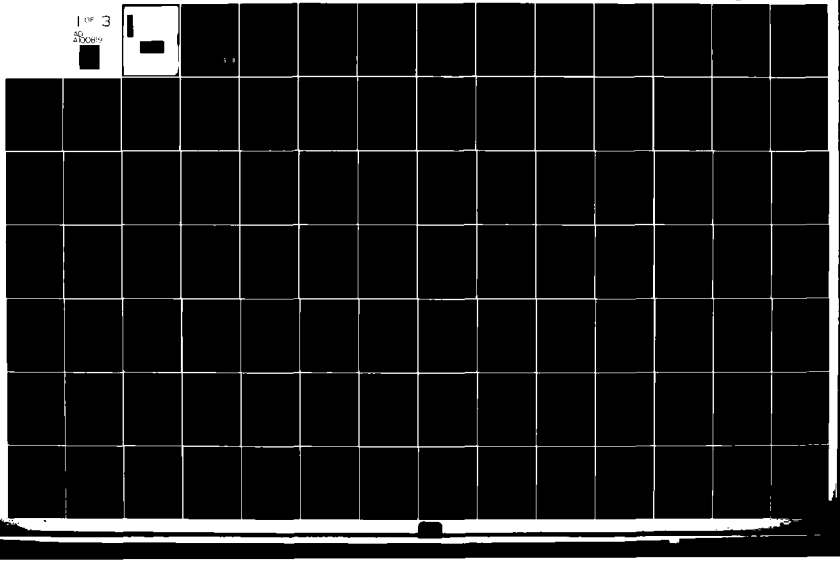
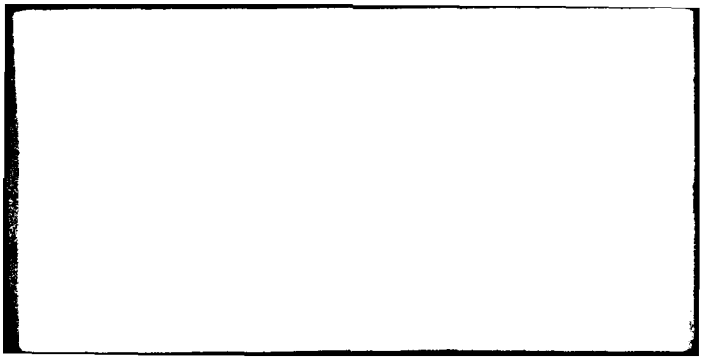
AD-A100 819 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 9/2
CONSTRUCTION OF A GENERAL PURPOSE COMMAND LANGUAGE FOR USE IN C--ETC(U)
SEP 80 W D GRIESS
UNCLASSIFIED AFIT/GCS/EE/80S-15 NL





CONSTRUCTION OF A GENERAL PURPOSE COMMAND LANGUAGE FOR USE IN COMPUTER TO COMPUTER DIALOG

THESIS

AFIT/GCS/EE/80S-15 Wayne D. Griess
Captain USAF

Approved for public release; distribution unlimited.

AFIT/GCS/EE/80S-15

CONSTRUCTION OF A GENERAL PURPOSE COMMAND LANGUAGE FOR USE IN COMPUTER TO COMPUTER DIALOG

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

by

Wayne D. Griess, B.S.
Captain USAF
Graduate Computer Systems

September 1980

Preface

The AFIT School of Engineering installed, in April 1980, a Data General ECLIPSE S/250 and NOVA 2/10 computer system to serve as the foundation for a digital signal processing facility. To expand the processing capability of the new facility, a means to interface the NOVA/ECLIPSE computers with the locally accessible Aeronautical Systems Division CDC CYBER computer system was desired. I undertook this project and constructed a general purpose command language that could be used in varied applications.

I heartily thank Captain Larry Kizer, my thesis advisor, for his steady encouragement and sure support. I also thank Lt Colonel James Rutledge and Professor Gary Lamont for their support. On several occasions, Mr. Hurst D. Carlstrom of the Avionics Laboratory provided invaluable and timely assistance. His experience and knowledge of the NOVA/ECLIPSE computers were eagerly exploited. Finally, I thank my wife Betty and my two children for their patience, sacrifice, and love. They have been a source of strength and sustenance throughout this project. I also reverently give thanks and praise to my Lord and Savior Jesus Christ for His providence and counsel through it all.

Wayne D. Griess

Contents

List of Figures

AFIT/GCS/EE/80S-15

Abstract

Two computer programs were developed and implemented to enable intercommunication between a Data General NOVA/ECLIPSE computer system and another modem linked computer system. One program, called TTERMOP, allows a user to sit at a NOVA terminal and interact with a connected system in a transparent mode. The other program, called MONITOR, is a command language interpreter that examines and executes instructions contained within an action file. An action file, consisting of instruction strings and associated control parameters, is designed to be dependent upon a connected system with regard to contents, yet independent of such a connected system with regard to structure and format. The interpreter is written in FORTRAN IV with FORTRAN and assembly language modules. Actual implementation of the programs is accomplished between the NOVA/ECLIPSE and the Aeronautical Systems Division Control Data CYBER computer system. ASCII data files between 20 and 35,000 bytes have been transferred between the two interconnected systems, each transfer initiated by a single string command acceptable to the interpreter and compatible with a tailored action file for the CYBER system. The programs were designed to be flexible enough for use with several different connected systems, and general enough to be hosted on a system other than the NOVA/ECLIPSE. However, no attempt is made to implement the programs outside of the NOVA/ECLIPSE - CYBER environment.

CONSTRUCTION OF A GENERAL PURPOSE COMMAND LANGUAGE FOR USE IN COMPUTER TO COMPUTER DIALOG

I. Introduction

Background

Under sponsorship of the United States Air Force Electronic Systems Division and the United States Air Force Aerospace Medical Research Laboratory, the Air Force Institute of Technology (AFIT) Electrical Engineering (EE) department is undertaking research in digital signal processing, specifically digital speech processing and digital image processing. A Data General Corporation (DGC) NOVA 2/10 computer, a DGC ECLIPSE S/250 computer, and associated peripheral equipment are being combined and integrated to form the signal processing facility.

To expand the local processing capability of the facility, the EE department proposed to interface the DGC computers with the larger and more sophisticated Control Data Corporation (CDC) CYBER computer system, which is operated by the United States Air Force Aeronautical Systems Division (ASD) and which provides support to AFIT. (Both AFIT and ASD are located at Wright-Patterson AFB, Ohio.) Interfacing the two systems would allow intercommunication between the two systems, wherein the advantage of each system's features could be utilized. The CDC CYBER system, consisting of a dual CYBER 175 and a CYBER 750, could readily provide the "number crunching" and file repository required to help analyze signals. The NOVA/ECLIPSE system could then be devoted to other signal processing functions, such as

analog to digital/digital to analog conversions. Furthermore, though the systems would be capable of intercommunication, the failure of one of the two systems would not mean the failure of the other system. Thus, each system could stand alone -- or be interconnected for supplementary processing power.

Expanding the capability of the signal processing facility by interconnection to the CYBER computer system may not be the only expansion possible. Many other computer facilities are available in the AFIT School of Engineering, as well as throughout Wright-Patterson AFB. The EE department also proposed, therefore, that the method of interfacing the local NOVA/ECLIPSE computers be flexible enough to be used with systems other than the CYBER, and that the interfacing be general enough to be applied by separate and distinct systems, if possible.

Problem Statement

Interfacing the NOVA/ECLIPSE computers to the CYBER system or any other system may be viewed in two parts. One part of the interfacing would be to connect the NOVA/ECLIPSE computers to the CYBER system just as any other peripheral would be connected or appear to be connected. In this case, the CYBER system would treat the NOVA/ECLIPSE computers as one of its many terminals, not recognizing that it is a complete computer system. This type of interface is not complex, and involves accessing an input and output port on the NOVA or ECLIPSE via a data line of the local telephone system to the CYBER system. Connection to

the CYBER system would be accomplished in the same fashion as other terminals are connected. A user would call a prescribed telephone number and couple the telephone line to the data rt line via a standard modem. After connection, a resident software program designed for the purpose would be called into execution to control input and output transfers. Any local terminal of the NOVA or ECLIPSE, whichever owned the selected data port, would appear transparently as a terminal directly connected to the CYBER.

The limitations of this first part are the same as those limitations any typical terminal has when connected to the CYBER system. First of all, the language for communication must be that of the "host" system, and in this instance, it is the CYBER operating system. Presently, the CYBER operating system language is called NOS/BE (Network Operating System/Batch Environment). The users of the signal processing facility desire a language of intercommunication other than NOS/BE, that simplifies intercommunication and reduces the complexity of user access. Furthermore, they desire that such a language be understandable and portable, i.e., that may be used on more than one machine for the same purpose. A second and even greater limitation of the transparent terminal interface is that no direct computer to computer dialog may take place. Under a transparent terminal operation, users may access files of the CYBER or connected system, but cannot access local files. Also, there is no provision for the direct exchange of information, such as file transfers, in the transparent mode.

A second part of interfacing, then, would be to connect the NOVA/ECLIPSE computers to the CYBER system or any other system such that direct information exchanges are possible. Within this type of environment, a user would have access to files on the NOVA/ECLIPSE and the connected computer system. For example, files from the local system could be transferred to the connected system, executed, and returned. This type of interfacing is significantly more complex, even though basic connection techniques remain the same, since files must be identified and manipulated. Because of this need to manipulate files, the software program to control information transfers becomes much more complex particularly; and, hence, requires the greater concentration of design and development efforts. Just as in the case of the transparent mode, the users of the signal processing facility desire a language of intercommunication that is understandable and portable. The language must be simple and provide a means to issue commands both to the local NOVA/ECLIPSE system and the connected computer system, such as the CYBER.

The problem, then, is to develop a method of intercommunication for use in computer to computer dialog. In its narrower focus, the problem becomes one of developing a command language on the DGC NOVA/ECLIPSE computer system for intercommunication with the CDC CYBER computer system.

Scope

An examination of this problem involves researching computer to computer interconnections, with particular emphasis upon software development. The compatability of operating systems is briefly considered, with respect to the degree of interconnection required at the software level. The scope of the study also includes the guidelines that surround computer to computer dialog, humanized input and output, and man-machine interfaces. Though hardware considerations in the actual implementation will be considered, the study is not oriented toward hardware, nor will hardware be examined in any depth. Further, the study will not specifically be concerned with the actual digital signal processing capabilities of the computer system. Although the topics above will be considered, the study will be devoted almost exclusively to the analysis, design, development, implementation, testing, and review of software for interfacing the NOVA/ECLIPSE computers to a connected computer system (specifically the CDC CYBER) via a defined command language.

General Approach and Preliminary Results

The first step in constructing a command language for use on the NOVA/ECLIPSE computers was to search the literature for ideas and projects of a similar nature. Few articles were found that directly impacted upon the project at hand, but methods and parameters for interfacing systems in general were investigated.

Other thesis efforts were reviewed for applicability to this effort, and possible expansion (Ref 4 and 8). The literature investigation was followed by a period of machine familiarization, both of the NOVA/ECLIPSE and the CYBER. As the NOVA/ECLIPSE computer system was installed just prior to the full scale thesis effort, much time was spent learning the characteristics of the NOVA/ECLIPSE Real-Time Disk Operating System (RDOS). Initially, programs taken from operating manuals were copied and executed. Later, original programs were developed to duplicate the same actions. These initial efforts concentrated on assembly language facilities, as these were the least familiar. The CYBER had been used before and was not unfamiliar. Nonetheless, operating system characteristics were examined in more depth.

Design of the software to implement a transparent terminal operation was next. It is at this point that the details of interrupts and their operation were examined and tested in experiments. Once the transparent mode was workable from a local terminal to another local terminal, efforts turned to developing the actual command language. Using top-down design techniques and successive refinement, individual modules were developed as needed to implement parts of the language. Early on, the project was divided into two parts, the output from the NOVA/ECLIPSE and the input to the NOVA/ECLIPSE. Multitasking was used to allow asynchronous operation of these two parts. A further division was made to the design. An action file was created to actually contain command sequences as needed by the

user. The main program became an interpreter to examine and act upon this action file. Once a working subset of the final product was available, access to the CYBER system was tried. The transparent mode software required minor modifications and transitioned well to on-line execution. The transition of the command language itself was more painstaking and slow.

Eventually the modules were all developed and combined, and then several tests and trials were conducted. Once the program was in a reasonably workable state, efforts began to more fully and comprehensively document design, development, implementation, and validation findings. Finally, the project was put into written form and a user's manual was written to instruct users in the use of the command language. Follow-on steps to this process are recommended in the Conclusion, Chapter V.

Sequence of Presentation

The introduction to this project is followed by an analysis of the literature regarding man-machine interfaces and command language structures, an analysis of the computer systems involved, the transparent terminal operation mode, and the command language itself. The analyses are followed by an explanation of the development theory behind the software programs. Once the theory has been presented, the actual development of the software will be discussed, concentrating first on a program called TTERMOP -- the transparent terminal operation mode software, and then a program called MONITOR --

the actual command language mode software. The discussion of MONITOR will include a look at various subordinate modules and routines that comprise MONITOR. A section regarding validation of the software will follow the software development description, including its current state and usefulness. The project will then be summarized, pointing out several possibilities for follow-on work to this thesis.

II. Detailed Analysis

An analysis of the problem of interconnecting the NOVA/ECLIPSE computers to another computer system begins with a look at interfacing in general. Several elements of the literature are examined with respect to the concepts of man-machine interfaces, computer to computer dialog, and command languages. This introductory look at the problem is followed by an introduction to the NOVA/ECLIPSE computers and their primary features. Once the computers have been described, a closer look at the methods of interfacing for the NOVA/ECLIPSE are mentioned, particularly with regard to the modes of operation envisioned.

Man-Machine Interface

It is becomming increasingly important, though it has always been important, that there be a proper perspective with regard to the interaction of people with computers. The past emphasis seems to have been on the usage of computers; the newer emphasis is on the usage of people (Ref 5:4). This change in emphasis is an outgrowth of the volumes of information computers generate and the ever wider proliferation of computers. How do, or how should, the two cooperate? James Martin states:

> This difference in thinking talent -- the computer being good for ultrafast sequential logic and the human being capable of slow but highly parallel and associative thinking -- is the basis for cooperation between man and machine (Ref 5:7).

The logical answer to cooperation, then, lies in utilizing both people and computers in a manner consistent with their strengths. It is obvious, therefore, that people should not be called upon to do the kind of work that a computer can do better. It should be equally obvious, as well, that computers should not be called upon to be substitute people. The future will undoubtedly lead to ever greater computer maturity, to the point where some human functions can be paralleled or duplicated. The present, however, would seemingly be better served if people and computers, each with their relatively mutually exclusive spheres of advantage, were deliberately and constructively meshed together.

With this view in mind, human beings could more profitably gain from computers if computers were creatively linked to enhance their speed, throughput, and intercommunication. And, such computer systems would better profit people who use them, if their output and input facilities were readily understood, easily manipulated, functionally controlable, and consumer dependent. The point where interaction culminates in identifying the most effective techniques for enabling a user to access needed data quickly, easily, and in a relatively natural manner, is the user-computer system interface, called the man-machine interface (Ref 3:1-1).

Computer to Computer Interface

Linking computers together is an action to improve upon the individual computer's capability and to provide a more powerful resource for users. Just as input and output data is a dominant factor in the application of single computers (Ref 10:119), so it is with multiple computer systems. It should not be surprising, therefore, that intercomputer or interprocessor communication is primarily at the data level (Ref 12:67).

Interfacing, or linking computers together, is widely discussed throughout the computer literature. Many types of interfacing are possible, and many terms are used to describe differing methods and degrees of interface. An article in Computing Surveys presented a naming scheme or taxonomy for identifying various systems of interconnected computers (Ref 2:197-213). The scheme was presented as a tree diagram of four levels, wherein the two highest levels were concerned with strategic (policy) issues and the two lower levels were concerned with tactical (implementation) issues. The authors defined two terms of interest with regard to communication interconnection. The first term, path, is a medium by which a message is transferred between the other system elements, such as wires or buses. The other term was switching element, an "intervening intelligence" between the sender and receiver of a message. The switching element affects the destination of the message in some way.

Within this scheme, communication interconnection is either direct or indirect. A transfer path can be either dedicated or shared, and a variety of system architechures may be employed to interconnect computers. As an example, a multiprocessor system is described as direct, having no need for a transfer control method, and using a shared path transfer structure with a central memory system architechure. As another example, an irregular network system architecture follows from an indirect communication interconnection, decentralized routing, and a shared transfer structure. In the authors' scheme, the variety of interconnections may range from a complete, dedicated interconnection to an indirectly, decentralized, shared path window. Nonetheless, the common element used to describe all is the data level communications path.

In any interconnected computer system, a protocol is necessary. "A protocol is essentially a set of conventions between communicating processes on the format and content of messages to be exchanged (Ref 1:4-3)." The protocol can be easily determined in some computer links by simply adapting the internal protocol of one of the individual computers.

In all cases, computers are interconnected to benefit the users. The better and clearer the interconnection of computers, the less likely the confusion between man and machine. Further, the more likely will be the usefulness of the system for the people it was meant to serve. From simple interconnections of a dedicated nature to large networks, computer to computer communications is dealt with at the data

level and is the starting point for future interconnection efforts.

Command Language Interfacing

The problem of interfacing the NOVA/ECLIPSE computers with other computer systems is one of command language interfacing, in which the goal is to construct a general purpose command language that provides the means for computer to computer dialog. This is another element of computer to computer interfacing, in which the connecting feature is a command language itself. Command languages collectively form a category of intercomputer connectivity, in which the type and purpose of a command language may vary widely from use to use. One of the most widely recognized command languages is that of the UNIX (trademark of Bell Laboratories) Time-Sharing System developed by the Bell Telephone Laboratories (Ref 11:1905), in which the most visible system interface is the "shell" or command language interpreter, through which other programs are called into execution singly or in combination (Ref 6:1900). The UNIX shell is a high-level programming language that provides users with an interface into process related facilities of the UNIX operating system. The language is powerful, concise, flexible, and, once it becomes familiar, easy to use. It serves as a useful basis of comparison for other command languages and was a source of guidance for this project. However, the degree of difference between the developed command language for the NOVA/ECLIPSE system and the UNIX shell is singularly

significant. To meet the particular constraints of this project, the NOVA/ECLIPSE command language was developed as an interpretive process that examines tailored files, called action files, for individually connective systems. In fact, the pattern of development parallels the so-called PROCEDURE files utilized within the CDC NOS/BE (Ref 7:5-21 - 5-38).

The CDC PROCEDURE files allow several CDC CYBER commands to be combined and ordered as desired into a single package for execution. Each file has a header and body, of which the header statement starts the header. The header starts with the keyword PROC., contains the name of the procedure, and ends with the names of arguments to be used within the procedure. The next statements of the file are the body, and the body is terminated with an end-of-record. The body is made up of control statements, which are inserted into the job control stream when the PROCEDURE file is called into execution. The PROCEDURE file is called into execution by a call-by-name statement or by a call statement that begins with BEGIN. The operating system makes the appropriate parameter and variable substitutions within the PROCEDURE file, and then executes the file's body of statements. Thus, via PROCEDURE files, users of NOS/BE may combine a sequence of control statements (commands) into a single command with a variable number of arguments. The entire sequence may be executed by a single reference to the name of the PROCEDURE file, eliminating the need for a user to repeat minor, often used commands, such as REWIND. It also allows the computer operating system to keep track of the entire command

sequence without user intervention.

The NOVA/ECLIPSE Computers

The DGC NOVA and ECLIPSE computers are both 16 bit machines installed within the digital signal processing facility of the AFIT EE department. The NOVA 2/10 and the ECLIPSE S/250 share a ten megabyte disk through an Inter-Processor Buffer, which arbitrates simultaneous disk access. Each computer operates under a Real-Time Disk Operating System (RDOS), which is partially resident in core memory as well as on the disk itself. Each operating system starts with generation of SYSGEN, a program that permits the RDOS to be tailored to the hardware and software configurations available at a location. This system generated program includes, among other things, the number of printers, the number of teletypes, and the number of other devices to be connected to and recognized by the operating system. The SYSGEN created program for the ECLIPSE is called ESYS; the SYSGEN created program for the NOVA is called NSYS. NSYS, however, includes the generation of a secondary teletype for both input and output, denoted by device codes $TTI1 and $TTO1, respectively. Once a device code is system generated, the operating system identifies the interrupts of the device and appropriate handling procedures. This investigation required user defined interrupts and handlers for these two particular devices. Hence, another program was generated, called NSYS1, in which the operating system (RDOS) did not "know" about device codes $TTI1 and $TTO1.

Though both the NOVA and ECLIPSE may share disk files, there are differences in their hardware features and capabilities. The AFIT EE department decided to use the NOVA computer as the interface link to any other connected system and most peripherals, thereby freeing the larger and faster ECLIPSE for actual processing of data. In the remainder of this thesis, therefore, the term NOVA/ECLIPSE will be used to indicate that both the NOVA and ECLIPSE computers are available, yet only the NOVA is utilized in the actual interfacing to the connected systems. Thus, all device codes and descriptions may be thought of as pertaining only to the NOVA 2/10.

In order to connect any other computer system to the NOVA/ECLIPSE computers, access is required to the computers themselves. It was decided to make use of standard RS-232 interface connectors and modem links, wherein the NOVA device codes of $TTI1 and $TTO1 would serve as device ports for the connected system. Information from the NOVA/ECLIPSE to the connected computer system would be transmitted via device code $TTO1. Information from the connected computer system to the NOVA/ECLIPSE would be transmitted via device code $TTI1. Because of the way RDOS operates, each device code used must be assigned an integer channel number. Accordingly, throughout the software and documentation that is developed, device codes and channel numbers will refer to the same entities, particular device codes and individual channel numbers being equated at various times.

The basic method of interaction with the RDOS is through the Command Line Interpreter (CLI), a program that accepts command lines from the console and translates them into RDOS commands. Through the CLI, high-level compilers like FORTRAN may be invoked, as well as the assembler and other utilities. Interface into the RDOS can also be achieved via system calls and task calls. Both of these calls were utilized within the development of the software for this project. Essentially, the RDOS can be addressed directly via system calls in a program. The general form of a system call is (Ref 9:3-1):

```
.SYSTM
command mnemonic
error return
normal return
```

The mnemonic .SYSTM precedes each system command, which passes control through the RDOS task scheduler to the system call processor, a core-resident portion of RDOS. The task monitor saves the program (or task) environment in a special block called a Task Control Block (TCB) and saves the contents of location 16, the User Stack Pointer (USP), before passing control to the call processor. Upon execution of a system call, the RDOS takes the error return if it encounters some impediment while executing the call. Accumulator two contains an error code that describes any error condition and may be displayed by the user. If the call succeeds, the RDOS takes the normal return. System calls are detailed in the RDOS Reference Manual, Chapter 3 (Ref 9:3-1 - 3-12). A task call is similar to a system call, except for the following points. Task calls have no .SYSTM mnemonic before the task command mnemonic. The RDOS

executes task calls in user address space, whereas system calls are executed in operating system space. And finally, many task calls have no error returns. The details of task calls may be found in the RDOS Reference Manual as well (Ref 9:5-1 - 5-8). Other conventions of the RDOS and NOVA/ECLIPSE computers will be introduced and explained as encountered in the discussion of the project development.

Modes of Operation

The most direct method of connecting the local NOVA/ECLIPSE computers to any other computer system, particularly the locally accessible CDC CYBER system, is via a standard RS-232 interface link to one of many available modems. The simplest method of interconnection is to allow a local NOVA terminal to act as a transparent CYBER terminal, i.e., as if the NOVA terminal were directly connected to the CYBER and not the NOVA. This requires a software interface that allows the NOVA computer to accept data in from the CYBER, display to the local terminal, and vice-versa. This at least allows access via the NOVA computer to the connected computer system, but it does not allow file transfers or any other direct intercommunication between systems. Thus, to extend the concept further, additional software is needed to enable direct access from one computer to the other. One method of doing this is to create a program to mesh the NOVA with a particular system to the degree that files on either system are accessed by the user on the NOVA. It was recognized early that such software should be designed to be

independent of a particular system. Hence, a more general software product was desired that allows interaction between the NOVA/ECLIPSE and a variety of other computer facilities.

Since each computer system is somewhat different, a method was needed to confront these differences in a general fashion by the local NOVA/ECLIPSE system. If a particular segment of the software was designed to be compatible with a particular connected system, then the NOVA would have to be able to select that software and use it on demand. Further, the act of selection should not tie the NOVA to the selected system, as a different system may be desired at a later time. To meet these constraints, the pattern of a so-called action file was conceived, based upon the general concepts of a PROCEDURE file as utilized in the CDC NOS/BE.

To extend this concept to the local NOVA/ECLIPSE system, both the action file (patterned after the PROCEDURE file) and a method of reading and acting upon the action file was required. Once one or more action files could be constructed on the NOVA system, how could they be utilized? In the simplest context, the action file may be thought of as a simple list of command sequences to be executed by the system. A method was needed to read each sequence of instructions and send them to the connected system to be executed. This presented additional considerations. First of all, the action file sequences, if more than one, must be distinguishable from each other. When and where would one sequence start and end? Secondly, the connected system may respond at some time and in some fashion to

inputs, perhaps requiring that responses somehow be acknowledged. Thus, what responses would be expected and how or when should they be acknowledged? Thirdly, sending instructions to the connected system limits the local system in taking independent action. Thus, there must be some capability of instructing the local system in the midst of action file execution. Each of these considerations and questions led to the idea of an action file interpreter -- a program that would seek out the desired action file, initiate and terminate selected command sequences, allow for responses to the connected system, and alert the local NOVA/ECLIPSE system to necessary inputs and outputs. Thus, the basic idea for computer to computer dialog regarding the NOVA/ECLIPSE system revolved around the design and development of action files peculiar to possible connected systems and a general interpreter for the action files. The process of designing these began by considering a specific system to work with, and the natural choice was the CYBER system.

III. Development of the Program

The development of the software to permit the NOVA/ECLIPSE computers to be interconnected to another computer system was done in a top-down manner, using successive refinement. This chapter describes the software development of three specific software parcels: (a) program TTERMOP, (b) program MONITOR, and (c) the action files. The program TTERMOP is a program that allows the user on the NOVA/ECLIPSE terminal to operate in the transparent terminal only mode. The program MONITOR is the command language interpreter that reads and acts upon selected action files. The action files are created in a prescribed manner that meets the expectations of the interpreter. The action file referenced within this text is that created and implemented to intercommunicate with the CDC CYBER computer system. The development of MONITOR and the action files was interdependent and cannot be artificially separated. Nonetheless, the following text does treat the development of each somewhat separately, first describing the generation of the action files and then the program MONITOR. Before these discussions are presented, some theory and background concerning the development are presented, in order to acquaint the reader with the overall concepts and implementation conventions.

Development Theory and Background

After analyzing the problem and deciding that three partitionable software products were to be developed -- program TTERMOP, program MONITOR, and the action files -- the alternative methods of accomplishment and the tools to be used in the development were considered. Some factors bearing upon the considerations were the stated objectives of the solution to the problem itself. The program MONITOR should be as flexible as possible, so as to be utilized with as many connected systems as possible. Also, program MONITOR should be as general as possible, so as to be a completely portable (to the degree possible) program that could be hosted on other systems, in addition to the NOVA/ECLIPSE. This dictated that the action files be dependent upon the connected system for content, but independent of the connected system as far as structure. One final factor was to develop a system that enabled the NOVA/ECLIPSE computers to interact with the CDC CYBER computer system.

In consideration of these factors, the software needed to be developed in as high-level a language as permissible, thereby supporting its generality. PASCAL was a logical choice. A structured language, such as PASCAL, leads to simplified data structures and more straightforward file structure manipulations. In addition, PASCAL has powerful features, such as recursion, that enable complex algorithms to be more readily designed and developed. Nonetheless, PASCAL was not available

on the NOVA/ECLIPSE computers at the time of development. Other structured languages that seemed to lend themselves to this type of undertaking were not available either. FORTRAN IV and FORTRAN V were available. FORTRAN IV was selected because it was available, is a high-order language, and is, perhaps, as universal a language as exists. FORTRAN V was not selected because of the case for FORTRAN IV and the fact that FORTRAN V is essentially a superset of FORTRAN IV. Thus, if a user chooses, FORTRAN IV may be extended to FORTRAN V. It also seemed reasonable at the outset to expect to use some assembly language programs to implement device handlers and routines that depended heavily upon the operating system characteristics. Thus, both the DGC FORTRAN IV and assembly languages are used to develop the software parcels. Some variations from the American National Standards Institute (ANSI) standard FORTRAN were utilized, but only to enhance readability and understanding. None of the uses would preclude adaptation to other system FORTRAN versions, depending upon the assembly language that may be required for another system.

Transparent Terminal Mode. The idea to create a transparent terminal mode of operation was two-fold. First, the terminal mode would at least allow access to another system. Depending upon the ultimate limitations of a command language, such a mode would be desirable to permit detailed interaction with a connected system not possible at the higher level of the command language. Further, such a mode would serve as a handy option for transitioning in the middle of any command langauge

execution. Secondly, the development of a terminal mode would be a prelude to other developments. Such a prelude would provide system familiarization and understanding in a less complex environment.

The first and major obstacle in developing the program TTERMOP was the method of interaction to be implemented between connected systems. The transparent terminal mode needs to be controlled by the user at the local NOVA terminal. Outputs to the connected computer system are generated by entries made by the terminal user. However, responses from the connected system are less controlled, and, more generally, may vary from one system to another. There may be responses during the access initiation process, after individual instructions are received, or upon other occasions unexpected by the user. Implementation of interrupt servicing seemed the logical alternative to overcome this control problem in the most general case. This, in turn, also lead to a problem in implementation, for interrupts are usually serviced by the operating system for devices known by the system, i.e., generated when the system is initially brought up to operation. In order to use interrupts, then, either the system generated interrupt service routines had to be changed to allow different interrupt handling, or the interrupts had to be removed from the purview of the operating system and be generated by the developed software. The latter alternative was chosen, since changing the operating system handlers was more complex and would still leave the handlers in the system. Other users with other programs would not

necessarily find changed operating system handlers useful or expected. As mentioned in Chapter II, the device codes to be used for purposes of intercommunication are $TTO1 and $TTI1. Both these devices and their codes are removed in the system generated program NSYS1. Thus, development of TTERMOP was now possible using defined interrupts and handlers within the software itself.

A second obstacle existed that hindered TTERMOP development. Even though interrupts were available to permit servicing for the two NOVA/ECLIPSE port devices ($TTO1 and $TTI1), the program software needed to respond to both the local terminal inputs and outputs -- interrupts that remained defined by the operating system -- and the connected system inputs and outputs -- interrupts that were program defined. The possibility existed that the program might be servicing one of these sets of inputs/outputs, and lose inputs/outputs from the other set. Some method of asynchronous interaction was needed to insure both sets of inputs/outputs were equally and promptly handled. The NOVA/ECLIPSE system has such a capability, called multitasking (Ref 7:5-1+). Multitasking permits a single program to contain multiple, competitive tasks. A task is a complete, self-contained execution path through a program, which demands system resources. In a single task environment, a program has a single unified path connecting all its program logic, no matter how complex the logic branches. In a multitask environment, a program may have two or more logically distinct tasks, each with its own priority. Each of these

distinct tasks performs a specified function asynchronously and in real-time. In the RDOS, the Task Scheduler allocates central processor control to the highest priority task that is ready to perform its function. The multitasking feature of the RDOS is used, then, to put the interrupts for the connected system into one task for processing, while the interrrupts for the NOVA terminal are put into a logically distinct task. To insure the user maintains control, the task that controls the NOVA terminal is given the higher priority.

The Action Files. The concept for the action file came directly from the CDC NOS/BE PROCEDURE files (Ref 7:5-21 - 5-38). A separate action file was conceived for particular computer systems to be interfaced with the NOVA/ECLIPSE computers. Each action file contains a sequence of commands, of which each command in the sequence instructs either the NOVA/ECLIPSE or the connected system. These instructions serve the same purpose as the body of the PROCEDURE file, i.e., they provide the control statements necessary for execution of a command. The header statement of the action file was conceived to be essentially the same as that for the PROCEDURE file. It contains a keyword that identifies the action file, such as .CACT for the CYBER action file. The keyword is followed by a command name and argument parameters. To facilitate segregation of each sequence from the next, another keyword is inserted into the file between sequences. This keyword is END.. Similarly, to denote the end of the action file, the keyword FINISH. is used. To allow the action file to be as powerful as

possible, and ,hence, the interpreter as general as possible, a method to incorporate expected system responses is developed. At the beginning of each action file, the control character "I" must be included . It is followed by up to two separate responses that may be expected of the system. (These responses are used in the interpreter to detect the occurrence of system responses and to detect the end of such responses.) Other features of the action files follow logically from the requirements of the program MONITOR.

The Command Language Interpreter Mode. The command language interpreter is developed to serve not only as the interpreter of the selected action files, but as the executive for the entire software development. As such, the program MONITOR is the fundamental part of the software package. Program MONITOR is developed to instruct the interactive user step by step in the execution of the software that forms the command language. Thus, the program first identifies itself via a display on the user's terminal (NOVA/ECLIPSE) and identifies the action files that are available for selection. (Four action files are available to the user, of which only one is totally implemented. The others are shells that await future completion. The complete action file is for the CDC CYBER computer system, called .CACT . Two action files are for computers that are accessible locally, even though connections have not been attempted. They are called .DACT and .VACT . The last action file is completely arbitrary, and is available for any user to generate as desired. This file, called .MACT,

presents the ultimate in flexibility -- the creation of a totally user-defined and user-modifiable action file for any system.) Program MONITOR requests the user to select an action file and then requests the user to LOGON. (Procedures for LOGON and the other commands are contained in the MONITOR User Manual, Appendix A.) To notify the user that a response is required, program MONITOR provides a prompt. Program MONITOR also screens input commands to determine if they are valid; for example, rejecting commands not within the action file, commands that are too long, commands that are without the necessary arguments, and commands that contain syntax errors.

Program MONITOR was developed in a simple configuration to expedite its implementation. In this configuration, commands are required to be entered and followed by exactly the number of arguments expected, that is, the number of arguments indicated after the command name in the action file. Furthermore, commands must be in the same order. There are no optional arguments nor default arguments. This configuration simplifies the decision-making of the interpreter with regard to parameter substitution. Also, there are two control options that the user may exercise at any time in lieu of command strings. One of these is the entry "^T", (an up-arrow and T) which causes the program MONITOR to relinquish control to the terminal operation only mode. The other option is the entry "^L", (an up-arrow and L) which causes a return to the RDOS CLI.

A similar obstacle to that of TTERMOP hindered development of MONITOR. This was the control of inputs and outputs of the

connected system and inputs and outputs of the NOVA/ECLIPSE. The solution was again multitasking, with one task monitoring NOVA/ECLIPSE input/output and the other task monitoring the connected system's input/output. An additional obstacle not encountered before concerned instructing the NOVA/ECLIPSE RDOS from within the program MONITOR. As the program MONITOR executes in RDOS already, some method to exit the program MONITOR is needed in order to instruct RDOS via its own command language, the CLI. It is equally necessary, however, to be able to reenter MONITOR immediately after any instruction is passed to RDOS, and to enter at the location from which the exit occurred. One feasible means of doing this is to swap out the program MONITOR, swap in the RDOS CLI, and then swap right back to the program MONITOR. The process of swapping is available on the NOVA/ECLIPSE system via system and/or task calls to the operating system.

Swapping as implemented on the DGC system is a process whereby a program is called by name into execution. The calling program is swapped out of memory and the called program is swapped into memory. Within the RDOS, the CLI normally operates on what is called level zero. This is the highest of five possible levels of program execution. When a typical program is called into execution by the CLI, the program is executed on level one. In effect, the CLI is swapped out of memory and the executing program is swapped into memory. Going from one level to another is essentially a stack operation. If the CLI is executing on level zero and a program is called into execution,

the executing program is pushed onto the stack. Returning to the CLI at the program's conclusion is essentially a pop of the stack. The latest program on the stack is the level being executed. If the CLI is executing on level zero, it is not possible to execute another program from the CLI without that program starting execution at its beginning. The ability to communicate with RDOS, however, can be accomplished as desired by calling the CLI into operation on level two. Then, when the CLI execution is complete, an appropriate pop of the stack will return to the level one program. Furthermore, the return will be to the location from which the program called the CLI into execution on level two. Executing the CLI on level two can be limited to a single instruction or a group of instructions that are passed at the time of the call to swap to the CLI. The process is described more completely in the RDOS Reference Manual, Chapter 4 (Ref 7:4-2).

Development of TTERMOP

As mentioned earlier, the port device codes $TTI1 and $TTO1 were incorporated into the software of TTERMOP and MONITOR, rather than leaving the operating system to define them in its SYSGEN program. The advantage of identifying the device codes at run time is that new interrupt handlers can be designed to handle interrupts as desired. By doing this, any input to the NOVA/ECLIPSE computers from the connected computer system generates an interrupt via device code $TTI1. The interrupt service routine essentially accepts the input and stores it in a

buffer for later disposition. Similarly, any output from the NOVA/ECLIPSE computers to the connected computer system generates an interrupt via device code $TTO1. This interrupt service routine simply clears the particular interrupt, allowing the connected computer to handle the NOVA/ECLIPSE output in its own fashion. (It should be noted that all input and output is limited to ASCII characters in all the software developed.)

Once the input from the connected computer system is received by the NOVA (where in fact the device ports exist) and is stored in a buffer, a separate section of code disposes of the buffer itself. Two pointers are established. One pointer, called INPTR, points to the next empty location in the buffer. The other pointer, called OUTPTR, points to the last buffer location actually disposed of by program TTERMOP. If the buffer has not been accessed by the program, OUTPTR started and remains at the buffer's beginning. If the INPTR and OUTPTR point to the same location, the program knows that the buffer is "empty." Conversely, whenever the pointers are not the same, input has been received and stored in the buffer. The buffer itself is designed to be 132 characters long. This arbitrary length was selected with the reasonable expectation that the buffer would never overflow, i.e., that the INPTR would not be 132 characters ahead of the OUTPTR at any time. The expectation seemed particularly reasonable when considering the 300 baud rate of the CDC CYBER, used as the implementing connected system. This should be the case for higher baud rates as well, particularly 1200 baud. Even at rates up to 4800 baud, the buffer should be

adequate.

The section of code in program TTERMOP that looks at the buffer continually is named LINERD, and is called as a separate task within the multitasking framework of the program TTERMOP. LINERD has a priority of ten, whereas the main task of TTERMOP, called TERMRD, has a priority of zero. By the conventions of the RDOS, the task with the lower number has the higher priority. The main task TERMRD reads input from the NOVA terminal and transmits it directly to the connected system via device code $TTO1. The task LINERD simply takes input from the input buffer, and displays it to the NOVA terminal. In operation, then, the user generates output and usually awaits input. Since the output program has higher priority, the user controls the program. While the user awaits input, the lower priority task LINERD seizes control of the processor to look for that input. At rates up to at least 1200 baud, both tasks appear to operate simultaneously and are in fact seizing control on a character by character basis.

Program TTERMOP has several parts to it. The program contains the separate, logical tasks LINERD and TERMRD. The program also has the system calls to identify and generate device codes $TTO1 and $TTI1. Finally, TTERMOP has the interrupt handlers defined within its source code also. Altogether, the program provides the necessary software to make the NOVA/ECLIPSE user operate off of the connected system, in particular the CYBER system, transparently. Since the majority of the functions dealt with in this program are at the device

driver level, the entire source code is assembly language. A higher-level flowchart of the program TTERMOP is not explicitly provided. However, a flowchart for program TERMOP contained in Appendix B is exactly the same for TTERMOP, and may be referred to for information concerning TTERMOP.

Program TTERMOP is executed by entering the directory DIALOG and typing on the NOVA terminal TTERMOP. Once a telephone connection via a modem is established between the CYBER and the NOVA/ECLIPSE, all subsequent terminal action is as if the NOVA terminal were directly connected to the CYBER. All NOVA/ECLIPSE operations are transparent to the user during the execution of the program TTERMOP. Exit from TTERMOP may be accomplished by typing in an up-arrow "^" at any time, thus reverting the user to the RDOS CLI.

Development of the Action Files

One of the very first things considered when starting the construction of the general purpose interface for the NOVA/ECLIPSE computers was the minimum number of instructions required as input by a connected computer system, in order for that system to execute functions on its own file structure. This minimal set of instructions formed the initial action file and served as a forerunner of the final product. As stated in Chapter II, the CDC CYBER PROCEDURE files were the pattern behind the design of the action files. Thus, the initial action file was closely structured after the PROCEDURE file. Both files have header statements that declare the name and arguments

of a particular command sequence. Both files have a body of statements that consist of the control command sequences, and both files have all entries arranged sequentially. This sequential arrangement is, in turn, the logical and convenient means by which to examine the action files. Once the interpreter selects an appropriate action file and receives a command input, the file is sequentially examined from the beginning until a command name matches the input command. Once the match is made, each instruction within the sequence is acted upon and executed sequentially as well. Keywords are inserted within the action file to isolate instruction sequences and indicate file beginning and end. Again borrowing heavily from the PROCEDURE file, the start of each action file is simply the file itself. The start of a particular command sequence in the file is denoted by a single period appearing as the first character in the line of the sequence, followed by a descriptor. The descriptor for the CDC CYBER action file is the word ".CACT" . This descriptor is followed by the name of the command sequence and any arguments as required. The arguments are simply a series of sharpsigns (#) and integers, which denote the order and position of the arguments in the header statement. (As noted earlier, position is fixed.) Each entry is arbitrarily separated by commas. After this first line, referred to as the action file header, is a list of command instructions that comprise a command sequence. To delimit individual command instruction sequences, the keyword "END." is used. Finally, to indicate the last line in the action file,

the keyword "FINISH." is inserted. Figure 1 shows a skeletal outline of an action file for the CDC CYBER as presented thus far.

```
.CACT,COMMANDNAME1,#1,#2
YYYYY                    dummy instructions that
ZZZZZ                    form a command sequence
END.
.CACT,COMMANDNAME        no arguments
YYYYY
  :
ZZZZZ
END.
  :
.CACT,COMMANDNAMEN,#1,#2,...,#N
YYYYY
  :
ZZZZZ
END.
FINISH.
```

Fig 1. Skeletal Command Action File

With respect to the action file, the command language interpreter reads the action file, matches the desired command name, and then sends the instructions within the matched command sequence to the connected system or to the local NOVA/ECLIPSE system for execution. To simplify the decision process of the interpreter, the action file includes control characters within each command sequence to direct interpreter handling. Each of these control character sets has two control characters. For example, an instruction within the sequence that calls for writing to the local NOVA terminal is preceded by the control character set WL. Other examples of control

character sets are WS, write to the connected system; RS, read from the connected system; and RW, read from the local system terminal and then write a single carriage return to the connected system. Two particular control character sets are RC and WC. These character sets precede an instruction that is directed to the NOVA/ECLIPSE RDOS. The first set results in a disk file on the NOVA/ECLIPSE being read and copied to the connected system. The second set results in a file being written to the NOVA/ECLIPSE disk from information copied from the connected system. These character sets put into motion a swap to the RDOS CLI, as explained briefly in Chapter II and more fully described in the next section.

```
C  THIS COMMAND PERMITS LOCAL FILES TO BE SENT
C  CORRECT INPUT IS:    PUT,LFN,SFN,ID,SFPASSWRD
.CACT,PUT,#1,#2,#3,#4
WS       COPYBF,INPUT,ZQY
WC       XFER/A #1 QQVV/R
WS       %EOF
WS       REWIND,ZQY
WS       REQUEST,ZQZ,*PF
WS       COPYBF,ZQY,ZQZ
WS       CATALOG,ZQZ,#2,ID=#3,RP=999,PW=#4
WS       RETURN,ZQZ,ZQY
END.
C  THIS COMMAND RECEIVES SYSTEM FILES
C  CORRECT INPUT IS:    GET,SFN,ID,SFPASSWRD,LFN
.CACT,GET,#1,#2,#3,#4
WS       ATTACH,QZQ,#1,ID=#2,PW=#3
RR
WS       COPYSBF,QZQ,OUTPUT
RC       XFER/A VVQQ #4/R
WS       RETURN,QZQ
END.
```

Fig 2. Sample from CYBER Action File

An example portion of the final CYBER action file is illustrated in Figure 2. The first command name shown is PUT. This command takes a disk file resident on the NOVA/ECLIPSE system and transfers the file to permanent storage on the CDC CYBER system. The command name PUT is preceded by the keyword .CACT and followed by the place holders for four arguments. Therefore, the PUT command requires four arguments. In order, these arguments must be the local file name (including any directory specifiers) of the NOVA/ECLIPSE disk file, the system file name to be used on the CYBER, the user identification number or problem number the file is to be stored under, and the CYBER system passwords. (Since each argument parameter is required, a parameter must be used for the password. In this case, and only in this case, a zero entry may be used to indicate no password.) The first instruction in the PUT command sequence is preceded by WS, and, as with all instructions, arbitrarily starts in column nine. Thus, the instruction consisting of the string -- COPYBF,INPUT,ZQY -- is to be sent to the CYBER. The next command in the sequence is preceded by WC. This means that the file with the name entered in the position of argument one is to be copied from the NOVA/ECLIPSE disk to the CYBER system. All the rest of the commands in the sequence are preceded by WS, and explicitly instruct the CYBER to store the transferred file into permanent file storage, the user input arguments having been substituted for the argument parameters within the action file by manipulation of the interpreter. The command sequence terminates with END.

The second command name shown is GET. This command takes a file in permanent storage on the CYBER and transfers the file to the NOVA/ECLIPSE disk. The command GET is also preceded by the keyword .CACT and followed by the placeholders for four arguments. The treatment of these four arguments is exactly the same as that for PUT, with the order and position strictly observed. Hence, the arguments in order must be the system file name, the user identification number, the system file password, and the local file name. The commands in the sequence preceded by WS are handled exactly as described for PUT above. A blank command line is preceded by the control character set RR, which is sufficient to tell the interpreter that the local NOVA/ECLIPSE needs to ready itself for a read from the CYBER. Essentially, the character set RR activates a subroutine to create a temporary file that will subsequently be written into, when the next instruction in the action file preceded by RC is executed. The command line preceded by RC is just the converse of the command line preceded by WC. In this case, file transfer takes place from the CYBER to the NOVA/ECLIPSE. Again, the command sequence is terminated by the keyword END.

Finally, single control characters are used outside of the command sequence. The control character C, for example, indicates that the remainder of the line is a comment that the interpreter may ignore. (There is one other single control character -- I -- used elsewhere in the action file. It indicates that up to two responses of the connected system may follow. Even if no responses are expected nor entered, the

control character I must be in the action file as the interpreter always looks for it.)

Development of MONITOR

Program MONITOR is the largest software partition utilized in interfacing the NOVA/ECLIPSE computers to other connected computer systems. MONITOR serves as the basic command string interpreter of the action files discussed previously, and acts as the executive to control functional interactions of all subroutines and any tasks. There are two tasks that execute asynchronously within MONITOR. The main task is the interpreter/executive program MONITOR, and the other task is called SYSIN, for system input. This latter task continually monitors and seeks any input from the connected system. In fact, SYSIN is an extension to TTERMOP, doing all the same functions of TTERMOP, plus others to be developed below.

The MONITOR Task. In the narrowest focus, the program MONITOR reads the action file selected by a user, and, upon matching the desired command name and related command sequence, sends the command sequence lines to the connected computer system or to the NOVA/ECLIPSE RDOS. In order to interpret the action file, the file itself has to be read into storage. The simplest method to accomplish this is to use FORTRAN read statements that store the file in a one-dimensional array, one line at a time. As the action files are exclusively ASCII, the A format specification is used in the READ statements. As each line is read into the array, the keywords or control characters

are examined to determine further actions. Prior to taking any of these further actions, the arguments must be resolved.

A typical command instruction within the command language may include no arguments or up to four arguments. Argument substitution then allows the user flexibility in selecting arguments to be used with any particular action file. The method of argument substitution developed is simple and proceeds as follows. A command string consists of a command name optionally followed by arguments. Each argument of a command string is entered sequentially after the command instruction name. The command and arguments are separated either by commas or blanks, where two blanks or two commas together indicate a null argument. Order must be fixed, and is determined by the action file. An easy way to identify the arguments in the action file is to number them sequentially, using the sharpsign as an indicator. For example, argument two is denoted #2 and argument x is denoted #x. Not all command sequence instructions within the action file contain argument parameters. In fact, there are certain sequences that may be checked, while the others may be ignored for this purpose. The interpreter, therefore, checks all command sequence instructions that are preceded by the control character set in which the second character is either an "S" or a "C" . In these instances, the process of argument substitution is four-fold. First of all, the array containing the command sequence line is checked to determine if any sharpsigns exist. If not, the argument substitution process ceases. If a sharpsign exists, the array

is collapsed about the sharpsign, i.e., the sharpsign and its corresponding number are removed from the array. Next, in the same location where the sharpsign and number were, the array is expanded to the size of the argument entered by the user. Once this step is complete, spaces exist where the sharpsign and number used to be. In the last step of the substitution procedure, these spaces are replaced by the argument. This four-fold process is then repeated until all argument placeholders in any particular command sequence line have been substituted with actual arguments. Several error checking procedures are in effect prior to and throughout this process. If an argument is too long (maximum length 40 characters) or if arguments supplied by the user do not match exactly the arguments expected by the action file, error messages are provided to the user.

Once arguments are resolved in any command sequence instruction, the control characters dictate what action the interpreter is to take next. Each action is supported by one or more subroutines that program MONITOR calls into execution. For example, in the command PUT, one of the command sequence instructions is preceded by the control character set WS. Upon resolving the arguments in this instruction, the program MONITOR calls FORTRAN subroutine WRITSYSTM to implement the action of writing the instruction to the connected system. WRITSYSTM is really just a transition subroutine that smooths the transfer from the FORTRAN program MONITOR to the assembly language program WRSYS, a subroutine called by WRITSYSTM. WRSYS actually

does the work of transmitting the command instruction to the connected system. The transfer is made character by character. Several other subroutines parallel these two. For a control character set of RW (read from the local system and write a carriage return to the connected system), the interpreter calls FORTRAN subroutine READLWRITS, which transitions to the assembly language routine RDAWR. For a control character set of WL (write to the local terminal), program MONITOR calls FORTRAN subroutine WRITLOCAL, which does not call any other routines. The control character set RR (ready the NOVA/ECLIPSE for a subsequent read that will take place), causes the interpreter to call FORTRAN subroutine READYREAD. READYREAD simply creates a file to be used as temporary storage for the information that will subsequently be copied from the connected system. Two other control character sets are similar, though converses of each other. The set WC (write a copy of a NOVA/ECLIPSE disk file to the connected system) prompts program MONITOR to call FORTRAN subroutine SENDFILE. SENDFILE serves primarily as a transition routine and, secondarily, as a mini-executive of the actions necessary for such a transfer. SENDFILE calls assembly language subroutines EXCLI and GETFILE. Program EXCLI is called to execute the RDOS CLI on level two by swapping the CLI in and the program EXCLI (or effectively MONITOR) out of memory. During the swap, the instruction contained in the one-dimensional action file array (called IACTFILE) is sent to the RDOS. Upon return from the swap, program GETFILE is called. This program actually gets the disk file that is to be

transmitted and outputs the entire contents of the file character by character to the connected system. The converse of the character set WC is RC (read a copy of a connected system file and store on the NOVA/ECLIPSE). The interpreter responds to the control character set RC by calling FORTRAN subroutine RECEVFILE. Like SENDFILE, RECEVFILE also calls assembly language routine EXCLI in order to send instructions to the RDOS. Because of the structure of SYSIN, described below, the information from the connected system has already been received and stored in a temporary file. Thus, execution of EXCLI at this time simply transfers the input file from this temporary disk storage to the file location input by the user as an argument.

The remaining programs called by MONITOR are more a part of its executive functions, rather than interpretive functions. Two programs that blur this distinction somewhat are GETRSPS and CNVRT. FORTRAN program GETRSPS is called by MONITOR shortly after initiation, yet subsequent to user selection of the desired action file. This program reads the action file selected, looking for control character I (mentioned earlier). Entries in the action file after this character are initial responses (up to two) that may be expected from the connected system during interconnection. For example, in the CYBER action file the entry in the line after control character I is "COMMAND-,.." . The comma serves as a separator, indicating two possible responses that may be expected from the CYBER during interconnection with the NOVA/ECLIPSE. The first response is

"COMMAND-" and the second response is ".." . GETRSPS simply gets these responses, if any, from the selected action file and stores them in arrays. Upon return from GETRSPS, MONITOR immediately calls assembly language routine CNVRT. The sole function of CNVRT is to convert the FORTRAN array storage of GETRSPS into assembly language storage. For example, the ASCII character "A" is stored in the FORTRAN array as two bytes in one 16 bit word, of which the octal representation is <101><40>. The same character is stored in assembly language as <0><101>. The difference between these two stems from the difference between the FORTRAN A format specification used in reading in the array and the assembly language construction that does not use special formating. In essence, the FORTRAN read statement puts a single character into one word, the left byte the actual ASCII character code and the right byte a space. Assembly language looks at bytes only. Thus, a 16 bit word will have a null in the left byte and the actual ASCII representation in the right byte. CNVRT then stores the needed character representation of the responses and their size in buffers that are used by subsequent programs to assist the interpreter in detecting responses from the connected system and determining whether or not they're expected.

Those programs that are purely executive are the programs BGIN, PROMPT, REVERT, and TOTERM. Each of these carries out a specific function that needs to be accomplished to help integrate the various modules of MONITOR into a working whole. BGIN simply opens channels to the local NOVA/ECLIPSE terminal

input and output device codes -- $TT1 and $TTO -- respectively. PROMPT provides prompt character ">" to the user's terminal whenever called. The program REVERT is selected by the user as an option in lieu of a command string. By entering the string "^L" , the user instructs the interpreter to return to the RDOS CLI. REVERT simply executes the return to the CLI. The user also has the option of entering the string "^T" . This string instructs the interpreter to go to the terminal operation only mode. In essence, the program TTERMOP is called, although as a subroutine it is slightly modified and renamed TERMOP. MONITOR actually calls TOTERM, which first removes the previously defined device codes $TTO1 and $TTI1, and then calls TERMOP. Appendix B contains a flowchart description of program MONITOR and its various modules.

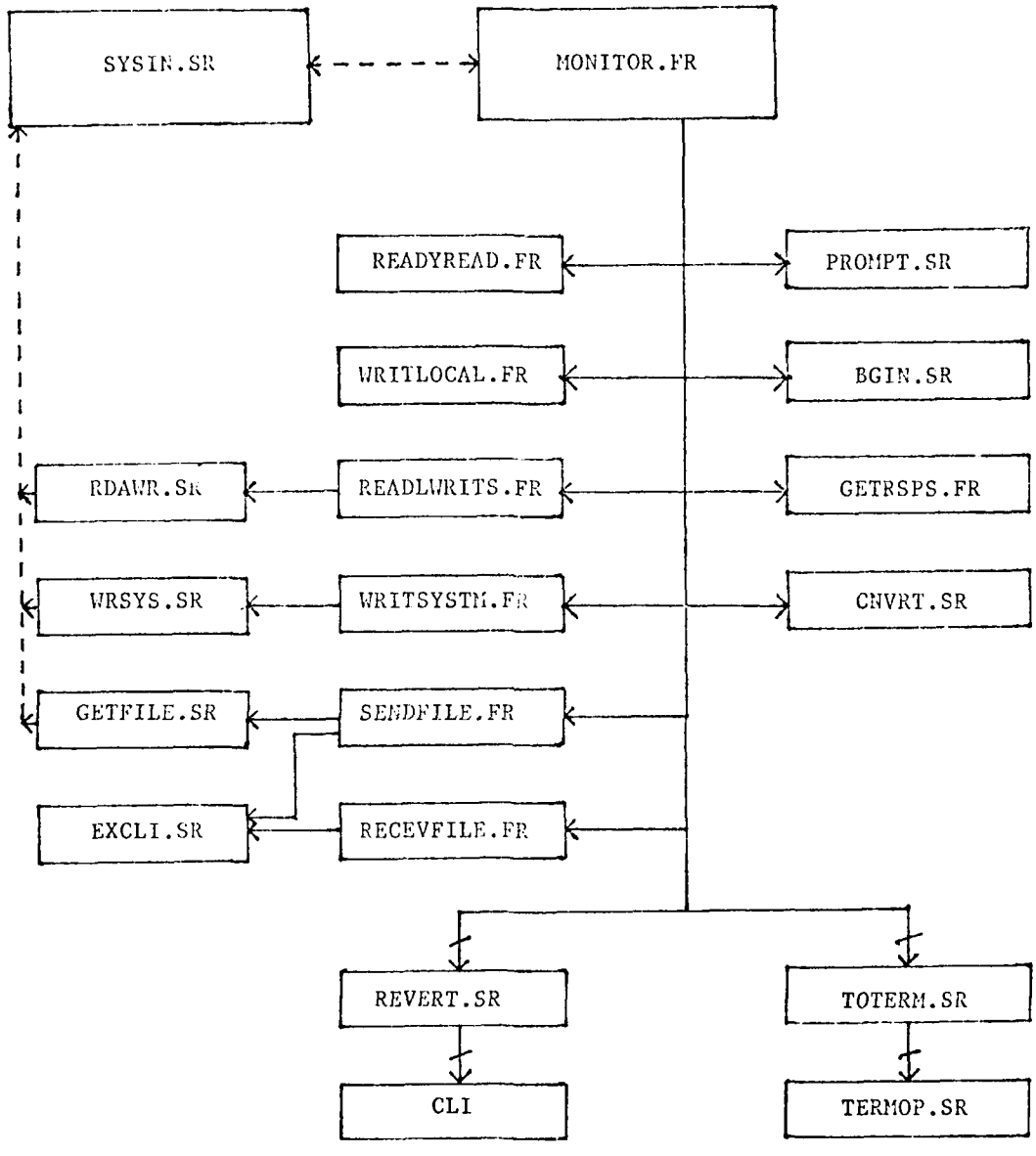
The SYSIN Task. As each of the interpretive and executive functions of MONITOR are being executed, a separate task is also called into execution by MONITOR. Using a DGC FORTRAN version task call, program SYSIN is activated at the beginning of program MONITOR. SYSIN is given a priority of one, while MONITOR is automatically assigned a priority of zero. In keeping with the RDOS conventions, MONITOR has the higher priority.

SYSIN has four basic actions to accomplish. First, when SYSIN is initially called, it identifies and defines the needed device codes $TTO1 and $TTI1. The remaining three actions occur as the processor schedules the task itself. The first action taken by SYSIN is to continually check the input buffer that may

have been receiving data from the connected system. Any data received is put into another buffer, called MATBUFR, for matching purposes. Next, the program SYSIN compares the data put into the match buffer with the expected responses from the connected system. (These responses were stored in retrievable locations by program CNVRT.) If the responses are as expected, the match buffer is cleared and new information is entered into it. If the responses are not expected, SYSIN displays them on the NOVA terminal for the user to see. Finally, SYSIN always checks to see if a temporary disk storage file has been created by the program READYREAD. If no such file is created, then SYSIN does nothing extra and returns to repeat the other actions. If the file has been created, however, then SYSIN writes the information in the match buffer into the temporary disk file as well. Thus, SYSIN creates the device interrupt routines for device codes/ports $TTO1 and $TTI1, monitors the input buffer, detects and appropriately displays responses from the connected system, and writes to a temporary disk file when such a file exists. Flowcharts that describe SYSIN are also contained in Appendix B.

Handshaking Conventions. Because of peculiarities in the RDOS with regard to its scheduling of user created tasks and because of the relatively slow response from the connected computer system (currently operating at 300 baud), several handshaking conventions available via task calls have been utilized throughout the program MONITOR. Basically, whenever an instruction has been sent to the connected computer system by

any routine or subroutine of the command language, the main task MONITOR is suspended. The main task remains suspended until the secondary task SYSIN readies the MONITOR. SYSIN readies MONITOR after any response from the connected system or after a built-in time delay has been exceeded. The asynchronous nature of the two tasks is affected by this implementation, but only after a suspension of the main task. Both tasks remain independent and asynchronous when both are active. The handshaking permits the user to control events through the interpreter, as MONITOR is the task that the user communicates with. This handshaking and the relationship of all programs that constitute the command language are presented graphically in Figure 3. Each assembly language program depicted is given the file name extension .SR, while each FORTRAN program is given the file name extension .FR. All the programs shown in the figure are loaded together and executed under the file name MONITOR.SV. The mechanics of the loading and execution are briefly discussed in Appendix C.



indicates task interaction

indicates subroutine call and return

indicates subroutine call and no return

Fig 3. Program MONITOR Structure Chart

IV. Validation

The general purpose command language was constructed in a modular manner from the top down. The initial program devloped and tested was the main task MONITOR. Until other needed modules were completely developed, stubs were created and used to interact with the main task. Tasking itself was not needed and was not introduced until the software effort was fifty percent complete. The remainder of this chapter discuss the testing and validation efforts with respect to the c. and language development and construction. The first section discusses efforts during the development, while the second section discusses efforts once a workable product was produced.

During Development

Every effort was made during development to completely test and debug modules as they were created and modified. Most of the modules were short and uncomplicated, enabling repeated assembling and loading without extensive time delays. Those modules that were more unwieldly, such as SYSIN and MONITOR, were developed and tested in stages. A tool frequently used to great advantage was the DGC RDOS Symbolic Debugger. The debugger saved many manhours in tracking down sporadically appearing errors. Perhaps the main feature of the debugger that permitted this savings was the ability to set breakpoints and execute up to these breakpoints.

Many trial modules were created in the project's beginning to test system calls, FORTRAN calls, and subroutine interactions. A significant amount of time was spent trying to integrate FORTRAN programs with assembly language programs. In this regard, several small scale FORTRAN programs and their called FORTRAN subroutines were created and compiled. The NOVA/ECLIPSE RDOS requires each distinct routine or subroutine to be separately compiled or assembled. Because of this, compiled FORTRAN programs produce an optional assembly language file. These files were examined repeatedly to determine the exact relationship between FORTRAN and assembly language programs. Essentially, the assembly language routines require calls to specific FORTRAN runtime library routines at the program beginning and end. These library routines initialize variables and organize stacks automatically, thereby enabling communication and interaction from a FORTRAN module to an assembly language module. Another complication was establishing the FORTRAN address locations for compiler generated and temporary variables within the assembly language routines. This is always accomplished at the end of an assembly language subroutine by setting the variables used in the assembly language program to FORTRAN address locations expected by the FORTRAN main module. FORTRAN calls to RDOS routines and system calls within assembly language routines also required repeated examination and trials to determine their effects. Once familiar, these types of calls proved quite powerful and convenient. It should be noted, however, that complications can

and did arise when mixing user defined operations that executed in user space with system calls that executed in system space. In fact, system calls cannot be used at all in user defined interrupt service routines. Finally, extra code was used regularly to provide messages to the user terminal regarding decision points encountered and overcome. For example, the one-dimentional array IACTFILE that contained the command sequence instruction from the action file was repeatedly displayed on the terminal screen throughout the argument substitution process. This provided clear and immediate feedback as to the program's progress and correctness during execution.

No computer link was available for testing the interfacing programs until late in the development, nor was it needed until late in the development. Initially, therefore, a simple line printer was connected to the input and output ports of the NOVA/ECLIPSE. Early programs were tested by writing to this printer and ignoring inputs (since there were none). About halfway through the development, the printer was replaced by a separate terminal. Later versions of the interfacing programs provided for writing to this terminal and reading from this terminal. Thus, two-way communication was established. The program TTERMOP was essentially proofed in this testing configuration, since computer to computer dialog for this program was easily simulated. Interrupts were generated and serviced, and responses were user-simulated as if the terminal that was connected to the access ports of the NOVA/ECLIPSE was

indeed the CYBER computer. During this entire period of time, no major attempt was made to streamline or enhance coded procedures. The first goal was to make the program execute successfully.

After Development

Once the programs were complete and individually tested in the terminal to terminal environment, the emphasis shifted to integrated program validation. All independent aspects of the individual modules were correctly compiled, assembled, and loaded together without any explicit errors discerned by the RDOS. By this same time, all stubs had been replaced by working routines and the intial refinements had been made. At this same time, the telephone link to the local CDC CYBER computer system was installed. Initial attempts to interconnect the NOVA/ECLIPSE computers to the CYBER system failed due to pin mismatches in the RS-232 connectors. Once these pin connections were changed, computer interconnection was achieved. Program TTERMOP worked well almost immediately, only requiring modifications that eliminated sections of code that simulated modem functions for operation in the terminal to terminal mode. Program MONITOR had more severe problems, revolving around handshaking considerations for the multiple tasks in the command language interpreter. These problems were most difficult to isolate for two reasons. First, the debugger was limited to setting breakpoints only within a single task. That is, if breakpoints were set in each task, the debugger would halt at

the first breakpoint encountered in either task and would remain in that task. The other task was not visible through the debugger in this instance. The second reason quickly overshadowed the first reason. The debugger, and in fact the entire MONITOR program would not execute at all once the entire program was loaded together. The debugger and all other programs loaded without error, but a FORTRAN runtime error indicating a stack overflow resulted whenever an attempt was made to execute the complete program. It appears that resident memory is too small to handle the execution of the integrated program with the debugger. Nonetheless, the handshaking problems were corrected by using task calls to appropriately suspend and ready the main task, which allowed the secondary task to seize control of the processor as desired.

Repeated failure of the program SYSIN to suppress displays of expected responses from the CYBER computer system lead to the discovery of another problem. The CYBER terminal sends out sequences of null characters at various times. These nulls are accepted in the input buffer, but are not visible to the terminal user. Even so, they cause the matching routine of SYSIN to give erroneous results. The solution is simply to discard all nulls received by the NOVA/ECLIPSE. This corrected the problem of detecting and suppressing expected responses from the connected CYBER system.

Early attempts to execute commands of the developed command language were intermittently successful. Numerous minor changes were made to the action files to correct command sequences,

syntax, and spacing. To save some time and preclude potential problems, the interpreter was modified so as to send only the exact number of characters comprising a command instruction. Prior to this modification, an entire record line of 80 characters was sent, even if the command string was less than 80 characters. As these changes were made, the program became progressively more reliable.

File transfers have been accomplished and the files have been checked for completeness and uniformity. Files are transferred as expected with one exception. Files received from the CDC CYBER system have an extra carriage return attached at the file beginning and end. These can easily be removed using the text editor. Also, there were occassions when a file would be transmitted to the CYBER system from the NOVA/ECLIPSE, but upon completion of the transmission, the NOVA computer would hang up about location 12 (octal). The problem was intermittent, and could not always be duplicated. Nevertheless, files with less than 20 bytes of information and files with upwards of 35,000 bytes of information have been successfully and repeatedly transferred both ways without program failure.

V. Conclusions and Recommendations

A general purpose command language for computer to computer dialog has been designed, developed, and implemented. The command language, called MONITOR, meets the basic constraints that were established when the problem was defined. The program is flexible, which allows the same software to be used for more than one connected computer system to the NOVA/ECLIPSE computers. However, until another system other than the CDC CYBER is actually interconnected and used, the degree of flexibility is speculative and subjective. The command language is general, to the degree that it seems reasonable to be able to implement MONITOR on another system other than the NOVA/ECLIPSE, with some modifications. The assembly language routines must be fitted to the assembler to be used, and the interaction between the FORTRAN and the assembly language routines may require changes. Only a test will demonstrate the real generality of this programming. The concept and overall structure, however, seem most feasible. The interpreter is the basic entity and should remain static in most cases. The action files are system dependent, and they are dynamic by design. Thus, the remainder of this chapter will cover the conclusions and recommendations regarding this project.

Conclusions

The general purpose command language works well overall. There are a variety of available commands and all have been successfully implemented. ASCII files have been transferred to and from the NOVA/ECLIPSE repeatedly, files have been printed on the CYBER printer, and files have been punched on the CYBER punch.

The command language is easy to use, simply defined, and brief. There are just a few commands, although more can be added or deleted. With these limited commands, much work can be accomplished. The user need enter a limited number of words only, and is relieved of the tedium of repeated entries. Fewer commands encourage use; simple commands encourage use.

The command language is not as universal as conceivable. Commands are structured to fit the environment in which they are to be used, limiting their universality somewhat. Commands are restricted to size, the number of arguments possible, and the order and position of arguments are fixed.

The program is relatively slow executing. Much of this is due to the handshaking that has been used.

The program is not as efficient as it could be with some design changes. The handshaking has circumvented the independence and asynchronization of some of the multitasking features of the operating system. Swapping takes place although some other means to avoid swapping may exist.

Finally, the programs are structured modularly, lending to ease of modification, readability, and understanding.

Recommendations

The following recommendations are presented to indicate areas in which the general pupose command language constructed may be improved and enhanced. Some of the recommendations follow logically from the conclusions presented above. Other recommendations are just observations and suggestions that have accumulated over the time of the project. Each recommendation is presented in the hope that it will either lead to follow-on project opportunities or a better understanding of the project as a whole.

The current structure and content of the general purpose language is not necessarily the most efficient nor the most effective. There are several assembly language routines that might readily be convertible to FORTRAN, and several features that have been implemented might be simplified. Logical candidates for simplification are the routines EXCLI and SYSIN. There may be a more straightforward way to interact with the RDOS from the command language itself, for example.

Much more flexibility could be built into the command language with respect to the functioning of commands. For example, the commands need not be limited to simple command strings. Perhaps global and local switches could be introduced to increase the power of the command instructions themselves. Further, argument defaults and optional arguments would

certainly improve the power of the language.

The current software has no convenient means to internally adapt to connected systems that are full-duplex. All the source code has been written with a half-duplex link in mind. An internal flag or switch to appropriately select code for either half-duplex or full-duplex operation would certainly increase the universality of the software.

Current implementation of the MONITOR command language is limited to ASCII file transfers only. Expanding the capability to handle binary files would be most useful, especially in support of the digital signal processing demands.

With the language implemented as it is, access is now available to another computer system for dialog from the NOVA/ECLIPSE. Devising a method of access to the NOVA/ECLIPSE from the other computer system would open new avenues of dialog between computers.

More sophisticated command languages often implement pipelining, i.e., stringing command instructions sequentially for sequential execution. Implementing pipelining within MONITOR would provide a terse language of greater power and convenience.

MONITOR lends itself to coding in a high-level structured language. Several advantages may be gained by redesigning MONITOR in PASCAL, for example. Data structure and file structure manipulations might be greatly simplified and facilitate greater creativity in developing command sequences.

Bibliography

1. Abrams, M., R. P. Blanc, and I. W. Cotton. Computer Networks: Test and References for a Tutorial (Revised Edition). JH3100-5C. IEEE Computer Society, October 1975.

2. Anderson, G. A. and D. E. Jensen. "Computer Interconnection Structures: Taxonomy, Characteristics, and Examples," Computing Surveys, 7: 197-213 (December 1975).

3. Bales, M., W. Bowman, D. Fisher, and P. Gruber. Man-Machine Interface/Intelligent Terminal Study Report. DTIC AD-B027 395L. Command and Control Technical Center, Washington, D.C.: 30 January 1978.

4. Carter, Benjamin F., III. A Special Purpose Computer to Computer Interface. MS Thesis. Ames, Iowa: Iowa State University, 1975.

5. Martin, J. Design of Man Computer Dialogues. Englewood Cliffs: Prentice-Hall, Inc., 1973.

6. McIlroy, M. D., E. N. Pinson, and B. A. Tague. "UNIX Time-Sharing System: Forward," The Bell System Technical Journal, 57 (6) Part II: 1899-1904 (July-August 1978).

7. Network Operating System/Batch Environment (NOS/BE) Version 1 Reference Manual. 60493800, Revision H. Control Data Corporation (CDC) Computer Systems, St. Paul, Minnesota, 1979.

8. Ravenscroft, Donald L. Electrical Engineering Digital Design Laboratory Communication Network. MS Thesis. Wright-Patterson AFB, Ohio: Air Force Institute of Technology, December 1978.

9. Real-Time Disk Operating System (RDOS) Reference Manual. 093-000075-08. Data General Corporation (DGC), Westboro, Massachusetts, 1979.

10. Rinder, R. "The Input/Output Architecture of Minicomputers," Datamation, 16: 119-124 (May 1970).

11. Ritchie, D. M. and K. Thompson. "The UNIX Time-Sharing System," The Bell System Technical Journal, 57 (6) Part II: 1905-1929 (July-August 1978).

12. Russo, P. M. "Interprocessor Communication for Multi-Microcomputer Systems," Computer, 10: 67-76 (April 1977).

Appendix A

MONITOR User Manual

Contents

1. Introduction

Program MONITOR is a general purpose command language that may be used for computer to computer dialog. The MONITOR software resides within the Data General Corporation (DGC) NOVA/ECLIPSE computer system. By calling upon particularly designed action files, MONITOR enables a user to intercommunicate with various computer systems that are linked to the NOVA/ECLIPSE via a standard RS-232 modem connection. The input/output ports for the NOVA/ECLIPSE are called device codes $TTI1 and $TTO1, respectively. These ports are on the NOVA 2/10 computer only. However, since the NOVA 2/10 and ECLIPSE S/250 share a ten megabyte disk via the DGC Real-Time Disk Operating System (RDOS), access to the ECLIPSE may also be gained through the NOVA by appropriate operating system instructions.

The user desiring to utilize MONITOR may use a NOVA video display terminal within the Air Force Institute of Technology (AFIT) Electrical Engineering department's Digital Signal Processing Laboratory. This user manual will describe the access procedures, the commands available for use, the errors that may be encountered, and the particular design of an action file.

Though the command language MONITOR was designed to be used with different connected computer systems, only the locally accessible Control Data Corporation (CDC) CYBER computer system has actually been interconnected. Thus, all examples and specifics will refer to this interconnection throughout the

manual. A general extension to other computer systems is relatively straightforward.

2. How to Access/Use MONITOR

The executable software package that enables computer to computer dialog via the NOVA/ECLIPSE computers is the binary save file MONITOR.SV. This file is composed of several FORTRAN language and assembly language source routines that are loaded together to form MONITOR.SV. The main partition of the entire software package is the FORTRAN source program MONITOR.FR, which forms the basic command language interpreter and executive for the command language. (Unless otherwise stated hereafter, the term MONITOR will refer to the binary save file MONITOR.SV.)

The software for MONITOR resides upon the NOVA/ECLIPSE disk, specifically the NOVA disk platter. The directory under which MONITOR is stored is named DIALOG. Therefore, to access MONITOR, the user must enter the directory DIALOG. The appropriate command by which to enter the directory from any other directory is:

```
DIR DP0F:DIALOG
```

Once the user has entered the directory DIALOG, the user is ready to execute the command language. The simple entry

```
MONITOR
```

will call the binary save file MONITOR.SV into execution. The user will next see the following display upon the terminal screen:

```
The MONITOR program you have entered provides
intercommunication between the NOVA/ECLIPSE computer
system and your choice of another system.


Please enter the digit opposite the action file
you desire to use:

          1 -- CDC CYBER
          2 -- DEC 10
          3 -- VAX 11/780
          4 -- Your own
>
```

The character ">" is a prompt that signals the user to make an entry. In the above instance, an entry of 1 would select the CDC CYBER action file, a 2 the DEC 10 action file, a 3 the VAX 11/780 action file, and a 4 an action file developed and coded by the user. (Originally, only the CDC CYBER action file was developed. Thus, the other action files existed, but had no useful entries. Effectively, these other files contained no commands and always returned an error condition if commands were attempted from them.) Entry of a number other than those indicated causes an error message as follows:

```
You have entered an illegal number.  Try again!
>
```

The user is again at the initial starting point.

Once the user has entered a selected integer, such as 1, the following type of message is displayed:

```
You have selected the CYBER.
Thank you.  Please enter a command.
>
```

At this point, all preliminary initializations of the program have taken place. The user is now ready to enter a specific command from the available commands contained within the selected action file, or two additional commands not contained within the action file.

The two commands not contained within any action file are a command that reverts the user back to the RDOS Command Language Interpreter (CLI) and a command that calls the terminal only mode of operation -- program TERMOP -- into execution. The two commands are:

```
^L  -  revert to local CLI
^T  -  change to terminal only mode
```

The next display a user sees upon entry of ^L is:

```
You have returned to the local CLI mode.
R
```

The "R" is the RDOS CLI prompt. The next display a user sees upon entry of ^T is:

```
You have entered into the terminal only mode.  Proceed!
```

The user's terminal is now connected as a transparent terminal to the computer system selected, such as the CYBER. To exit this mode and return to the MONITOR mode requires the user to enter an up-arrow "^", which returns the user to the RDOS CLI, and then to enter MONITOR and repeat the sequences described above.

All other permissible entries are contained in the action file. To see a complete list of these commands, see the action file itself. The next chapter describes the original commands developed and implemented within the CDC CYBER action file. As action files are designed to change, however, the user must look at the current action file to find the current status of commands.

3. Original Commands Available

The original commands developed specifically for the CDC CYBER action file are listed and briefly described below. These commands (names) may be used in other action files, provided command sequences contained in the action files are appropriately designed and controlled. See Chapter 5 for a discussion of the design and structure of a particular action file.

PUT,LFN,SFN,ID,SFPASSWRD

> This command selects a local file name (LFN) from the NOVA/ECLIPSE disk and transfers the file to the connected computer system. The transferred file is stored permanently under a system file name (SFN) with an identification (ID) and system file password (SFPASSWRD) supplied by the user.

GET,SFN,ID,SFPASSWRD,LFN

> This command selects a SFN stored on the connected computer system and transfers the file to the NOVA/ECLIPSE disk with the LFN input. The user supplies the ID and SFPASSWRD for the SFN.

SPRINT,SFN,SFPASSWRD

> This command selects the SFN with appropriate SFPASSWRD on the connected computer system and prints the file out on the connected system printer.

SPUNCH,SFN,SFPASSWRD

> This command selects the SFN with appropriate SFPASSWRD on the connected computer system and punches the file out in Hollerith code on the system card punch.

DELETE,SFN,SFPASSWRD

This command deletes/purges the permanent file on the connected computer system with the SFN and SFPASSWRD input.

FILES

This command causes the connected system to display local files in use.

PFILES

This command causes the connected system to display the permanent files in use for the ID supplied.

LOGON,USER ID,USER PASSWRD

This command initiates access to the connected system. The user must supply the specific ID and protected password (USER PASSWRD).

LOGOFF

This command terminates access to the connected system.

LPRINT,LFN

This command selects the LFN on the NOVA/ECLIPSE and prints the file out on the connected system printer.

LPUNCH,LFN

This command selects the LFN on the NOVA/ECLIPSE and punches the file out in Hollerith code on the connected system card punch.

SBATCH,SFN,SFPASSWRD,DISPOSITION,TERMINAL ID

This command selects the SFN with appropriate SFPASSWRD for execution on the connected system in the batch mode. The output of the file (DISPOSITION) and location (TERMINAL ID) are supplied by the user.

LBATCH,LFN,DISPOSITION,TERMINAL ID

This command selects the LFN on the NOVA/ECLIPSE for execution on the connected system in the batch mode. The DISPOSITION and TERMINAL ID are supplied by the user.

4. Error Handling

There are four general types of errors that may occur in execution of the program MONITOR. There are (a) interpreter errors, (b) executive function errors, (c) NOVA/ECLIPSE RDOS errors, and (d) connected computer system errors. The first two types are errors that arise within the MONITOR software itself. NOVA/ECLIPSE RDOS errors occur whenever the RDOS generates an error condition, and the last type of errors results from operating system error conditions generated by the connected computer system.

Connected Computer System Errors. MONITOR was designed to display error conditions generated by the connected computer system to the user. Each action file reserves a location after the control character "I" for up to two expected responses from the connected system. Any response not matching these expected responses is displayed to the user. Thus, any error conditions of the connected computer system are simply passed to the user for action. Most of these error conditions are non-fatal and do not cause MONITOR to fail. However, unexpected results may arise if error conditions are encountered.

NOVA/ECLIPSE RDOS Errors. MONITOR executes within the direct control of the RDOS. Error conditions generated by the RDOS may be displayed to the user or cause the RDOS to cease execution of the MONITOR program. In the worst case, RDOS errors will be severe enough to cause a NOVA/ECLIPSE system "panic." In this case, the computer halts and displays the

error condition and accumulator values. The RDOS Reference Manual (Ref 9:F-1 - F-2) details these circumstances. In a less severe, but fatal error condition, the RDOS will halt execution without a "panic." Generally, a "Control A" input will restore the CLI to the user. On other occasions, "Control A" will have no effect. In these circumstances, the user must reset the computer and bring it up as if it had been powered down. In the best case, the RDOS error condition will cause an automatic return to the RDOS CLI and the error condition will be displayed to the user. In any case, RDOS error conditions are generally symptomatic of software errors in the executing program. All such error conditions should be examined and tracked in order to correct apparent errors in the source code.

MONITOR Executive Function Errors. Executive function errors are closely related to NOVA/ECLIPSE errors. In fact, these errors are produced by the RDOS, but are generally anticipated. Thus, such errors are caught by the MONITOR program and serviced automatically, or a specific error condition causes a software halt of the program. For example, whenever a FORTRAN call to open a file is executed, a potential error may occur that indicates that the file is already open or that such a file doesn't exist. MONITOR handles all of these types of errors by executing a STOP and displaying the cause of the stop. Specifically, if the CYBER action file would not open properly when a call was issued to open it, the following message would be displayed for the user:

STOP CACT NOT OPENED PROPERLY

Other executive function errors are handled identically. In each instance, the cause must be isolated before program MONITOR can be executed successfully.

MONITOR Interpreter Errors. The interpreter errors are those that the interpreter MONITOR.FR anticipates and handles. These errors are handled entirely within the MONITOR software. Whenever such as error occurs, the user is notified via display, and a prompt is given to indicate that a connected input is needed. Each of the anticipated error conditions and their reasons are provided below:

INVALID COMMAND - EMPTY STRING

Indicates that a command string is comprised of all blanks or nulls. (Entering a carriage return alone will result in this error condition.)

SYNTAX ERROR
FIRST OR LAST LITERAL INVALID SEPARATOR

Indicates a command string began with a comma and/or ended with a comma. (Only commas and spaces are separators. However, commas cannot start or end a command string. Spaces are ignored except between other literals. Thus, an entry of a command string that ends with a comma and then a space will result in this error condition. NOTE: Internal null arguments within a command string are allowed. For example,

PUT,,THIS ,THERE

will be accepted as a valid string with the command PUT and four arguments.)

INVALID COMMAND - TOO FEW CHARACTERS

Indicates a command string with just one character. Again, spaces are ignored except between other literals.

INVALID [illegible]

The [illegible] to four arguments. This indicates [illegible] were supplied with a single [illegible].

INVALID COMMAND - [illegible]

Each command and each argument may be up to 30 characters long. This indicates a command or argument entered was 31 or more characters.

INVALID COMMAND
COMMAND NOT IN ACTION FILE
OR SUPPLIED NOT EQUAL REQUIRED ARGUMENTS

Indicates one of two possibilities: (1) there was not a match of the command entered with any command contained within the action file. (could be a misspelling.) (2) the action file command required x arguments and either less than x or more than x arguments were input. (The number of arguments must be exactly as specified within the action file.)

COMMAND ABORT
UNEXPECTED ENTRY IN ACTION FILE

Indicates an undefined or nonexistent control character set within a command sequence. (The action file must be corrected.)

5. CYBER Action File Design

The basic structural design of any action file is detailed in the main body of this report. This chapter describes the overall design of action files and details the design of the specific CYBER action file developed and implemented. The listing of the action file itself is in Appendix D.

Essentially, the interpreter initially looks for the first character or first two characters of each line in the action file. Depending upon these characters, action is taken to read the next line, stop reading the file, or execute a line just read. Thus, only specific characters are expected in the first two columns of the action file. Any other characters are simply ignored. If no recognizable characters or no characters at all are in columns one or two, the interpreter cannot find any useful information and aborts reading. This feature does allow, however, comments to be inserted anywhere. The character "C", for example, is not expected by the interpreter. Thus, all lines beginning with "C" are ignored and serve as comment lines. Any other unexpected character, such as "A", could also have been used, but "C" was arbitrarily chosen to match with FORTRAN comment lines.

Spacing is predetermined by the expectation of the interpreter as well. The exact spacing established is arbitrary, but once established, it must be followed explicitly. Thus, columns one and two are reserved for control characters. Anything may be in the rest of a comment line. Lines that begin

with "END." and "FINISH." may have comments after them, as the interpreter will simply read the first character and ignore the remainder of the line. Each command sequence starts with a period, followed by the name of a particular action file. Thus, for the CDC CYBER, each sequence starts with ".CACT" . This name is followed by a specific command, as indicated in Chapter 3. The command is followed by the exact number of arguments required in the form of a sharpsign and an accompanying integer. Argument three is #3, for example. Each of these entries must be separated (arbitrarily) by commas. As mentioned in the body of this report, this header line is followed by executable statements preceded by control character sets. The executable statements must (arbitrarily) begin in column nine. (NOTE: A tab will not suffice to separate the control character sets from the executable statements.) In each executable statement that requires an argument, the exact argument parameter from the header statement must be used. Thus, if an executable line requires argument three, the executable line must contain #3 exactly where argument three is required. The interpreter will substitute the arguments entered by the user into the specified locations. As noted before, each command sequence ends with the keyword "END." . and the entire action file command sequence ends with the keyword "FINISH." .

All executable statements in the action file are sent to the CYBER for execution if preceded by the control character set WS. Thus, these statements are exactly as required by the CDC CYBER. Similar requirements must be met for other action

files. The order of these statements must also be acceptable to the connected computer system. If not in the proper order or syntactically incorrect, the CYBER system will generate error conditions and display them to the user. Essentially, the method of transferring files to and from the CYBER utilizes the command execution mode of the CYBER system. In this mode, the CYBER always responds after correct user input with "COMMAND-" , and this is indicated in the action file response line -- the line preceded by the character "I" . Up to two expected responses from the connected system may be in the line after the control character "I", beginning in column nine. Instructions entered in the command sequences for PUT, GET, and similar command strings, simply tell the CYBER to copy all input to a file name or copy all output from a file name. The NOVA/ECLIPSE access ports, when connected to the CYBER, send to the input file and receive from the output file. To terminate this copying, the command instruction %EOF is issued. All other command sequences are straightforward and detailed in the CYBER user documents.

All other executable statements not preceded by WS tell the NOVA/ECLIPSE RDOS to do something. Most are straightforward and all are explained within the body of this report. The lines preceded by WC and RC contain executable statements for the RDOS CLI. For example, the line containing

XFER/A #1 QQVV/R

is a CLI instruction to transfer the ASCII file named as user-supplied argument one to the temporary disk file QQVV, and

to make file QQVV a random file. Control character sets RC and WC precede these kind of executable lines, causing a swap to the CLI to execute these instructions and then a swap back to the main interpreter program. Those lines that are blank, except for the control character set, have no statements to be executed per se. The MONITOR software proceeds strictly upon the basis of the control sequence read. In the case of the set RR, the interpreter creates a temporary file on the NOVA/ECLIPSE disk that will be used by a subsequent RC statement. Hence, in order for the command sequence to execute properly from start to finish, any line with the RR control character set must be followed by a line with the RC control character set. The RW character set indicates that a carriage return needs to be sent to the CYBER to initiate access. This particular command instruction is CYBER system dependent, and may require some modification if expected to be used with another connected computer system. The entry beginning in column nine after the control set WL will be sent verbatim to the user terminal for display. No other control character sets have been established for the interpreter.

It should be noted that the RDOS console interrupts have not been deactivated anywhere within the software developed. Thus, the console interrupts for the RDOS CLI are effective when executed within MONITOR. For example, an entry by the user that is incorrect may be totally replaced by simply entering a backslash "\" . The next keyboard entry will begin a new line. The backspace works, and the "Control A" and "Control C" inputs

work as well. "Control A" and "Control C" cause the currently executing program, such as MONITOR, to be interrupted and control returned to the RDOS CLI. "Control C" also provides a "Break" file that displays a dump of memory at the time of the interrupt.

Finally, the action file for the CYBER has been set up with the following results. The products produced upon the CYBER card punch and/or printer are to be output to the terminal located in the AFIT School of Engineering. Further, all products may be picked up at the bin labeled "M/N" , since all output products will be tagged with the banner NEO. The NOVA terminal identification number for logging in purposes is arbitrarily set at 777. Lastly, the order of the commands entered into the action file is based upon use. Those commands used frequently are placed in the action file's beginning, whereas those used less frequently are placed later in the action file. Since the interpreter searches the action file sequentially from the beginning each time a command is needed, those used more often will be found in less time. (NOTE: The same named command may appear more than once within an action file. In order for the interpreter to distinguish one command from another, in this case, the number of arguments must be different for each command name.)

6. Summary

The instructions and guidelines contained within this user manual are the minimum needed to operate MONITOR. Once a user becomes familiar with the action files and interpreter, many other opportunities may occur to experiment with the program MONITOR that would alter the guidelines set forth herein. Such changes are expected and desirable, if this command language is to truly be general purpose. Furthermore, there may be areas in which even the basic structure of the interpreter may be profitably altered. See the recommendations covered in Chapter V of the main report for some potential examples. Finally, the source listings and interactive guidelines have been developed to provide the user with necessary instructions governing MONITOR execution. Thus, a user should be able to use MONITOR quite readily with only a listing of the selected action file to consult, once program execution has begun.

Appendix B

Program Descriptive Flowcharts

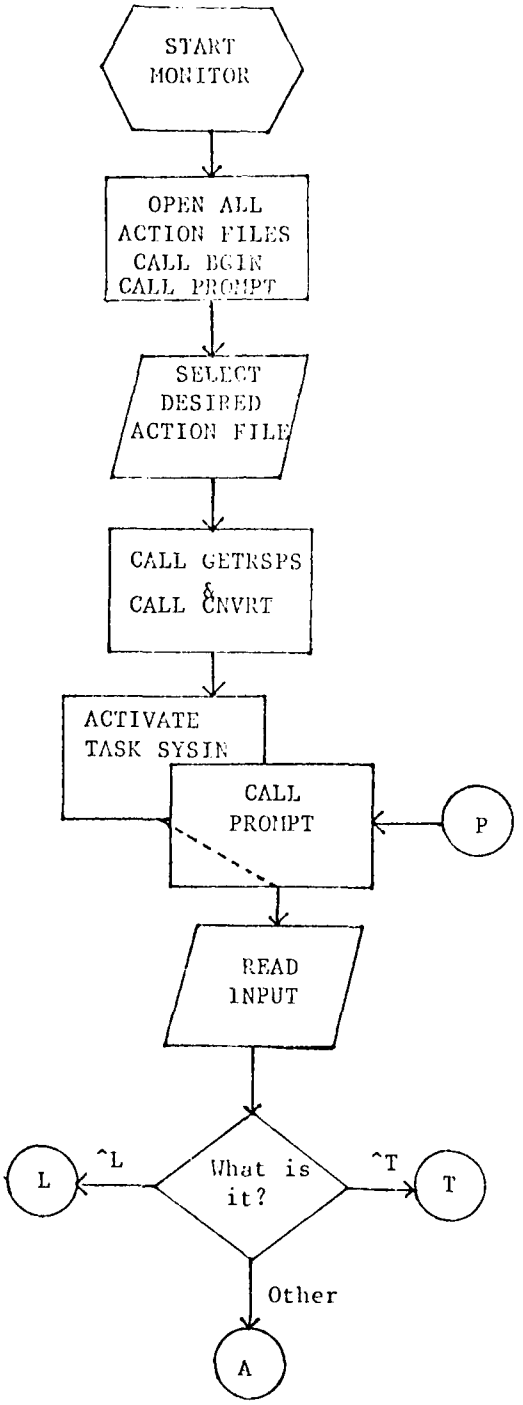


Fig 4. MONITOR.FR (Part 1)

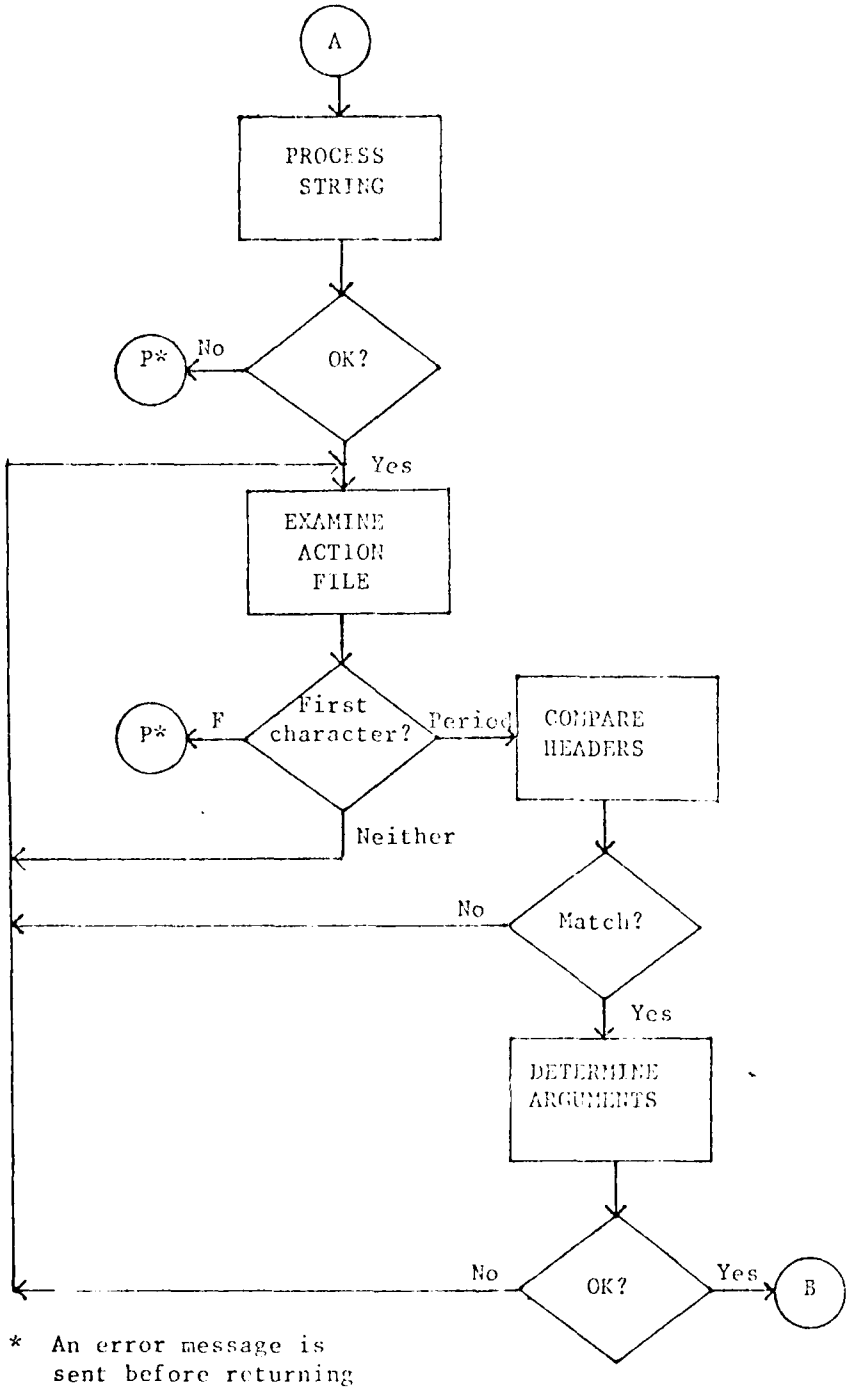
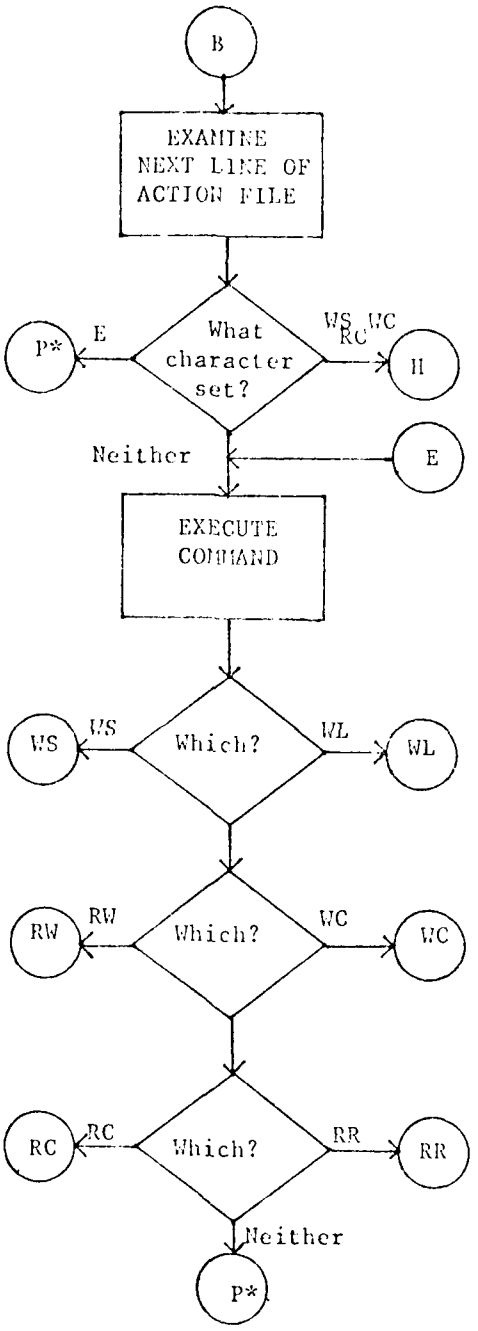


Fig 5. MONITOR.FR (Part 2)



* An error message is sent before returning

Fig 6. MONITOR.FR (Part 3)

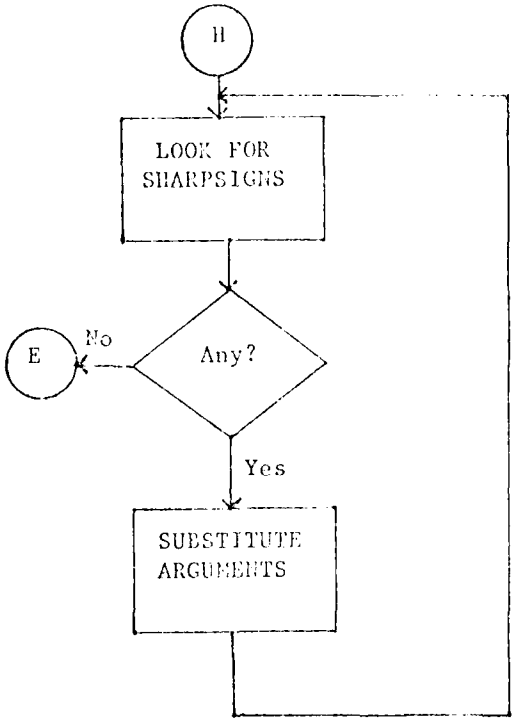


Fig 7. MONITOR.FR (Part 4)

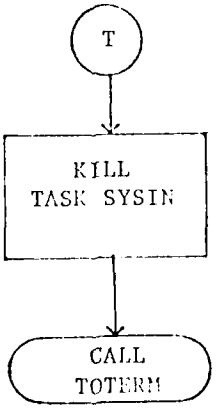


Fig 8. MONITOR.FR (Part 5)

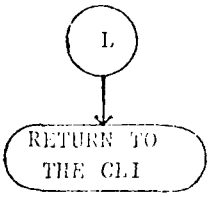


Fig 9. MONITOR.FR (Part 6)

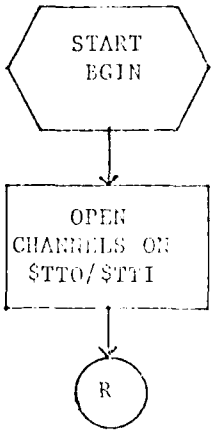


Fig 10. BGIN.SR

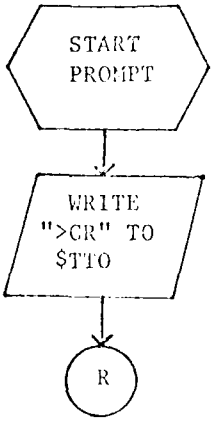


Fig 11. PROMPT.SR

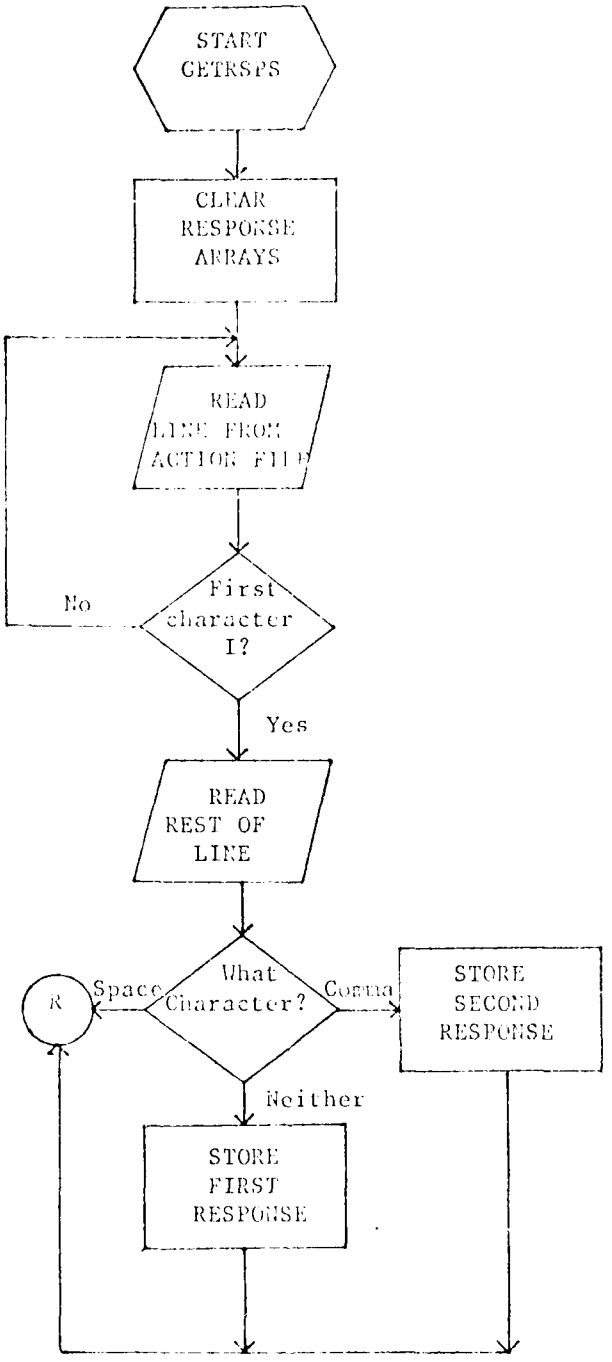
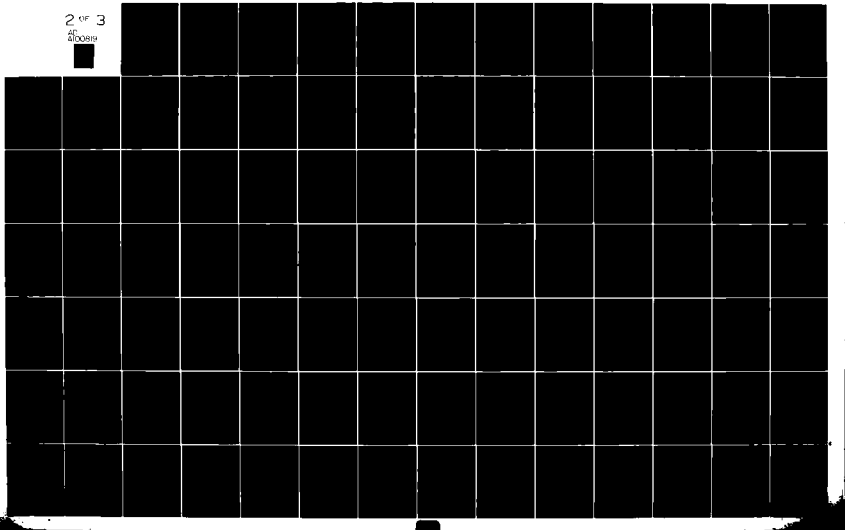


Fig 12. GETRSPS.FR

AD-A100 819 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL--ETC F/G 9/2
CONSTRUCTION OF A GENERAL PURPOSE COMMAND LANGUAGE FOR USE IN C--ETC(U)
SEP 80 W D GRIESS
UNCLASSIFIED AFIT/GCS/EE/80S-15 NL

2 OF 3
AD
A100819



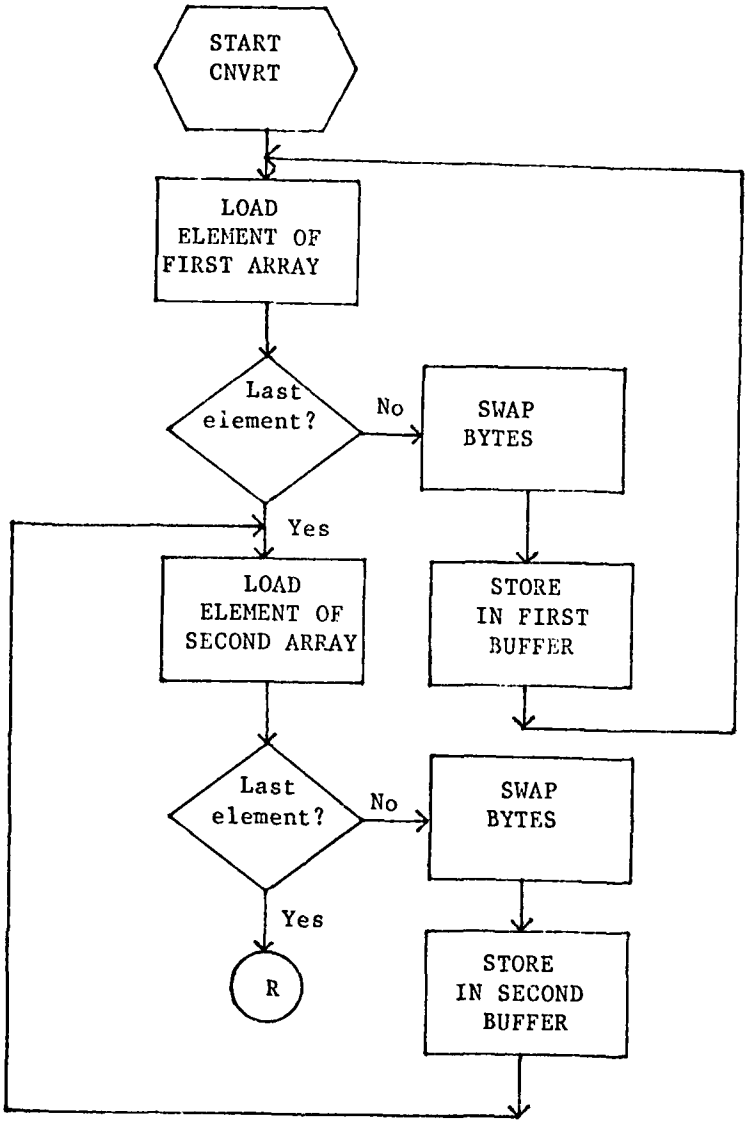


Fig 13. CNVRT.SR

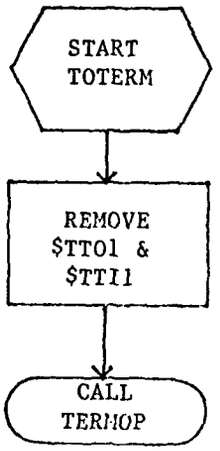
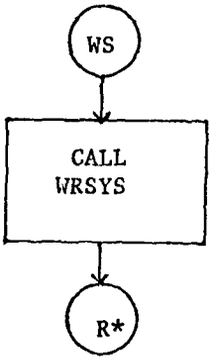


Fig 14. TOTERM.SR



* Return to a specified line number (602)

Fig 15. WRITSYSTM.FR

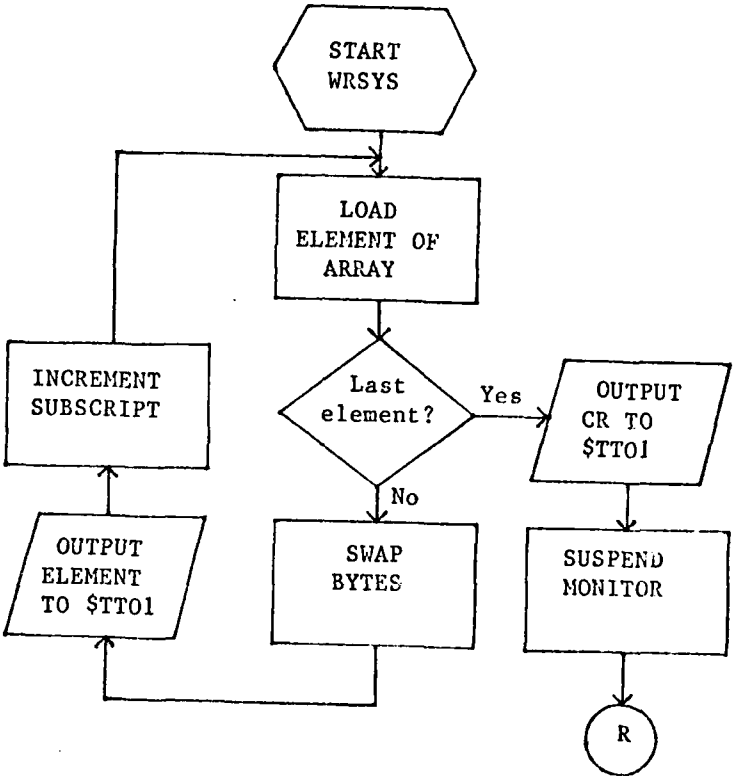
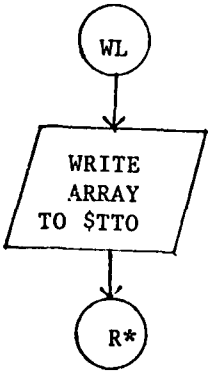
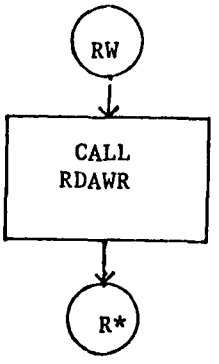


Fig 16. WRSYS.SR





* Return to a specified line number (602)

Fig 17. WRITLOCAL.FR

Fig 18. READLWRITS.FR

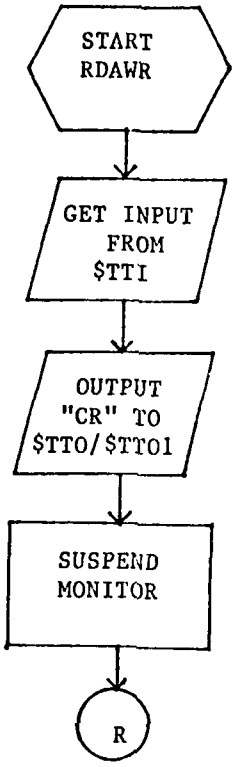
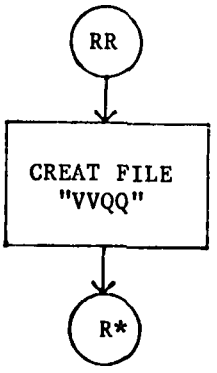
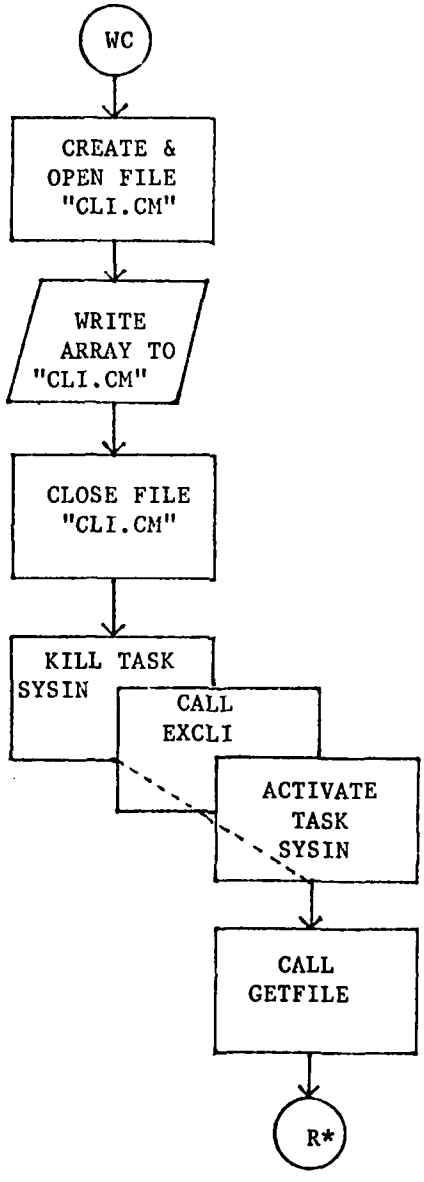


Fig 19. RDAWR.SR



* Return to a specified line number (602)

Fig 20. READYREAD.FR



* Return to a specified line number (602)

Fig 21. SENDFILE.FR

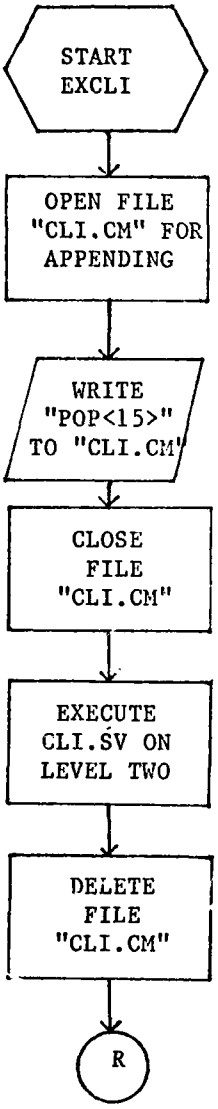


Fig 22. EXCLI.SR

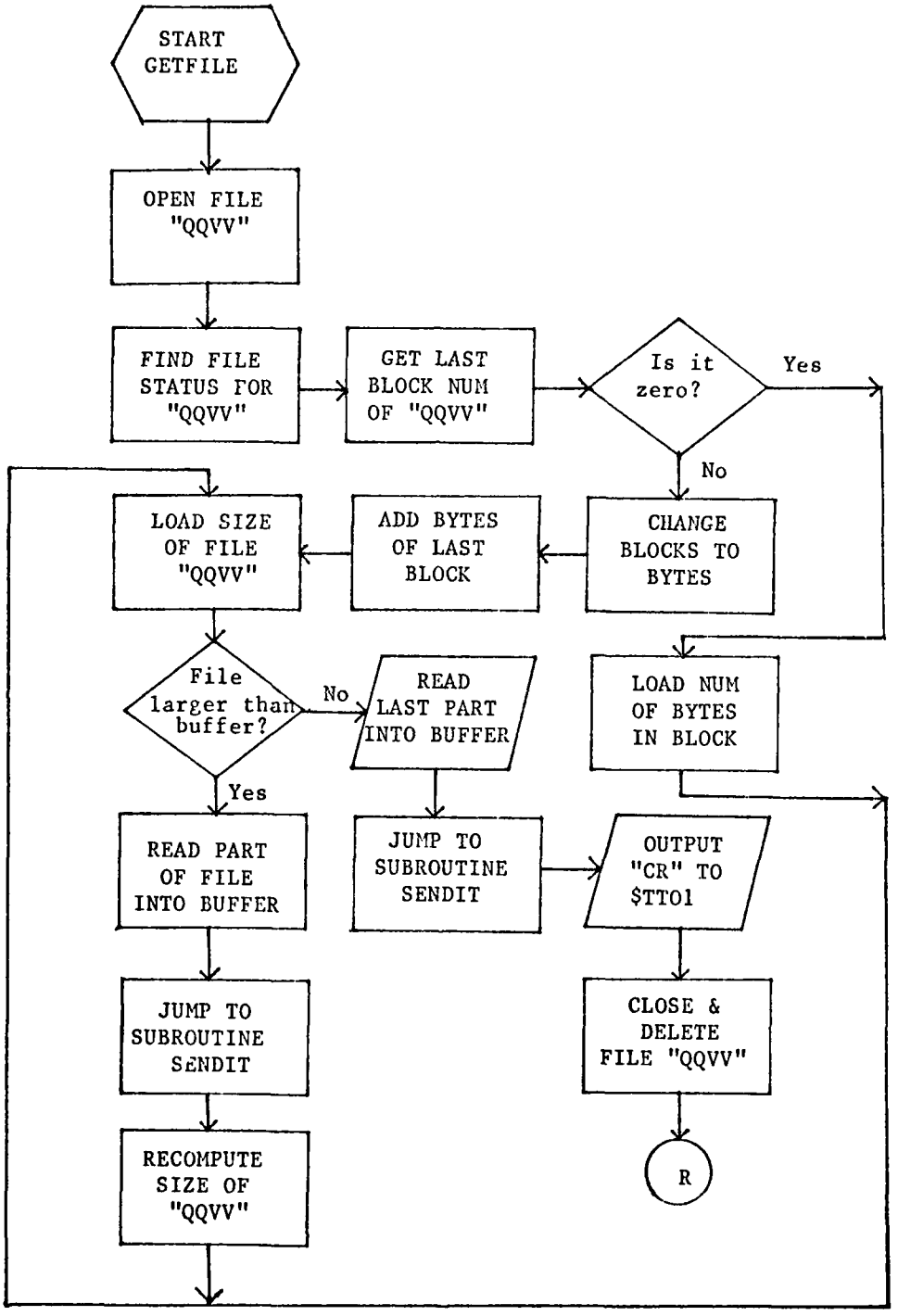


Fig 23. GETFILE.SR

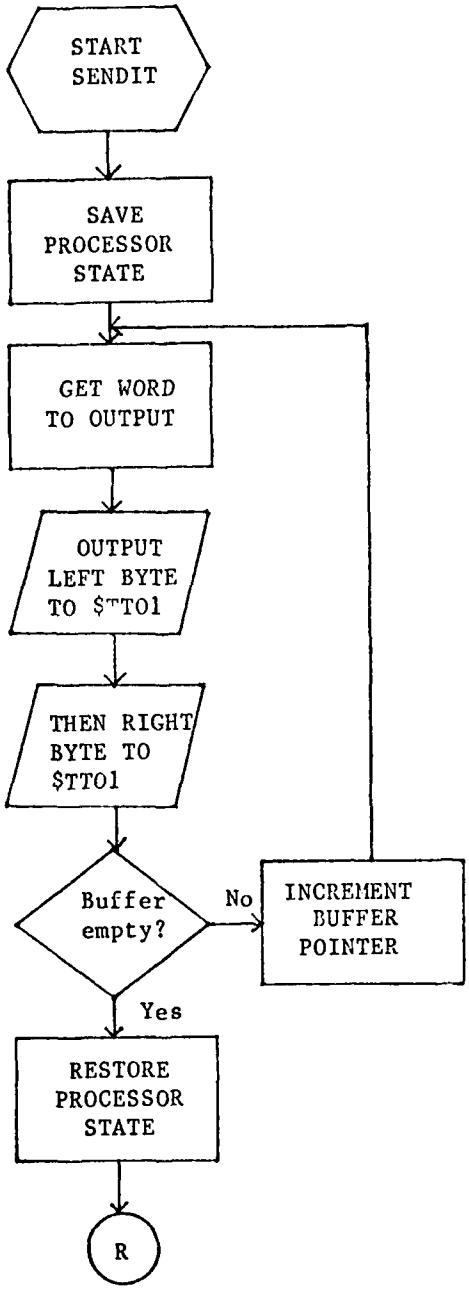
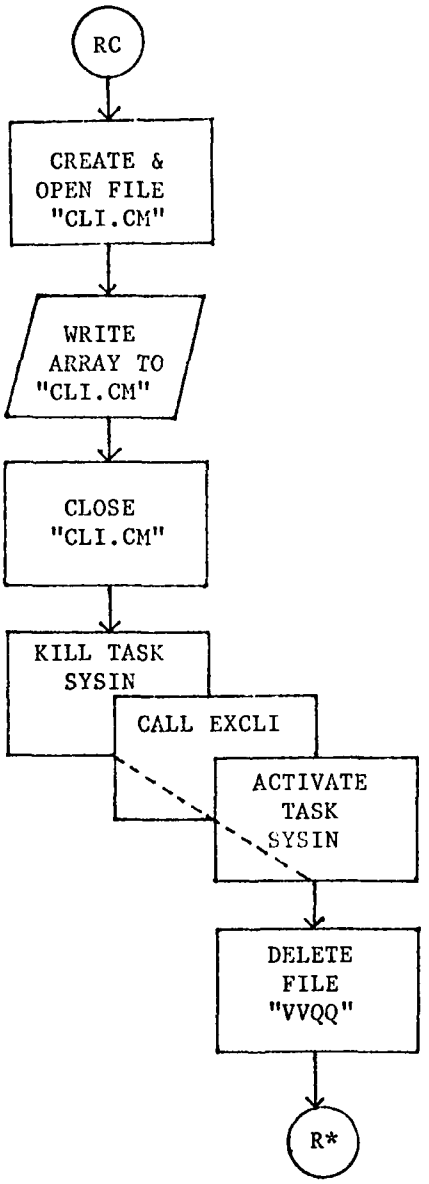


Fig 24. GETFILE.SR (Part 2)



* Return to a specified line number (602)

Fig 25. RECEVFILE.FR

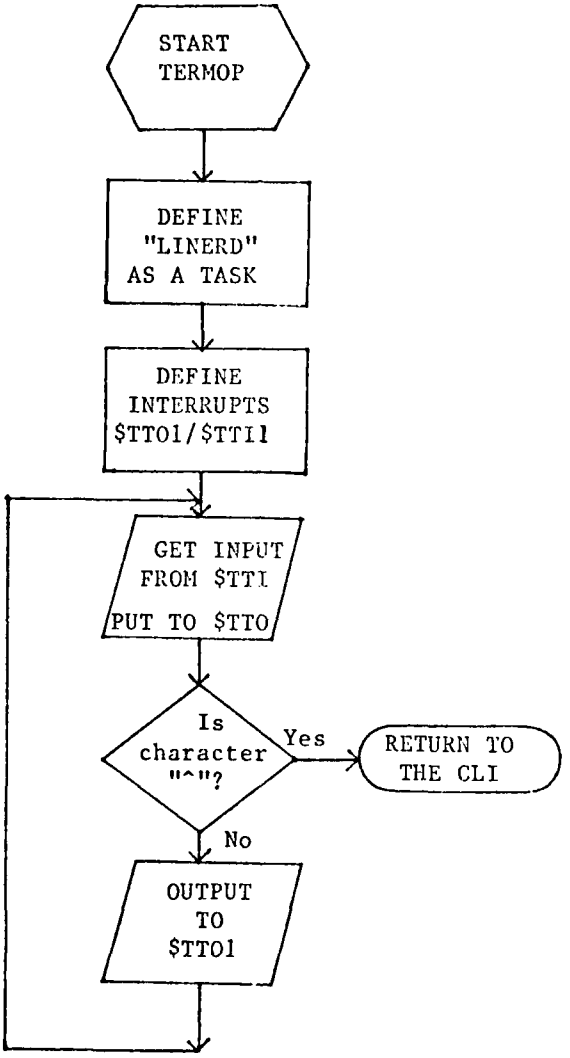


Fig 26. TERMOP.SR (Part 1)

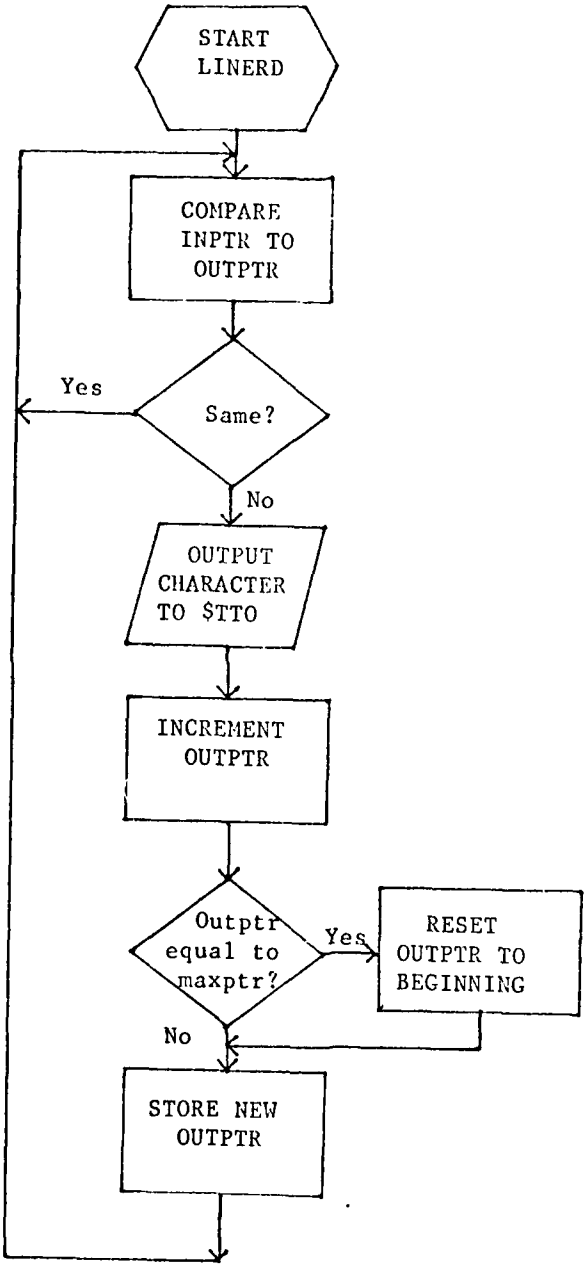


Fig 27. TERMOP.SR (Part 2)

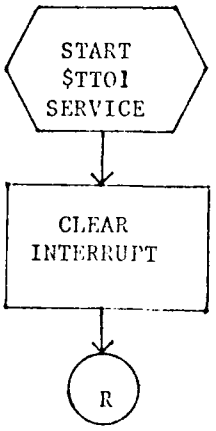


Fig 28. TERMOP.SR (Part 3)

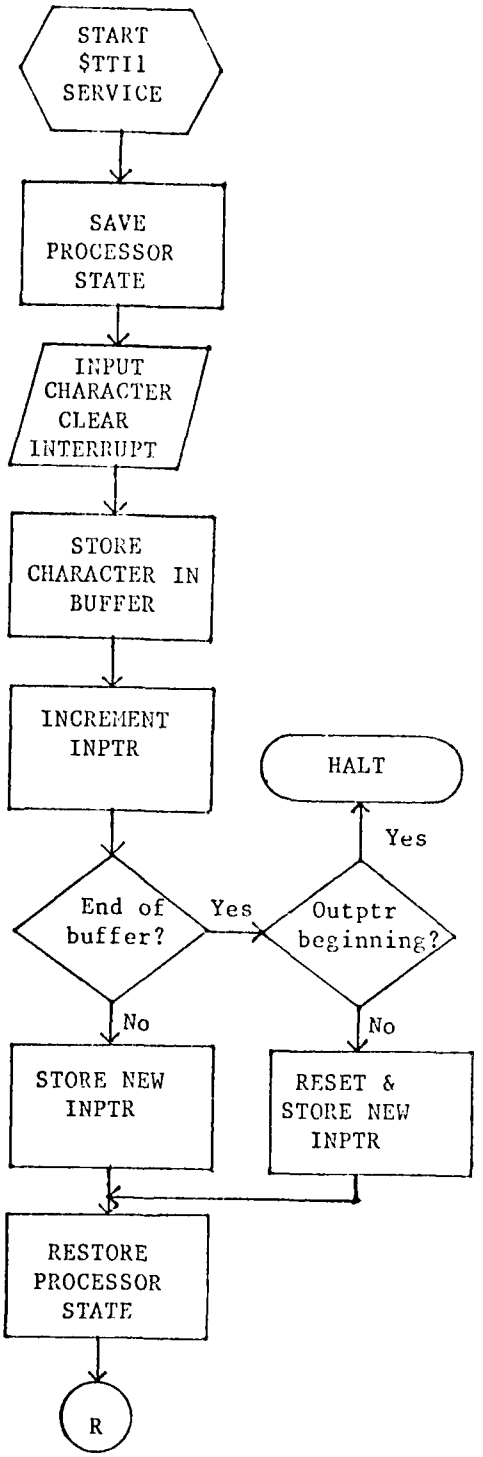


Fig 29. TERMOP.SR (Part 4)

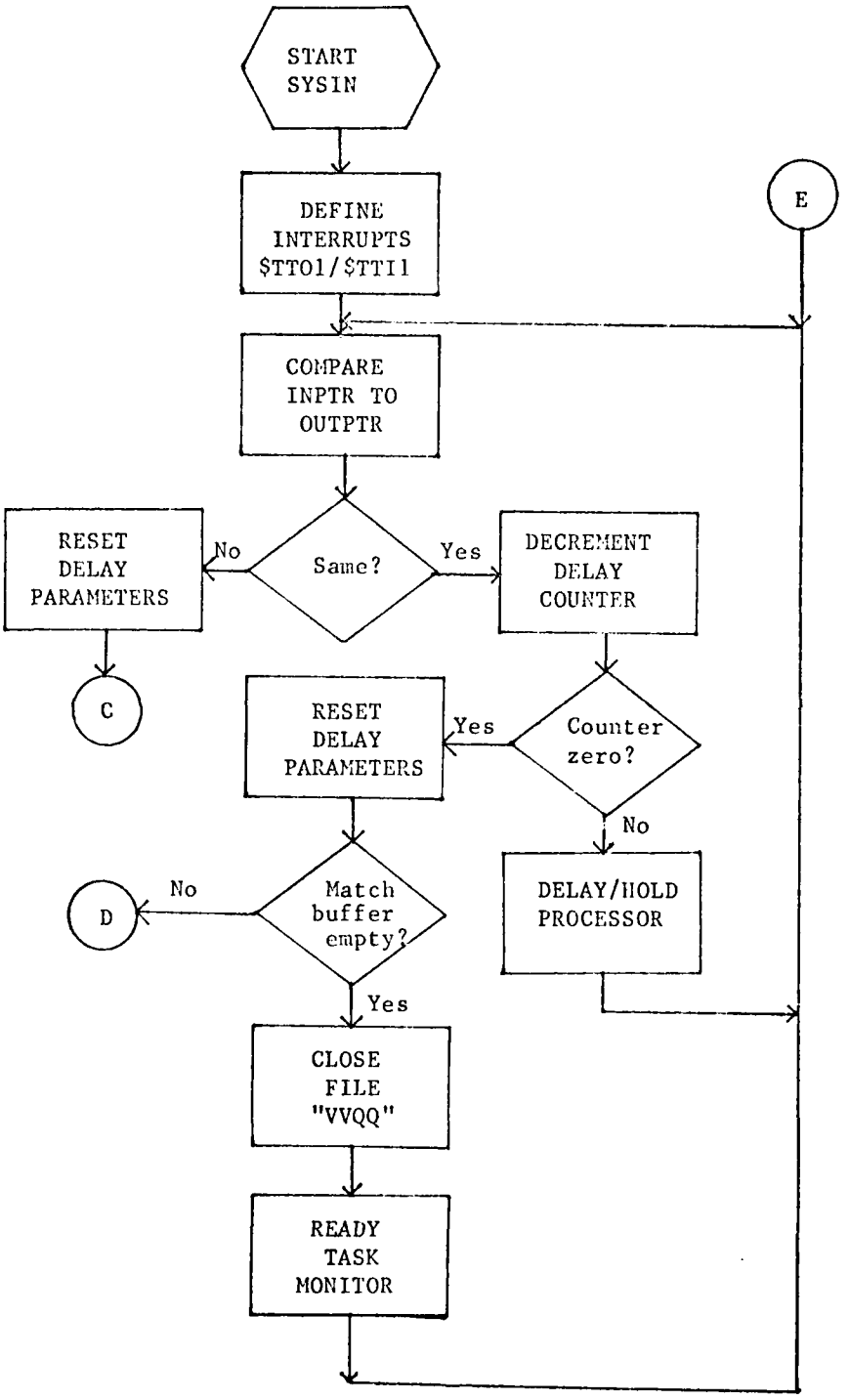


Fig 30. SYSIN.SR (Part 1)

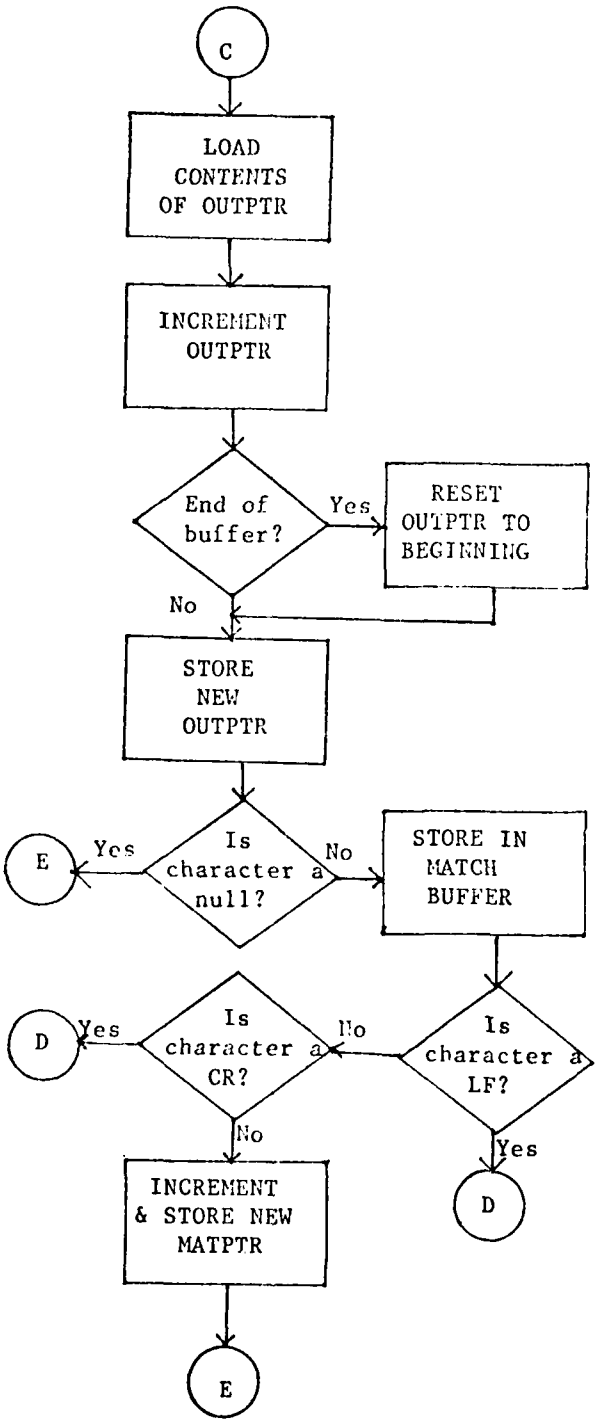


Fig 31. SYSIN.SR (Part 2)

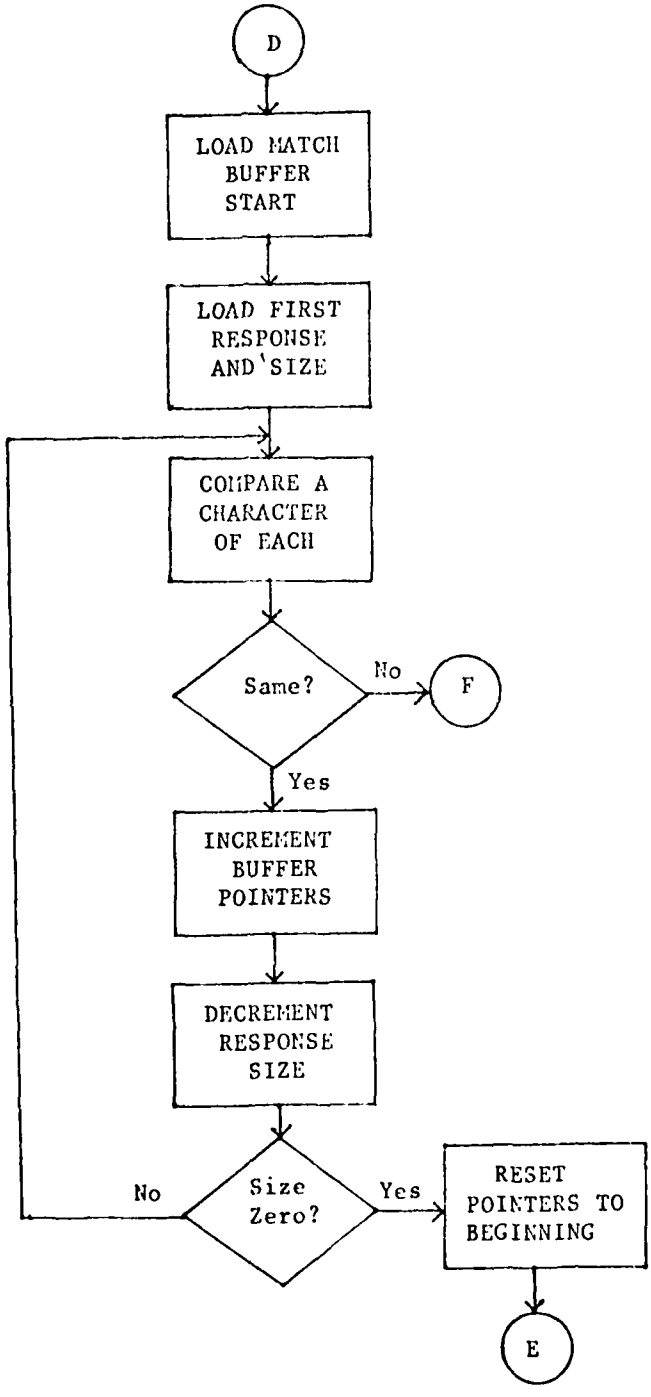


Fig 32. SYSIN.SR (Part 3)

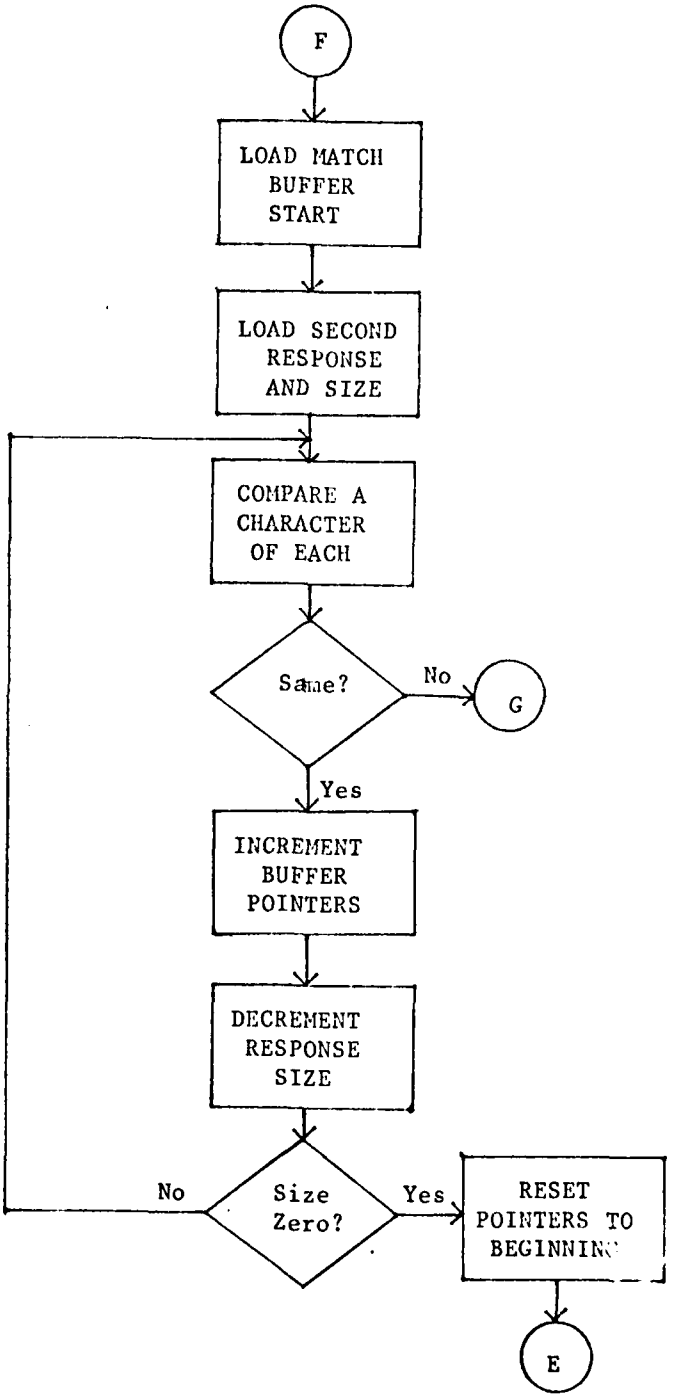


Fig 33. SYSIN.SR (Part 4)

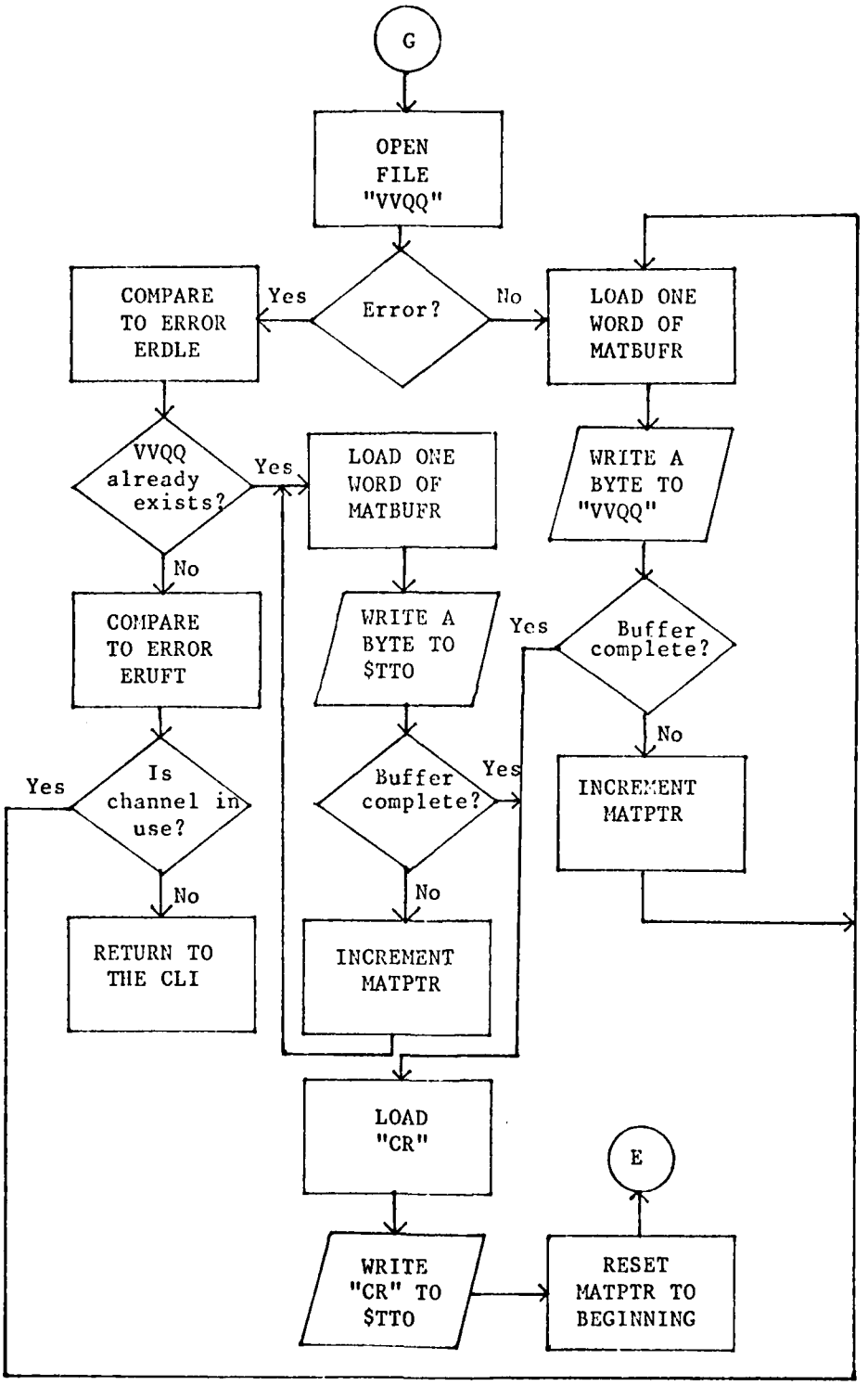


Fig 34. SYSIN.SR (Part 5)

Appendix C

Loading and Executing MONITOR

Each individual routine or subroutine created for the NOVA/ECLIPSE must be separately compiled or assembled. In accordance with the RDOS conventions, all assembly language routines have been given the file name extension ".SR" . All FORTRAN routines have been given the file name extension ".FR" . For example, the source code for the main interpreter program is named MONITOR.FR, and the source code for the separate task program is named SYSIN.SR. To assemble each assembly langauge source program, the MACRO assembler was used. A typical instruction to cause assembly follows:

```
MAC SYSIN
```

The extension is not needed, since the assembler automatically searches for the file name with the extension ".SR" . To compile each FORTRAN language source program, the FORTRAN compiler was used. A typical instruction to cause compiling follows:

```
FORT MONITOR
```

The extension is not needed, since the compiler automatically searches for the file name with the extension ".FR" . The local switch "/L" (slash and L) may be used to create a disk file to contain the assembled/compiled results.

The RDOS relocatable loader was used to load all previously assembled and compiled programs. The first file name listed in the loader instruction sequence becomes the executable save

file. All such files are given the file name extension ".SV" . The local switch "/L" may be used to create a disk file to contain the load map of the results. The following string command was used to load all programs/sources that constitute the MONITOR command language interpreter:

```
RLDR/P/U MONITOR BGIN PROMPT REVERT GETRSPS CNVRT ^
WRITSYSTM WRSYS WRITLOCAL READLWRITS RDAWR SENDFILE ^
EXCLI GETFILE READYREAD RECEVFILE TOTERM TERMOP ^
SYSIN FMT.LB FORT.LB MINE5/L
```

Once executed, file MINE5 contains the load map and file MONITOR.SV is the executable binary file. The local switch "/P" causes the normal relocatable value of each separate binary file to be printed out to file MINE5, and the switch "/U" causes a chain of undefined symbols to be maintained. File FORT.LB supplies needed FORTRAN runtime libraries and file FMT.LB supplies the needed multitasking library. The order and sequence of the loader command is as specified in the RDOS Reference Manual (Ref 9:D-6).

Appendix D

Program MONITOR Source Listing

```
C-------------------------------------------------------------------
C-------------------------------------------------------------------
C
C    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
C    +                                                             +
C    +    MONITOR.FR                                               +
C    +        ****  CREATED 30 AUGUST 1980; REV 01  **** +
C    +                                                             +
C    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
C
C-------------------------------------------------------------------
C-------------------------------------------------------------------
C
C                              **********
C
C       Program MONITOR.FR provides a simplified, general purpose
C       Command Language for use in computer to computer dialog.
C       The host system is a Data General (DG) NOVA 2/10 that is
C       connected via a shared disk operating system to a DG
C       ECLIPSE S/250.  Via a standard RS 232 modem link, this
C       program allows intercommunication to any compatible computer
C       system.  Hereafter, the NOVA/ECLIPSE is referred to as
C       "local" and the connected computer as "system."  The
C       program was initially designed for intercommunication with
C       the ASD Control Data Corp CYBER 750 as the "system."
C
C       MONITOR serves as a basic command string interpreter.  Upon
C       execution of MONITOR, the user is provided instructions on
C       how to access the "system" and a prompt -">"- to enter
C       commands.  Command structure and use are described in the
C       MONITOR Command Language User's Manual.
C
C       "Local" input is received on a "local" device channel - $TTO1.
C       "Local" output is transmitted on device channel - $TTI1.  Via
C       multitasking, a task (SYSIN.SR) is always monitoring input
C       while another task (MONITOR.FR) is always conducting output.
C       MONITOR acts as the executive control of these functions, in
C       addition to its role as interpreter.
C
C       The structure, concept, design, and implementation of program
C       MONITOR and its assorted subroutines are detailed in the thesis
C       that accompanied the development - AFIT/GCS/EE/80-2.
C
C                              **********
C
C-------------------------------------------------------------------
C-------------------------------------------------------------------
```

```
C
C     **  Start the program by defining tasks, channels, parameters,  **
C         etc.  CHANTASK 77,2 declares that up to 77 distinct channel
C         numbers may be used in the program and that 2 asynchronous
C         tasks may be executing "simultaneously."
C
          CHANTASK 77,2
C
C     **  PARAMETER MDIM1 is the size of the one-dimensional command  **
C         instruction arrays.  MDIM2 is the size of the one-dimensional
C         argument arrays and valid instruction string array.  MDIM3
C         is the size of the one-dimensional response arrays.
C
          PARAMETER MDIM1 = 82, MDIM2 = 30, MDIM3 = 40
C
C     **  SYSIN.SR is a separately-compiled assembly language program **
C         that serves as the task for monitoring all "system" input.
C         DG FORTRAN requires this program to be externally defined in
C         the calling program (MONITOR), before it is activated via a
C         call to ITASK.
C
          EXTERNAL SYSIN
C
C     **  INPUT is the initial array store for input commands.  After **
C         a determination of the command's validity, IACTFILE is the
C         array store for command strings read from the action file.
C         Up to four (4) arguments are possible with any single
C         input command, and the arguments are stored in the IARG
C         arrays.  Up to two (2) separate responses from the "system"
C         may be provided, and they are stored in the IRSP arrays.
C
          DIMENSION INPUT(MDIM1), ICOMMAND(MDIM2), IARG1(MDIM2)
          DIMENSION IARG2(MDIM2), IARG3(MDIM2), IARG4(MDIM2)
          DIMENSION IACTFILE(MDIM1), IRSP1(MDIM3), IRSP2(MDIM3)
C
C     **  Characters used for comparisons and decisions are data      **
C         initialized.  Therefore, they must be declared COMMON
C         as well.  The data are mnemonic.  For example, KLTRW is
C         the letter "W" and KOMMA is a "," .
C
          COMMON /KONST/ KSPACE, KSLSH, KOMMA
          COMMON /KONST/ KUPAROW, KLTRL, KLTRT
          COMMON /KONST/ KNULL, KPERIOD, KLTRF
          COMMON /KONST/ KLTRE, KLTRC, KLTRW
          COMMON /KONST/ KLTRR, KSHARPSGN ,KLTRS
          COMMON /KONST/ KNUM1, KNUM2, KNUM3
          COMMON /KONST/ KNUM4
C
          DATA KSPACE,KSLSH,KOMMA/ "<40><40>","<57><40>","<54><40>"/
          DATA KUPAROW,KLTRL,KLTRT/ "<136><40>","<114><40>","<124><40>"/
          DATA KNULL,KPERIOD,KLTRF/ "<0><40>","<56><40>","<106><40>"/
          DATA KLTRE,KLTRC,KLTRW/ "<105><40>","<103><40>","<127><40>"/
          DATA KLTRR,KSHARPSGN,KLTRS/ "<122><40>","<43><40>","<123><40>"/
          DATA KNUM1,KNUM2,KNUM3/ "<61><40>","<62><40>","<63><40>"/
          DATA KNUM4/ "<64><40>"/
```

```
C
C     **  DG FORTRAN allows labels to be tagged.  The following are   **
C         thus defined:
C
          ASSIGN 102 TO IDOOVER
          ASSIGN 106 TO IILGNUM
          ASSIGN 108 TO I1CONT
          ASSIGN 202 TO IPROMPT
          ASSIGN 206 TO IEND206
          ASSIGN 304 TO ISYNERR
          ASSIGN 306 TO IRMSTOR
          ASSIGN 404 TO ILMSTOR
          ASSIGN 406 TO I2CONT
          ASSIGN 412 TO ICMDSTOR
          ASSIGN 420 TO I1ARGSTOR
          ASSIGN 428 TO I2ARGSTOR
          ASSIGN 436 TO I3ARGSTOR
          ASSIGN 444 TO I4ARGSTOR
          ASSIGN 446 TO ILGTHERR
          ASSIGN 502 TO IEXAMFILE
          ASSIGN 506 TO INOMOENTRY
          ASSIGN 508 TO ICHKHDR
          ASSIGN 512 TO I3CONT
          ASSIGN 602 TO I4CONT
          ASSIGN 606 TO IEXECUTE
          ASSIGN 608 TO ICHKSUBS
          ASSIGN 610 TO ICONTCHK
          ASSIGN 614 TO NRMSTOR
          ASSIGN 615 TO I5CONT
          ASSIGN 702 TO ITERMOP
C
C     **  BGIN.SR opens channels 21 and 22 for "local" input and      **
C         output.
C
          CALL BGIN
C
C     **  The various possible action files to be interpreted are     **
C         opened on the channels indicated.
C
          CALL OPEN (1,"CACT",2,IEROR,82)
                  IF (IEROR .NE. 1) STOP  CACT NOT OPENED PROPERLY.
          CALL OPEN (2,"DACT",2,JEROR,82)
                  IF (JEROR .NE. 1) STOP  DACT NOT OPENED PROPERLY.
          CALL OPEN (3,"VACT",2,KEROR,82)
                  IF (KEROR .NE. 1) STOP  VACT NOT OPENED PROPERLY.
          CALL OPEN (4,"MACT",2,LEROR,82)
                  IF (LEROR .NE. 1) STOP  MACT NOT OPENED PROPERLY.
C
C
```

```
        TYPE "The Monitor program you have entered provides "
        TYPE "intercommunication between the NOVA/ECLIPSE computer"
        TYPE "system and your choice of another system."
        TYPE " "
        TYPE " "
C IDOOVER = 102
  102   TYPE "Please enter the digit opposite the action file"
        TYPE "you desire to use: "
        TYPE " "
        TYPE "                  1 -- CDC CYBER"
        TYPE "                  2 -- DEC 10    "
        TYPE "                  3 -- VAX 11/780 "
        TYPE "                  4 -- Your own  "
C
C    **  Call the program PROMPT to signal the user to              **
C        provide some kind of terminal entry.
C
        CALL PROMPT
C
C    **  Get the correct action file requested by the user.         **
C
        READ (11,104) INTRY
  104   FORMAT (I1)
C
        IF (INTRY .LE. 0 .OR. INTRY .GE. 5) GO TO IILGNUM
        GO TO (1,2,3,4) INTRY
C
C IILGNUM = 106
  106   TYPE "You have entered an illegal number.  Try again!"
        GO TO IDOOVER
C
  1     TYPE  "You have selected the CYBER."
        GO TO I1CONT
C
  2     TYPE  "You have selected the DEC."
        GO TO I1CONT
C
  3     TYPE  "You have selected the VAX."
        GO TO I1CONT
C
  4     TYPE  "You have selected your own file."
        GO TO I1CONT
C
C    **  Once the desired action file has been selected, GETRSPS.FR **
C        finds the action file and stores the responses to be sought
C        from the "system" from this point forward.
C
C I1CONT = 108
  108   CALL GETRSPS (INTRY,IRSP1,IRSP2,I1SSZ,I2SSZ)
C
C    **  CNVRT.SR converts the characters stored in FORTRAN format, **
C        and found in GETRSPS, to assembly language format.  It
C        uses the same arrays and also returns the size of the
C        response arrays.
C
        CALL CNVRT (IRSP1,IRSP2,I1SSZ,I2SSZ)
```

```
C
C-----------------------------------------------------------------
C
C    **  The preliminaries are over and the program is now ready     **
C        for user instruction inputs.
C
       TYPE "Thank you.  Please enter a command."
C
C    **  ITASK activates task SYSIN.SR with identity number ten (10) **
C        and priority one (1).  Thus, SYSIN has lower priority than
C        the calling program - MONITOR.  MONITOR has priority zero (0).
C
       CALL ITASK (SYSIN,10,1,IER,1)
              IF (IER .NE. 1) STOP  SYSIN NOT ACTIVATED PROPERLY.
C
C    **  Provide the user with the prompt ">" .                      **
C
C IPROMPT = 202
  202  CALL PROMPT
C
C    **  After each access to the action file, its pointer needs     **
C        to be reset to the file's beginning.  The call to FSEEK
C        resets the pointer accordingly.
C
       CALL FSEEK (INTRY,0)
C
C    **  Read what the user inputs and store in INPUT.  Word entries **
C        may be separated by individual commas or single spaces.
C
       READ (11,204) (INPUT(I), I = 1, MDIM1)
  204  FORMAT (82A1)
C
C    **  Check to see if user desires terminal only operation or a   **
C        return to the "local" command language - CLI.  An input of
C        "^L" reverts user to the "local" CLI.  An input of "^T"
C        causes terminal only operation.
C
       DO 206 INDXA = 1, MDIM1
       IF (INPUT(INDXA) .NE. KUPAROW .OR. INDXA .GE. MDIM1)
     +  GO TO IEND206
              INDXA = INDXA + 1
              IF (INPUT(INDXA) .EQ. KLTRL) CALL REVERT
              IF (INPUT(INDXA) .EQ. KLTRT) GO TO ITERMOP
              INDXA = INDXA - 1
C IEND206 = 206
  206  CONTINUE
C
C-----------------------------------------------------------------
```

```
C
C     **  Search array input right to left to find the first non-null/**
C         non-blank character.
C
          DO 302 INDXB = 1, MDIM1
                  IF (INPUT((MDIM1 + 1) - INDXB) .EQ. KOMMA)
     +                    GO TO ISYNERR
                  IF (INPUT((MDIM1 + 1) - INDXB) .NE. KNULL .AND.
     +                  INPUT((MDIM1 + 1) - INDXB) .NE. KSPACE)
     +                    GO TO IRMSTOR
  302     CONTINUE
C
C     **  If none, so state and return to prompt.                      **
C
          TYPE " "
          TYPE "  *****  INVALID COMMAND - EMPTY STRING  *****"
          GO TO IPROMPT
C
C     **  If the string ended with a separator comma, so state         **
C         and return to the prompt.
C
C ISYNERR = 304
  304     TYPE " "
          TYPE "  *****  SYNTAX ERROR ***** "
          TYPE "  *****  FIRST OR LAST LITERAL INVALID SEPARATOR  *****  "
          GO TO IPROMPT
C
C     **  Otherwise, store the index of the rightmost character.       **
C
C IRMSTOR = 306
  306     IRTMSTINDX = (MDIM1 + 1) - INDXB
C
C-------------------------------------------------------------------------
C
C     **  Now find the leftmost character.                             **
C
          DO 402 INDXC = 1, MDIM1
                  IF (INPUT(INDXC) .EQ. KOMMA) GO TO ISYNERR
                  IF (INPUT(INDXC) .NE. KSPACE) GO TO ILMSTOR
  402     CONTINUE
C
C     **  This error return should never be taken.                     **
C
          TYPE " "
          TYPE "  *****  INVALID COMMAND - EMPTY STRING  *****"
          GO TO IPROMPT
C
C     **  Discard initial blanks/spaces and store it.                  **
C
C ILMSTOR = 404
  404     LTMOSTINDX = INDXC
C
```

```
C     **  Check for obvious error condition.                          **
C
        IF (LTMOSTINDX .NE. IRTMSTINDX) GO TO 12CONT
C
        TYPE " "
        TYPE "  *****  INVALID COMMAND - TOO FEW CHARACTERS  ***** "
        GO TO IPROMPT
C
C     **  Now search the input string for the command portion, i.e.,  **
C         until a separator or a rightmost character is encountered.
C
C I2CONT = 406
  406   DO 408 INDXD = LTMOSTINDX, IRTMSTINDX
                IF (INPUT(INDXD) .EQ. KSPACE .OR. INPUT(INDXD) .EQ.
     +                  KOMMA) GO TO ICMDSTOR
  408   CONTINUE
C
C     **  If the string is a single command, store it in COMMAND.     **
C         First, initialize index for the COMMAND array.
C
        NDIM1 = (IRTMSTINDX - LTMOSTINDX) + 1
        IF (NDIM1 .GT. MDIM2) GO TO ILGTHERR
        DO 410 INDXE = 1, NDIM1
                ICOMMAND(INDXE) = INPUT(INDXE + (LTMOSTINDX - 1))
  410   CONTINUE
C
C     **  At each point that the number of arguments is determined,   **
C         jump ahead to execute the command.  In this case, for example,
C         there is just a command word and no arguments.
C
C
        NUMARGS = 0
        GO TO IEXAMFILE
C
C
C     **  A separator was encountered, so there is more than just a   **
C         single command.  Store the command portion and resolve the
C         rest of the string.
C
C ICMDSTOR = 412
  412   NDIM1 = INDXD - LTMOSTINDX
        IF (NDIM1 .GT. MDIM2) GO TO ILGTHERR
        DO 414 INDXF = 1, NDIM1
                ICOMMAND(INDXF) = INPUT(INDXF + (LTMOSTINDX - 1))
  414   CONTINUE
C
C     **  Proceed to establish the value of the first argument.       **
C
        IDXP1 = INDXD + 1
        DO 416 INDXG = IDXP1, IRTMSTINDX
                IF (INPUT(INDXG) .EQ. KSPACE .OR. INPUT(INDXG) .EQ.
     +          KOMMA) GO TO I1ARGSTOR
  416   CONTINUE
C
```

```
C     **  If a single argument, process it.                          **
C
        NDIM2 = IRTMSTINDX - INDXD
        IF (NDIM2 .GT. MDIM2) GO TO ILGTHERR
        DO 418 INDXH = 1, NDIM2
                IARG1(INDXH) = INPUT(INDXH + INDXD)
  418   CONTINUE
C
C
        NUMARGS = 1
        GO TO IEXAMFILE
C
C
C     **  A separator was encountered.  Store the first argument      **
C         portion and resolve the rest of the string.
C
C I1ARGSTOR = 420
  420   NDIM2 = INDXG - (INDXD + 1)
        IF (NDIM2 .GT. MDIM2) GO TO ILGTHERR
        DO 422 INDXI = 1, NDIM2
                IARG1(INDXI) = INPUT(INDXI + INDXD)
  422   CONTINUE
C
C     **  Proceed to get the next argument for array 2.              **
C
        IGXP1 = INDXG + 1
        DO 424 INDXJ = IGXP1, IRTMSTINDX
                IF (INPUT(INDXJ) .EQ. KSPACE .OR. INPUT(INDXJ) .EQ.
     +          KOMMA) GO TO I2ARGSTOR
  424   CONTINUE
C
C     **  Just two arguments - process the second.                   **
C
        NDIM3 = IRTMSTINDX - INDXG
        IF (NDIM3 .GT. MDIM2) GO TO ILGTHERR
        DO 426 INDXK = 1, NDIM3
                IARG2(INDXK) = INPUT(INDXK + INDXG)
  426   CONTINUE
C
C
        NUMARGS = 2
        GO TO IEXAMFILE
C
C
C     **  A separator was encountered.  Store the second argument     **
C         portion and resolve the rest of the string.
C
C I2ARGSTOR = 428
  428   NDIM3 = INDXJ - (INDXG + 1)
        IF (NDIM3 .GT. MDIM2) GO TO ILGTHERR
        DO 430 INDXL = 1, NDIM3
                IARG2(INDXL) = INPUT(INDXL + INDXG)
  430   CONTINUE
C
```

```
C     **  Procede to get the next argument for array 3.                **
C
         IJXP1 = INDXJ + 1
         DO 432 INDXM = IJXP1, IRTMSTINDX
                 IF (INPUT(INDXM) .EQ. KSPACE .OR. INPUT(INDXM) .EQ.
     +           KOMMA) GO TO I3ARGSTOR
  432    CONTINUE
C
C     **  Just three arguments - process the third.                    **
C
         NDIM4 = IRTMSTINDX - INDXJ
         IF (NDIM4 .GT. MDIM2) GO TO ILGTHERR
         DO 434 INDXN = 1, NDIM4
                 IARG3(INDXN) = INPUT(INDXN + INDXJ)
  434    CONTINUE
C
C
         NUMARGS = 3
         GO TO IEXAMFILE
C
C
C     **  A separator was encountered.  Store the third argument       **
C         and resolve the rest of the string.
C
C I3ARGSTOR = 436
  436    NDIM4 = INDXM - (INDXJ + 1)
         IF (NDIM4 .GT. MDIM2) GO TO ILGTHERR
         DO 438 INDXO = 1, NDIM4
                 IARG3(INDXO) = INPUT(INDXO + INDXJ)
  438    CONTINUE
C
C     **  Procede to get the next argument for array 4.                **
C
         IMXP1 = INDXM + 1
         DO 440 INDXP = IMXP1, IRTMSTINDX
                 IF (INPUT(INDXP) .EQ. KSPACE .OR. INPUT(INDXP) .EQ.
     +           KOMMA) GO TO I4ARGSTOR
  440    CONTINUE
C
C     **  Just four arguments - process the fourth.                    **
C
         NDIM5 = IRTMSTINDX - INDXM
         IF (NDIM5 .GT. MDIM2) GO TO ILGTHERR
         DO 442 INDXQ = 1, NDIM5
                 IARG4(INDXQ) = INPUT(INDXQ + INDXM)
  442    CONTINUE
C
C
         NUMARGS = 4
         GO TO IEXAMFILE
C
C
```

```
C    **  A separator was encountered.  There are too many arguments. **
C        Give an error message and return to the prompt.
C
C I4ARGSTOR = 444
  444   TYPE " "
        TYPE "  *****  INVALID COMMAND - TOO MANY ARGUMENTS  *****"
        GO TO IPROMPT
C
C ILGTHERR = 446
  446   TYPE " "
        TYPE "  *****  INVALID COMMAND - TOO MANY CHARACTERS  ***** "
        GO TO IPROMPT
C
C-----------------------------------------------------------------------
C
C    **  Once a potentially valid command string has been accepted,  **
C        it is time to examine the action file for that command.
C        The strings of the action file are read into IACTFILE.
C
C IEXAMFILE = 502
  502   READ (INTRY,504) (IACTFILE(J), J = 1, MDIM1)
  504   FORMAT (82A1)
C
C    **  Check the first character of the line just read.            **
C        If an "F", there are no more entries to read.  If a
C        period, then the encountered header line needs to be checked.
C
        IF (IACTFILE(1) .EQ. KLTRF) GO TO INOMOENTRY
        IF (IACTFILE(1) .EQ. KPERIOD) GO TO ICHKHDR
        GO TO IEXAMFILE
C
C INOMOENTRY = 506
  506   TYPE " "
        TYPE "  *****  INVALID COMMAND  ***** "
        TYPE "  *****  COMMAND NOT IN ACTION FILE  ***** "
        TYPE "  *****  OR SUPPLIED NOT EQUAL REQUIRED ARGUMENTS  ***** "
        GO TO IPROMPT
C
C    **  All commands are ten (10) characters or less.  Here,        **
C        the command portion of the header is determined.
C
C ICHKHDR = 508
  508   DO 510 INDXZ = 7, 17
                IF (IACTFILE(INDXZ) .EQ. KSPACE .OR.
     +          IACTFILE(INDXZ) .EQ. KOMMA) GO TO I3CONT
  510   CONTINUE
C
```

```
C     **  Compare the header command to that input by the user.       **
C         If correct, proceed.  Otherwise, return to read the action
C         file again until finished, or the next header is encountered.
C
C I3CONT = 512
  512    INDXY = INDXZ - 7
C
         IF (NDIM1 .NE. INDXY) GO TO IEXAMFILE
         DO 514 INDXX = 1, INDXY
                 IF (IACTFILE(INDXX + 6) .NE. ICOMMAND(INDXX))
     +           GO TO IEXAMFILE
  514    CONTINUE
C
C     **  Look at the number of required arguments for this command.  **
C         If there are no arguments, then two spaces after the
C         command string in the header will be a space.  Similarly,
C         if there is one argument, five spaces after the command string
C         in the header will be a space, and so forth.  The INDX
C         numbers are the location of the sharpsigns in the header
C         string.  Compare the command string in IACTFILE with the
C         sharpsign location to determine how many arguments are required.
C
         INDX1 = INDXZ + 1                ;The sharpsign is 2,5,8,
         INDX2 = INDXZ + 4                ;and 11 spaces after command
         INDX3 = INDXZ + 7                ;string - 1,4,7, or 10 spaces
         INDX4 = INDXZ + 10               ;after comma, if arguments exist
C
         NNUMARGS = 4
         IF (IACTFILE(INDX4) .EQ. KSPACE) NNUMARGS = 3
         IF (IACTFILE(INDX3) .EQ. KSPACE) NNUMARGS = 2
         IF (IACTFILE(INDX2) .EQ. KSPACE) NNUMARGS = 1
         IF (IACTFILE(INDX1) .EQ. KSPACE) NNUMARGS = 0
C
C     **  Compare the required number of arguments with the supplied  **
C         number of arguments.
C
         IF (NUMARGS .EQ. NNUMARGS) GO TO I4CONT
C
C     **  If the arguments supplied are not equal the number required,**
C         then continue to examine the action file for the same named
C         command with the appropriate number of arguments.  (NOTE:  This
C         means that more than one command with the same name may be entered
C         and found within the action file, but each must have a different
C         number of arguments.)
C
         GO TO IEXAMFILE
C
C-------------------------------------------------------------------
```

```
C
C    **  If arguments required equal arguments supplied, then read   **
C        the next line in the action file, which is the first command
C        in the command sequence.  The last command in the sequence
C        precedes "END." .
C
C I4CONT = 602
  602   READ(INTRY,604) (IACTFILE(K), K = 1, MDIM1)
  604   FORMAT (82A1)
C
C    **  Look at the first two control letters to determine          **
C        specific actions to take.  If "END." is encountered, the command
C        sequence is over.  If an "S" or "C" is encountered in column
C        two (2), then the string needs to be checked for substitution
C        of arguments for the sharpsigns in the action file.  Then the
C        remaining control letters are examined.  Appropriate subroutines
C        are called to execute the strings as required.  Each subroutine
C        returns to the place where a new line out of the action file
C        may be read and examined.  For further discussion of control
C        characters, look at the action file documentation or the
C        MONITOR Command Language User's Manual.
C
        IF (IACTFILE(1) .EQ. KLTRE) GO TO IPROMPT
        IF (IACTFILE(2) .EQ. KLTRS .OR. IACTFILE(2) .EQ. KLTRC)
     +  GO TO ICHKSUBS
C IEXECUTE = 606
  606   IF (IACTFILE(1) .EQ. KLTRW .AND. IACTFILE(2) .EQ. KLTRS)
     +  CALL WRITSYSTM (IACTFILE,NRTMSTINDX,$602)
        IF (IACTFILE(1) .EQ. KLTRW .AND. IACTFILE(2) .EQ. KLTRL)
     +  CALL WRITLOCAL (IACTFILE,MDIM1,$602)
        IF (IACTFILE(1) .EQ. KLTRR .AND. IACTFILE(2) .EQ. KLTRW)
     +  CALL READLWRITS ($602)
        IF (IACTFILE(1) .EQ. KLTRW .AND. IACTFILE(2) .EQ. KLTRC)
     +  CALL SENDFILE (IACTFILE,MDIM1,$602)
        IF (IACTFILE(1) .EQ. KLTRR .AND. IACTFILE(2) .EQ. KLTRC)
     +  CALL RECEVFILE (IACTFILE,MDIM1,$602)
        IF (IACTFILE(1) .EQ. KLTRR .AND. IACTFILE(2) .EQ. KLTRR)
     +  CALL READYREAD ($602)
C
C    **  If unexpected letters are encountered, the action file is   **
C        suspect.  Abort and try again.
C
        TYPE " "
        TYPE "  *****  COMMAND ABORT  ***** "
        TYPE "  *****  UNEXPECTED ENTRY IN ACTION FILE  ***** "
        GO TO IPROMPT
```

```
C
C    **  Check for and make any required substitutions.                **
C
C ICHKSUBS = 608
  608   IBGININDX = 9
C
C    **  Start looking in column nine (9) for sharpsigns to replace. **
C        Then use new value of IBGININDX on subsequent iterations.
C
C ICONTCHK = 610
  610   DO 612 NAINDX = IBGININDX, MDIM1
              IF (IACTFILE(NAINDX) .EQ. KSHARPSGN) GO TO I5CONT
  612   CONTINUE
C
C    **  There were no sharp signs or there are no more sharp signs. **
C        Now find the rightmost index of the current command line
C        and return to execute string.
C
        DO 613 NCINDX = 1, MDIM1
              IF (IACTFILE (( MDIM1 + 1) - NCINDX) .NE. KNULL .AND.
     +        IACTFILE (( MDIM1 + 1) - NCINDX) .NE. KSPACE)
     +                 GO TO NRMSTOR
  613   CONTINUE
C
C NRMSTOR = 614
  614   NRTMSTINDX = (MDIM1 + 1) - NCINDX
C
        GO TO IEXECUTE
C
C    **  Collapse the array about the sharp signs.                     **
C
C I5CONT = 615
  615   NAP1INDX = NAINDX + 1
        NAM1INDX = NAINDX - 1
        IF (IACTFILE(NAP1INDX) .EQ. KNUM1) IVALOFSGN = 1
        IF (IACTFILE(NAP1INDX) .EQ. KNUM2) IVALOFSGN = 2
        IF (IACTFILE(NAP1INDX) .EQ. KNUM3) IVALOFSGN = 3
        IF (IACTFILE(NAP1INDX) .EQ. KNUM4) IVALOFSGN = 4
        M2MDIM1 = MDIM1 - 2
        DO 616 NBINDX = NAINDX, M2MDIM1
              IACTFILE(NBINDX) = IACTFILE(NBINDX +2)
  616   CONTINUE
C
```

```
C     **  Determine the size of the argument arrays (IARG) and expand **
C         the command sequence in IACTFILE to make room for the
C         substitution of the IARG arrays, where the corresponding
C         sharpsigns had been.  IVALOFSGN is the value of the particular
C         sharpsign being substituted with IARG.
C
          GO TO (618,620,622,624), IVALOFSGN
C
  618     ISIZEARG = NDIM2
          GO TO 626
  620     ISIZEARG = NDIM3
          GO TO 626
  622     ISIZEARG = NDIM4
          GO TO 626
  624     ISIZEARG = NDIM5
C
C     **  Expand the array to make room for the substitution of IARC. **
C
  626     IRGHTMST = MDIM1 - ISIZEARG
          INCRMAX = IRGHTMST - NAINDX
          IACTFILE(IRGHTMST + ISIZEARG) = IACTFILE(IRGHTMST)
          DO 628 M1 = 1, INCRMAX
                  IACTFILE((IRGHTMST - M1) + ISIZEARG) =
     +            IACTFILE(IRGHTMST - M1)
  628     CONTINUE
C
C     **  Replace each sharp sign -- #1, #2, #3, and #4 -- if used.   **
C
          GO TO (630,634,638,642), IVALOFSGN
C
  630     DO 632 M2 = 1, ISIZEARG
                  IACTFILE(NAM1INDX + M2) = IARG1(M2)
  632     CONTINUE
          GO TO 646
C
  634     DO 636 M3 = 1, ISIZEARG
                  IACTFILE(NAM1INDX + M3) = IARG2(M3)
  636     CONTINUE
          GO TO 646
C
  638     DO 640 M4 = 1, ISIZEARG
                  IACTFILE(NAM1INDX + M4) = IARG3(M4)
  640     CONTINUE
          GO TO 646
C
  642     DO 644 M5 = 1, ISIZEARG
                  IACTFILE(NAM1INDX + M5) = IARG4(M5)
  644     CONTINUE
C
C     **  Recalculate IBGININDX for the next iteration.              **
C
  646     IBGININDX = NAINDX + ISIZEARG
C
          GO TO ICONTCHK
```

```
C
C-----------------------------------------------------------------------
C
C    **  Before going to the terminal operation mode, inactivate     **
C        (kill) the task SYSIN.  Then call the program TOTERM.SR,
C        which removes defined device codes utilized within SYSIN.
C
C ITERMOP = 702
  702   CALL AKILL(1)
C
        CALL TOTERM
C
C
C       STOP  END OF THE PROGRAM
C
        END
C
C-----------------------------------------------------------------------
C-----------------------------------------------------------------------
C
C    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
C    +                                                         +
C    +                  END MONITOR.FR                         +
C    +                                                         +
C    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
C
C-----------------------------------------------------------------------
C-----------------------------------------------------------------------
```

```
;-------------------------------------------------------------------
;-------------------------------------------------------------------

;    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
;    +                                                             +
;    +    BGIN.SR                                                  +
;    +        ****  CREATED  3 JULY 1980; REV 01  ****            +
;    +                                                             +
;    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +

;-------------------------------------------------------------------
;-------------------------------------------------------------------

;                          **********

;       Program BGIN.SR is called from MONITOR.FR and returns to
;       MONITOR.  It is called at the beginning of the MONITOR program
;       to open devices $TT0 and $TT1 for all subsequent programs.
;       There are no arguments or parameters that are passed via BGIN.

;                          **********

;-------------------------------------------------------------------
;-------------------------------------------------------------------

        .TITL BGIN                      ;Program name - Begin

        .ENT BGIN                       ;Enables outside entry into this
                                        ;program

        .TXTM 1                         ;Packs ASCII strings left to right

        .EXTU                           ;Undefined variables are treated
                                        ;as External Displacement variables

        .EXTN .I                        ;Provides some FORTRAN initialization

;    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

        .NREL                           ;Normal relocatable space starts

        FS.
```

```
;     BEGIN TO OPEN DEVICES
;     - - - - - - - - - - -

BGIN :  JSR @ .FARL

START:  SUB 1,1                         ;Load default mask
        LDA 0, NTTO                     ;Load bytepointer to $TTO
        .SYSTM
        .OPEN 21                        ;Open $TTO on channel 21
        JMP ERROR
        LDA 0, NTTI                     ;Load bytepointer to $TTI
        .SYSTM
        .OPEN 22                        ;Open $TTI on channel 22
        JMP ERROR

        JMP RT                          ;Jump to return location when complete

;     ROUTINE TO RETURN TO THE CLI ABNORMALLY
;     - - - - - - - - - - - - - - - - - - - -

ERROR:  .SYSTM
        .ERTN                           ;Abnormal return - error
        JMP ERROR

;     BYTEPOINTERS DEFINED
;     - - - - - - - - - - -

NTTO:   .+1*2                           ;Bytepointer to device $TTO
        .TXT "$TTO"

NTTI:   .+1*2                           ;Bytepointer to device $TTI
        .TXT "$TTI"

;     - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

RT:     JSR @ .FRET

        FS.=0
        TMP=-167

        .END BGIN

;-------------------------------------------------------------------------
;-------------------------------------------------------------------------

;     + + + + + + + + + + + + + + + + + + + + + + + + + + + +
;     +                                                     +
;     +                    END BGIN.SR                      +
;     +                                                     +
;     + + + + + + + + + + + + + + + + + + + + + + + + + + + +

;-------------------------------------------------------------------------
;-------------------------------------------------------------------------
```

```
;-----------------------------------------------------------------------
;-----------------------------------------------------------------------

;    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
;    +                                                               +
;    +    PROMPT.SR                                                  +
;    +        ****  CREATED  3 JULY 1980;  REV 01  ****             +
;    +                                                               +
;    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +

;-----------------------------------------------------------------------
;-----------------------------------------------------------------------

;                              **********

;       Program PROMPT.SR is called from MONITOR.FR and returns to
;       MONITOR.  It is called several times within MONITOR, each time
;       to provide the user a prompt that information may be entered.
;       The prompt character for MONITOR is ">" .  There are no arguments
;       or parameters that are passed via PROMPT.

;                              **********

;-----------------------------------------------------------------------
;-----------------------------------------------------------------------

        .TITL PROMPT                    ;Program name - Prompt

        .ENT PROMPT                     ;Enables outside entry into this
                                        ;program

        .TXTM 1                         ;Packs ASCII strings left to right

        .EXTU                           ;Undefined variables are treated
                                        ;as External Displacement variables

        .EXTN .I                        ;Provides some FORTRAN initialization

;    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

        .NREL                           ;Normal relocatable space starts

        FS.
```

```
;    WRITE PROMPT TO $TTO
;    - - - - - - - - - - -

PROMPT: JSR @ .FARL

START:  SUB 1, 1                        ;Load default mask
        LDA 0, PRMT                     ;Load bytepointer to prompt
        .SYSTM
        .WRL 21                         ;Write prompt to channel 21
        JMP ERROR
        JMP RT                          ;Jump to return location when complete

;    ROUTINE TO RETURN TO THE CLI ABNORMALLY
;    - - - - - - - - - - - - - - - - - - - -

ERROR:  .SYSTM
        .ERTN                           ;Abnormal return - error
        JMP ERROR

;    BYTEPOINTER DEFINED
;    - - - - - - - - - -

PRMT:   .+1*2                           ;Bytepointer to prompt - > (with
        .TXT "><15>"                    ;carriage return)

;    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

RT:     JSR @ .FRET

        FS.=0
        TMP=-167

        .END PROMPT
;-------------------------------------------------------------------------
;-------------------------------------------------------------------------

;    + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
;    +                                                       +
;    +                  END PROMPT.SR                        +
;    +                                                       +
;    + + + + + + + + + + + + + + + + + + + + + + + + + + + + +

;-------------------------------------------------------------------------
;-------------------------------------------------------------------------
```

```
;-------------------------------------------------------------------
;-------------------------------------------------------------------

;    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
;    +                                                             +
;    +    REVERT.SR                                                +
;    +        ****  CREATED 15 JULY 1980;  REV 01  ****           +
;    +                                                             +
;    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +

;-------------------------------------------------------------------
;-------------------------------------------------------------------

;                          **********

;       Program REVERT.SR is called from MONITOR.FR and always returns
;       to the "local" CLI.  This program is called by the user entering
;       "^L" anywhere in the line that follows a prompt.  There are no
;       parameters or arguments that are passed via REVERT.

;                          **********

;-------------------------------------------------------------------
;-------------------------------------------------------------------

        .TITL REVERT                    ;Program name - Revert to CLI

        .ENT REVERT                     ;Enables outside entry into this
                                        ;program

        .TXTM 1                         ;Packs ASCII strings left to right

        .EXTU                           ;Undefined variables are treated
                                        ;as External Displacement variables

        .EXTN .I                        ;Provides some FORTRAN initialization

;    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

        .NREL                           ;Normal relocatable space starts

        FS.
```

```
;     NOTIFY USER OF RETURN TO CLI
;     - - - - - - - - - - - - - - - -

REVERT: JSR @ .FARL

        SUB 1, 1                        ;Load default mask
        LDA 0, NOTEA                    ;Load bytepointer to message
        .SYSTM
        .WRL 21                         ;Write message to channel 21 ($TTO)
        JMP ERROR

;     ROUTINE FOR NORMAL RETURN TO THE CLI
;     - - - - - - - - - - - - - - - - - - - -

START:  .SYSTM
        .RTN                            ;Normal return - no error
        JMP ERROR
        JMP RT                          ;Jump to return location when complete

;     ROUTINE TO RETURN TO THE CLI ABNORMALLY
;     - - - - - - - - - - - - - - - - - - - -

ERROR:  .SYSTM
        .ERTN                           ;Abnormal return - error
        JMP ERROR

;     BYTEPOINTER DEFINED
;     - - - - - - - - - -

NOTEA:  .+1*2                           ;Bytepointer to message (NOTEA)
        .TXT "You have returned to the local CLI mode.<15>"

;     - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

RT:     JSR @ .FRET

        FS.=0
        TMP=-167

        .END REVERT

;-------------------------------------------------------------------------------
;-------------------------------------------------------------------------------

;     + + + + + + + + + + + + + + + + + + + + + + + + + + + +
;     +                                                     +
;     +                   END REVERT.SR                     +
;     +                                                     +
;     + + + + + + + + + + + + + + + + + + + + + + + + + + + +

;-------------------------------------------------------------------------------
;-------------------------------------------------------------------------------
```

```
C-------------------------------------------------------------------
C-------------------------------------------------------------------
C
C    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
C    +                                                           +
C    +    GETRSPS.FR                                             +
C    +        ****  CREATED 8 AUGUST 1980;  REV 00 ****         +
C    +                                                           +
C    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
C
C-------------------------------------------------------------------
C-------------------------------------------------------------------
C
C                              **********
C
C       Program GETRSPS.FR is called by MONITOR.FR and returns to
C       MONITOR.  Its sole function is to read the selected action file
C       (seeking that line that begins with the control character "I" in
C       column one (1)) and store up to two (2) possible responses that
C       are expected from the "system" when intercommunication is taking
C       place.
C
C       The call to GETRSPS is as follows:
C
C               CALL GETRSPS (INTRY,IRSP1,IRSP2,I1SSZ,I2SSZ)
C
C       (Integer) INTRY is the user inputed channel number of the
C       selected action file and is the only parameter passed from
C       MONITOR to GETRSPS.  The remaining parameters are returned from
C       GETRSPS to MONITOR.  They are:
C
C               IRSP1 - one-dimensional array containing first response,
C                       if any
C               IRSP2 - same for second response, if any
C               I1SSZ - integer that indicates size of IRSP1
C               I2SSZ - same for IRSP2
C
C                          **********
C
C-------------------------------------------------------------------
C-------------------------------------------------------------------
```

```
C
C     **  PARAMERTER MDIM1 is the size of the one-dimensional command **
C         sequence array.  MDIM3 is the size of the one-dimensional
C         response arrays.
C
          PARAMETER MDIM1 = 82, MDIM3 = 40
C
C     **  GETRSPS receives an input channel and returns the first and **
C         second responses (if any), as well as their sizes.
C
          SUBROUTINE GETRSPS (INCHAN,I1STRSP,I2NDRSP,I1STSZ,I2NDSZ)
C
C     **  IINITAR is an initialized array that stores the command     **
C         sequence line as read from the action file.  It parallels the
C         use of IACTFILE in MONITOR.
C
          DIMENSION  IINITAR(MDIM1), I1STRSP(MDIM3), I2NDRSP(MDIM3)
C
C     **  Characters used for comparisons and decisions are data-     **
C         initialized.  They must therefore be declared COMMON.
C
          COMMON /IKONST/ KLTRI, KSPACE, KOMMA
C
          DATA KLTRI,KSPACE,KOMMA/ "<111><40>","<40><40>","<54><40>"/
C
C     **  Since the response arrays may be empty, they must be        **
C         cleared.  Then, if no responses are provided, the expected
C         response will be blanks or spaces by default.
C
          DO 1002 JJ = 1, MDIM3
                  I1STRSP(JJ) = KSPACE
                  I2NDRSP(JJ) = KSPACE
 1002     CONTINUE
C
C     **  Read a line out of the action file and begin search.        **
C
 1004     READ (INCHAN,1006) (IINITAR(II), II = 1, MDIM1)
 1006     FORMAT (82A1)
C
```

```
C     **  If the control character "I" is not encountered, read the   **
C         next line of the action file until such a line is encountered.
C         (NOTE:  Each action file must have a line with control
C         character "I" or GETRSPS will generate an error condition.)
C         When the desired line is encountered, begin searching for a
C         response in column nine (9) and beyond.  If nothing is there,
C         return to the calling program.  If a separator comma is
C         encountered, there are two responses.  Get the first and then
C         get the second.  Otherwise, get the first response only and
C         return to the calling program.
C
        IF (IINITAR(1) .NE. KLTRI) GO TO 1004
        DO 1008 ISUB1 = 9, MDIM1
                IF (IINITAR(ISUB1) .EQ. KSPACE) GO TO 1014
                IF (IINITAR(ISUB1) .EQ. KOMMA) GO TO 1010
                I1STRSP(ISUB1 - 8) = IINITAR(ISUB1)
                I1STSZ = ISUB1 - 8
 1008   CONTINUE
C
C     **  Begin the search for the second response, if any.  Return   **
C         to the calling program when completed.
C
 1010   IP1SUB1 = ISUB1 + 1
        DO 1012 ISUB2 = IP1SUB1, MDIM1
                IF (IINITAR(ISUB2) .EQ. KSPACE .OR. IINITAR(ISUB2)
     +          .EQ. KOMMA) GO TO 1014
                I2NDRSP(ISUB2 - ISUB1) = IINITAR(ISUB2)
                I2NDSZ = ISUB2 - ISUB1
 1012   CONTINUE
C
C
C     **  Before returning to the calling program, the pointer to the **
C         action file must be reset to the beginning of the file.  FSEEK
C         resets the pointer to the beginning.
C
 1014   CALL FSEEK (INCHAN,0)
C
        RETURN
        END
C
C-------------------------------------------------------------------------
C-------------------------------------------------------------------------
C
C     + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
C     +                                                                 +
C     +                  END GETRSPS.FR                                 +
C     +                                                                 +
C     + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
C
C-------------------------------------------------------------------------
C-------------------------------------------------------------------------
```

```
;-----------------------------------------------------------------------
;-----------------------------------------------------------------------

;    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
;    +                                                               +
;    +    CNVRT.SR                                                   +
;    +        ****  CREATED 15 JULY 1980;  REV 04  ****             +
;    +                                                               +
;    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +

;-----------------------------------------------------------------------
;-----------------------------------------------------------------------

;                              **********

;       Program CNVRT.SR is called from MONITOR.FR and returns to
;       MONITOR.  It is called after GETRSPS.FR and serves to convert
;       the FORTRAN array storage to assembly language storage.  For example,
;       the character "A" is stored in FORTRAN array IRSP1 as <101><40>,
;       whereas the same character is stored in assembly language as
;       <0><101>.  Thus, this program simply converts between the two.
;       The call to CNVRT is as follows:

;               CALL CNVRT (IRSP1,IRSP2,I1SSZ,I2SSZ)

;       IRSP1 and IRSP2 are one-dimensional FORTRAN arrays that are passed
;       from MONITOR to CNVRT.  They contain the responses as found by
;       the program GETRSPS.  I1SSZ and I2SSZ are the size of the response
;       arrays, and are parameters that are passed from CNVRT to MONITOR.
;       Upon completion of CNVRT, the assembly language responses and their
;       sizes are stored in BUF1, BUF2, .FB1, and .FB2, respectively.
;       Subsequent programs then use these locations for comparison
;       purposes, when checking for appropriate responses.

;                          **********

;-----------------------------------------------------------------------
;-----------------------------------------------------------------------

        .TITL CNVRT                     ;Program name - Convert

        .ENT CNVRT                      ;Enables outside entry into
        .ENT .BUF1, .BUF2, .FB1, .FB2   ;this program and these locations

        .TXTM 1                         ;Packs ASCII strings left to right

        .EXTD .FSUB, .FREDI             ;FORTRAN runtime library routines must
                                        ;be decleared external displacements

        .EXTU                           ;Undefined variables are treated as
                                        ;External Displacement variables

        .EXTN .I                        ;Provides some FORTRAN initialization
```

```
;    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

        .ZREL                           ;Zero relocatable space starts


.FB1:   0                               ;Store for the size of first response
.FB2:   0                               ;Same for second response
.BUF1:  BUF1                            ;Buffers that contain responses may be
.BUF2:  BUF2                            ;addressed indirectly via these location

;    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

        .NREL                           ;Normal relocatable space starts

        FS.

;    ESTABLISH THE ARRAYS PASSED IN AS DUMMY ARGUMENTS
;    - - - - - - - - - - - - - - - - - - - - - - -

CNVRT:  JSR @ .FARL

        JSR @ .FRED                     ;Redimension an array, passed as dummy
        ARY2IN                          ;argument.  This is array specifier
        @ STORA+1                       ;This is array size - dummy argument
        STORA+4                         ;This is three word stack specifier

        JSR @ .FRED
        ARY1IN                          ;Same for the other array
        @ STORA+0
        STORA+7

;    STORE SIZE OF RESPONSES
;    - - - - - - - - - - - - -

        LDA 1, @TMP+3, 3                ;Load size of second array from
        STA 1, .FB2                     ;FORTRAN stack - store in .FB2

        LDA 1, @TMP+2, 3                ;Same for first array, but
        STA 1, .FB1                     ;store in .FB1
```

```
;    SELECT EACH ELEMENT OF THE FIRST ARRAY
;    - - - - - - - - - - - - - - - - - - - -

        LDA 0, MIN12SUB                 ;Load subscript one
        STA 0, TMP+12, 3                ;Store on stack
        JMP STRT1                       ;Start iterations

INC1S:  LDA 0, TMP+12, 3                ;On repeated passes, get the
        INC 0, 0                        ;last subscript and increment it
        STA 0, TMP+12, 3                ;Store new subscript

STRT1:  LDA 1, @TMP+2, 3                ;Load maximum subscript
        SUBZ 0, 1, SNC                  ;If maximum exceeded,
        JMP NXTRSP                      ;start same type iteration on next array

        JSR @ .FSUB                     ;Otherwise, get the array element
        3                               ;Number of arguments for library call
        STORA+7                         ;Stack specifier for first array
        STORP+1                         ;Temporary location for FSUB result
        STORA+12                        ;FORTRAN address of subscript

;    CONVERT FROM FORTRAN TO ASSEMBLY LANGUAGE STORAGE
;    - - - - - - - - - - - - - - - - - - - - - - - -

        LDA 0, @TMPP+1, 3               ;Load selected array element
        LDA 2, MSKSAP                   ;Delete second byte, strip parity from
        ANDS 2, 0                       ;first byte, and swap bytes
        STA 0, @BUF1PTR                 ;Store result in BUF1
        LDA 1, BUF1PTR                  ;Load pointer to BUFF1
        INC 1, 1                        ;Increment and store the new
        STA 1, BUF1PTR                  ;pointer

        JMP INC1S                       ;Continue the next iteration

;    ROUTINE TO RETURN TO THE CLI ABNORMALLY
;    - - - - - - - - - - - - - - - - - - - -

ERROR:  .SYSTM
        .ERTN                           ;Abnormal return - error
        JMP ERROR
```

```
;     DEFINE INITIAL SUBSCRIPT SIZE FOR BOTH ARRAYS
;     - - - - - - - - - - - - - - - - - - - - - - -

MIN12SUB:1

;     REPEAT THE ABOVE PROCEDURE FOR THE SECOND ARRAY
;     - - - - - - - - - - - - - - - - - - - - - - - - - -

NXTRSP: LDA 0, MIN12SUB
        STA 0, TMP+12, 3
        JMP STRT2

INC2S:  LDA 0, TMP+12, 3
        INC 0, 0
        STA 0, TMP+12, 3

STRT2:  LDA 1, @TMP+3, 3
        SUBZ 0, 1, SNC
        JMP RETN                        ;Return to calling program

        JSR @ .FSUB
        3
        STORA+4
        STORP+2
        STORA+12

        LDA 0, @TMPP+2, 3
        LDA 2, MSKSAP
        ANDS 2, 0

        STA  0, @BUF2PTR                ;Store result in BUF2
        LDA 1, BUF2PTR
        INC 1, 1
        STA 1, BUF2PTR

        JMP INC2S
```

```
;    DEFINE POINTERS, VARIABLES, AND BUFFERS
;    - - - - - - - - - - - - - - - - - - - - - - -

MSKSAP: 077400                          ;Mask for stripping parity, deleting
                                        ;second byte

BUF1PTR:BUF1                            ;Pointer to BUF1
BUF2PTR:BUF2                            ;Pointer to BUF2

BUF1:   .BLK 40.                        ;Define buffer sizes
BUF2:   .BLK 40.

;    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

RETN:   JSR @ .FRET

;    SPECIFY THE ARRAYS
;    - - - - - - - - - - -

ARY2IN: 3                               ;This is - 2*(number of subscripts)+1
        401                             ;This is - 400*1+1, length=1 and type=1
        MIN12SUB                        ;This is lower subscript size
        @STORA+3                        ;This is higher subscript size

ARY1IN: 3                               ;This is the same for first array
        401
        MIN12SUB
        @STORA+2

;    DEFINE STACK PARAMETERS
;    - - - - - - - - - - - - -

        FS.=17                          ;Frame size for all stack variables
        TMP=-167                        ;Middle of the user's stack
        STORA=200 .MP
        TMPP=TMP+12                     ;First word available for temporary
                                        ;storage
        STORP=STORA+12

        .END CNVRT
```

```
;-----------------------------------------------------------------------
;-----------------------------------------------------------------------

;    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
;    +                                                         +
;    +                    END CNVRT.SR                         +
;    +                                                         +
;    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +

;-----------------------------------------------------------------------
;-----------------------------------------------------------------------
```

```
C-------------------------------------------------------------------
C-------------------------------------------------------------------
C
C    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
C    +                                                             +
C    +    WRITSYSTM.FR                                             +
C    +        ****  CREATED  8 AUGUST 1980;  REV 00  ****+
C    +                                                             +
C    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
C
C-------------------------------------------------------------------
C-------------------------------------------------------------------
C
C                              **********
C
C       Program WRITSYSTM.FR is called by MONITOR.FR and returns to
C       MONITOR, whenever a command sequence from an action file is
C       ready to be sent to the "system."  WRITSYSTM provides a
C       transition to and from an assembly language subroutine that
C       actually transmits data to the "system."  The call to
C       WRITSYSTM is as follows:
C
C               CALL WRITSYSTM (IACTFILE,NRTMSTINDX,$602)
C
C       IACTFILE is one line from the action file that is to be sent
C       to the "system."  NRTMSTINDX is the size of the one-dimensional
C       array IACTFILE, and $602 is the return line number in MONITOR
C       that will next be executed upon return from WRITSYSTM.  All
C       parameters are passed to WRITSYSTM from MONITOR.
C
C                              **********
C
C-------------------------------------------------------------------
C-------------------------------------------------------------------
```

```
C
C     **  WRITSYSTM receives an input array, the dimension of that    **
C         array, and an assigned dummy return variable.
C
          SUBROUTINE WRITSYSTM (IN1ARRAY,I1DIMAR,I1DUMRTN)
          DIMENSION IN1ARRAY(I1DIMAR)
C
C     **  WRSYS actually transmits the contents of IN1ARRAY (of size  **
C         IN1DIMAR) to the "system."
C
          CALL WRSYS (IN1ARRAY, I1DIMAR)
C
C     **  Return to the statement number passed in I1DUMRTN           **
C
          RETURN I1DUMRTN
          END
C
C-------------------------------------------------------------------------
C-------------------------------------------------------------------------
C
C     + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
C     +                                                         +
C     +                END WRITSYSTM.FR                         +
C     +                                                         +
C     + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
C
C-------------------------------------------------------------------------
C-------------------------------------------------------------------------
```

```
;-------------------------------------------------------------------
;-------------------------------------------------------------------

;    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
;    +                                                             +
;    +    WRSYS.SR                                                 +
;    +        ****  CREATED 15 JULY 1980; REV 03  ****            +
;    +                                                             +
;    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +

;-------------------------------------------------------------------
;-------------------------------------------------------------------

;                              **********

;       Program WRSYS.SR is called from WRITSYSTM.FR and returns to
;       WRITSYSTM.  Its sole function is to transmit data to the "system."
;       The data is always a command of the action file.  The call to
;       WRSYS is as follows:

;               CALL WRSYS (IN1ARRAY, I1DIMAR)

;       IN1ARRAY is one line from the action file that is to be sent to
;       the "system."  I1DIMAR is the size of the one-dimentional array
;       IN1ARRAY.  Both these parameters are passed from WRITSYSTM to
;       WRSYS.

;                             **********

;-------------------------------------------------------------------
;-------------------------------------------------------------------

        .TITL WRSYS                     ;Program name - Write to System

        .ENT WRSYS                      ;Enables outside entry into this
                                        ;program

        .TXTM 1                         ;Packs ASCII strings left to right

        .EXTD .FSUB, .FREDI, .LD1       ;FORTRAN runtime library routines must
                                        ;be declared external displacements

        .EXTU                           ;Undefined variables are treated as
                                        ;External Displacement variables

        .EXTN .I, .ASUSP                ;.I provides some FORTRAN initialization
                                        ;.ASUSP is a task call for suspension
                                        ;and must be declared external normal

;   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

        .NREL                           ;Normal relocatable space starts

        FS.
```

```
;     ESTABLISH THE ARRAY PASSED IN AS DUMMY ARGUMENT
;     - - - - - - - - - - - - - - - - - - - - - - - -

WRSYS:  JSR @ .FARL

        JSR @ .FRED                     ;Redimension an array, passed as dummy
        ARYIN                           ;argument.  This is array specifier
        @ STORA+0                       ;This is array size - dummy argument
        STORA+2                         ;This is three word stack specifier

;     SELECT EACH ELEMENT OF THE ARRAY
;     - - - - - - - - - - - - - - - - -

        LDA 0, LOWSUB                   ;Load subscript one
        STA 0, TMP+5, 3                 ;Store on stack
        JMP START                       ;Start iterations

INCSUB: LDA 0, TMP+5, 3                 ;On repeated passes, get the
        INC 0, 0                        ;last subscript and increment it
        STA 0, TMP+5, 3                 ;Store new subscript

START:  LDA 1, @TMP+1, 3                ;Load maximum subscript
        SUBZ 0, 1, SNC                  ;If maximum exceeded,
        JMP RET                         ;return to calling program

        JSR @ .FSUB                     ;Otherwise, get the array element
        3                               ;Number of arguments for library call
        STORA+2                         ;Stack specifier for the array
        STORP+1                         ;Temporary location for FSUB result
        STORA+5                         ;FORTRAN address of subscript

;     CONVERT FROM FORTRAN TO ASSEMBLY LANGUAGE
;     - - - - - - - - - - - - - - - - - - - - -

        LDA 0, @TMPP+1, 3               ;Load selected array element
        LDA 2, MSKIT                    ;Delete second byte, strip parity from
        ANDS 2, 0                       ;first byte, and swap bytes

;     OUTPUT CHARACTER TO SYSTEM
;     - - - - - - - - - - - - - -

        SKPBZ TTO1                      ;If the output line (channel device $TTO
        JMP .-1                         ;is busy, try again
        DOAS 0, TTO1                    ;Otherwise, output the character
        JMP INCSUB                      ;Continue the next iteration

;     DEFINE VARIABLES
;     - - - - - - - - -

CR:     15                              ;Carriage return is octal 15
LOWSUB: 11                              ;9 decimal - subscript that starts
                                        ;action file text
MSKIT:  077400                          ;Mask for stripping parity, deleting
                                        ;second byte
TSKPRI: 0                               ;MONITOR's task priority
MINSUB: 1                               ;The lower subscript starts at one
```

```
;    SPECIFY THE ARRAY
;    - - - - - - - - - -

ARYIN:  3                               ;This is - 2*(number of subscripts)+1
        401                             ;This is - 400*1+1, length=1 and type =1
        MINSUB                          ;This is lower subscript size
        @STORA+1                        ;This is higher subscript size

;    OUTPUT CARRIAGE RETURN, SUSPEND MONITOR, AND RETURN
;    - - - - - - - - - - - - - - - - - - - - - - - - - - -

RET:    LDA 0, CR                       ;Load carriage return
        SKPBZ TI01                      ;Output it when $TT01 not busy
        JMP .-1
        DOAS 0, TT01

        LDA 0, TSKPRI                   ;Load MONITOR task priority
        .ASUSP                          ;Suspend MONITOR - task SYSIN will
                                        ;ready MONITOR

;    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

        JSR @ .FRET

;    DEFINE STACK PARAMETERS
;    - - - - - - - - - - - -

        FS.=7                           ;Frame size for all stack variables
        TMP=-167                        ;Middle of the user's stack
        STORA=200+TMP
        TMPP=TMP+5                      ;First word available for temporary
                                        ;storage
        STORP=STORA+5

        .END WRSYS

;-------------------------------------------------------------------------
;-------------------------------------------------------------------------

;    + + + + + + + + + + + + + + + + + + + + + + + + + + + +
;    +                                                     +
;    +                 END WRSYS.SR                        +
;    +                                                     +
;    + + + + + + + + + + + + + + + + + + + + + + + + + + + +

;-------------------------------------------------------------------------
;-------------------------------------------------------------------------
```

```
C-------------------------------------------------------------------------
C-------------------------------------------------------------------------
C
C    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
C    +                                                               +
C    +    WRITLOCAL.FR                                               +
C    +        ****  CREATED  8 AUGUST 1980;  REV 00  ****           +
C    +                                                               +
C    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
C
C-------------------------------------------------------------------------
C-------------------------------------------------------------------------
C
C                          **********
C
C       Program WRITLOCAL.FR is called by MONITOR.FR and returns to
C       MONITOR, whenever a command sequence from an action file is
C       ready to be sent to the "local" terminal.  WRITLOCAL simply
C       writes a string contained in array IN2ARRAY to the "local"
C       terminal.  The call to WRITLOCAL is as follows:
C
C               CALL WRITLOCAL (IACTFILE,MDIM1,$602)
C
C       IACTFILE is one line from the action file that is to be sent
C       to the "local" terminal.  MDIM1 is the size of the one-dimentional
C       array IACTFILE, and  $602 is the return line number in MONITOR
C       that will be next executed upon return from WRITLOCAL.  All
C       parameters are passed from MONITOR to WRITLOCAL.
C
C                          **********
C
C-------------------------------------------------------------------------
C-------------------------------------------------------------------------
```

```
C
C     **  WRITLOCAL receives an input array, the dimension of that    **
C         array,and an assigned dummy return variable.
C
          SUBROUTINE WRITLOCAL (IN2ARRAY,I2DIMAR,I2DUMRTN)
          DIMENSION IN2ARRAY(I2DIMAR)
C
C     **  The contents of the action file line are written to channel **
C         10, the "local" terminal device ($TTO).
C
          WRITE (10, 2001) (IN2ARRAY(J1), J1 = 3, I2DIMAR)
 2001     FORMAT (80A1)
C
C     **  Return to the statement number passed in I2DUMRTN.          **
C
          RETURN I2DUMRTN
          END
C
C-------------------------------------------------------------------------
C-------------------------------------------------------------------------
C
C     + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
C     +                                                                 +
C     +                  END WRITLOCAL.FR                               +
C     +                                                                 +
C     + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
C
C-------------------------------------------------------------------------
C-------------------------------------------------------------------------
```

```
C-------------------------------------------------------------------
C-------------------------------------------------------------------
C
C    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
C    +                                                             +
C    +    READLWRITS.FR                                            +
C    +        ****  CREATED  8 AUGUST 1980;  REV 00  ****         +
C    +                                                             +
C    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
C
C-------------------------------------------------------------------
C-------------------------------------------------------------------
C
C                          **********
C
C       Program READLWRITS.FR is called by MONITOR.FR and returns to
C       MONITOR, whenever a command sequence from an action file
C       indicates that something is to be read from the "local" terminal
C       and a carraiage return is to be sent to the "system."
C       READLWRITS provides a transition to and from an assembly
C       language subroutine that actually reads data from the "local"
C       terminal and writes a carriage return to the "system."
C       The call to READLWRITS is as follows:
C
C               CALL READLWRITS ($602)
C
C       The $602 is the return line number in MONITOR that will next
C       be executed upon return from READLWRITS.
C
C                          **********
C
C-------------------------------------------------------------------
C-------------------------------------------------------------------
C
C    **  READLWRITS receives an assigned dummy return variable.      **
C
       SUBROUTINE READLWRITS (I3DUMRTN)
C
C    **  RDAWR actually implements the read and write functions.     **
C
       CALL RDAWR
C
C    **  Return to the statement number passed in I3DUMRTN.          **
C
       RETURN I3DUMRTN
       END
C
C-------------------------------------------------------------------
C-------------------------------------------------------------------
C
C    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
C    +                                                             +
C    +                END READLWRITS.FR                            +
C    +                                                             +
C    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
C
C-------------------------------------------------------------------
C-------------------------------------------------------------------
```

```
;-----------------------------------------------------------------------
;-----------------------------------------------------------------------

;    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
;    +                                                               +
;    +    RDAWR.SR                                                   +
;    +        ****  CREATED  3 JULY 1980;  REV 03  ****             +
;    +                                                               +
;    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +

;-----------------------------------------------------------------------
;-----------------------------------------------------------------------

;                              **********

;       Program RDAWR.SR is called from READLWRITS.FR and returns to
;       READLWRITS.  Its sole purpose is to await any keyboard input
;       and then transmit a carriage return to the "system."  This
;       routine is only called during LOGON processing.  There are no
;       arguments or parameters that are passed via RDAWR.

;                              **********

;-----------------------------------------------------------------------
;-----------------------------------------------------------------------
```

```
        .TITL RDAWR                     ;Program name - Read And Write

        .ENT RDAWR                      ;Enables outside entry into this
                                        ;program

        .TXTM 1                         ;Packs ASCII strings left to right

        .EXTU                           ;Undefined variables are treated
                                        ;as External Displacement variables

        .EXTN .I, .ASUSP                ;.I provides some FORTRAN initialization
                                        ;.ASUSP is a task call for suspension
                                        ;and must be declared external normal

;   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

        .ZREL                           ;Zero relocatable space starts

.ER:    ERROR                           ;Access to ERROR may be gained via
                                        ;indirect addressing to this location

;   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

        .NREL                           ;Normal relocatable space starts

        FS.

;   WAIT FOR KEYBOARD INPUT
;   - - - - - - - - - - - - -

RDAWR:  JSR @ .FARL

RERD:   .SYSTM
        .GCHAR                          ;Get input character from keyboard
        JMP @ .ER
```

```
;     SEND CARRIAGE RETURN TO $TTO1
;     - - - - - - - - - - - - - - - - - -

         LDA 0, CR                        ;Load a carriage return
         .SYST1
         .PCHAR                           ;Output carriage return to $TTO
         JMP @ .ER

         SKPBZ TTO1                       ;Is $TTO1 busy?
         JMP .-1                          ;Yes - try again
         DOAS 0, TTO1                     ;No - output carriage return to $TTO1

;     SUSPEND THE TASK MONITOR TO ALLOW SYSIN TO EXECUTE
;     - - - - - - - - - - - - - - - - - - - - - - - - - - - -

         LDA 0, TSKPR1                    ;Load MONITOR's task priority (0)
         .ASUSP                           ;Suspend MONITOR until readied in
                                          ;task SYSIN

         JMP RTN                          ;Jump to return location when complete

;      DEFINE VARIABLES
;     - - - - - - - - - - - -

TSKPR1: 0                                 ;MONITOR's task priority is zero
CR:     15                                ;A carriage return is octal 15

;     ROUTINE TO RETURN TO THE CLI ABNORMALLY
;     - - - - - - - - - - - - - - - - - - - - -

ERROR:  .SYSTM
         .ERTN                            ;Abnormal return - error
         JMP @ .ER

;     - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
RTN:     JSR @ .FRET

         FS.=0
         TMP=-167

         .END RDAWR

;-------------------------------------------------------------------------------
;-------------------------------------------------------------------------------

;     + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
;     +                                                         +
;     +                    END RDAWR.SR                         +
;     +                                                         +
;     + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +

;-------------------------------------------------------------------------------
;-------------------------------------------------------------------------------
```

```
C-------------------------------------------------------------------
C-------------------------------------------------------------------
C
C    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
C    +                                                             +
C    +    SENDFILE.FR                                              +
C    +        ****  CREATED  8 AUGUST 1980;  REV 00  ****+
C    +                                                             +
C    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
C
C-------------------------------------------------------------------
C-------------------------------------------------------------------
C
C                          **********
C
C       Program SENDFILE.FR is called by MONITOR.FR and returns to
C       MONITOR, whenever a command sequence from an action file
C       requires a "local" disk file to be sent to the "system."
C       SENDFILE uses the RDOS file CLI.CM to "send" action file commands
C       to the "local" system.  Thus, the action file command is inserted
C       into CLI.CM and then a program swap takes place to execute that
C       command.  When swapping takes place, functions within tasks are
C       disabled (such as .IDEF).  Therefore, the task SYSIN is inactivated
C       (killed) and reactivated before and after the swap - program
C       EXCLI.SR.  Finally, program GETFILE.SR actually transmits the
C       data retrieved from the "local" disk and sends it to the
C       "system."  The call to SENDFILE is as follows:
C
C               CALL SENDFILE (IACTFILE,MDIM1,$602)
C
C       IACTFILE is one line from the action file that is to be
C       acted upon by the "local" system.  MDIM1 is the size of the
C       one-dimensional array IACTFILE, and $602 is the return line
C       number in MONITOR that will next be executed upon return from
C       SENDFILE.  All parameters are passed to SENDFILE from MONITOR.
C
C                          **********
C
C-------------------------------------------------------------------
C-------------------------------------------------------------------
```

```
C
C     **  SENDFILE receives an input array, the dimension of that      **
C         array, and an assigned dummy return variable.
C
          SUBROUTINE SENDFILE (IN4ARRAY,I4DIMAR,I4DUMRTN)

C     **  SYSIN.SR is the task program that monitors "system" input    **
C         and was activated by MONITOR.  As it is to be killed and then
C         reactivated, DG FORTRAN required that it be externally defined.
C
          EXTERNAL SYSIN
          DIMENSION IN4ARRAY(I4DIMAR)
C
C     **  This call creates file CLI.CM.  If it already exists, an     **
C         error of 12 is returned in KERR1.  If the call is okay, an
C         error of 1 is returned.  Any other error is printed out for
C         reference.  This insures that CLI.CM is available.
C
          CALL CFILW ("CLI.CM",2,KERR1)
          IF (KERR1 .NE. 1 .AND. KERR1 .NE. 12) TYPE  "KERR1 IS  ",KERR1
C
C     **  This call opens file CLI.CM on channel 25.  State the error **
C         condition if not opened properly.
C
          CALL OPEN (25,"CLI.CM",2,KERR2,82)
          IF (KERR2 .NE. 1) TYPE "KERR2 IS  ", KERR2
C
C     **  Insert command sequence from action file into CLI.CM.        **
C
          WRITE (25,4001) (IN4ARRAY(K1), K1 = 3, I4DIMAR)
 4001     FORMAT (1H , 80A1)
C
C     **  Close CLI.CM so it can be deleted in EXCLI.SR, after it is   **
C         no longer needed.
C
          CALL CLOSE (25,KERR3)
          IF (KERR3 .NE. 1) TYPE  "KERR3 IS  ", KERR3
```

```
C
C     **  Before executing the swap (EXCLI), inactivate SYSIN.          **
C         SYSIN is the only task with priority one (1).
C
          CALL AKILL (1)
C
C     **  EXCLI swaps to level two (2) to execute the instruction       **
C         just inserted into CLI.CM.  Level two (2) is the RDOS CLI.
C         Upon completion of the swapped program, control returns to the
C         next instruction in this program.
C
          CALL EXCLI
C
C     **  Reactivate SYSIN just as it was before the swap.              **
C
          CALL ITASK (SYSIN,10,1,KERR4,1)
          IF (KERR4 .NE. 1) TYPE  "KERR4 IS  ", KERR4
C
C     **  GETFILE.SR transmits the data on file QQVV to the "system." **
C
          CALL GETFILE
C
C     **  Return to the statement number passed in I4DUMRTN.           **
C
          RETURN I4DUMRTN
          END
C
C-------------------------------------------------------------------------
C-------------------------------------------------------------------------
C
C     + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
C     +                                                               +
C     +                  END SENDFILE.FR                              +
C     +                                                               +
C     + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
C
C-------------------------------------------------------------------------
C-------------------------------------------------------------------------
```

```
;-------------------------------------------------------------------
;-------------------------------------------------------------------

;    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
;    +                                                             +
;    +    EXCLI.SR                                                 +
;    +        ****  CREATED 22 JULY 1980;  REV 01  ****           +
;    +                                                             +
;    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +

;-------------------------------------------------------------------
;-------------------------------------------------------------------

;                          **********

;       Program EXCLI.SR is called from SENDFILE.FR and RECEVFILE.FR,
;       and returns to  SENDFILE and/or RECEVFILE, whichever program
;       called EXCLI last.  The purpose of EXCLI is to execute the
;       RDOS CLI on level two (2) by swapping the CLI in and swapping out
;       EXCLI.  A command from the action file has been inserted into
;       CLI.CM by either SENDFILE or RECEVFILE; this swap executes
;       these commands.  EXCLI also appends a POP command in the CLI.CM
;       file, in order to return to level one (1) - the program EXCLI -
;       upon completion of the command sequence inserted into CLI.CM.
;       There are no arguments or parameters that are passed via EXCLI.

;                          **********

;-------------------------------------------------------------------
;-------------------------------------------------------------------

        .TITL EXCLI                     ;Program name - Execute the CLI

        .ENT EXCLI                      ;Enables outside entry into this
                                        ;program

        .TXTM 1                         ;Packs ASCII strings left to right

        .EXTU                           ;Undefined variables are treated as
                                        ;External Displacement variables

        .EXTN .I                        ;Provides some FORTRAN initialization

;    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

        .NREL                           ;Normal relocatable space starts

        FS.
```

```
;     ADD THE POP COMMAND TO FILE CLI.CM
;     - - - - - - - - - - - - - - - - - -

EXCLI:  JSR @ .FARL

        SUB 1,1                         ;Load the default mask
        LDA 0, CLICM                    ;Load the bytepointer to file CLI.CM
        .SYSTM
        .APPEND 23                      ;Open CLI.CM for appending on channel 23
        JMP ER

        LDA 0, CMAND                    ;Load bytepointer to additional command
        LDA 1, BCOUNT                   ;Load number of bytes to be written

        .SYSTM
        .WRS 23                         ;Write the added command to CLI.CM
        JMP ER
        .SYSTM
        .CLOSE 23                       ;Now close file CLI.CM
        JMP ER

;     CALL SWAP TO EXECUTE THE CLI
;     - - - - - - - - - - - - - - - -

        LDA 0, CLISV                    ;Load bytepointer to file CLI.SV
        SUB 1, 1                        ;Load zero - indicates swap
        SUBZL 2, 2                      ;Send no message to swap program
        .SYSTM
        .EXEC                           ;Swap to CLI on level two (2)
        JMP ER
        JMP RT                          ;Jump to return location when complete

;     ROUTINE TO RETURN TO THE CLI ABNORMALLY
;     - - - - - - - - - - - - - - - - - - - -

ER:     .SYSTM
        .ERTN                           ;Abnormal return - error
        JMP ER

;     DEFINE BYTEPOINTERS, ETC.
;     - - - - - - - - - - - - -

CLISV:  .+1*2                           ;Bytepointer to file CLI.SV
        .TXT "CLI.SV"

CLICM:  .+1*2                           ;Bytepointer to file CLI.CM
        .TXT "CLI.CM"

CMAND:  .+1*2                           ;Bytepointer to POP command
        .TXT ";POP<15>"

BCOUNT: (BCOUNT-CMAND)*2                ;Number of bytes in POP command
```

```
;     DELETE FILE CLI.CM BEFORE RETURNING TO CALLING PROGRAM
;     - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

RT:       LDA 0, CLICM                    ;Load bytepointer to file CLI.CM
          .SYSTM
          .DELET                          ;Delete file CLI.CM
          JMP ER

;     - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

          JSR @ .FRET

          FS.=0
          TMP=-167

          .END EXCLI

;-------------------------------------------------------------------------
;-------------------------------------------------------------------------

;     + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
;     +                                                               +
;     +                    END EXCLI.SR                               +
;     +                                                               +
;     + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +

;-------------------------------------------------------------------------
;-------------------------------------------------------------------------
```

```
;-----------------------------------------------------------------------
;-----------------------------------------------------------------------

;    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
;    +                                                               +
;    +    GETFILE.SR                                                 +
;    +        ****  CREATED 26 JULY 1980; REV 02  ****              +
;    +                                                               +
;    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +

;-----------------------------------------------------------------------
;-----------------------------------------------------------------------

;                              **********

;       Program GETFILE.SR is called from SENDFILE.FR and returns to
;       SENDFILE.  It is called after EXCLI has placed a disk file of
;       the "local" system into file QQVV.  It gets this file and
;       outputs it to the "system" character by character.  It calls
;       upon a system call to determine the User File Discription
;       status, which provides the size of the file.  There are no
;       arguments or parameters that are passed via GETFILE.

;                              **********

;-----------------------------------------------------------------------
;-----------------------------------------------------------------------

        .TITL GETFILE                   ;Program name - Get File

        .ENT GETFILE                    ;Enables outside entry into this
                                        ;program

        .TXTM 1                         ;Packs ASCII strings left to right

        .EXTU                           ;Undefined varaibles are treated as
                                        ;External Displacement variables

        .EXTN .I                        ;Provides some FORTRAN initialization

;    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

        .NREL                           ;Normal relocatable space starts

        FS.
```

```
;     OPEN AND FIND STATUS OF FILE QQVV
;     - - - - - - - - - - - - - - - - - -

GETFILL:JSR @ .FARL

        LDA 0, NAMOFFL                  ;Load bytepointer to file QQVV
        SUB 1, 1                        ;Load default mask
        .SYSTM
        .OPEN 26                        ;Open file QQVV on channel 26
        JMP ER

        LDA 1, STRLOC                   ;Load pointer to beginning of UFD store
        .SYSTM
        .STAT                           ;Get the UFD status of QQVV
        JMP ER

        JMP CALCSZ                      ;Now calculate the size of QQVV

;     READ QQVV BY PORTIONS AND SEND OUT
;     - - - - - - - - - - - - - - - - - -

AGNRD:  LDA 3, SZOFFL                   ;Load the size of file QQVV
        LDA 1, BYTSZ                    ;Load the size of the CNTS store
        SUBZ# 1, 3, SNC                 ;Is current size of file greater
                                        ;than size of CNTS?
        JMP LSTRD                       ;No - read QQVV for the last time

        STA 1, WRSSZ                    ;Yes - read at least two times
        LDA 0, PTRCNTS                  ;Load bytepointer to buffer CNTS
        .SYSTM
        .RDS 26                         ;Read a portion of QQVV into CNTS
        JMP ER

        JSR SENDIT                      ;Send this portion out

        LDA 1, BYTSZ                    ;Load the size of the CNTS buffer
        LDA 3, SZOFFL                   ;and the current size of QQVV not read
        SUB 1, 3                        ;Find the difference between the
        STA 3, SZOFFL                   ;two and store in SZOFFL
        JMP AGNRD                       ;Return to read the next portion

LSTRD:  LDA 0, PTRCNTS                  ;Load the bytepointer to CNTS
        LDA 1, SZOFFL                   ;Load the current size of QQVV
        STA 1, WRSSZ                    ;Store this size
        .SYSTM
        .RDS 26                         ;Read in the last portion of QQVV
        JMP ER

        JSR SENDIT                      ;Send this portion also

        LDA 0, CR                       ;Load a carriage return

        SKPBZ TTO1                      ;If the output line is not busy,
        JMP .-1                         ;send a carriage return as the last
        DOAS 0, TTO1                    ;character
```

```
;    ONCE COMPLETE, CLOSE AND DELETE QQVV
;    - - - - - - - - - - - - - - - - - - - -

TRYCLOS:LDA 0, NAMOFFL                  ;Load bytepointer to QQVV

        .SYSTM
        .CLOSE 26                       ;Close file QQVV
        JMP CHKERR                      ;Check to see if there is an expected
                                        ;error
        .SYSTM
        .DELET                          ;If not, delete QQVV
        JMP ER

        JMP RETERN                      ;Then return to the calling program

CHKERR: LDA 1, .ERFIU                   ;Is QQVV still in use?
        SUB 2, 1, SNR                   ;Compare to error code ERFIU
        JMP TRYCLOS                     ;Yes - return to try for closing again
        JMP ER                          ;Otherwise, state the unexpected error

;    ROUTINE TO RETURN TO THE CLI ABNORMALLY
;    - - - - - - - - - - - - - - - - - - - -

ER:     .SYSTM
        .ERTN                           ;Abnormal return - error
        JMP ER

;    DEFINE VARIABLES, BYTEPOINTERS, AND BUFFERS
;    - - - - - - - - - - - - - - - - - - - - - - -

.ERFIU: ERFIU                           ;ERFIU = 60, file in use

NAMOFFL:.+1*2                           ;Bytepointer to file QQVV
        .TXT "DPOF:DIALOG:QQVV"

STRLOC: UFDSTR                          ;Pointer to UFD store
UFDSTR: .BLK 22                         ;18 decimal UFD word store

SZOFFL: 0                               ;Store for current size of file
BYTSZ:  122                             ;82 decimal bytes - CNTS buffer
PTRINIT:CNTS                            ;Pointer to the beginning of CNTS
TEXTPTR:CNTS                            ;Pointer to text entries for CNTS
PTRCNTS:CNTS*2                          ;Bytepointer to CNTS buffer
WRSSZ:  0                               ;Number of bytes to be written
```

```
;     CALCULATE THE SIZE OF QQVV
;     - - - - - - - - - - - - - -

CALCSZ: LDA 3, ZERO                     ;Load zero (0)
        LDA 1, UFDSTR+10                ;Load the number of the last
        SUB# 3, 1, SNR                  ;block in QQVV.  Is it zero?
        JMP ADONCE                      ;Yes - just calculate once the size

        STA 1, CNTER                    ;No - load the number of blocks in
        LDA 0, ZERO                     ;counter and load zero
        LDA 2, ONEBLK                   ;Load the size of one block

MULT:   ADD 2, 0                        ;Multiply the number of blocks
        DSZ CNTER                       ;by the number of bytes in a block
        JMP MULT                        ;Continue till all blocks are included

        LDA 1, UFDSTR+11                ;Load the number of bytes in the
        ADD 1, 0                        ;last block - add to the total
        STA 0, SZOFFL                   ;Store total in size of file
        JMP AGNRD                       ;Return to read portions of QQVV

ADONCE: LDA 2, UFDSTR+11                ;If just one block, store the
        STA 2, SZOFFL                   ;number of bytes in size of file
        JMP AGNRD                       ;Return to read the portions

;     DEFINE VARIABLES AND STORAGE LOCATIONS
;     - - - - - - - - - - - - - - - - - - -

ONEBLK: 1000                            ;One block is 512 decimal bytes
ZERO:   0
CNTER:  0                               ;Counter storage location
SAVE3:  0                               ;Storage location for accumulator
SAVE2:  0                               ;3, 2, 1, 0, and the carry bit
SAVE1:  0
SAVE0:  0
SAVEC:  0
CR:     15                              ;Carriage return is an octal 15
```

```
;     SEND FILE QQVV OUT CHARACTER BY CHARACTER
;     - - - - - - - - - - - - - - - - - - - - -

SENDIT: STA 3, SAVE3                    ;Upon entering routine, store
        STA 2, SAVE2                    ;accumulators and carry bit
        STA 1, SAVE1
        STA 0, SAVE0
        MOVL 0, 0
        STA 0, SAVEC

        LDA 1, WRSSZ                    ;Load number of bytes to write
        MOVZR 1, 1                      ;Divide the number by two, as two
        STA 1, WRSSZ                    ;bytes will be written per iteration

REPT:   LDA 0, @TEXTPTR                 ;Load pointer to text character
        LDA 1, MSK1                     ;Load mask to isolate left byte
        ANDS 1, 0                       ;Strip parity and swap

        SKPBZ TTO1                      ;If output line not busy, output
        JMP .-1                         ;this isolated character
        DOAS 0, TTO1

        LDA 0, @TEXTPTR                 ;Load same text
        LDA 1, MSK2                     ;Load mask to isolate right byte
        AND 1, 0                        ;Strip parity

        SKPBZ TTO1                      ;Output this character when output
        JMP .-1                         ;line not busy
        DOAS 0, TTO1

        LDA 3, TEXTPTR                  ;Load the pointer to the text
        INC 3, 3                        ;Increment the pointer
        STA 3, TEXTPTR
        DSZ WRSSZ                       ;Have the words all been sent?
        JMP REPT                        ;No - return and repeat for the next
                                        ;word
        LDA 1, PTRINIT                  ;Yes - load the pointer to beginning of
        STA 1, TEXTPTR                  ;text buffer - CNTS

        LDA 0, SAVEC                    ;Restore accumulators and carry
        MOVR 0, 0
        LDA 0, SAVE0
        LDA 1, SAVE1
        LDA 2, SAVE2
        LDA 3, SAVE3

        JMP 0, 3                        ;Return to next line from location
                                        ;in which subroutine called
```

```
;    DEFINE MASKS
;    - - - - - - - -

MSK1:   177400                          ;Mask to isolate left byte
MSK2:   000377                          ;Mask to isolate right byte

;    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

RETURN: JSR @ .FRET

;    DEFINE CONTENTS BUFFER
;    - - - - - - - - - - - - -

CNTS:   .BLK 122                        ;82 decimal bytes of storage

        FS.=0
        TMP=-167

        .END GETFILE

;-----------------------------------------------------------------------
;-----------------------------------------------------------------------

;    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
;    +                                                               +
;    +                END GETFILE.SR                                 +
;    +                                                               +
;    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +

;-----------------------------------------------------------------------
;-----------------------------------------------------------------------
```

```
C-------------------------------------------------------------------
C-------------------------------------------------------------------
C
C    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
C    +                                                             +
C    +    READYREAD.FR                                             +
C    +        ****  CREATED  9 AUGUST 1980;  REV 00  ****+
C    +                                                             +
C    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
C
C-------------------------------------------------------------------
C-------------------------------------------------------------------
C
C                              **********
C
C       Program READYREAD.FR is called by MONITOR.FR and returns to
C       MONITOR, whenever a disk file needs to be prepared (i.e.,
C       created) to accept data from the "system."  File VVQQ is created
C       to receive the data from the "system", which will transpire on the
C       next execution of task call SYSIN.SR.  This latter action is
C       triggered by a repeated call to open the file VVQQ, which only
C       succeeds after VVQQ has been created.  The call to READYREAD
C       is as follows:
C
C               CALL READYREAD ($602)
C
C       The $602 is the return line number in MONITOR that will next
C       be executed upon return from READYREAD.
C
C                             **********
C
C-------------------------------------------------------------------
C-------------------------------------------------------------------
```

```
C
C     **  READYREAD receives an assigned dummy return variable.       **
C
         SUBROUTINE READYREAD (I5DUMRTN)
C
C     **  This call creates file VVQQ, that will be accepting data    **
C         from the "system."  Data will actually be entered only when
C         VVQQ is opened by SYSIN.SR.
C
         CALL CFILW ("DPOF:DIALOG:VVQQ",2,IERR1)
         IF (IERR1 .NE. 1) TYPE  "IERR1 IS  ", IERR1
C
C     **  Return to the statement number passed in I5DUMRTN.          **
C
         RETURN I5DUMRTN
         END
C
C-------------------------------------------------------------------------
C-------------------------------------------------------------------------
C
C     + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
C     +                                                               +
C     +                  END READYREAD.FR                             +
C     +                                                               +
C     + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
C
C-------------------------------------------------------------------------
C-------------------------------------------------------------------------
```

```
C-------------------------------------------------------------------
C-------------------------------------------------------------------
C
C    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
C    +                                                           +
C    +    RECEVFILE.FR                                           +
C    +        ****  CREATED  9 AUGUST 1980;  REV 00  ****       +
C    +                                                           +
C    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
C
C-------------------------------------------------------------------
C-------------------------------------------------------------------
C
C                          **********
C
C       Program RECEVFILE.FR is called by MONITOR.FR and returns to
C       MONITOR, whenever a command sequence from an action file
C       requires a "local" disk file to be created to receive data
C       transmitted from the "system."  RECEVFILE uses the RDOS file
C       CLI.CM to "send" action file commands to the "local" system.
C       Thus, the action file command is inserted into CLI.CM and then
C       a program swap takes place to execute that command.  When
C       swapping takes place, functions within tasks are disabled
C       (such as .IDLF).  Therefore, the task SYSIN is inactivated
C       (killed) and reactivated before and after the swap - program
C       EXCLI.SR.  Lastly, RECEVFILE deletes the temporary disk file
C       VVQQ, as it is no longer needed.  The call to RECEVFILE is as
C       follows:
C
C               CALL RECEVFILE (IACTFILE,MDIM1,$602)
C
C       IACTFILE is one line from the action file that is to be
C       acted upon by the "local" system.  MDIM1 is the size of the
C       one-dimentional arrray IACTFILE, and $602 is the return line
C       number in MONITOR that will be next executed upon return from
C       RECEVFILE.  All parameters are passed from MONITOR to RECEVFILE.
C
C                          **********
C
C-------------------------------------------------------------------
C-------------------------------------------------------------------
```

```
C
C     **  RECEVFILE receives an input array, the dimension of that     **
C         array, and an assigned dummy return variable.
C
          SUBROUTINE RECEVFILE (IN6ARRAY,I6DIMAR,I6DUMRTN)
C
C     **  SYSIN.SR is the task program that monitors "system" input    **
C         and was activated by MONITOR.  As it is to be killed and then
C         reactivated, DG FORTRAN requires that it be externally defined.
C
          EXTERNAL SYSIN
          DIMENSION IN6ARRAY(I6DIMAR)
C
C     **  This call creates file CLI.CM.  If it already exists, an     **
C         error of 12 is returned in JERR1.  If the call is okay, an
C         error of 1 is returned.  Any other error is printed out for
C         reference.  This insures that CLI.CM is available.
C
          CALL CFILW ("CLI.CM",2,JERR1)
          IF (JERR1 .NE. 1 .AND. JERR1 .NE. 12) TYPE  "JERR1 IS  ", JERR1
C
C     **  This call opens file CLI.CM on channel 25.  State the error **
C         condition if not opened properly.
C
          CALL OPEN (25,"CLI.CM",2,JERR2,82)
          IF (JERR2 .NE. 1) TYPE  "JERR2 IS  ", JERR2
C
C     **  Insert command sequence from action file into CLI.CM.        **
C
          WRITE (25,6001) (IN6ARRAY(L1), L1 = 3, I6DIMAR)
 6001     FORMAT (1H , 80A1)
C
C     **  Close CLI.CM so it can be deleted in EXCLI.SR, after it is   **
C         no longer needed.
C
          CALL CLOSE (25,JERR3)
          IF (JERR3 .NE. 1) TYPE  "JERR3 IS  ", JERR3
C
```

```
C    **  Before executing the swap (EXCLI), inactivate SYSIN.          **
C         SYSIN is the only task with priority one (1).
C
         CALL AKILL(1)
C
C     **  EXCLI swaps to level two (2) to execute the instruction      **
C         just inserted into CLI.CM.  Level two (2) is the RDOS CLI.
C         Upon completion of the swapped program, control returns to the
C         next instruction in this program.
C
         CALL EXCLI
C
C     **  Reactivate SYSIN just as it was before the swap.             **
C
         CALL ITASK (SYSIN,10,1,JERR4,1)
         IF (JERR4 .NE. 1) TYPE  "JERR4 IS  ", JERR4
C
C     **  Upon return from the swap, delete file VVQQ, as it is no     **
C         longer required.
C
         CALL DFILW ("DPOF:DIALOG:VVQQ",JERR5)
         IF (JERR5 .NE. 1) TYPE  "JERR5 IS  ", JERR5
C
C     **  Return to the statement number passed in I6DUMRTN.           **
C
         RETURN I6DUMRTN
         END
C
C-------------------------------------------------------------------------
C-------------------------------------------------------------------------
C
C     + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
C     +                                                         +
C     +                  END RECEVFILE.FR                       +
C     +                                                         +
C     + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
C
C-------------------------------------------------------------------------
C-------------------------------------------------------------------------
```

```
;-----------------------------------------------------------------------
;-----------------------------------------------------------------------

;    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
;    +                                                               +
;    +    TOTERM.SR                                                  +
;    +        ****  CREATED 27 JUNE 1980;  REV 02 ****              +
;    +                                                               +
;    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +

;-----------------------------------------------------------------------
;-----------------------------------------------------------------------

;                              **********

;       Program TOTERM.SR is called from MONITOR.FR and always calls
;       TERMOP.SR in turn.  This program transitions the user from the
;       MONITOR Command Language to the transparent terminal only mode.
;       The transition is effected whenever the user enters "^T" anywhere
;       in the command line after MONITOR provides its prompt.  The
;       program, in addition to jumping to TERMOP.SR, also removes the
;       "system" device codes that were identified (via .IDEF) in SYSIN.SR.
;       This is necessary, since TERMOP identifies its own device codes.
;       Both TERMOP and SYSIN use the same device codes -- $TTO1 and $TTI1.
;       There are no arguments or parameters that are passed via TOTERM.

;                              **********

;-----------------------------------------------------------------------
;-----------------------------------------------------------------------

        .TITL TOTERM                    ;Program name - To Terminal Operation

        .ENT TOTERM                     ;Enables o[illegible]le entry into this
                                        ;program

        .EXTD TERMOP                    ;Dec[illegible]s program name (address)
                                        ;TERMOP.SR as external displacement
                                        ;TERMOP can now be accessed by TOTERM

        .TXTM 1                         ;Packs ASCII strings left to right

        .EXTU                           ;Undefined variables are treated
                                        ;as External Displacement variables

        .EXTN .I                        ;Provides some FORTRAN initialization

;    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
        .ZREL                           ;Zero relocatable space starts

.ER:    ERR                             ;Access to ERR and TERMOP may be
.TOTERM:TERMOP                          ;gained via indirect addressing to
                                        ;these locations

;   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

        .NREL                           ;Normal relocatable space starts

        FS.

;   FIRST REMOVE IDENTIFIED DEVICE CODES
;   - - - - - - - - - - - - - - - - - -

TOTERM:JSR @ .FARL

        LDA 0, D1CODE                   ;Load device code for $TT11
        .SYSTM
        .IRMV                           ;Remove it from the "system"
        JMP CHKER                       ;If an error, check to see if
                                        ;device has already been removed

        LDA 0, D2CODE                   ;Load device code $TT01
        .SYSTM
        .IRMV                           ;Remove it from the "system"
        JMP CHKER                       ;If an error, check to see if
                                        ;device has already been removed

;   JUMP TO TERMOP.SR
;   - - - - - - - - -

        JMP @ .TOTERM

;   ROUTINE TO CHECK FOR EXPECTED ERROR
;   - - - - - - - - - - - - - - - - - -

CHKER:  LDA 1, .ERDNM                   ;Has device been removed?
        SUB 2, 1, SNR
        JMP @ .TOTERM                   ;Yes - jump to TERMOP
        JMP @ .ER                       ;No - state the abnormal error

;   ROUTINE TO RETURN TO THE CLI ABNORMALLY
;   - - - - - - - - - - - - - - - - - - - -

ERR:    .SYSTM
        .ERTN                           ;Abnormal return - error
        JMP @ .ER
```

```
;     VARIABLES DEFINED
;     - - - - - - - - - -

.ERDNM: ERDNM                           ;ERDNM = 36, device not in system error

D1CODE: TTI1                            ;$TTO1 is the first device code
D2CODE: TTO1                            ;$TTO1 is the second device code

;     - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

         JSR @ .FRET

         FS.=0
         TMP=-167

         .END TOTERM

;-------------------------------------------------------------------------
;-------------------------------------------------------------------------

;     + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
;     +                                                               +
;     +                    END TOTERM.SR                              +
;     +                                                               +
;     + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +

;-------------------------------------------------------------------------
;-------------------------------------------------------------------------
```

```
;-------------------------------------------------------------------------
;-------------------------------------------------------------------------

;    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
;    +                                                                   +
;    +    TERMOP.SR                                                      +
;    +        ****  CREATED 30 JUNE 1980;  REV 02  ****                 +
;    +                                                                   +
;    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +

;-------------------------------------------------------------------------
;-------------------------------------------------------------------------

;                           **********

;       Program TERMOP.SR is called by TOTERM.SR, which is called from
;       MONITOR.FR.  TERMOP is the terminal only mode of operation for
;       intercommunication with the "system."  From the "local"
;       terminal, the mode of operation is as a transparent terminal
;       connected to the "system."  TERMOP serves as a device driver
;       that enables intercommunication between the "local"
;       system (devices $TTO and $TTI) and a "system" connected to it
;       via devices (devices $TTO1 and $TTI1).

;                           **********

;-------------------------------------------------------------------------
;-------------------------------------------------------------------------

        .TITL TERMOP                    ;Program name - Terminal Operation

        .EXTN .TASK                     ;Permits program to externally
                                        ;access task call TASK

        .ENT TERMOP                     ;Enables outside entry into this
                                        ;program

        .TXTM 1                         ;Packs ASCII strings left to right

        .EXTU                           ;Undefined varaibles are treated as
                                        ;External Displacement variables

        .EXTN .I                        ;Provides some FORTRAN initialization

;   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
        .ZRFL                           ;Zero relocatable space starts

.ER:    ERROR                           ;Access to ERROR and NOTE1
.NOT1:  NOTE1                           ;may be gained via indirect
.NOT2:  NOTE2                           ;addressing to these locations

;   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

        .NRFL                           ;Normal relocatable space starts


;   DEVICE CONTROL TABLE (DCT) LAYOUT
;   - - - - - - - - - - - - - - - - -

A1DCT:  TTI1AD                          ;Address of TTI1 DCT
A2DCT:  TTO1AD                          ;Address of TTO1 DCT

TTI1AD: STA1SA                          ;Interrupt state save area - TTI1
        -1                              ;Mask word for no interrupts
        TTI1RA                          ;TTI1 interrupt service routine
                                        ;address

TTO1AD: STA2SA                          ;Interrupt state save area - TTO1
        -1                              ;Mask word
        TTO1RA                          ;TTO1 interrupt service routine
                                        ;address

;   ADDITIONAL VARIABLES FOR $TTI1/$TTO1 HANDLERS
;   - - - - - - - - - - - - - - - - - - - - - - -

D1CODE: TTI1                            ;Device code - TTI1
D2CODE: TTO1                            ;Device code - TTO1

STA1SA: .BLK 10                         ;Eight word state save area - TTI1
STA2SA: .BLK 10                         ;Same - TTO1
```

```
;   START OF TERMINAL OPERATION PROGRAM
;   - - - - - - - - - - - - - - - - - - -

        FS.

TERMOP: JSR @ .FARL

        SUB 1, 1                        ;Load default mask
        LDA 0, @.NOT1                   ;Load bytepointer to NOTE1
        .SYSTM
        .WRL 21                         ;Write NOTE1 to TTO
        JMP @ .ER

;   DEFINE DEVICES $TTI1/$TTO1
;   - - - - - - - - - - - - - - - -

DEF1DEV:LDA 0, D1CODE                   ;Define TTI1 via the .IDEF
        LDA 1, A1DCT                    ;call, using device code
        .SYSTM                          ;TTI1 and its DCT address
        .IDEF
        JMP @ .ER

DEF2DEV:LDA 0, D2CODE                   ;Define TTO1 via the .IDEF
        LDA 1, A2DCT                    ;call, using device code TTO1
        .SYSTM                          ;and its DCT address
        .IDEF
        JMP @ .ER

;   DEFINE LINERD AS AN ASYNCHRONOUS TASK
;   - - - - - - - - - - - - - - - - - - -

DEFTASK:LDA 0, IDANDPR                  ;Load the ID and priority of
        LDA 1, TSKPTR                   ;task LINERD; load task pointer
        .TASK                           ;to LINERD
        JMP @ .ER

;       DEVICES $TTI1/$TTO1 BEFORE STARTING
;   -   - - - - - - - - - - - - - - - - - -

        NIOC TTI1
        NIOC TTO1
```

```
;   FIRST TASK -- READ AND WRITE CHARACTERS FROM/TO TTI AND TTO
;   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
;   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

TERMRD: .SYSTM                          ;Get character from TTI
        .GCHAR
        JMP @ .ER
        .SYSTM                          ;Put character to TTO
        .PCHAR
        JMP @ .ER

        LDA 1, UPAROW                   ;Compare character to "^" ;
        SUB# 0, 1, SNR                  ;If a match, return to
        JMP LVTRMOP                     ;the CLI.  Otherwise,

        SKPBZ TTO1                      ;output the character to TTO1
        JMP .-1
        DOAS 0, TTO1

        JMP TERMRD                      ;Return to terminal read/write

LVTRMOP:LDA 0, @ .NOT2
        SUB 1, 1
        .SYSTM
        .WRL 21
        JMP @ .ER

;    ROUTINE TO RETURN TO THE CLI NORMALLY
;    - - - - - - - - - - - - - - - - - - -

        .SYSTM
        .RTN                            ;Normal return -- no error
        JMP @ .ER


;   DEFINE ADDITIONAL VARIABLES USED ABOVE
;   - - - - - - - - - - - - - - - - - - -

IDANDPR:10B7+10                         ;LINERD ID is ten (arbitrary);
                                        ;priority is also ten (arbitrary)
TSKPTR: LINERD                          ;Second task is LINERD
UPAROW: .TXT "<0>^"                     ;PCHAR and GCHAR zero out the left
```

```
;   SECOND TASK -- READ AND WRITE FROM/TO $TTI1 AND $TTO1
;   - - - - - - - - - - - - - - - - - - - - - - - - - - -
;   - - - - - - - - - - - - - - - - - - - - - - - - - - -

        ;This routine operates on a first-in/first-out buffer concept.
        ;A single buffer is defined (BUFF) that is 133 characters
        ;long.  (NOTE:  The buffer is 133 words long, but each word
        ;contains just one character due to the way the reads and
        ;writes pack ASCII strings.)  Two pointers are defined to
        ;keep track of the latest character entered (INPTR)  and the
        ;latest character exited (OUTPTR).  A third  pointer (CHKPTR)
        ;always remains at the beginning of the buffer and is used for
        ;initialization, when required.  A fourth pointer (MAXPTR) is used to
        ;insure the buffer length is not exceeded when entering and
        ;exiting characters.  Pictorially, this looks as follows:


;                                BUFFER "BUFF"
;                                - - - - - - -

;  OUTPTR                                                    MAXPTR
;    \/                                                       \/
; ++++++++++++++++++++++++++++      / ++++++++++++++++++++++ + + + + +
; !        !         !            /      !        !        !        !
; !        !         !           /       !        !        !        !
; ! BUFF   ! BUFF+1  ! BUFF+2    /        !BUFF+131!BUFF+132!BUFF+133!
; ++++++++++++++++++++++++++    /   ++++++++++++++++++++++++ + + + + +
;      /\                      /                    /\
;     CHKPTR                                       INPTR


;   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

LINERD: LDA 1, INPTR                    ;Compare in and out pointers
        LDA 2, OUTPTR                   ;If they are the same, there are
        SUB# 2, 1, SNR                  ;no further characters to print
        JMP LINERD                      ;Return to the line reader

        LDA 0, @OUTPTR                  ;Otherwise, output to TTO the
        .SYSTM                          ;contents of BUFF pointed to by
        .PCHAR                          ;OUTPTR
        JMP @ .ER

        INC 2, 2                        ;Compare OUTPTR+1 and MAXPTR
        LDA 3, MAXPTR                   ;If they are the same, jump
        SUB# 2, 3, SNR                  ;to routine to re-initialize
        JMP INIT1                       ;OUTPTR with the CHKPTR. Otherwise,
        STA 2, OUTPTR                   ;store new value of OUTPTR
        JMP LINERD                      ;and return to the line reader

INIT1:  LDA 2, CHKPTR                   ;Re-initialize OUTPTR by storing
        JMP INIT1-2                     ;value of CHKPTR in OUTPTR
```

```
;   ADDITIONAL POINTER AND MASK VARIABLES DEFINED
;   - - - - - - - - - - - - - - - - - - - - - - -

NOTE1:  .+1*2                           ;Bytepointer to NOTE1 message
        .TXT "You have entered into the terminal only mode.  Proceed!<15>"

NOTE2:  .+1*2
        .TXT "You have returned to the local CLI mode!<15>"

OUTPTR: BUFF                            ;Initialize pointers to
INPTR:  BUFF                            ;beginning of buffer BUFF
CHKPTR: BUFF
PMSK:   177                             ;Mask to strip parity bit

;   $TTI1 INTERRUPT SERVICE ROUTINE
;   - - - - - - - - - - - - - - - - -
;   - - - - - - - - - - - - - - - - -

TTI1RA: STA 3, USP                      ;Save the previous processor
        STA 2, SAVE2                    ;state by saving all accumulators
        STA 1, SAVE1                    ;and the carry bit.  USP is the
        STA 0, SAVE0                    ;User Stack Pointer, location
        MOVL 0, 0                       ;016 (octal)
        STA 0, SAVEC

        DIAC 0, TTI1                    ;Input the character from TTI1 to
        LDA 3, PMSK                     ;accumulator zero, and strip the
        AND 3, 0                        ;the parity bit in the right byte
        STA 0, @INPTR                   ;Put character in next empty
                                        ;buffer location
        LDA 1, INPTR                    ;Is the INPTR at the end of BUFF?
        INC 1, 1                        ;Increment INPTR to compare with
        LDA 2, MAXPTR                   ;MAXPTR.  If they are the same,
        SUB# 1, 2, SNR                  ;jump to check the location of
        JMP CHECK                       ;OUTPTR.  Otherwise,
        STA 1, INPTR                    ;store the new value of INPTR

RETURN: LDA 0, SAVEC                    ;Restore the state of the processor
        MOVR 0, 0                       ;before returning to the next
        LDA 0, SAVE0                    ;instruction in the program
        LDA 1, SAVE1                    ;interrupted by input from TTI1
        LDA 2, SAVE2                    ;(This could be either of the
        LDA 3, USP                      ;two tasks of TERMOP)
        JMP 0, 3

CHECK:  LDA 3, OUTPTR                   ;If outptr is still at the
        LDA 2, CHKPTR                   ;beginning of the buffer BUFF,
        SUB# 2, 3, SNR                  ;there is a buffer overflow
        JMP PROB1                       ;potential requiring the processor
                                        ;to halt!  Otherwise,

        STA 2, INPTR                    ;store the new value of INPTR
        JMP RETURN                      ;and then return

PROB1:  HALT                            ;Not a recoverable errror - STOP!
```

```
;   $TT01 INTERRUPT SERVICE ROUTINE
;   - - - - - - - - - - - - - - - - -
;   - - - - - - - - - - - - - - - - -

TT01RA: NIOC TT01                       ;Idle the device TT01 and
        JMP 0, 3                        ;return.  No need to save the
                                        ;the processor state, as it is
                                        ;not changed

;   ROUTINE TO RETURN TO THE CLI ABNORMALLY
;   - - - - - - - - - - - - - - - - - - - - -

ERROR:  .SYSTM
        .ERTN                           ;Abnormal return - error
        JMP @ .ER

;   DEFINE NECESSARY STORAGE AREAS TO BE USED
;   - - - - - - - - - - - - - - - - - - - - -

SAVE0:  0                               ;Processor save state
SAVE1:  0                               ;locations
SAVE2:  0
SAVEC:  0
MAXPTR: BUFF+133.                       ;Pointer to the end of BUFF
BUFF:   .BLK 133.                       ;BUFF is 133 words long (decimal)

;   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

        JSR @ .FRET

        FS.=0
        TMP=-167

        .END TERMOP

;-------------------------------------------------------------------------
;-------------------------------------------------------------------------

;    + + + + + + + + + + + + + + + + + + + + + + + + + + + +
;    +                                                     +
;    +                  END TERMOP.SR                      +
;    +                                                     +
;    + + + + + + + + + + + + + + + + + + + + + + + + + + + +

;-------------------------------------------------------------------------
;-------------------------------------------------------------------------
```

```
;-------------------------------------------------------------------
;-------------------------------------------------------------------

;    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
;    +                                                               +
;    +    SYSIN.SR                                                   +
;    +        ****  CREATED  4 AUGUST 1980;  REV 04  ****           +
;    +                                                               +
;    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +

;-------------------------------------------------------------------
;-------------------------------------------------------------------

;                          **********

;       Program SYSIN.SR is an asynchronous task that is activated via
;       task calls and inactivated via task calls.  Initially, SYSIN is
;       activated by MONITOR.FR.  It is subsequently inactivated and
;       reactivated by SENDFILE.FR and RECEIVFILE.FR.  MONITOR also
;       inactivates SYSIN just before shifting to the terminal only
;       mode of operation.  SYSIN does several things simultaneouly.
;       First, it defines device codes $TTO1 and $TTI1 that are not
;       system generated.  Second, SYSIN reads the input line from the
;       "system" - $TTI1 - after the device defined interrupt routines
;       store the input in a buffer.  Third, SYSIN readies task MONITOR
;       whenever it is suspended.  And fourth, SYSIN writes to file
;       VVQQ whenever it is created by program RECEIVFILE.  SYSIN is of
;       priority 1, which is lower in precedence than MONITOR, which is
;       of priority 0.  SYSIN also has the identity number 10.  As
;       SYSIN is not a subroutine, there are no arguments or parameters
;       that are passed via SYSIN.

;                          **********

;-------------------------------------------------------------------
;-------------------------------------------------------------------

        .TITL SYSIN                     ;Program name - System Input

        .ENT SYSIN                      ;Enables outside entry into this
                                        ;program

        .EXTD .BUF1, .BUF2, .FB1, .FB2  ;These storage locations were created
                                        ;by CNVRT.SR and must be accessed by
                                        ;SYSIN.  They are external displacements

        .EXTN .ARDY                     ;.ARDY readies suspended tasks and
                                        ;must be declared external normal

        .TXTM 1                         ;Packs ASCII strings left to right

        .EXTU                           ;Undefined variables are treated as
                                        ;External Displacement variables

;    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
        .ZREL                           ;Zero relocatable space starts

.ER:    ERR                             ;ERR may be addressed indirectly
                                        ;via this location

;   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

        .NREL                           ;Normal relocatable space starts

;   DEVICE CONTROL TABLE LAYOUT
;   - - - - - - - - - - - - - - -

D1DCT:  I1AD                            ;Address of $TTI1 DCT
D2DCT:  O1AD                            ;Address of $TTO1 DCT

I1AD:   SSA1                            ;Interrupt state save area - $TTI1
        -1                              ;Mask word for no interrupts
        I1RA                            ;$TTI1 interrupt service routine address

O1AD:   SSA2                            ;Interrupt state save area - $TTO1
        -1                              ;Mask word
        O1RA                            ;$TTO1 interrupt service routine address

;   ADDITIONAL VARAIBLES FOR $TTI1/$TTO1 HANDLERS
;   - - - - - - - - - - - - - - - - - - - - - - -

CODE1:  TTI1                            ;Device code - $TTI1
CODE2:  TTO1                            ;Device code - $TTO1

SSA1:   .BLK 10                         ;8 decimal word state save areas
SSA2:   .BLK 10

;   DEFINE BYTEPOINTER AND NULL WORD
;   - - - - - - - - - - - - - - - - - -

TMPFL:  .+1*2                           ;Bytepointer to file VVQQ
        .TXT "DP0F:DIALOG:VVQQ"

NULL:   0                               ;A null word

;   DEFINE THE DEVICES
;   - - - - - - - - - -

SYSIN:  SUB 1, 1                        ;This is no operation - holds a place

DEV1:   LDA 0, CODE1                    ;Define $TTI1 via the .IDEF call,
        LDA 1, D1DCT                    ;using device code $TTI1 and its
        .SYSTM                          ;DCT address
        .IDEF
        JMP @ .ER

DEV2:   LDA 0, CODE2                    ;Define $TTO1 via the .IDEF call,
        LDA 1, D2DCT                    ;using device code $TTO1 and its
        .SYSTM                          ;DCT address
        .IDEF
        JMP @ .ER
```

```
;     DEFINE AN ASYNCHRONOUS TASK TO READ LINE INPUT
;     - - - - - - - - - - - - - - - - - - - - - - -

RDLINE: LDA 1, INPTR                    ;Compare in and out pointers.  If
        LDA 2, OUTPTR                   ;they are the same, there are no further
        SUB# 2, 1, SNR                  ;characters from input
        JMP CHKDLAY                     ;Check a delay time-out before
                                        ;readying task MONITOR

        LDA 1, DLAYCNT                  ;Otherwise, insure time-out parameters
        STA 1, CNTER                    ;are reset and initialized

;     EXAMINE INPUT
;     - - - - - - -

        LDA 0, @OUTPTR                  ;Load the contents of outptr

        INC 2, 2                        ;Increment the outpointer itself
        LDA 3, MAXPTR                   ;Load the maxpointer and compare to
        SUB# 2, 3, SNR                  ;outpointer.  If they're equal,
        JMP RSET1                       ;reset pointer to beginning of buffer

SET:    STA 2, OUTPTR                   ;Otherwise, give outpointer a new value
        LDA 3, NULL                     ;Load a null word
        SUB# 3, 0, SNR                  ;Is the character a null?(Input)
        JMP RDLINE                      ;Yes - throw it away and get next one

        STA 0, @MATPTR                  ;No - store character in Match Buffer
        LDA 3, MATPTR                   ;Save this location in Full Match Buffer
        STA 3, FMATBUF
        LDA 1, LF                       ;Is character a line feed?
        SUB# 0, 1, SNR
        JMP COMPAR                      ;Yes - compare all of Match Buffer

        LDA 1, CR                       ;No - Is character a carriage return?
        SUB# 0, 1, SNR
        JMP COMPAR                      ;Yes - compare all of Match Buffer

        INC 3, 3                        ;No - increment the match buffer pointer
        STA 3, MATPTR                   ;Store the new value
        JMP RDLINE                      ;Return to get next input character

RSET1:  LDA 2, CHKPTR                   ;If outpointer is at the end of
        JMP SET                         ;its buffer, reset to the beginning
```

```
;     AFTER A DELAY TIME-OUT, READY MONITOR
;     - - - - - - - - - - - - - - - - - - -

REDYTSK:LDA 1, MATPTR                   ;Insure nothing is in the match buffer
        LDA 2, MATSTRT                  ;If something is there, compare the
        SUB# 2, 1, SZR                  ;entire buffer
        JMP COMPAR

        .SYSTM                          ;If not, then
        .CLOSE 27                       ;close file VVQQ
        JMP .+1                         ;If already closed, just continue

        LDA 0, TSKPRI                   ;Load the priority of MONITOR
        .ARDY                           ;Ready MONITOR
        JMP RDLINE                      ;Return to get next character

;     DELAY BEFORE READYING MONITOR
;     - - - - - - - - - - - - - - - - -

CHKDLAY:DSZ CNTER                       ;Decrement the counter - if not zero
        JMP DLAY                        ;delay a while longer
        LDA 1, DLAYCNT                  ;Otherwise, reset delay values and
        STA 1, CNTER
        JMP REDYTSK                     ;allow MONITOR to take control

DLAY:   LDA 1, PULSC                    ;Load the number of pulse counts
        .SYSTM
        .DELAY                          ;Delay processing for the time allotted
        JMP @ .ER

        JMP RDLINE                      ;Return to check the input again

;     DEFINE VARIABLES AND BUFFERS
;     - - - - - - - - - - - - - - - - -

PULSC:  3                               ;3 decimal counts -- 3/10 sec
CNTER:  32                              ;Initial count is 26 decimal
DLAYCNT:32                              ;This creates a 7-1/2 sec delay
TSKPRI: 0                               ;MONITOR's task priority is 0

MATPTR: MATBUFR                         ;A pointer to start of Match Buffer
MATSTRT:MATBUFR                         ;Same
FMATBUF:0                               ;Location of a full Match Buffer

RSPSZ:  0                               ;Temporary location to store size of
                                        ;responses gathered from CNVRT.SR
LF:     12                              ;Line feed is an octal 12
CR:     15                              ;Carriage return is an octal 15
PMSK:   177                             ;Mask to strip parity bit
```

```
;     LOOK AT THE MATCH BUFFER AND COMPARE TO EXPECTED RESPONSES
;     - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

COMPAR: LDA 0, MATSTRT                  ;Load start of Match Buffer
        LDA 1, .FB1                     ;Load size of first response expected
        LDA 2, .BUF1                    ;Load starting location of first
        STA 1, RSPSZ                    ;response buffer.  Store size

CPR1AG: STA 0, MATPTR                   ;After setting pointer,
        LDA 0, @MATPTR                  ;get the character input
        LDA 3, 0, 2                     ;Get first character of response buffer
        SUB# 0, 3, SNR                  ;Are they the same?
        JMP CMPR1                       ;Yes - reset values to check next input

        LDA 0, MATSTRT                  ;No - start at the buffer beginning
        LDA 1, .FB2                     ;Load size of second expected response
        LDA 2, .BUF2                    ;Load start of second response buffer
        STA 1, RSPSZ                    ;Store size

CPR2AG: STA 0, MATPTR                   ;Look at characters just as before
        LDA 0, @MATPTR
        LDA 3, 0, 2
        SUB# 0, 3, SNR
        JMP CMPR2

;     OPEN VVQQ IF IT HAS BEEN CREATED
;     - - - - - - - - - - - - - - - - -

        LDA 0, TMPFL                    ;Load bytepointer to file VVQQ
        SUB 1, 1                        ;Load the default mask
        .SYSTM
        .OPEN 27                        ;Open VVQQ on channel 27
        JMP ERCHK                       ;Check for expected error condition
        JMP WRTOFL-1                    ;If no error, then begin output of
                                        ;file VVQQ to "system"
```

```
;     OUTPUT UNEXPECTED RESPONSES TO TERMINAL SCREEN
;     - - - - - - - - - - - - - - - - - - - - - - - -

        LDA 1, MATSTRT                  ;If VVQQ does not exist, output
                                        ;Match Buffer contents to terminal

OUTPUT: STA 1, MATPTR                   ;Start at beginning of Match Buffer
        LDA 0, CHKPTR
        .SYSTM
        .PCHAR                          ;Put a character to terminal - $TTO
        JMP @ .ER

        LDA 2, FINBUF                   ;Load location of last buffer character
        SUB# 1, 2, SKR                  ;Is output complete?
        JMP TORDLIN                     ;Yes - send carriage return and read
                                        ;the next line for input

        INC 1, 1                        ;No - increment buffer pointer and
        JMP OUTPUT                      ;repeat

;     DEFINE BUFFER POINTERS TO BUFFERS
;     - - - - - - - - - - - - - - - - -

OUTPTR: BUFR                            ;Pointer to next output character
CHKPTR: BUFR                            ;Pointer to buffer start
INPTR:  BUFR                            ;Pointer to next input buffer location
MAXPTR: BUFR+133                        ;Pointer to the end of input buffer

;     OUTPUT CARRIAGE RETURN AS LAST CHARACTER OF A WRITE
;     - - - - - - - - - - - - - - - - - - - - - - - - - -

TORDLIN:LDA 0, CR                       ;Load carraige return
        .SYSTM
        .PCHAR                          ;Put carraige return to terminal
        JMP @ .ER
        LDA 1, MATSTRT                  ;Reset pointers to beginning of Match
        STA 1, MATPTR                   ;Buffer and return to read the
        JMP RDLINE                      ;next line
```

```
;     CONTINUE TO COMPARE EXPECTED RESPONSES WITH INPUT
;     - - - - - - - - - - - - - - - - - - - - - - - - -

CMPR1:  LDA 0, MATPTR                   ;Look at next character in
        INC 0, 0                        ;response buffer and match buffer
        INC 2, 2
        DSZ RSPSZ                       ;Has the size of the expected response
        JMP CPR1AG                      ;been exceeded?  No - continue comparing
        LDA 1, MATSTRT                  ;Yes - reset buffer pointers
        STA 1, MATPTR                   ;and return to read the next line
        JMP RDLINE

CMPR2:  LDA 0, MATPTR                   ;Do the same for second response
        INC 0, 0
        INC 2, 2
        DSZ RSPSZ
        JMP CPR2AG
        LDA 1, MATSTRT
        STA 1, MATPTR
        JMP RDLINE

;     CHECK FOR AN EXPECTED ERROR CONDITION
;     - - - - - - - - - - - - - - - - - - -

ERCHK:  LDA 1, .ERDLE                   ;Load expected error code - ERDLE
        SUB# 1, 2, SNR                  ;Does file VVQQ exist?
        JMP OUTPUT-1                    ;No - jump to output to terminal
        LDA 1, .ERUFT                   ;Load another expected error
        SUB# 1, 2, SZR                  ;Is this channel in use?
        JMP @ .ER                       ;No - state unexpected error condition

;     IF FILE VVQQ EXISTS, WRITE ALL INPUT TO IT
;     - - - - - - - - - - - - - - - - - - - - -

        LDA 1, MATSTRT                  ;Yes - write input buffer toVVQQ
WRTOFL: STA 1, MATPTR
        LDA 1, ONE                      ;Load one (1) - number of bytes to write
        LDA 0, MATPTR                   ;Load pointer to match buffer
        MOVOL 0, 0                      ;Make this a bytepointer to right byte
        .SYSTM
        .WRS 27                         ;Write right byte to VVQQ
        JMP @ .ER

        LDA 1, MATPTR                   ;Look at pointer to match buffer
        LDA 2, FMATBUSH                 ;and location of last character
        SUB# 1, 2, SNR                  ;Is buffer writing complete?
        JMP TORDLINE                    ;Yes - return to read next line
        INC 1, 1                        ;No - increment pointer and
        JMP WRTOFL                      ;write next character to VVQQ
```

```
;     DEFINE VARIABLES
;     - - - - - - - - - -

.ERUFT: ERUFT                          ;ERUFT = 21, channel in use
.ERDLE: ERDLE                          ;ERDLE = 12, file does not exist
ONE:    1

;     $TTO1 INTERRUPT SERVICE ROUTINE
;     - -- - - - - - - - - - - - - - - -

CIRA:   NIOC TTO1                      ;Idle device $TTO1 and return to next
        JMP 0, 3                       ;line after interrupt.  No need to save
                                       ;the processor state, as it is not
                                       ;changed

;     $TTI1 INTERRUPT SERVICE ROUTINE
;     - - - - - - - - - - - - - - - - -

IIRA:   STA 3, USP                     ;Save the previous processor state by
        STA 2, SAVE2                   ;saving all accumulators and the carry
        STA 1, SAVE1                   ;bit.  USP is User Stack Pointer
        STA 0, SAVE0
        MOVL 0, 0
        STA 0, SAVEC

        DIAC 0, TTI1                   ;Input character from $TTI1
        LDA 3, PMSK                    ;Strip parity bit and
        AND 3, 0
        STA 0, @ INPTR                 ;store in BUFR

        LDA 1, INPTR                   ;Load pointer to next BUFR location
        INC 1, 1                       ;Increment it
        LDA 2, MAXPTR                  ;Load pointer to last buffer entry
        SUB# 1, 2, SNR                 ;Is buffer full? Yes -
        JMP CHEK                       ;jump to see if buffer has been used
        STA 1, INPTR                   ;No - store new pointer value

RETRN:  LDA 0, SAVEC                   ;Restore the state of the processor
        MOVR 0, 0
        LDA 0, SAVE0
        LDA 1, SAVE1
        LDA 2, SAVE2
        LDA 3, USP
        JMP 0, 3                       ;Return to next location after interrupt

CHEK:   LDA 3, OUTPTR                  ;If outptr is still at the beginning
        LDA 2, CHKPTR                  ;of the buffer BUFR, there is a
        SUB# 2, 3, SNR                 ;buffer overflow potential requiring
        JMP PROB1                      ;the processor to halt.  Otherwise,

        STA 2, INPTR                   ;store new value of INPTR and
        JMP RETRN                      ;return

PROB1:  HALT                           ;Not a recoverable error - STOP!
```

```
;     ROUTINE TO RETURN TO THE CLI ABNORMALLY
;     - - - - - - - - - - - - - - - - - - - - -

ERR:      .SYSTM
          .ERTN                              ;Abnormal return - error
          JMP @ .ER

;     DEFINE STORAGE AREAS AND BUFFERS
;     - - - - - - - - - - - - - - - - - -

SAVE0:  0                                    ;Storage area for accumulator 0,
SAVE1:  0                                    ;1, 2, and carry bit
SAVE2:  0
SAVEC:  0

MATBUFR:.BLK 133.                            ;Match buffer has 91 decimal
BUFR:   .BLK 133.                            ;locations.  So does input buffer

;     - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

          .END SYSIN

;-------------------------------------------------------------------------------
;-------------------------------------------------------------------------------

;     + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
;     +                                                                       +
;     +                        END SYSIN.SR                                   +
;     +                                                                       +
;     + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +

;-------------------------------------------------------------------------------
;-------------------------------------------------------------------------------
```

AD-A100 819 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL--ETC F/G 9/2
CONSTRUCTION OF A GENERAL PURPOSE COMMAND LANGUAGE FOR USE IN C--ETC(U)
SEP 80 W D GRIESS
UNCLASSIFIED AFIT/GCS/EE/80S-15 NL

```
C-----------------------------------------------------------------------
C-----------------------------------------------------------------------
C
C                          THIS IS THE CYBER ACTION FILE.
C                                +++++++ CREATED 7 AUGUST 1980; REV 01 ++
C
C  THE FIRST CONTROL CARD IMAGE (I) CONTAINS EXPECTED RESPONSES FROM
C  THE CYBER SYSTM.
I       COMMAND-,..
C
C  THIS COMMAND PERMITS LOCAL FILES TO BE SENT TO THE SYSTEM.
C  CORRECT INPUT IS:    PUT,LFN,SFN,ID,SFPASSWRD
.CACT,PUT,#1,#2,#3,#4
WS      COPYBF,INPUT,ZQY
WC      XFER/A #1 QQVV/R
WS      %EOF
WS      REWIND,ZQY
WS      REQUEST,ZQZ,*PF
WS      COPYBF,ZQY,ZQZ
WS      CATALOG,ZQZ,#2,ID=#3,RP=999,PW=#4
WS      RETURN,ZQZ,ZQY
END.
C  THIS COMMAND PERMITS SYSTEM FILES TO BE RECEIVED LOCALLY.
C  CORRECT INPUT IS:    GET,SFN,ID,SFPASSWRD,LFN
.CACT,GET,#1,#2,#3,#4
WS      ATTACH,QZQ,#1,ID=#2,PW=#3
RR
WS      COPYSBF,QZQ,OUTPUT
RC      XFER/A VVQQ #4/R
WS      RETURN,QZQ
END.
C  THIS COMMAND PERMITS SYSTEM FILES TO BE PRINTED ON SYSTEM PRINTER.
C  CORRECT INPUT IS:    SPRINT,SFN,SFPASSWRD
.CACT,SPRINT,#1,#2
WS      ATTACH,ZXQ,#1,PW=#2
WS      REQUEST,ZYQ,*Q
WS      COPYSBF,ZXQ,ZYQ
WS      REWIND,ZYQ
WS      ROUTE,ZYQ,DC=PR,TID=BB,FID=NEO,ST=CSB
WS      RETURN,ZXQ,ZYQ
END.
C  THIS COMMAND PERMITS SYSTEM FILES TO BE PUNCHED ON SYSTEM PUNCH.
C  CORRECT INPUT IS:    SPUNCH,SFN,SFPASSWRD
.CACT,SPUNCH,#1,#2
WS      REQUEST,ZJQ,*Q
WS      ATTACH,ZJJ,#1,PW=#2
WS      COPYSBF,ZJJ,ZJQ
WS      REWIND,ZJQ
WS      ROUTE,ZJQ,DC=PU,FID=NEO,TID=BB,ST=CSB
WS      RETURN,ZJQ,ZJJ
END.
```

```
C  THIS COMMAND PERMITS SYSTEM FILES TO BE DELETED.
C  CORRECT INPUT IS:    DELETE,SFN,SFPASSWRD
.CACT,DELETE,#1,#2
WS       PURGE,UZU,#1,PW=#2
WS       RETURN,UZU
END.
C  THIS COMMAND PERMITS DISPLAY OF SYSTEM FILES IN USE (CREATED,ATTACHED).
C  CORRECT INPUT IS:    FILES
.CACT,FILES
WS       FILES
END.
C  THIS COMMAND PERMITS DISPLAY OF SYSTEM PERMANENT FILES (AUDIT).
C  CORRECT INPUT IS:    PFILES,USER ID (PROB NUM)
.CACT,PFILES,#1
WS       AUDIT,AI=P,ID=#1
END.
C  THIS COMMAND PERMITS USER ACCESS TO THE SYSTEM.
C  CORRECT INPUT IS:    LOGON,USER ID (PROB NUM),USER PASSWRD
.CACT,LOGON,#1,#2
WL       Dial the CDC CYBER telephone number (currently 5180 or 5159),
WL       wait for the tone, and then place the telephone headset
WL       into the modem receiver.  Now strike any key
WL       on the keyboard.
RW
WS       LOGIN,#1,#2,777
END.
C  THIS COMMAND TERMINATES ACCESS TO THE SYSTEM.
C  CORRECT INPUT IS:    LOGOFF
.CACT,LOGOFF
WS       LOGOUT
WL       The CDC CYBER is logged out.  Enter uparrow L, "^L",  to return
WL       to the local NOVA/ECLIPSE CLI.
END.
```

```
C  THIS COMMAND PERMITS LOCAL FILES TO BE PRINTED ON SYSTEM PRINTER.
C  CORRECT INPUT IS:    LPRINT,LFN
.CACT,LPRINT,#1
WS      REQUEST,XZX,*Q
WS      COPYBF,INPUT,XZY
WC      XFER/A #1 QQVV/R
WS      %EOF
WS      REWIND,XZY
WS      COPYSBF,XZY,XZX
WS      REWIND,XZX
WS      ROUTE,XZX,DC=PR,FID=NEO,TID=BB,ST=CSB
WS      RETURN,XZX,XZY
END.
C  THIS COMMAND PERMITS LOCAL FILES TO BE PUNCHED ON SYSTEM PUNCH.
C  CORRECT INPUT IS:    LPUNCH,LFN
.CACT,LPUNCH,#1
WS      REQUEST,JZX,*Q
WS      COPYBF,INPUT,JZY
WC      XFER/A #1 QQVV/R
WS      %EOF
WS      REWIND,JZY
WS      COPYBF,JZY,JZX
WS      REWIND,JZX
WS      ROUTE,JZX,DC=PU,FID=NEO,TID=BB,ST=CSB
WS      RETURN,JZX,JZY
END.
C  THIS COMMAND PERMITS SYSTEM FILES TO BE EXECUTED (BATCHED) ON SYSTEM.
C  CORRECT INPUT IS:    SBATCH,SFN,SFPASSWRD,DISPOSITION,TERMINAL ID
.CACT,SBATCH,#1,#2,#3,#4
WS      ATTACH,VQY,#1,PW=#2
WS      BATCH,VQY,#3,Y=#4
WS      RETURN,VQY
END.
C  THIS COMMAND PERMITS LOCAL FILES TO BE EXECUTED (BATCHED) ON SYSTEM.
C  CORRECT INPUT IS:    LBATCH,LFN,DISPOSITION,TERMINAL ID
.CACT,LBATCH,#1,#2,#3
WS      COPYBF,INPUT,VQX
WC      XFER/A #1 QQVV/R
WS      %EOF
WS      REWIND,VQX
WS      BATCH,VQX,#2,Y=#3
WS      RETURN,VQX
END.
FINISH.
```

```
C
C-------------------------------------------------------------------
C
C  THIS ACTION FILE MAY BE EXPANDED OR CONTRACTED, PROVIDED ENTRIES MEET
C  THE FOLLOWING FORMAT AND CONTENT GUIDELINES:
C
C       A.  COMMAND IDENTIFICATION LINES ARE PRECEDED BY ".CACT" .
C       B.  THE LAST LINE IN EACH DISTINCT COMMAND SEQUENCE MUST BE "END." .
C       C.  THE LAST LINE OF THE ENTIRE FILE MUST BE "FINISH.", WITH THE
C           EXCEPTION OF COMMENT LINES.
C       D.  COMMENT LINES MAY BE PRECEDED BY ANY CHARACTER NOT RESERVED
C           AS INDICATED ABOVE OR BELOW.  FOR CONVENIENCE, "C" HAS BEEN
C           CHOSEN.
C       E.  THE FIRST TWO COLUMNS OF EACH LINE SATISFY CONTROL FUNCTIONS:
C
C               1)  "I " PRECEDES THE EXPECTED RESPONSES FROM THE SYSTEM.
C                   THEY ARE USED TO INITIALIZE THE EXPECTED RESPONSE ARRAYS.
C               2)  "WS" PRECEDES INFORMATION TO BE WRITTEN TO THE SYSTEM.
C               3)  "WL" PRECEDES INFORMATION TO BE WRITTEN TO THE LOCAL TERM.
C               4)  "RW" PRECEDES BLANK LINE; INDICATES A BACK-TO-BACK LOCAL
C                   TERM READ FOLLOWED BY A SYSTEM WRITE.
C               5)  "WC" PRECEDES CALL TO COPY A LOCAL FILE AND WRITE IT
C                   TO THE SYSTEM.
C               6)  "RC" PRECEDES CALL TO COPY A LOCAL FILE THAT HAS BEEN
C                   READ FROM THE SYSTEM.
C               7)  "RR" PRECEDES BLANK LINE; INDICATES THAT AN "RC" CONTROL
C                   WILL FOLLOW AND MAKES READY A LOCAL FILE FOR THE READ.
C               8)  SEE A THROUGH D ABOVE FOR OTHER CONTROL FUNCTIONS.
C
C       F.  ALL LINES (CONTROL CHARACTERS) MUST BEGIN IN COLUMN ONE (1);
C           ALL INFORMATION IN LINES OTHER THAN THOSE DESCRIBED IN A THROUGH D
C           ABOVE MUST CONTINUE IN OR AFTER COLUMN NINE (9), EXCEPT FOR COMMENT
C           LINES.  (TABS CANNOT BE USED ANYWHERE IN THE ACTION FILE, EXCEPT
C           IN COMMENT LINES AFTER THEIR CONTROL CHARACTERS.)
C
C
C-------------------------------------------------------------------
C-------------------------------------------------------------------
```

```
C       THIS IS THE DEC ACTION FILE.  THERE ARE NO ENTRIES
C       AS OF 15 JULY 1980.
C
I
FINISH.
```

```
C       THIS IS THE VAX ACTION FILE.  THERE ARE NO ENTRIES
C       AS OF 15 JULY 1980.
C
I
FINISH.
```

```
C       THIS IS MY OWN ACTION FILE.  THERE ARE NO ENTRIES
C       AS OF 15 JULY 1980.
C
I
FINISH.
```

VITA

Wayne Griess was born on 5 April 1949 in Scottsbluff, Nebraska. He graduated from high school in Cheyenne, Wyoming in 1967 and attended the University of Wyoming from which he received the degree of Bachelor of Science in 1971. Upon graduation, he received a commission in the USAF through the ROTC program. He entered active duty as a communications operations officer, first serving with the 1879th Communications Squadron, Richards-Gebaur AFB, Missouri. He then served as Voice Operations Branch Chief, 1931st Communications Group, Elmendorf AFB, Alaska. During this time he earned a Masters of Public Administration degree from the University of Alaska, Anchorage. He was then the Commander, 2064th Communications Squadron, Shemya AFB, Alaska until entering the School of Engineering, Air Force Institute of Technology, in January 1979.

Permanent address: 1950 E. 18th Street
Cheyenne, Wyoming 82001

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/GCS/EE/80S-15	2. GOVT ACCESSION NO. AD-A100 819	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* CONSTRUCTION OF A GENERAL PURPOSE COMMAND LANGUAGE FOR USE IN COMPUTER TO COMPUTER DIALOG		5. TYPE OF REPORT & PERIOD COVERED MS Thesis 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Wayne D. Griess Captain		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Air Force Institute of Technology (AFIT-EN) Wright-Patterson AFB, Ohio 45433		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS		12. REPORT DATE September, 1980 13. NUMBER OF PAGES 198
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*		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

Approved for public release; IAW AFR 190-17

FREDRIC C. LYNCH, Major, USAF
Director of Public Affairs

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

Command Language
Computer to computer dialog
Computer Interfacing/Interconnection
Command Language Interpreter

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

Two computer programs were developed and implemented to enable intercommunication between a Data General NOVA/ECLIPSE computer system and another modem linked computer system. One program, called TTERMOP, allows a user to sit at a NOVA terminal and interact with a connected system in a transparent mode. The other program, called MONITOR, is a command language interpreter that examines and executes instructions contained within an action file. An action file, consisting of instruction

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

strings and associated control parameters, is designed to be dependent upon a connected system with regard to contents, yet independent of such a connected system with regard to structure and format. The interpreter is written in FORTRAN IV with FORTRAN and assembly language modules. Actual implementation of the programs is accomplished between the NOVA/ECLIPSE and the Aeronautical Systems Division Control Data CYBER computer system. ASCII data files between 20 and 35,000 bytes have been transferred between the two interconnected systems, each transfer initiated by a single string command acceptable to the interpreter and compatible with a tailored action file for the CYBER system. The programs were designed to be flexible enough for use with several different connected systems, and general enough to be hosted on a system other than the NOVA/ECLIPSE. However, no attempt is made to implement the programs outside of the NOVA/ECLIPSE - CYBER environment.

END

DATE
FILMED

7-81

DTIC