,
    LTR BYTE,
    FOUND BYTE INITIAL (FALSE),
    M BYTE;

    /* READS CHARACTERS FROM THE CONSOLE AND ALLOWS EDITING
    USING THE PROCEDURES OF A LINE EDITOR  */

    OPTR = 0; NPTR = 0;
    DO WHILE NPTR < SIZE$NBUF;
    IF (CHAR:=READC) = AMPERSAND THEN
    /* BASIC LINE EDITOR COMMAND */
       DO;
        CALL PRINTCHAR(ED$BUF(0):=AMPERSAND);
        I = 1;
        DO WHILE (CHAR:=READC) <> CR;
        CALL PRINTCHAR(ED$BUF(I):=CHAR);
             I = I + 1;
       END;
       IF (ED$BUF(1) = 'C') AND (ED$BUF(2) = 'H') THEN
       /* CHANGE COMMAND */
            DO;
                 I = 3;
                 DO WHILE ED$BUF(I) = BLANK;
                      I = I + 1;
                 END;
                 IF ED$BUF(I:=I + 1) <> SLASH THEN
            /* VIRGULE EXPECTED */
                      CALL ERROR(8);
                 LTR = (I:=I + 1);
                 CALL COPY$ON(ED$BUF(LTR), FALSE);
                 DO WHILE FOUND = FALSE;
                      DO WHILE (ED$BUF(I) <> SLASH) AND
                           (ED$BUF(I) = OBUF(OPTR));
                           I = I + 1;
                           IF (OPTR:=OPTR + 1) > OBUF THEN
                                CALL ERROR(9);
                      END;
                      IF (ED$BUF(I) <> OBUF(OPTR)) THEN
                      /* LOOK FOR NEXT OCCURRENCE */
                      DO;
                           I = LTR;
                           CALL COPY$ON(ED$BUF(I), FALSE);
                           END;
                      ELSE
                           FOUND = TRUE;
                 END;
                 CALL ENTER;
                 DO WHILE ED$BUF(I+1) <> SLASH;
                      NBUF(NPTR:=NPTR+1)=ED$BUF(I:=I+1);
                 END;
                 CALL ENTER;
                 CALL P$MOVE$ON;
            END;
```

```
      IF (ED$BUF(1) = 'D') AND (ED$BUF(2) = 'E') THEN
      /* DELETE COMMAND */
           CALL DELETE(FALSE);
       /* DELETE OLD BUFFER, DO NOT ECHO PERCENT SIGN */
      IF (ED$BUF(1) = 'F') AND (ED$BUF(2) = 'I') THEN
      /* FILE COMMAND */
           CALL MOVE$TO$OLD;
      IF (ED$BUF(1) = 'Q') AND (ED$BUF(2) = 'U') THEN
      /* QUIT COMMAND */
           GO TO ENDEDIT2;
      IF (ED$BUF(1) = 'N') AND (ED$BUF(2) = 'X') THEN
      /* NEXT COMMAND */
      CALL PRINT(.'NEXT COMMAND IS NOT IN CREATE PROGRAM
           EDITOR $');

      IF (ED$BUF(1) = 'P') AND (ED$BUF(2) = 'R') THEN
      /* PRINT COMMAND */
      CALL PRINT(.'PRINT COMMAND IS NOT IN CREATE PROGRAM
           EDITOR $');

      IF (ED$BUF(1) = 'U') AND (ED$BUF(2) = 'P') THEN
      /* UP COMMAND */
      CALL PRINT(.'UP COMMAND IS NOT IN CREATE PROGRAM
           EDITOR $');

      IF (ED$BUF(1) = 'P') AND (ED$BUF(2) = 'O') THEN
      /* PROMPT ON COMMAND */
      CALL PRINT(.'PROMPT-ON COMMAND IS NOT IN CREATE
           EDITOR $');

      IF (ED$BUF(1) = 'N') AND (ED$BUF(2) = 'P') THEN
      /* NO PROMPT COMMAND */
      CALL PRINT(.'NO-PROMPT COMMAND IS NOT IN CREATE
           EDITOR $');

      IF (ED$BUF(1) = 'I') AND (ED$BUF(2) = 'N') THEN
      /* INSERT COMMAND */ ;
      CALL PRINT(.'INSERT COMMAND IS NOT IN CREATE
           EDITOR $');

END;
IF CHAR <= CTLZ THEN /* CONTROL CHAR */
    DO CASE CHAR;
           /* CAS 0 NULL */
           ;

           /* CASE 1 CONTROL A  */
           CALL BACKUP;

           /* CASE 2 CONTROL B  */
           CALL MOVE$TO$OLD;

           /* CASE 3 CONTROL C  */
                CALL COPY$ONE;

           /* CASE 4 CONTROL D  */
           DO;
           CALL P$MOVE$ON;
           GO TO ENDEDIT1;
           END;

           /* CASE 5 CONTROL E  */
           CALL ENTER;

           /*  CASE 6 CONTROL F  */
           GO TO ENDEDIT2;

           /* CASE 7 CONTROL G  */
           CALL PRINT$BOTH;

           /* CASE 8 CONTROL H  */
           CALL P$MOVE$ON;
```

```
                /* CASE 9 CONTROL I  */
                CALL PRINT$TAB;

                /* CASE 10 CONTROL J  */
                GO TO ENDEDIT1;

                /* CASE 11 CONTROL K  */
                ;

                /* CASE 12 CONTOL L  */
                CALL COPY$RM$O$N;

                /* CASE 13 CONTROL M  */
                DO;
                CALL MOVE$TO$OLD;
                CALL END$LINE;
                END;

                /* CASE 14 CONTROL N  */
                CALL BS$O$N;

                /* CASE 15 CONTROL O  */
                CALL COPY$ON(READC,FALSE);

                /* CASE 16 CONTROL P  */
                CALL DELETE(TRUE);

                /* CASE 17 CONTROL Q  */
                CALL DEL$N;

                /* CASE 18 CONTROL R  */
                CALL DISPLAY$RM$O$N;

                /* CASE 19 CONTROL S  */
                CALL DEL$O;

                /* CASE 20 CONTROL T  */
                CALL CONT$FILL;

                /* CASE 21 CONTROL U */
                CALL COPY$ON(TAB,FALSE);

                /* CASE 22 CONTROL V  */
                CALL ESCAPE;

                /* CASE 23 CONTROL W  */
                CALL BEGIN$WRD;

                /* CASE 24 CONTROL X  */
                CALL DELETE(FALSE);

                /*CASE 25 CONTROL Y */
                DO;
                CALL P$MOVE$ON;
                CALL MOVE$TO$OLD;
                END;

                /* CASE 26 CONTROL Z  */
                CALL COPY$ON(READC,TRUE);


     END;
ELSE  /* CHECK SPECIAL CASES */
IF CHAR = RUBOUT THEN
   CALL BACKUP;
ELSE
IF CHAR = EOC THEN    /* INDICATES END OF CODED INFO */
      DO;
      CALL PRINTCHAR(CHAR);
      CALL MOVE$TO$OLD;
      CALL END$DL;
```

```
        END;
    ELSE
    IF CHAR = ERR THEN
          DO;
          CALL PRINTCHAR(CHAR);
          CALL MOVE$TO$OLD;
          CALL END$ERR;
          END;
    ELSE
    IF CHAR = EOP THEN /* END OF PROMPT INFORMATION */
          DO;
          CALL PRINTCHAR(CHAR);
          CALL MOVE$TO$OLD;
          CALL END$REC;
          END;
         ELSE
    IF CHAR = END$FILE THEN /* END OF FILE */
          DO;
          CALL PRINTCHAR(CHAR);
          CALL MOVE$TO$OLD;
          CALL END$F;
          END;
        ELSE
       DO;
          CALL PRINTCHAR(CHAR);
          NBUF(NPTR:=NPTR+1)=CHAR;
          IF NOT(INSERT) THEN OPTR = OPTR + 1;
          IF NPTR = 72 THEN CALL PRINTCHAR(BELL);
        END;
    END;  /* DO WHILE */

    /* ARRIVE HERE IF BUFFER FULL */

    CALL PRINTCHAR(BELL);
    ENDEDIT1: CALL MOVE$TO$OLD;
    ENDEDIT2:
    END LEDIT;

     /*    START MAIN PROGRAM HERE    */

    OLD$BUF = (NEW$BUF := .BUFFER)+90;
    OBUF = 0;
    CALL MOVE(.'DAT', DAT$FCB+9,3);
    DFCB,DFCB(12),DFCB(32) = 0;
    IF MON2(17, DAT$FCB) <> 255 THEN /*FILE EXISTS */
      DO;
       CALL PRINT(.' FILE ALREADY EXISTS  $');
       GO TO BOOT;
      END;
    IF MON2(22, DAT$FCB) = 255 THEN
       DO;
        CALL PRINT(.' OUT OF DIRECTORY SPACE  $');
        GO TO BOOT;
       END;
    IF (DCNT:=MON2(15,DAT$FCB)) = 255 THEN /* CAN'T OPEN */
         CALL ERROR(2);
    CALL MON1(12,0);    /* LIFT READ WRITE HEAD  */
    /*  ARRIVE HERE WITH NEW FILE CREATED   */
    DFCB(32) = 1; /* RESERVE FIRST RECORD FOR RECORD MAP */
    TMEM = (SBDOS - 1) AND MOD$128$MASK;
    CALL FILL(.MEMORY,TMEM,0);
    SPTR,RECORD$MAP = .MEMORY;
    BSTORE = .MEMORY + (128 * 12);
    STORE = BSTORE;
    SPTR = STORE - 128;
    NUM$REC = 12;
    CALL PROMPT(0);
    DO FOREVER;
       CALL LEDIT;
    END;
EOF
```

BIBLIOGRAPHY

1. Burns, J. C., "The Evolution of Office Information Systems," Datamation, v. 23, p. 60-64, April 1977.

2. Digital Research, CP/M Interface Guide, 1976.

3. Digital Research, An Introduction to CP/M Features and Facilities, 1976.

4. Holyoak, J. G., LT, USN, A Shipboard Report Origination System Utilizing a Microcomputer, M. S. Thesis, Naval Postgraduate School, 1976.

5. INTEL Corporation, 8008 and 8080 PL/M Programming Manual, 1975.

6. Naval Electronics System Command, Test and Evaluation Report X/C 13 Increment I (COMPREP), 30 June 1975.

7. Office ot the Chief of Naval Operations, NWIP 10-1(E), Operational Reports, 1 September 1974.

8. Office of the Chief of Naval Operations Instruction C3501.66A, NAVFORSTAT Reporting Guide (U), 5 January 1976.

9. Tollefsen, T. S., LCDR, USN, "Reports or Readiness: A Dilemma," Naval War College Review, v. 26, p. 74-82, May-June 1974.

10. Wohl, A. D., "What's Happening in Word Processing," Datamation, v. 23, p. 65-71, April 1977.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Chairman, Code 52 Computer Science Department Naval Postgraduate School Monterey, California 93940	2
4.	LCDR S. L. Holl, Code 52Hl Computer Science Department Naval Postgraduate School Monterey, California 93940	1
5.	Assoc Professor U. R. Kodres, Code 52Kr Computer Science Department Naval Postgraduate School Monterey, California 93940	1
6.	LCDR J. A. Dollard, Code 306 Naval Personnel Research and Development Center San Diego, California 92152	1
7.	MAJ F. J. Muller, Code MCC 010 Headquarters Battalion Headquarters, United States Marine Corps Washington, District of Columbia 20380	1

8. LCDR J. B. Godley 1
USS Shasta (AE-33)
Fleet Post Office
San Francisco 96601

Thesis
G5235 Godley
c.1 A microcomputer based generator of recurring operational reports.

17125