A MICROPROCESSOR-BASED COMMUNICATIONS INFORMATION SYSTEM

Robert Harry Ekstrom

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A MICROPROCESSOR-BASED COMMUNICATIONS INFORMATION SYSTEM

by

Robert Harry Ekstrom

and

William Henry Reinhardt III

June 1975

Thesis Advisor: V. M. Powers

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Microprocessor-Based Communications Information System		5. TYPE OF REPORT & PERIOD COVERED Master's thesis; June 1975 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Robert Harry Ekstrom and William Henry Reinhardt, III		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		12. REPORT DATE June 1975 13. NUMBER OF PAGES 109
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Naval Postgraduate School Monterey, California 93940		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

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

18. SUPPLEMENTARY NOTES

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

micro-processor-based information system
application of microprocessing system
NEDS-4 Communications System

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

A functional design of a microprocessor-based system is proposed as a model for the Naval Environmental Display Station for use by the Naval Weather Service Environmental Detachments. The design consists of four modules: control, communications, storage, and display. A software program prototype that simulates many of the proposed functions of the control module is discussed. The processing requirements for the communicatio

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE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

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. (cont.)

module are presented along with a proposed hardware configuration. The storage module, based on a floppy disk system, is explained and its required functions defined. A display module utilizing an intelligent terminal and two CRTs is considered. The microprocessing system was designed to show not only the power and flexibility of this system, but also to demonstrate a potential application of low cost microprocessor technology.

A Microprocessor-Based Communications Information System

by

Robert Harry Ekstrom
Lieutenant, United States Navy
B.S., Northern Illinois University, 1968

and

William Henry Reinhardt III
Lieutenant, United States Navy
B.S., University of Maryland, 1968

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

ABSTRACT

A functional design of a microprocessor-based system is proposed as a model for the Naval Environmental Display Station for use by the Naval Weather Service Environmental Detachments. The design consists of four modules: control, communications, storage, and display. A software program prototype that simulates many of the proposed functions of the control module is discussed. The processing requirements for the communications module are presented along with a proposed hardware configuration. The storage module, based on a floppy disk system, is explained and its required functions defined. A display module utilizing an intelligent terminal and two CRTs is considered. The microprocessing system was designed to show not only the power and flexibility of this system, but also to demonstrate a potential application of low cost microprocessor technology.

TABLE OF CONTENTS

I. BACKGROUND

A. OPERATIONAL REQUIREMENTS

The range of modern weapon systems and the speed with which present military operations often take place have dictated a need for worldwide and "real time" environmental information. This implies that military commanders have a need to know as quickly as possible not only what environment their forces will likely encounter in the future, but also the present environmental conditions throughout the world.

The meteorological community has made great strides towards providing the real time support required. Digital computers, already in use, can rapidly assimilate and produce acceptable analysis and forecasts from the appropriate raw environmental data. The processed environmental parameters are then disseminated for the immediate use of the operational units. Full real time support is dependent upon the timely receipt of raw data and the delivery of the products, once generated, into the hands of the operational users. High speed automated communications is the crucial factor in the attainment of the real time data requirement.

B. SUPPORT AGENCIES

The Naval Weather Service Command is responsible for providing all of the environmental support required by the Naval community. This support, both meteorological and oceanographic, is supplied through the Fleet Numerical Weather Central (FNWC) located at Monterey, California. A primary mission of the Fleet Numerical Weather Central is to provide a complete set of worldwide environmental services and products in support of the entire spectrum of Naval operations and planning. Fleet Numerical Weather Central is also tasked with having a near real time operation due to the perishability of the environmental data and the necessity of providing quick response to worldwide Naval operations.

Fleet Numerical Weather Central receives inputs through observations collected by the Air Force, the National Weather Service, and the Fleet Weather Centrals (FWC), via high speed data links. Fleet Numerical Weather Central then utilizes this data to produce environmental parameters. These products are distributed in a variety of forms (charts, graphs, messages) to the Fleet Weather Centrals, where further processing and relay to other Naval units is accomplished. This final relay is via radio facsimile, radio teletype and data links. The World Meteorological Organization, a United Nations activity, collects weather data from sources throughout the world. This data is distributed to the Naval community over the United States Air Force Automated Weather Network (AWN).

Fleet Numerical Weather Central is the primary source of data for all Naval units. Every twelve (12) hours a

complete cycle of data products is made and then distributed to Fleet Weather Centrals throughout the world. The data are transmitted over the Naval Environmental Data Network (NEDN) to the appropriate FWC activities. The Naval Weather Service Detachments (NWSED) and the weather units aboard ships receive the data products from the Fleet Weather Centrals over facsimile and teletype circuits. Communications networks operated by the National Weather Service transmit similar forecasts and analysis. The Naval Weather Service Detachments, within the United States, normally avail themselves of this additional service.

C. DISPLAY STATION CONCEPT

In an effort to upgrade the existing communications, to overcome systems limitations, and to expand the environmental support, the Naval Environmental Display Station (NEDS) was conceived and a prototype designed. The Naval Environmental Display Station was to be primarily a communications device, as set forth by the Chief of Naval Operations in CNO letter serial 1339 of 14 November 1972. It was envisioned that the users of this device would cover the spectrum from Fleet Numerical Weather Central (FNWC), with numerous environmentalists on watch, to the small afloat commands, with no meteorological expertise. Every activity having a critical requirement for the predictions and current status of environmental parameters, would be equipped with a NEDS. Reference 6 contains a general description of the Naval Weather Service Command, the communications networks, and the specifications for the orginal NEDS configuration.

The present NEDS prototype utilizes a mini-computer as the "central control". The system was designed to be

programmable only at FNWC and not at the individual station sites. All installations were to have an identical configuration of hardware, firmware and software. Every station would be capable of performing all of the various NEDS functions. The NEDS is unique in Naval systems, in that it utilizes state-of-the-art technology to provide communications and display of both alphanumeric and graphic messages. Figures 1 and 2 depict the "original NEDS" configuration, the planned NEDS applications and provide a general idea of the intended capabilities of the original NEDS concept.

Due to the spiralling costs involved with the original NEDS units, and the cutback in defense funding, it was determined to be impractical to equip every facility requiring meteorological data with the original NEDS configuration. Therefore, the Fleet Numerical Weather Central and the Computer Science Group of the Naval Postgraduate School, Monterey, California, became interested in developing an alternative NEDS prototype. This alternative prototype design is based upon microprocessor technology. The microprocessor system will provide the communication, data storage, and data retrieval control. The system design is an attempt to achieve maximum flexibility while minimizing the costs.

D. NWSED DESCRIPTION

Emphasis was placed upon the development of a functional specification for a Naval Environmental Display Station (NEDS) that would best be utilized at the Naval Weather Service Environmental Detachment (NWSED) level. An attempt was made to meet all system capability requirements, and yet keep costs far below the present prototype.

The Naval Weather Service Environmental Detachments are located at Naval Air Stations throughout the world. The NWSED is normally manned by one or two meteorological officers and from ten to twenty enlisted aerographers mates. Each NWSED provides all the environmental data required by the air station, which includes aviation squadrons, flag officers, staffs, and other activities. Each NWSED receives weather data over multiple communications circuits.

Two general categories of data are retained by a NWSED: alphanumeric and graphic. These two data types are currently transmitted to the NWSED over seperate communications lines. The graphic data is provided to the NWSED via analog (facsimile) circuits by the appropriate Fleet Weather Central. The alphanumeric data can arrive from any number of agencies in a variety of formats. Each NWSED has one or more teletype circuits (100-250 words per minute) on the Air Force Automated Weather Network (AWN). These circuits will soon be increased to 1200 bits per second (2100 words per minute). The AUTODIN, a computer communications network, is occasionally used for the transmission of weather information, and in addition, some NWSED units are served through local civil weather networks. Thus each NWSED unit normally receives facsimile and teletype information from the appropriate Fleet Weather Central, weather data from the AWN circuit, information from the AUTODIN, and reports from various civilian sources.

The information transmitted to a NWSED is normally assimilated by one or more computers in the weather network. Computer processing has increased the amount of weather information available and the speed of its delivery.

Each NWSED unit is required to maintain and update files of required information. This information is received throughout the day on the various teletype and facsimile

circuits. Each circuit serves several NWSED units, or similar units, and only a portion of the information carried will be applicable to the individual station. As required information is received, it is presently filed on a display board. Each new update of a teletype bulletin or map is placed over the previous existing one of the same type. Periodically, over-age messages and bulletins are removed and filed or destroyed. When personnel come into a NWSED to be briefed on the weather (locally, nationally, or worldwide) the forecaster shows them the particular maps and teletype bulletins of interest. In the case of an aviation crew, weather information is recorded on their flight clearance form. For flight crews, the forecaster must at times provide a cross section of expected weather along the intended flight path. Thus a forecaster must, except for immediate local weather information, prepare the briefing ahead of time. This preparation requires the forecaster to consider numerous teletype bulletins and consult various weather maps.

In addition to briefings, the NWSED unit is required to prepare a number of routine forecasts for local use. These forecasts, along with other local environmental parameters, are transmitted on the teletype networks to other units. The recipients of these reports include the Fleet Weather Central (FWC) which supports the detachment's data needs. The transmissions are prepared on an established schedule, using all available information.

In some cases information is needed which is not normally scheduled to be received at the NWSED. When such information is required, an Automatic Response to Query (ARQ) is used to obtain the necessary data on the AWN circuit. The center controlling the AWN circuit extracts the requested information from its data base and transmits it, usually in a matter of minutes, to the requesting unit.

E. DESIGN CONSIDERATIONS

A microprocessor-based display system for use by a Naval Weather Service Detachment (NWSED) should be able to receive both alphanumeric and graphic forms of data. The system must store this data, retrieve it on demand, and display it in a format convenient for the forecaster's use. The system should also assist in the preparation of outgoing messages, such as forecasts, warnings, and Automatic Response to Query requests. All data will be digital, utilizing a vector form of graphic data to replace the current analog (facsimile) form.

In order to effectively operate at the NWSED level, an information system must be both easy to operate and easy to maintain. The system users will be enlisted aerographers mates and meteorological officers. Both groups of potential users have had extensive training and experience working with coded data, but may have little or no background in data processing. To utilize the proposed retrieval and display system, only a short introduction to the system commands should be required.

Maintenance personnel qualified to repair computer systems will not normally be available. Software routines should be provided to aid personnel in system error detection. The Naval Environmental Display Station unit should be as modular as possible, allowing hardware problems to be isolated to a given section. The defective module may then be replaced locally and returned to a central repair facility.

The capabilities necessary for an information system to

be effective at a NWSED are available today in microprocessor circuitry. An information system has been proposed consisting of a central controller, a communications interface, a cathode ray tube (CRT) display, and a floppy disk storage device. This paper is intended to study the basic functions of such a system and to determine the feasibility of the proposed design. A program which exercises many of the required functions and demonstrates the feasibility of a microprocessor-based system has been developed and will be discussed in the following sections of this report.

II. SYSTEM OVERVIEW

A. HARDWARE CHARACTERISTICS

The Naval Environmental Display Station (NEDS), as proposed, will consist of a microprocessor-based central control with a communications interface, two cathode ray tube (CRT) displays, and a floppy disk storage device. The hardware will be organized into four major modules: communications, storage, control, and display. Figure 3 presents a diagram of the proposed system with the module interface characteristics depicted.

1. The Communications Module

The communications subsection is a straightforward design problem. Similar communications modules are already being utilized in other systems. Figures 4 and 5 show part of an envisioned design.

The major function of this module is the "simultaneous" transmission and reception of data. The communications module will be capable of handling several communication lines operating at various speeds (75-9600 bits per second). This subsection will recognize the incoming data, remove the synchronization pulses, perform some preprocessing, buffer the incoming data, and pass the data to the control module on demand. The communications module will contain a microprocessor, the necessary memory for processor control and data buffering, a universal

asynchronous receiver/transmitter (UART) for each communications line, and the circuitry needed to interface with the control module and the communications lines.

2. The Storage Module

The storage section consists of a dual drive floppy disk system. This subsection is available as an off-the-shelf component from a number of manufactures. No hardware or other modifications would be required to utilize such a disk system in the present design. The disk controller, and the necessary software for system operation, will be physically located in the control module and are currently under development. Figures 6 and 7 depict the controller interface and the commercial floppy disk system.

3. The Control Module

The control module will consist of a microprocessor, memory, and the components necessary to interface with the other modules. The functions of the control module are:

* receive data from the communications section, determine what data is to be retained, and format the data for storage

* direct the floppy disk in the storage and retrieval of information

* perform self test and error analysis

* interpret inputs from the operator's keyboard

* pass properly formatted data and control signals to the display module

* maintain and update the tables necessary to access the stored information

4. The Display Module

This module will consist of two, television type, cathode ray tubes (CRT) and one entry keyboard. The display module will function not only as a display, but will also perform some data manipulation and processing. One CRT subsection will accept graphic data and formatting commands from the control module, unpack the data according to the formatting commands, place the data in a display memory, and refresh the CRT from this memory. This section of the display module is being developed by Compata Inc. under contract number N62271-75-M-0646, and should be operational by June 1975. The second CRT will be used for alphanumeric information display.

The entry keyboard will be interfaced with the control section via the communications module. The keyboard will be handled as an additional communications line.

B. SOFTWARE CONSIDERATIONS

The proposed system would consist of five (5) software modules: executive controller, communications, data storage and retrieval, graphics, and alphanumeric display. All software should be designed to incorporate the following specifications:

* make maximum use of existing software.

* utilize modular programming in order to permit easier

modification, expansion, and future software development.

* utilize a high level language, such as PL/M, for all software modules. The advantages of high level language utilization are numerous. The manpower savings gained in simplified program maintenance, the reduction in programming time, and simplified documentation are but a few of the more obvious reasons a high level language should be used.

* select the optimal memory buffer size. Consider not only memory allocation, but also the number of disk access occurrences which will be required.

* consider the use of dynamic disk storage instead of allocating specific areas to individual data types. This would maximize the disk area available for data storage.

C. LOGICAL CONSIDERATIONS

The most important considerations in the design of the logical structure of the system were:

1) reception and storage of data

2) system commands desired

3) information retrieval

4) information display

Considering the small number of different queries required and the assumed familiarity of the user with the information system, items 2 and 4 above present little difficulty. Item 3 must be analyzed to determine if

information retrieval can be accomplished rapidly enough to satisfy the user. Item 1 must also be analyzed to ensure that processor and disk speeds can handle the input stream without losing data or dead locking the system.

The time between a request by the user and the appearance of the requested display should be in the order of a few seconds. System response is dependent upon the time to:

* locate the proper file

* transfer the file from the storage device

* format the file

* display

Tables 1, 2, and 3 delineate the system data requirements, a possible file structure, and timing calculations. Considering the relative complexity and the volume of data involved (see Table 1) the request for a graphic display will probably be the most time consuming. The information in Table 2 lists a proposed file structure with the resulting estimated response times presented in Table 3. The maximum of 7.5 seconds, as computed in Table 3, is within the user response time requirements. See pages 41 and 42 for amplification of the timing constraints.

Effective handling of the input stream is dependent upon the file structure used. Some of the major considerations in arriving at the proposed file structure were:

* The relative size of each graphic and alphanumeric message, and the NWSED requirements for such data, suggests that each graphic message be contained in one file and each

of the alphanumeric data types be grouped together as a single file.

* Disk seeking will be a major time delay factor and must be minimized. By keeping file sizes relatively small, disk head movement after the initial seek can be limited to one track.

* Information retrieval should not require excessive linear search. Small file sizes will reduce the data search time.

* The data request should uniquely determine the logical file.

* The logical file should be directly linked to the physical file. This will shorten the search time.

A commercial floppy disk operating system has been implemented and allows microprocessor control of a floppy disk. The disk file structure is adaptable to the information requirements of the NWSED system. Important characteristics of the disk operating system are shown in Table 4.

The proposed file structure is implemented by the control program. For specifics on the file structure, the data retrieval logic, and the command priority system, consult the sections which follow and the program listing in Appendix A.

III. CONTROL MODULE

A. BASIC SYSTEM ARCHITECTURE

A communications processing system must handle a large number of data lines, all operating simultaneously and independently, and all capable of demanding dynamically varying processing and storage loads. The system must be designed to respond to all of the requirements of all the lines at all times. In a communications processing system each message must be processed serially through the system by task, and each task must be executed within a specific time interval. Aggravated by the condition that a large number of messages are continually flowing through the system in various stages of processing, the complex nature of communications processing and computerized message switching becomes apparent. The computer must rapidly switch processor attention from message to message, performing each required task, for each message, within the time constraints for that task and that message. A system designed around a number of microprocessors, each performing certain functions, and coupled through shared memory, could have many advantages, both in cost and performance, over one large system.

In such a design, each processor module is assigned only certain specific operations, and it performs these operations on all messages received or transmitted by the system. The processing of each message is performed in a

serial fashion with the message passing from module to module via common/shared memory or input/output ports. Shared memory is utilized for information transfer between the control module and the graphic display. Direct memory access is available for data transfer between the disk system and the control module. Processor input/output ports are employed in data movement between the communications module and the rest of the Naval Environmental Display Station (NEDS).

The communications module is dedicated to the buffering and preprocessing of the communications lines. The communications section receives input characters, assembles them into memory buffers, and passes the information, on demand, to the control module via an input port. On output, the communications processor receives the message characters, under the command of the control module, from the terminal display processor. The message is buffered in the communications memory, and transmitted on the appropriate communications lines, when directed by the control section. The communications processor also maintains timing information, detects parity errors and other failures in the transmission lines, and passes this information to the control module for possible display to the user.

The control module processes the input characters serially as received from the communications processor on the data bus. The control section validates and interprets all polling responses and message headers, generates acknowledgements and diagnostic messages. In addition, the control module utilizes its "disk operating system" to allocate storage, perform read/write operations, and validates and processes all commands addressed to the floppy disk system. The "disk operating system" is discussed in a later section of this paper.

The control section continually verifies the operational status of each module, including the floppy disk, monitors the condition of each communications line and appropriately updates the polling base, analyzes and reports all detected errors and performs diagnostics as directed by the user.

Part of the control module memory, shared with the display module, is used for transient information passing. This transient information is the output of a process in the control module, and is the input the display module accepts for the processing it performs. The transient information is placed in allocated buffer areas of the shared main memory. The use of shared main memory by the two asynchronously operating processors requires the use of sophisticated software and hardware techniques.

B. POLLING ROUTINE

The method used to multiplex the different peripheral lines to the control module is the poll and select method. This method allows for all input lines to communicate with the control section, via the communications module, over a single line as illustrated in Figure 3.

The communications module sets a one bit flag whenever data is available in a buffer. Each communications line has a separate buffer area in the communications module memory. The control module uses the flags to determine which buffer to empty, thus indirectly selecting which communications line will be handled next. The polling routine establishes the priorities of the various incoming data lines. Consult the program listing (procedure find-a-line) for present polling characteristics.

C. MAIN TABLE

All input and output operations update and/or utilize the "main table". The main table is the index to the information stored on the disk for both the control program and the operator. The table contains a table line number, a title index, the date time group of the information, the background number, the security classification, and the number of check sum errors that were detected during reception.

Copies of the main table are always retained both on the disk and in the control module memory. The main table may be initialized at system "start-up" from data stored on the disk or by clearing the table area.

D. SOFTWARE

The proposed system operates in one of two states. In the command state all information received by the control module is treated as system commands. In the receive state the control program considers all incoming data to be part of a bulletin. These two states are discussed below and the control program implementation is presented in the program listing.

The control program utilizes multiprogramming to accomplish the required data manipulations. Each communications line, including the keyboard, is processed by an execution sequence independent of the other lines. The control program polls from line to line, executing the necessary program steps for that particular line. Figure 3 shows the data flow in a system configuration where one line

is in the command mode and all other lines are in the receive mode.

Two buffers and a line information area (HEADER/FCB) are allocated in the control module memory for each communications line. The buffers are 128 byte records with an additional byte allocated for each record for buffer access control. The information area contains program pointers, mode keys, and the other data necessary to implement multiprogramming.

The control program continues to process the same communications line as long as the communications module has data available in the line's buffer area, and an end of bulletin is not received. Once the control program empties the communications module buffer for the line in processing, or an end of bulletin is received, a return to the polling routine is executed. The polling routine currently in use always tests for available data in the same sequence (high speed line first, keyboard last). The first data line's buffer encountered by the polling routine with information present results in selection of that line for processing by the control module.

1. System Commands

Normally only the keyboard line is in the command mode, but any communications line can execute the control program in the command state. Any communications line can access the command mode by sending a unique sequence to the station where commands are to be executed. Once in the command mode the following inputs are allowed, all others being disregarded.

a. Reset

The system is completely re-initialized by the reset command. The disk is reset and the control program will execute the following "power-up" routine. Until the operator indicates via the keyboard whether the disk contains a valid main table, the system will idle. The main table is initialized from the disk or it is cleared (see main table description above). All buffer parameters and all information areas are set to a system start value. The keyboard line is placed in the command mode and all other lines are set to the receive state.

b. Table

The main table is displayed on the alphanumerics CRT. An ability to page or escape from the table mode is provided. The title index is converted to a plain English text title prior to CRT display.

c. Display

The control program requests a two-digit input to determine which file is to be displayed. The digits input are to correspond to the main table line number associated with the file requested for display. A system test ensures that the input is valid and that a file exists at the table line number entered. If the input is invalid a new command is requested.

If the file requested contains graphic data, the file and a background are sent from the disk to a shared

memory area. The display processor then formats the information received via the shared memory for visual display on the graphics terminal. All other files are output directly to the alphanumerics CRT.

During execution of the display command, the control program transfers only one record, 128 bytes, prior to returning to the polling routine. This enables a higher priority communications line to "interrupt" the display command.

d. Erase

The command program requests a two-digit entry, similar to the display request, in order to determine which file to erase. After verification from the operator that the correct digits were entered, the file is deleted from the main table and the disk space allocation and the disk table are updated.

e. ARQ

This command, not presently implemented, displays a message format on the alphanumerics display. The system user then provides the necessary information to complete the message. The completed message form is forwarded to the communications module for verification and transmission on the appropriate circuit.

f. Copy

This command is not presently implemented. The copy command will transfer the information from one disk to

another.

g. Receive

The receive command places the communications line which is being processed into the receive state. Any data following this command is regarded as message traffic.

h. Find

The find command requires a five digit or three letter station identifier from the data line currently being processed. A linear search through all "hourly reports" is then executed. Every occurrence of a match between the input identifier and a stored station list results in an alphanumeric output. The search and display can be terminated at any time. Notification is sent to the operator if either an unsuccessful search or a request for a station not retained by the system is made.

2. Message Handling

When in the receive mode the control program files the incoming data, if the information is to be retained, according to type. Either the system retains the complete bulletin or the system selectively retains messages within the bulletins. Consult the program listing for details on the exact system implementation of receive mode routines.

a. Bulletin Retention

Upon receipt of a start of bulletin indication

from the communications module, the control program begins processing the data stream. The bulletin title is loaded into the control module information area for the line being processed. A test is made to determine if this bulletin type is to be retained. If the bulletin is not retained, the communications module is instructed by the control processor to "dump" (destroy) all data received on the line until a new start of bulletin is received.

If the bulletin is to be retained, a disk file is created. The information received is stored a record at a time, utilizing the buffers in the control module, until an end of bulletin indicator is received. Upon the receipt of the end of bulletin the control program updates the appropriate main table entries in memory and on the disk. If excessive check sum errors are encountered or the bulletin is incomplete (no end of bulletin) the file is closed and deleted thus freeing any disk space that may have been utilized by the file.

b. Message Retention

The same basic process is utilized during retention of messages within a bulletin. Depending upon the date time group of the bulletin, either the data to be retained is added to an existing file or a new file is created. If a new file is created the table will be updated upon receipt of the end of bulletin mark.

If the bulletin is the proper type, the message headers in the bulletin are compared to a stored listing. If a match occurs, the information up to the next message header is filed.

IV. COMMUNICATIONS

A. TYPES OF INFORMATION

The two categories of information, alphanumeric and graphic, received at a Naval Weather Service Environmental Detachment (NWSED) are depicted in Tables 1 and 2. The techniques required for handling, and storage, of the two data types differ. For a better understanding of system requirements, a brief description of the two categories follows.

1. Alphanumerics

Weather reports, forecasts, and warnings are all in the alphanumerics category. The hourly airways observations are normally the most important data reports the NWSED receives. These bulletins are transmitted every hour, and describe the current weather on all major airways. Each NWSED is required to keep a file of these reports for all airports under its cognizance, and any other airfields in which it has an operational interest.

Adjunctive data of interest to the NWSED includes weather forecasts made elsewhere. Some of these forecasts predict severe weather which might endanger life or property and are called "weather warnings". These warnings, along with forecasts and reports, are generally grouped together

to form bulletins. Each bulletin consists of information which has similar time, location and data type characteristics. These bulletins make up the alphanumeric information that is transmitted to the individual NWSED units over the teletype circuits.

2. Graphics

Weather maps giving a pictorial description of environmental parameters make up the graphics category. These maps are weather bulletins formed into contour lines and plotted over a geographic background. The maps represent either analysis, meteorological conditions existing at a previous time, or forecasts, predicted conditions for a future time.

Graphic data is normally transmitted in analog form on a facsimile circuit to the individual NWSED units. Although the data is presently transmitted in analog form, it may be transmitted in digital form as a series of vectors. This can be done only if the NWSED has a processing unit available to convert the data into the proper display format. Tests at Fleet Numerical Weather Central (FNWC) have shown that weather maps, in vector format, can be compacted into 20,000-30,000 bits (2500-3750 bytes). This will reduce transmission time by at least a factor of two for each graphic report.

B. MESSAGE DRAFTING

Virtually all outgoing reports, forecasts, Automatic Response to Query (ARQ) requests and other messages, must meet mandatory format requirements prior to being accepted

by any communications circuit. A program should be provided to assist the operator in preparing these messages. Such a routine would display the proper format, allow the operator to fill in the necessary information, check the entries for proper form, and then transmit the message on the proper communications circuit. The major portion of this software would be located in the display and communications modules.

C. DATA TRANSMISSION/RECEPTION PROCESS

All point-to-point main line communications on the Naval Environmental Data Network (NEDN) are accomplished by computer systems utilizing a 12 bit per character binary coded data. The only exception to this is the weather data input from the Automated Weather Network (AWN), received from Carswell and Offutt Air Force Bases. The AWN data is formatted in a Baudot code using 5 bits per character. Each character is part of a special synchronous transmission format although the start-stop pulses, as in the regular baudot code, are omitted. These serial-by-bit streams of 5 bit codes are converted by the FNWC computer into the NEDN standard 12 bit code. Data exchanged back to the AWN, via Carswell and Offutt, is translated back to Baudot code.

Computer exchange of weather data and messages on the remainder of the NEDN lines, for Navy use, utilize the 12 bit code. Data is sent in streams of 64, 12 bit words in the serial-by-bit transmission state. Error detection and correction is accomplished through a check sum technique where an arithmetic sum, with end-around carry, is performed for each group of 64 words. This check sum is transmitted with the data and utilized by the receiving unit to ensure proper message reception. At the end of the transmission, a receive status is sent to the control station, and only

those 64 word blocks which do not meet the check sum tests are re-transmitted. For internal system use, two bits are added to each character for synchronization and control, making a 14 bit word. The two extra bits are removed when data is transmitted in the serial-by-bit state.

When data reaches a Fleet Weather Central (FWC) it is re-formatted and then relayed to the Naval Weather Service Environmental Detachments. Transmissions received by the NWSED, from the FWC, are in an 8 bit code. Thus, each NWSED receives the 8 bit coded data from the FWC and the 5 bit data on the AWN circuit. Implementation of the 8 bit code transmission lines is not complete, but the message format has been defined and is presented in Figure 8.

D. COMMUNICATIONS MODULE

The function of the communications module is to receive and transmit data on a number of different communications lines, all of which may have different operating characteristics. The communications module must be able to recognize the start of bulletin header on each circuit, compact it into one byte, and set a status flag when the byte is transferred to the control module. The communications section must also, when commanded by the control module, dump (destroy) all data received on a specific line until the start of a new bulletin is detected. End of bulletin sequences must also be compacted into one byte and the appropriate status flags must be set. If necessary, the communications module will re-format the data into a sequence, and code, compatible with the control module processing routines.

The general specifications of the communications module

are presented in order to define the interface between the communications module and the control module. These specifications not only further the developmental work on the communications module, they are necessary for the design and implementation of the control section software.

To better understand the operation of the communications processor, bulletin formats should be understood. All bulletins have a header, a text, and an end of bulletin sequence. Three communication controls that define these bulletin fields are:

SOB: Start of bulletin

STX: Start of text / divider in text

EOB: End of bulletin

These communications controls are actually specific sequences of characters which serve as boundaries for the bulletins. The bulletin preamble interval starts all transmissions and is used to connect and synchronize the receiver and transmitter. The connection sequence ends immediately prior to the control sequence characters for the SOB. The actual synchronization is accomplished through a combination of physical, electrical and logical sensing.

The remainder of the header contents must satisfy the Fleet Numerical Weather Central application delivery network requirements. Application heading items include the identity of the originator, data type and time of origin, bulletin type and classification. Network heading items may also include bulletin addressing and identification for re-transmission. References 2 and 6 contain information on transmission formats.

The hardware utilized should not commit the system to a communications capability on only specific data lines. All transmission/reception interface parameters should be program selectable or programmable in the communications module. In the communications line interface, transmission characteristics such as data rate, stop-bit length, character length, and parity, will be programmable and set by simple software instructions. To help accomplish this, bipolar mask-programmable read only memories (ROM) might be used. ROM programs could change the 5 bit AWN codes into the 8 bit parallel codes the microprocessor utilizes. Programmed ROM can also hold routines to strip off any unnecessary synchronization or control bits.

The communications interface will use a single universal asynchronous receiver/transmitter (UART), as depicted in Figure 5, for each communications line. There have been many complete, one-chip communication interfaces introduced, such as Motorola's ACIA and Rockwell International's TDI. These UART chips are capable of replacing 20 to 25 logic packages, thus simplifing the communication interfaces. These large-scale integrated circuits combine, in a single package, all the functions needed to connect a microprocessor system to a communications line.

The UART will convert the serial bits from the communications line into the byte (8 bit) characters required by the microprocessor system. While transmitting, the UART will perform the opposite conversion, receiving data bytes from the microprocessor and changing them into a stream of serial bits for the communications line. Each character will consist of 8 data bits, including parity. The UART checks the parity of incoming characters from the communications line and generates the parity bits during data transmission.

E. IMPLEMENTATION

The communications module will act as a buffer and a data line multiplexer. This section will also preprocess incoming data, aiding the control module in task completion. Buffer size will be determined by data rates and the number of lines which must be serviced. For a 9600 bit per second data rate, 773 bytes of buffer will provide enough memory space to prevent data loss during the disk seek period. The buffer is needed to compensate for the 643 millisecond access time required by the disk system (see page 42).

Every communications line utilized by the Naval Environmental Display Station, will be connected to the communications module via a UART. When a character (8 bits) is available in the UART, an interrupt will be generated. This interrupt is sent to the communications microprocessor (see Figures 4 and 5). Each UART in the communications module is allocated a unique line address code and sufficient circuitry is supplied to ensure automatic response when the address is called. The communications processor responds to the UART interrupt by sending the address code of the line involved to the transfer logic (see Figure 4). The UART then outputs the data byte on the bus. At the 9600 bit per second transmission rate, the microprocessor will have 833 microseconds in which to respond to the interrupt before the UART overwrites and destroys the data. With current microprocessor cycle times, approximately 2 microseconds, there should be little problem responding to every interrupt.

After a UART interrupt, the processor will transfer the byte to a memory buffer area allocated to the individual communications line. During the transfer, a test is made to determine if the byte completes any special control sequence

(SOB, STX, or EOB). If a control sequence is completed by this byte, the sequence is replaced by a one byte variable. This buffering and testing process is simultaneously occurring for each communications line connected to the module.

When the control module requests data via an interrupt, the communications processor tests an input port to determine with which data line to respond. Once the line is determined, one byte is removed from the line's buffer and sent to the control module, via an output port. If the buffer for the requested line is empty, a flag is set, and a coded byte returned to the control module.

Whenever a control sequence variable (SOB, STX, or EOB) or an empty file code is passed to the control module a status flag is also set. This ensures that no 8 bit data codes will be misinterpreted as a control character.

Another function required of the communications module is the capability to dump the remainder of a bulletin. The control module will send another interrupt type to request data destruction. The communications processor again tests to determine which line is being addressed. The buffer for that line is then emptied until the next SOB is encountered.

The communications module will continuously send system status information, over output port lines, to the control module.

During data transmission from the station, the communications processor will add synchronization bits, add parity and check sum codes, and re-format the data as required. Data will be released from the communications module and transmitted under the direction of the control processor.

V. FLOPPY DISK SYSTEM

A. DIRECT MEMORY ACCESS

The fourth major element of the system is the floppy disk storage device and the associated interface hardware. A crucial feature of the floppy disk system is a direct memory access (DMA) capability to the random access memory (RAM) in the control module. Direct memory access will allow the control processor to respond quickly to the multitude of system requirements. By using this direct memory access, the disk system can store data bytes into the main memory, or take them out, without receiving processor commands for each individual data transfer. Thus, the DMA transfers data while the processor is occupied with other tasks.

Direct memory access transfers are usually associated with devices that require extremely high data transfer rates, such as a floppy disk or a processor to processor transfer. In direct memory access, the interface transfers data directly to/from the main memory buffers. The control module accomplishes this type of transfer, via the disk operating system, by initiating and supplying the interface with a starting memory address and a word count. The processor then proceeds to other tasks.

The logic that performs the data transfer is usually referred to as a channel. This channel contains a memory

address register controlling the location in memory to or from which data is transferred. A counter must be provided to keep track of the total number of transfers performed. Logic must be provided for gaining access to the memory and for providing all the necessary timing and control signals associated with reading/writing from memory. Transfers are made without interfering with the operating program, since the channel essentially steals a memory cycle from the main processor. After each successfully stolen cycle, the memory address and word count are incremented and a test is made to see if the desired transfer has been completed.

Cycle stealing is the mechanism by which direct memory access is implemented. Normally, there are two processor cycles for each program instruction: a fetch cycle and an execute cycle. When the cycle steal comes up, for example during an execute, the processor will finish the execute cycle, and then pause for a one-cycle interval, before moving to the next instruction fetch cycle. It is during this pause that data bytes are moved between the memory and the disk. Cycle stealing generally reduces the time available to ongoing programs by approximately one per cent, in other words, just slightly slows down the processor.

B. INTERFACE

The actual interface between the control module and the disk is accomplished through a disk controller. A block diagram of the floppy disk interface is presented in Figure 6. The interface logic is activated by commands from the control module processor. The disk controller logic responds to information which has been previously passed to it by the control module. This information includes the data required for DMA operations: the disk number, the

track number, and the sector number. The disk controller positions the read/write head and generates all the required hardware control signals. When the head is above the desired sector and all required delays have elapsed, disk reading or writing is initiated. When a disk read is desired, the controller hardware locates the sector and synchronizes the clocking pulses. One record, 128 bytes, is framed and then transmitted to the main memory via the direct memory access channel. An interrupt is generated to signal the end of transmission. When writing a record to the disk, the controller again utilizes the DMA feature. After 128 bytes have been transferred to the disk, an interrupt signaling the end of transmission is sent to the control processor. The interface hardware is complicated by the direct memory access feature, but the advantages of DMA outweight the cost and complexity of the additional circuitry.

C. HARDWARE

The floppy disk used is the Shugart SA900/901 diskette. The floppy disk storage medium is an expendable cartridge, about the size of a 45 rpm record. Data is recorded on one side of an oxide-coated, flexible Mylar sheet, cut into a 7.8 inch disk. The disk is packaged in an 8 inch square envelope containing cut-outs for drive hub mounting, index mark sense, and read/write head access. During operation, the mylar disk rotates, while the envelope remains stationary. Each cartridge costs about five dollars.

The floppy disk cartridge is manually placed into the drive housing. The drive hub clamp then engages, holding the disk against a rotating drive spindle. During reading or writing, pressure pads hold the read/write head against

the disk, the head being in physical contact with the surface, as depicted in Figure 7. Since this causes wear, it is best to raise the head as soon as possible after completion of a read or write operation. After about 10,000 accesses to the same location, the disk has reached useful surface life.

The floppy disk cartridges store approximately 2.1 million bits when formatted. Data can be recorded on 77 tracks, at a density of about 3200 bits per inch. A disk format of 26 sectors per track, IBM standard format, allows storage of 256 thousand bytes of data. This is equivalent to 3200 80-column punch cards. Data (both information and timing bits) transfers at a rate of 250 kilohertz when the disk is rotating at 360 rpm. The disk hardware will cost approximately 1500 dollars per single unit. Reference 1 contains an overview of floppy disk systems.

D. TRANSFER RATES

Data transfer between the floppy disk interface and the disk occurs at the rate of about 160,000 information bits per second. This is a burst I/O rate of 20,000 bytes per second. But when evaluating the disk read/write mechanism, the average access time is the most important consideration. Generally, this quantity depends on how fast the disk rotates and what type of head positioning mechanism is utilized. The average access time is the sum of the rotational latency and the positioning (seek) time. The average positioning time equals one-third of the head's full-stroke time, plus the time for the head to settle and begin transferring data reliably. The track-to-track time for the Shugart disk is listed as approximately 10 milliseconds, but present disk control software allows 20

milliseconds. Multiplying the 20 milliseconds by one-third the total number of tracks (26), and adding the head loading time (40 milliseconds), results in a positioning time of 560 milliseconds. The average latency time, one-half the period of one revolution, equals 83.3 milliseconds.

The present design results in an average access time (seek and latency) of 643 milliseconds and a data record transfer time of about 6.5 milliseconds (1 second / 6 rps / 26 sectors). The total transfer and access time amounts to 650 milliseconds for a 128 byte record.

Because of the time delay involved in seeking, a disk queue will not be used. If the control processor tries to execute a disk command while the disk is busy, the processor will be forced to idle until the disk is available. A communications line operating at 9600 bits per second, transmits 128 bytes, one record, every 106.5 milliseconds. A disk queue, further delaying the 643 millisecond disk response time, would demand unrealistic amounts of communications buffering. With the present configuration a minimum of 773 bytes are required to buffer a 9600 baud line (643 / 106.5 * 128).

If a 9600 bit per second communications data rate is used, the disk system should be modified to reduce the disk seek time. The disk directory and the system table should be located in the middle of the disk tracks. Tracks should also be allocated to "high speed line" data types. These tracks would be located in the center of the disk, surrounding the directory and system table. By such an allocation of tracks, seek time to the high speed area would be reduced by approximately 250 milliseconds. By lowering the software track-to-track time, further gains could be made.

E. SOFTWARE

A "disk operating system" (DOS) will be needed to implement the disk controller capabilities. The disk operating system program will be located in the control module and will utilize the control processor. The DOS will provide the disk controller with the information necessary to accomplish direct memory access and request from the controller disk status information.

The disk operating system will execute various functions for the control program. Besides performing read and write instructions, the DOS will create new files, delete old files, close and open files, and monitor the disk operation. The following is a brief description of the disk operating system and the procedures provided by the DOS for the control program.

The disk operating system provides a software interface between the control program and the physical disk system. The DOS performs all logical to physical conversions of record location and passes the calculated track and sector data to the disk controller. The operating system monitors the disk controller and the disk, forwarding any malfunction signals to the control program for display to the operator. The disk operating system maintains a directory of disk files for use in space allocation. The directory contains the file name and an index to the disk blocks assigned to the file. Disk space, assigned in 1024 byte blocks, is allocated to a file during write operations. The directory allows for a total of 64 separate files and has a total of 248 blocks available for assignment on each of the two disks. Disk space is dynamically assigned using an

allocation vector to account for each disk block. An entry, similar to a disk directory entry, must be available in the control program whenever file operations are to be executed. The following is a list of disk operating system routines which can be invoked by the control program.

1. Reset

The "reset" command will force the DOS to interrupt the disk controller and reset all disk control parameters. The disk space allocation vector will be cleared, and a directory search will be made to determine which blocks are currently assigned.

2. Open

A call to the DOS routine "open" will initialize various file parameters in the control program directory entry. If the file is located in the directory, the disk space allocation is returned to the control program. If the file cannot be found, an error report will be generated.

3. Close

"Close" will result in an update of the disk directory. Prior to this directory update, the space allocated to the file is only temporary. In other words, any blocks added to a file since the last call to close would be lost during a disk reset.

4. Make

The "make" command creates a new file entry, initializes the space allocation descriptor for the file to "empty", and opens the file.

5. Delete

When the DOS is called via "delete", any disk space in use by the specified file is de-allocated and the directory entry is erased from the disk.

6. Read

A "read" call from the control program to the disk operating system results in the following DOS actions: ensures the file is open and the requested record exists; passes the memory load address to the DMA; passes the track, sector, and disk number to the disk controller; returns to the control program a successful or unsuccessful read attempt indicator.

7. Write

"Write" will cause the DOS to transfer, via DMA, one record from the control module memory to the disk. If the file requires additional disk space for the record, it will be allocated. The DOS returns to the control program status information if the write can be accomplished or whether a disk/file overflow exists.

VI. DISPLAY MODULE

A. GENERAL INPUT/OUTPUT CONSIDERATIONS

Interaction with the keyboard and display units will be asynchronous, that is, not time dependent upon the control module. The control module processor will not stop during input or output operations, waiting for a function in the communication module or display module to be completed. The input/output operations will continue independently in time, setting flags for system control upon completion of the input/output operation.

Input and output buffering for telecommunications is accomplished in the main computer on many systems, but the proposed system as described in this paper will have separate telecommunications buffering in a communications module. The proposed system will use a separate processor to handle all the communication line interfacing. Communication to and from the control module will be via an eight-bit parallel data bus and for each communications line two 128 character data buffers will be allocated in the main memory. The communications processor will pass edited input messages to the control section upon demand. Input to the control section from the keyboard will be handled in the same manner as any other input from a communications device.

B. DISPLAY TERMINAL

The information system terminal display must serve as the interface between the human operator and the communications network. Therefore, terminal choice will depend upon the needs of the user, the technical requirements of the communciations lines, and the general requirements of the Naval Weather Service.

In meeting the various needs of the NEDS user, Fleet Numerical Weather Central can choose from a vast array of terminals. It is possible to simplify the selection process by first categorizing the terminals by application and functional characteristics. The characteristics can be classified along two main lines.

1. Intelligent vs. hardwired terminals

2. Alphanumeric vs. graphic terminals

1. Intelligent Terminals

At the base of every intelligent terminal system is a stored program processor, giving these terminals a wide range of capabilities. Because of the internal programming flexibility, an intelligent terminal can "look like" other equipment to the central processor, and fit into an existing communications network with minimum re-programming of the control system. An intelligent terminal will be able to edit, format, compress, and manipulate data while providing flexibility in system implementation.

Format control, editing, and operator queuing are important advantages of the intelligent terminal. In the formatted mode, the terminal can check for different types of errors and prevent the entering of erroneous data. This includes parity and illegal entry checks. When the terminal displays a form, the cursor automatically skips over the format or protected data, so that it cannot be destroyed. The terminal can be programmed to accept only numeric or alphabetic characters in a given field. This is particularly applicable to a NWSED unit when formatting an Automatic Response to Query transmission.

Programmed function keys can allow the operator such controls as display, rotate, and zoom. This will minimize the number of required input keystrokes and thus reduce the number of input errors.

Check digit verification can be implemented, and field lengths checked, ensuring that the operator inputs the proper number of characters. Selected fields can be controlled to allow only specifically identified character sequences to be entered, such as station identification. The terminal can also force entry to specified fields. If the selected field is not filled, the operator will be unable to proceed.

For editing, use can be made of the terminal memory. Modification or correction of the data prior to a message transmission can be accomplished at the terminal. In the case of minor mistakes during message composition, use of the back-space key will permit repositioning of both the printing head and the memory pointer, thus allowing erasure and correction of the data input. Editing also allows the operator to address a specific field if a correction is needed, eliminating re-keying the entire data form. In addition, selected records may be automatically inserted

into the appropriate field.

The memory of a CRT terminal is large enough to store one "page", or screen, of data. The display must be refreshed, requiring the use of the display processor, 30-60 times per second.

Considering the specifications set forth by the Fleet Numerical Weather Central (see reference 6) and the desire to make the NEDS as flexible as possible, it was determined that an intelligent terminal should be used. This terminal would control two CRT displays, one for alphanumerics and the other for graphics.

The reasons for utilizing an intelligent terminal system instead of a hardwired terminal are:

1) Fast response: Some data and data entry formats can be stored locally thus avoiding the constant accessing of the control module.

2) Compatibility with different central processors: If multiple access to two different processor types is ever required, the terminal will not have to be re-wired to handle the change.

3) Multiple tasks: The terminal will be a data entry unit and an inquiry-response unit.

4) Editing: Data editing can be handled without the use of the control module.

5) Local processing: The intelligent terminal can process small amounts of data, instead of forwarding all information directly to the communications module.

6) Data reduction: Since communications costs are a significant part of any system, decreasing the amount of data sent reduces the total transfer costs. Data can be verified and edited before transmission, thus eliminating erroneous data at the source and freeing the control module from execution of these tasks.

2. The Display

All cathode ray tube (CRT) displays contain certain operating elements. The cathode ray tube, the display control hardware and the memory are common to all such devices. The component packaging varies, hence, several hardware configurations are available. In some systems, all controls and memory may be packaged in a separate controller, allowing operation of more than one display screen.

Besides the cathode ray tube, the basic CRT terminal includes a keyboard for data input, an internal memory, a character generator, and a communications interface. There are two types of displays available. At the low end of the price scale is the alphanumeric CRT, displaying only alphanumeric and special characters, similar to a standard typewriter. At the high end of the price scale is the graphic CRT display, which presents line drawings and schematics in addition to the alphanumeric information. Since this display is complicated, it includes a processor, memory, and the associated software required for system operation.

A system with a separate controller would allow greater flexibility and permit growth and expansion. The control electronics provide the necessary sequence, timing,

and operational signals the display unit requires in order to perform its functions. In conjunction with the interface electronics, the control electronics provide the needed addressing circuits, allowing each display to be individually served by a single processor.

It is desirable to have two display systems in the NEDS unit.

1) A graphics display terminal would perform the functions necessary to unpack and display weather maps.

2) An alphanumeric display terminal would be utilized to display messages composed of alphanumeric characters, and would serve as a backup to the graphic terminal.

C. DISPLAY INTERFACE

1. The Keyboard

All keyboard entries are edited by the terminal. After all corrections have been made for each data or command input, the information is sent to the communications module. The communications module handles the keyboard data in the same manner as a communications line. The data is sent from the terminal bit-serially to the communications section.

2. The Alphanumerics Display

Data will be sent serially from the control module

to the alphanumeric display. A maximum of one record (128 bytes) will be sent prior to resuming system polling. This ensures that the control module will be available to process input from the communications section.

3. The Graphics Display

Data is sent from the disk system to a shared memory buffer area when graphic data is being output for display. When four records are available in the shared memory area, control signals are sent from the control module to the graphics display controller. The graphics controller then unpacks the data and displays it on the graphics terminal. Approximately twenty-five records are needed for a complete graphics display. The control module directs the disk operating system to send all the data required for the complete display, including necessary background information.

VII. SUMMARY AND CONCLUSIONS

With the increasing development of large-scale integrated (LSI) circuits the entire spectrum of computer technology is experiencing a rejuvenation of its basic concepts. That is, there now exist single-chip processors that increase the scope of microcomputers such that they now perform applications formerly exclusive to medium and large scale computers. Complementing the development of these new microprocessors have been the advances in semiconductor memory and logic. All these advances have lent themselves to the emergence of the microprocessor-based systems, at a fraction of the cost of comparable large scale computing equipment. The microprocessing system can also be developed or modified in a significantly shorter period of time than that of hard-wired logic.

The functional design of the microprocessor-based system proposed is not intended to be, necessarily, the optimal systems design for this communications network application. It is presented only to demonstrate that a microprocessor-based system is feasible and should be considered when designing similar future systems that have dedicated functions and/or preestablished tasks. The basic ideas and the methods described here are totally feasible concepts and are well within the state-of-the-art technology.

The control software provided in Appendix A is a developmental program. As noted in section 3, not all of the control functions are implemented in the program.

Additional features, which will not be part of the normal control routines, are included in the program for software testing and evaluation.

The program currently utilizes four system states. Each communications line can be either in the supervisor, command, weather, or general mode. The supervisor state initializes the system and allows interchange between the other three states. The command state implements the various operator functions which were described earlier in this text. The weather state simulates an input line from the communications module where the data line is in the receive or message state. The general mode is a system test mode where the machine aids in operator simulation of a bulletin input. In actual implementation, there will be only two system states, command or receive.

The DMA feature is not utilized in this program. There are no interrupt routines from the disk system. Instead the control processor idles until the disk function completes. The second disk is not presently utilized, and the software to provide graphic backgrounds is not available. There is no provision for graphic output and the current program does not utilize or take advantage of any of the features available from an intelligent terminal system. All output is sent directly to a single alphanumeric CRT display.

The disk operating system currently under development is for a general purpose file system. For the use envisioned, some modifications could be made to improve the disk system performance. Besides the changes mentioned earlier, the disk system should also assign consecutive data blocks to a file when it is first opened. This would aid in minimizing seek time during storage and display.

The NEDS will have the ability to detect and recover any

data lost due to slow system response to a communications line. The software required to ensure that no data would be lost would be prohibitively large and complicated. Thus the design assumes that data will be occasionally lost. If the disk just started a seek operation high priority line when a high speed, begins reception, the communications buffering will probably overflow before the disk system will be able to respond to storage needs. This would be difficult to prevent and as noted above really of little consequence since the data can be recovered.

A functional design has been presented which describes the use of a low cost microprocessor system and the associated peripherals in a performance oriented application. This approach to the control of data communications has tried to demonstrate the significant savings over conventional approaches and could well be applied to other areas. Potential applications for similar systems exist wherever storage and retrieval of information is required at a local user level.

Table 1

Estimated Information Required at a Typical NWSED

Alphanumeric Information	
Hourly weather reports	
12 hours collection, 3 bulletins each hour 20 teletype lines, 1000 characters per bulletin	36,000 bytes
Forecasts	
20 airport forecasts 5 teletype lines each report	7,000
5 severe weather warnings 5 teletype lines each report	1,500
10 upper wind forecasts 5 teletype lines each report	1,500
5 route wind forecasts 10 teletype lines each report	1,500
Total Alphanumeric Data	47,500 bytes
Graphic Information	
40 maps and graphs each map averages 25,000 bits	125,000 bytes
Total Capacity Required -	172,500 bytes

Table 2

Proposed File Structure

Alphanumeric Information

Hourly Weather Reports

12 files, one for each of the past 12 hours
Each file contains reports segregated by bulletin
Individual reports are locatable by station identifier

Forecasts

4 files, one for each forecast type
Individual reports are locatable by time and location

Graphic Information

Maps and graphs

40 files, one for each graphic

Miscellaneous Information

Interface

X files, one for each communication line
1 file, for operator keyboard

NOTE: Second disk use:

Transfer of data message for ARQ
Back-up
Background charts
System and file expansion
Copy functions

Table 3

Response Time Estimates

Alphanumeric Information (transfer without interruptions)		
Seek data (average)		
Seek directory	.65	
Seek data	.65	
		1.30
Read (transfer) time		.20
Formatting		.01
Transfer to display module		.01
Time required by display module		.10
Total		1.62 secs
Graphic Information (transfer without interruptions)		
Seek to map		
Seek directory	.65	
Seek data (4 blocks)	2.60	
		3.25
Seek background		
Seek directory	.65	
Seek data	.65	
2 track moves	.01	
		1.31
Transfer of geographical background to graphic module		.20
Transfer of map to graphic module		.20
Estimated software overhead (return to polling routine)		1.00
Time required by display module		.50
Total		7.46 secs

NOTE: If a priority communications line interrupts the display, the interrupt period must be added to arrive at the total response time.

Table 4

Disk System Characteristics

Physical Organization

- 2 disks
- Each disk: 77 tracks
- Each track: 26 sectors
- Each sector: 128 bytes
- Total capacity per disk: 256K bytes

Logical Organization

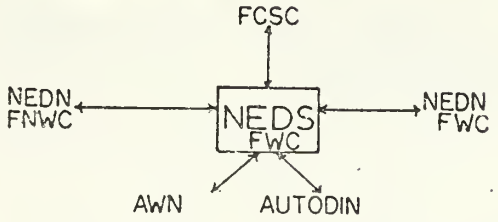
- 64 files
- Each file, maximum 128 records (16K bytes)
- Each record 128 bytes
- Space allocable to files in blocks of 1024 bytes

Timing

- Hardware step rate per track: 10 milliseconds
- Software step rate per track: 20 milliseconds
- Head settling time: 40 milliseconds
- Average latency at 360 rpm: 83 milliseconds
- Data (information and timing bits) transfer at 360 rpm: 250 KHZ
- Information transfer at 360 rpm: 20K bytes per second
- Average data access: 643 milliseconds

NEDS-1

COMMUNICATIONS

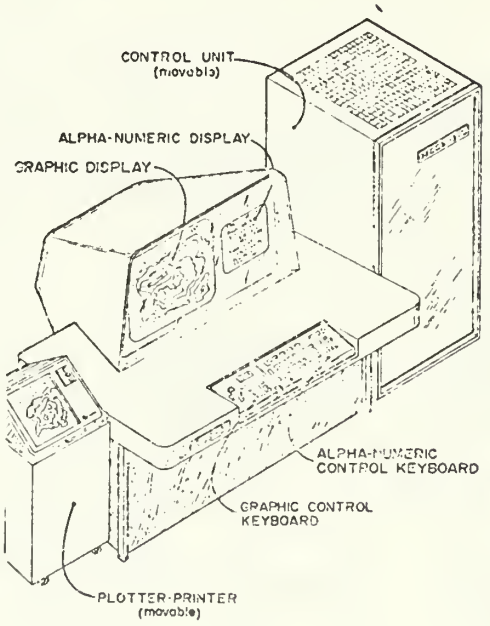


GRAPHICS

19" COLOR
OVERLAY RED, GREEN, BLUE
ZOOM
SCROLL
RAPID DISPLAY
SEQUENCING

STORAGE

80 MILLION BITS –
GRAPHICS
ALPHANUMERICS
GEOGRAPHY
OPERATING SYSTEM
STORE AND FORWARD



ALPHANUMERICS

DISPLAY MESSAGES
DRAFT MESSAGES
MASSAGE MESSAGES
PRODUCT INVENTORY
DISPLAY STATUS

ADD-ON

FOR TIME-SHARING -

MAG-TAPE CASSETTE

DISCIPLINES

COMM. - (RS 232)
NEDN
AWN
AUTODIN

OPERATING SYSTEM - (NOS)
RDOS COMPATIBLE

Figure 1

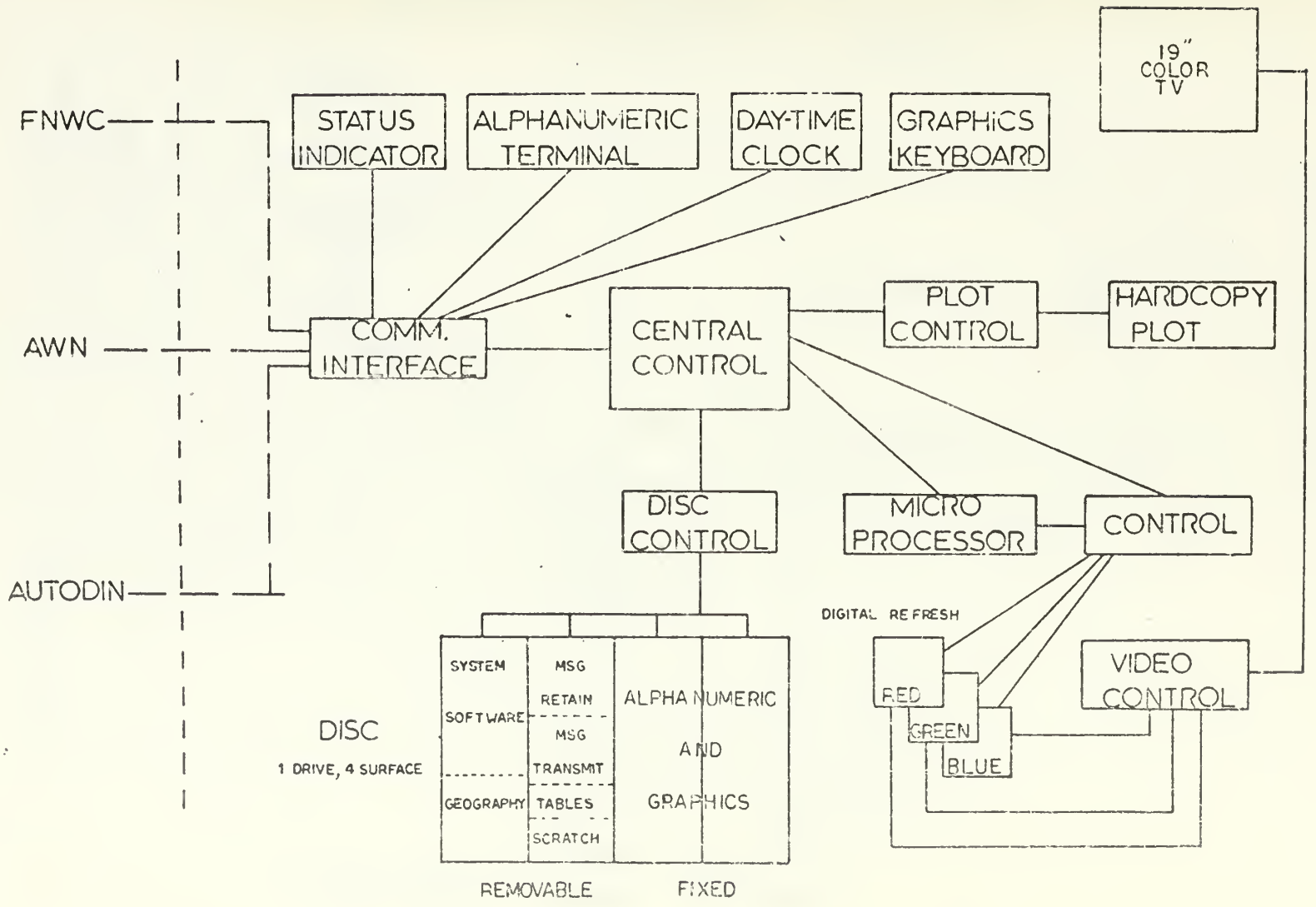
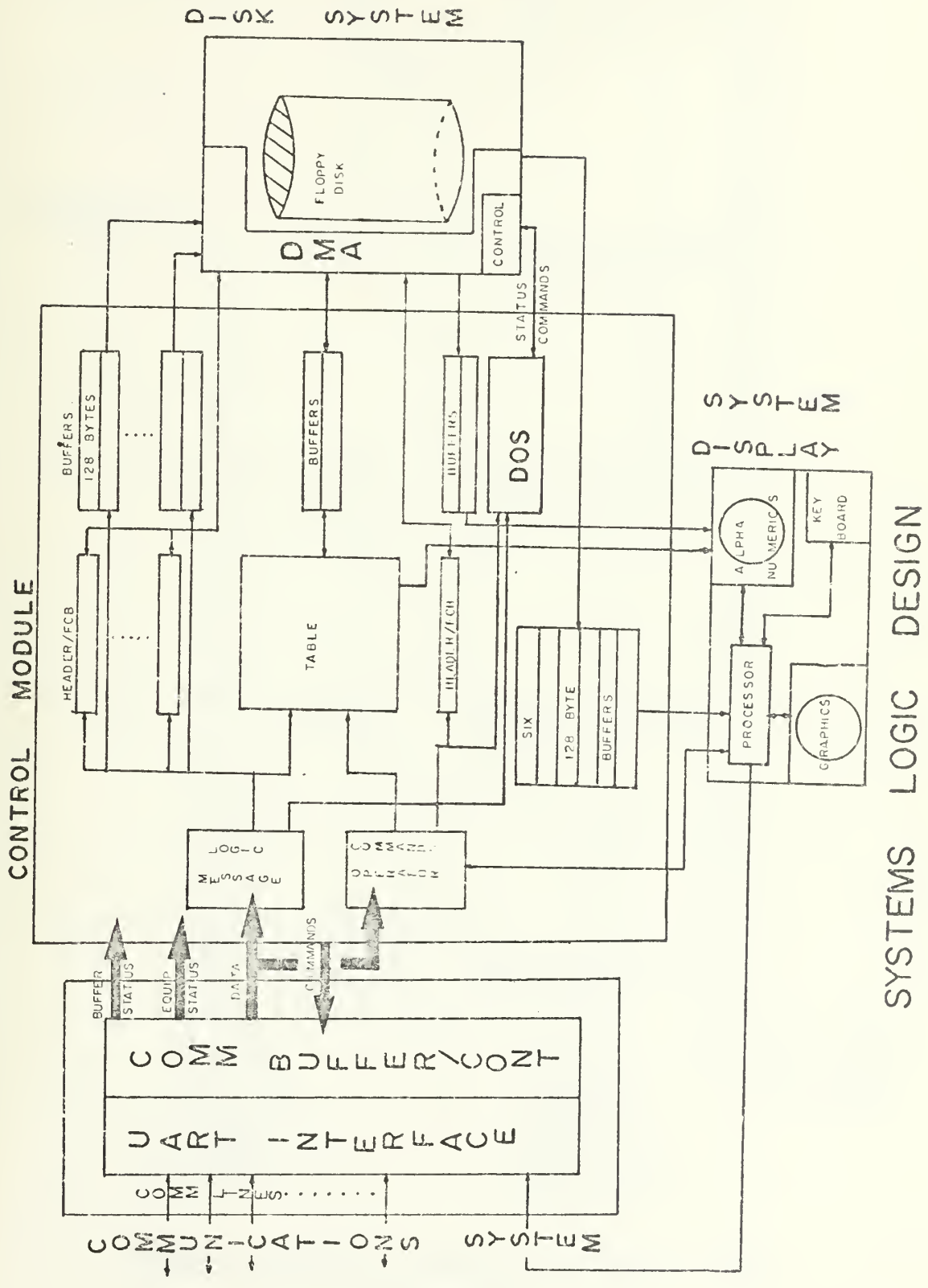

FNWC
AWN
AUTODIN
STATUS INDICATOR
ALPHANUMERIC TERMINAL
DAY-TIME CLOCK
GRAPHICS KEYBOARD
19" COLOR TV
COMM. INTERFACE
CENTRAL CONTROL
PLOT CONTROL
HARDCOPY PLOT
DISC CONTROL
MICRO PROCESSOR
CONTROL
DIGITAL REFRESH
RED
GREEN
BLUE
VIDEO CONTROL
DISC
1 DRIVE, 4 SURFACE
SYSTEM SOFTWARE
GEOGRAPHY
MSG RETAIN
MSG TRANSMIT
TABLES
SCRATCH
ALPHA NUMERIC AND GRAPHICS
REMOVABLE
FIXED

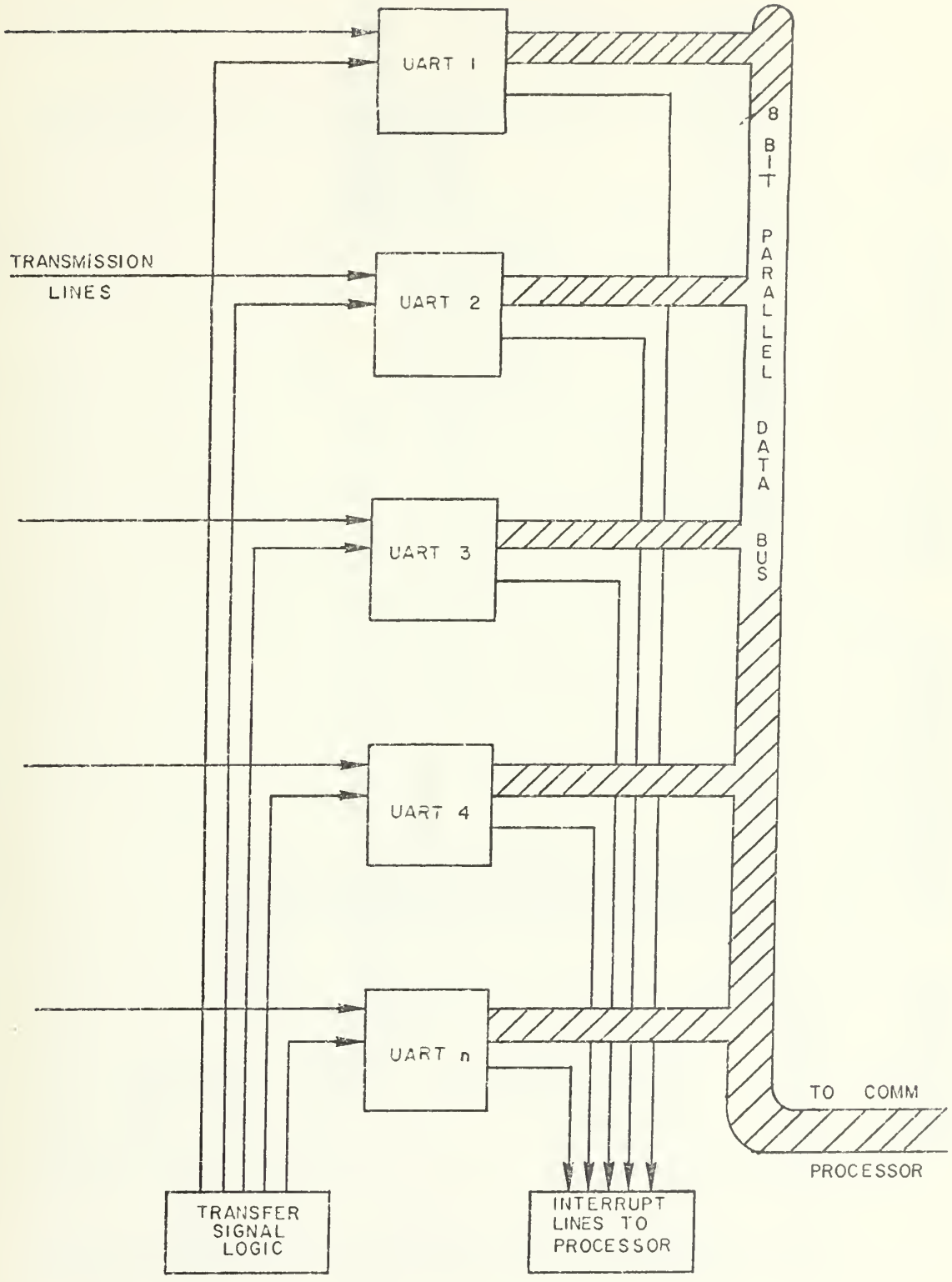
Figure 2

Figure 3



SYSTEMS LOGIC DESIGN

Figure 4



COMMUNICATION MODULE DATA LINE INTERFACE

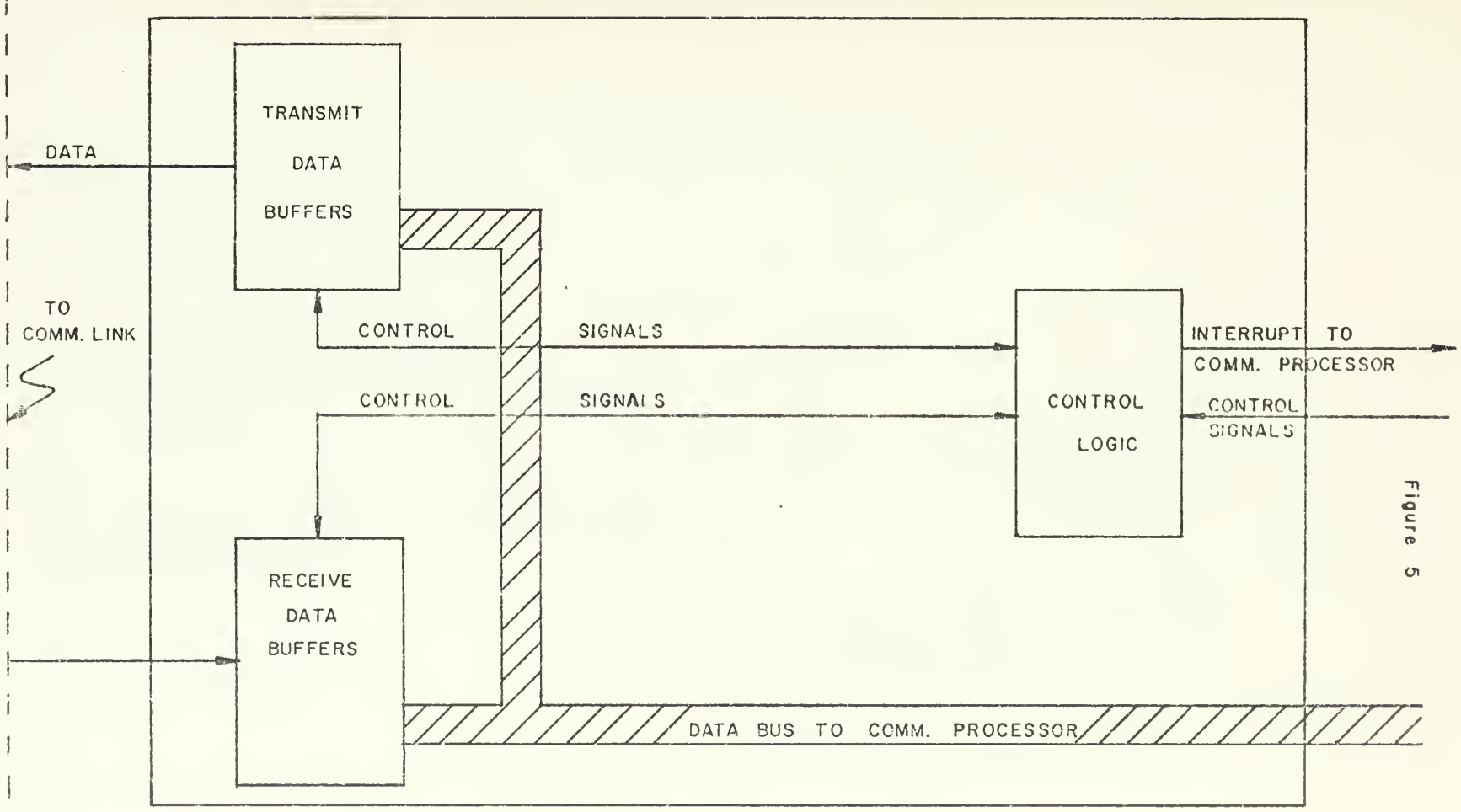


Figure 5

UNIVERSAL ASYNCHRONOUS RECEIVER-TRANSMITTER

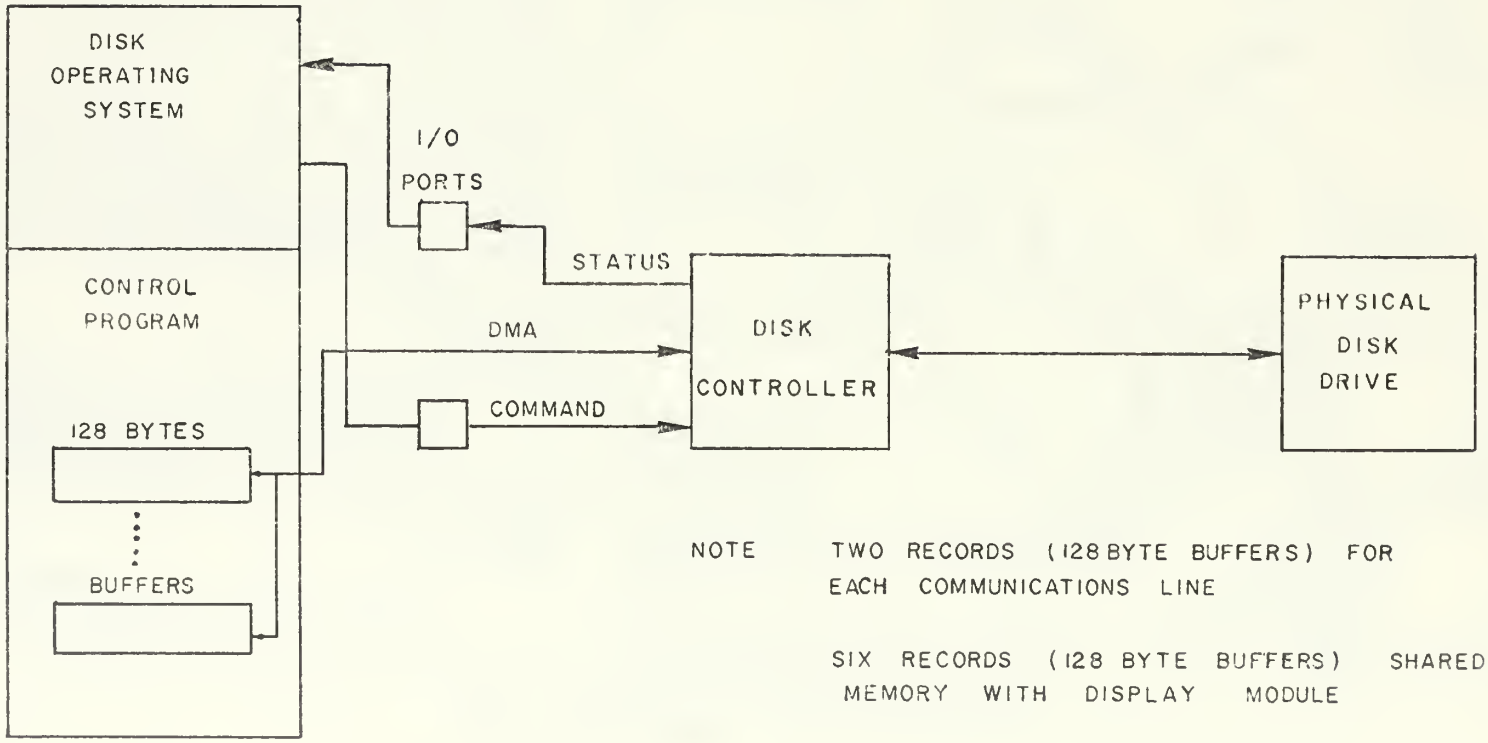
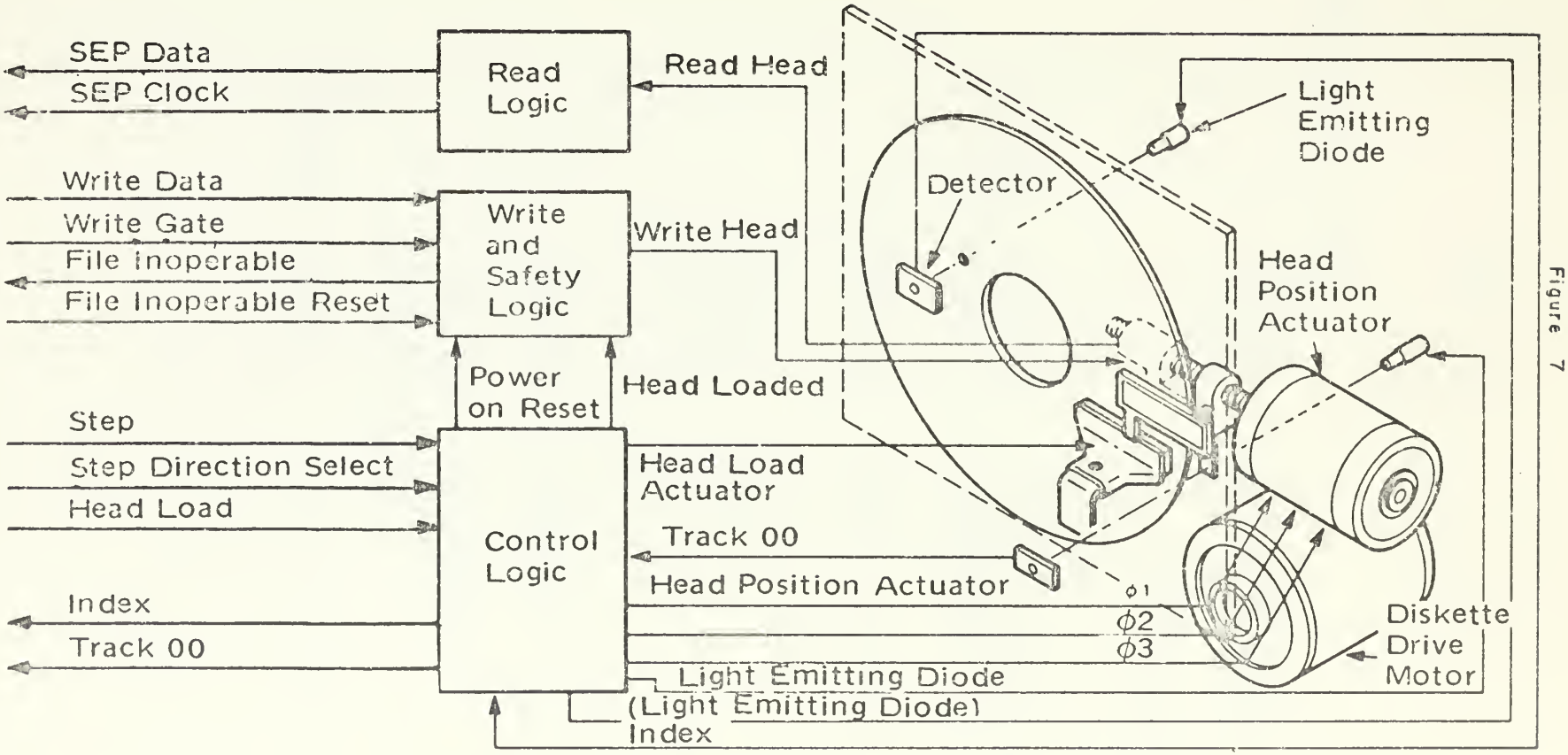


Figure 6

FLOPPY DISK INTERFACE ORGANIZATION

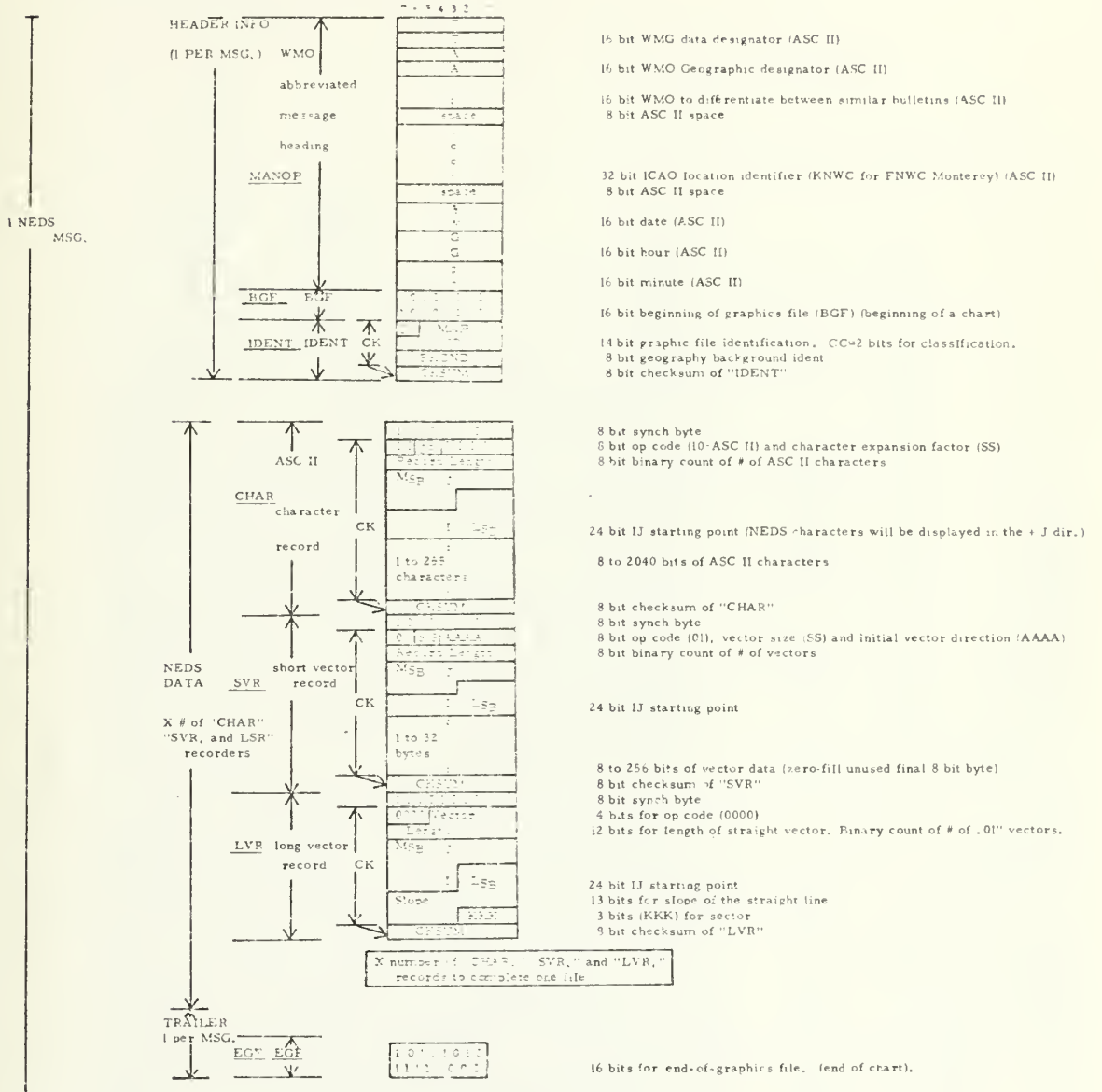


SHUGART SA900 FLOPPY DISK

Figure 7

Figure 8

NEDS DATA FORMAT



```
/*  NEDS CONTROL MODULE SOFTWARE  */

/*  THIS PROGRAM PROVIDES THE PROTOTYPE SOFTWARE
    FOR THE CONTROL MODULE OF A MICROPROCESSOR-BASED
    COMMUNICATIONS INFORMATION SYSTEM.  THE PROGRAM
    USES INDIRECT ADDRESSING TO ACHIEVE THE MULTIPROGRAMMING
    CAPABILITY NECESSARY TO EFFICIENTLY HANDLE THE MULTIPLE
    INPUT LINES.  THE MAIN TABLE (MAIN$TBL)
    IS THE BASE FOR THE LOGICAL STRUCTURE.  ALL INPUT
    AND OUTPUT MUST REFERENCE THE MAIN TABLE TO
    DETERMINE DATA STORAGE/RETRIEVAL LOCATIONS.  THERE
    ARE THREE DATA LINES SIMULATED IN THIS PROGRAM (ONE FROM
    THE KEYBOARD, ONE FROM A PAPER TAPE READER, AND ONE VIA
    A SOFTWARE PROGRAM).  EACH DATA LINE IS CAPABLE
    OF BEING IN ONE OF FOUR SYSTEM STATES (SUPERVISOR,
    'G'-GENERAL AIDED INPUT, 'W'-COMMUNICATIONS RECEIVE,
    'C'-COMMAND).  THE SUPERVISOR STATE IS USED FOR SYSTEM
    INITIALIZATION AND TRANSITION FROM STATE TO STATE.
    THE 'G' STATE IS A SYSTEM AIDED COMMUNICATIONS INPUT STATE.
    'W' MODE SIMULATES A COMMUNICATIONS LINE IN THE
    RECEIVE STATE AND ALL HEADINGS, END OF BULLETINS,
    ETC. MUST BE SUPPLIED BY THE INPUT LINE.  THE 'C' OR
    COMMAND STATE, AN INTERACTIVE MODE, ALLOWS THE
    EXECUTION OF VARIOUS OPERATOR DISPLAY ACTIONS.  THE
    PROGRAM USES A SERIES OF LIBRARY ROUTINES FOR
    INPUT/OUTPUT CONTROL AND DISK INTERFACING.  THE PROGRAM
    IS WRITTEN IN THE PL/M PROGRAMMING LANGUAGE AND THIS
    ROUTINE CAN BE IMPLEMENTED USING THE INTEL 3080
    MICROPROCESSOR  */

256:

  /*  START PROGRAM PAST PRESENT DISK PARAMETER AREA  */

DECLARE
  BEGIN$PROGRAM LABEL,  /*  LABEL TO JUMP TO PROGRAM START  */

  TRUE LITERALLY '1',
  FALSE LITERALLY '0',
  FOREVER LITERALLY 'WHILE TRUE',

  BUF$LEN LITERALLY '129',  /* LENGTH OF BUFFERS PLUS CONTROL BYTE  */
  BUF$LEN$M LITERALLY '128',  /*  BUF$LEN-1  */
```

```
FCB$LEN LITERALLY '33',  /*  NUMBER OF BYTES IN FCB  */
FCB$LEN$M LITERALLY '32',  /*  FCB$LEN-1  */
HDG$VEC$LEN LITERALLY '11',
DTG$LEN LITERALLY '10',

MAIN$TBL$MAX$AND$END LITERALLY '768',
MAIN$TBL$TOP LITERALLY '60',  /*  LOWER FOR ADDITIONAL COMM LINES  */
MAIN$TBL$END LITERALLY '64',
MAIN$TBL$MAX LITERALLY '704',  /*  64*11  */
MAIN$TBL$LEN LITERALLY '11',  /* MAIN$TBL HOLDS TITLE$TBL INDEX,
                                 DTG, CLASSIFICATION, BACKGROUND
                                 INDEX, CK$SUM ERRORS, AND ROTATION
                                 OR DISK RECORD POINTER */

TAI$TBL$END LITERALLY '9',
TAI$TBL$MAX LITERALLY '72',
TAI$TBL$LEN LITERALLY '8',  /*  TAI$TBL HOLDS REQUIRED BULLETIN
                               TITLES, TITLE$TBL INDEX, AND
                               PTR$TBL BASE */

STA$TBL$END LITERALLY '6',
STA$TBL$MAX LITERALLY '36',
STA$TBL$LEN LITERALLY '6',  /* STA$TBL HOLDS REQ MESSAGE TITLES */

TITLE$TBL$LEN LITERALLY '21',
TITLE$TBL$MAX LITERALLY '105',
TITLE$TBL$END$M LITERALLY '4',
TITLE$TBL$END LITERALLY '5',  /*  TITLE$TBL HOLDS PLAIN TITLE
                                  ADDRESS POINTERS  */

BLANK LITERALLY '20H',  /* SPACE */
CONSOLE$LINE LITERALLY '3',  /*  CHANGE IF MORE COMM LINES  */
TERM$MARK LITERALLY '1BH',  /*  THE ESCAPE KEY  */
SOB$MARK LITERALLY '3CH',  /*  START OF BULLETIN  <  */
STX$MARK LITERALLY '26H',  /*  START OF TEXT  &  */
EOB$MARK LITERALLY '3EH',  /*  END OF BULLETIN  >  */
CR LITERALLY '0DH',  /*  CARRIAGE RETURN  */
LF LITERALLY '0AH',  /*  LINE FEED  */
BACK$SPACE LITERALLY '08H',

CLEAN$M$TBL DATA (TITLE$TBL$END$M,'00000000U',0),
    /*  USED TO CLEAR THE MAIN TABLE  */

TAI$TBL DATA ('ROBERT',1,0,
              'TTAATI',1,0,
              'WARNIT',13,1,
              'BOBBIE',15,2,
              'NEWONE',30,2,
```

```
                'BILLYR',40,2,
                '000001',59,TITLE$TBL$END,
                'CMDMSG',59,3,
                OFFH,0,0,0,0,0,MAIN$TBL$TOP,TITLE$TBL$END$M),

STA$TBL DATA (0,0,0,0,0,'$31938$NPG  $RHE  $98765$BOB  $'),

TITLE$TBL DATA ('HOURLY REPORTS      $',
                'WARNING REPORTS     $',
                'THE THIRD TYPE      $',
                'COMMAND MESSAGE     $',
                'EMPTY FILE          $'),

/*  OUT** ARE CONTROL PROGRAM OUTPUTS TO THE OPERATOR  */
OUT01 DATA (CR,LF,'ENTER STATION   (3/A  OR  5/N)  $'),
OUT02 DATA (CR,LF,LF,'ENTER  G, A, OR C.  $'),
OUT03 DATA (CR,LF,LF,'SELECT MACHINE STATE.$'),
OUT04 DATA (CR,LF,LF,'COMMAND MODE:',CR,LF,'R--RESET; T--TABLE; ',
            'D--DISPLAY;',CR,LF,'F--FIND; A--ARQ; E--ERASE;',
            CR,LF,'C--CMDS;          ENTER CMD.  $'),
OUT05 DATA ('  $'),
OUT06 DATA (CR,LF,LF,'MSG NOT ON FILE',CR,LF,LF,'$'),
OUT07 DATA (CR,LF,LF,LF,LF,'     MAIN  TABLE',CR,LF,'$'),
OUT08 DATA (CR,LF,LF,'CALLED FILE IS EMPTY.$'),
OUT09 DATA (CR,LF,LF,'FILE INDEX > TBL END.   $'),
OUT10 DATA (CR,LF,'ENTER FILE INDEX  (TWO DIGITS).  $'),
OUT11 DATA (CR,LF,'ENTER TBL ENTRIES   (6 DTG,  3 A/N)',CR,LF,'$'),
OUT12 DATA (CR,LF,LF,'BULLETIN TYPE NOT FILED.$'),
OUT13 DATA (CR,LF,'INPUT MODE:',CR,LF,
            'ENTER BULLETIN TYPE  (6 A/N)',CR,LF,'$'),
OUT14 DATA (CR,LF,LF,'STATION CALLED IS NOT FILED.',CR,LF,LF,'$'),
OUT15 DATA (CR,LF,'TAI-TBL IMPROPERLY ORDERED.$'),
OUT16 DATA (CR,LF,'NEDS-WRITE ERR   DOS RETURNED $'),
OUT17 DATA ('  INVALID ENTRY.  $'),
OUT18 DATA (CR,LF,LF,'IS THIS A NEW DISK   (Y N)  $'),
OUT19 DATA (CR,LF,'ENTER DATA AND EOB-MARK.',CR,LF,'$'),
OUT20 DATA (CR,LF,LF,'DELETE FILE $'),
OUT21 DATA ('   (Y N)  $'),
OUT22 DATA (CR,LF,'ERASE ERROR',CR,LF,'$'),
OUT23 DATA (CR,LF,'WEATHER MODE:  COM LINE IS ON.',CR,LF,
            'ENTER SOB-MARK, WEATHER BULLETIN, EOB-MARK.',CR,LF,
            'ENTER "<000001" TO RETURN TO SUPERVISOR',CR,LF,'$'),
OUT24 DATA (CR,LF,LF,'ENTER P TO PAGE.  $'),
OUT25 DATA (CR,LF,LF,LF,'TABLE INPUT ERROR.',CR,LF,LF,LF,'$'),
OUT26 DATA (CR,LF,'DISK WILL BE CLEARED.',CR,LF,
            'IS THIS OK  (Y N)  $'),
OUT27 DATA (CR,LF,LF,'BUF-WAIT ERROR  $'),
OUT28 DATA (CR,LF,'OUTPUT CHART TO GRAPHICS DISPLAY',CR,LF,'$'),
```

```
OUT29 DATA (CR,LF,LF,'ENTER CMD  $'),

HS$CHAR BYTE,  /*  USED TO SIMULATE A HIGH SPEED DATA LINE  */
HS$CTR ADDRESS,
HS$VEC DATA ('WW<BILLYRFNWC11223344CH'),

            /*  PTR$TBL IS THE FIRST COLUMN OF THE MAIN TABLE.
                MUST BE DECLARED TOGETHER IN THIS WAY.  */
PTR$TBL (MAIN$TBL$END) BYTE,  /* CONTAINS POINTERS TO MAIN$TBL  */
MAIN$TBL (MAIN$TBL$MAX) BYTE,  /* HOLDS TITLE$TBL INDEX, DTG, ETC. */

SA$LEVEL LITERALLY '13',  /*  DIVIDES MAIN TABLE INTO DATA AREAS  */
GR$LEVEL LITERALLY '30',

NO$DATA$IN BYTE,  /*  WHEN TRUE COM$LINE BUFFER HAS NO DATA  */
                  /*  WOULD ACTUALLY BE A PROCEDURE
                      TEST A BYTE FOR NO DATA INDICATION
                      THEN TEST FLAG TO CONFIRM  */
LINE$HAS$DATA BYTE,  /*  INPORT PORT FROM UARTS  */
LINE$NUM BYTE,  /*  CURRENT COMM LINE IN PROCESSING  */
PTR$INDEX BYTE,  /*  CURRENT PTR$TBL LEVEL USED FOR FILE CREATION  */
FLICK$ON BYTE,  /*  IF LINE$NUM HAS BEEN OUTPUT, SET TRUE  */

                /*  BUFFERS FOR CURRENT DOS  */
(DBUF$B,DFCB$B) ADDRESS,
(DBUF BASED DBUF$B,DFCB BASED DFCB$B) BYTE,

/*  BUFFERS FOR 3 COMM LINES  */
MODE1 BYTE,
MODE1$STATE BYTE,
LOW$PTR1 BYTE,
HIGH$PTR1 BYTE,
CK$SUM1 BYTE,
HDG$VEC1 (HDG$VEC$LEN) BYTE,
DTG1(DTG$LEN) BYTE,
FCB1 (FCB$LEN) BYTE,
ABUF1 (BUF$LEN) BYTE,
BBUF1 (BUF$LEN) BYTE,

MODE2 BYTE,
MODE2$STATE BYTE,
LOW$PTR2 BYTE,
HIGH$PTR2 BYTE,
CK$SUM2 BYTE,
HDG$VEC2 (HDG$VEC$LEN) BYTE,
DTG2 (DTG$LEN) BYTE,
FCB2 (FCB$LEN) BYTE,
ABUF2 (BUF$LEN) BYTE,
```

```
    BBUF2 (BUF$LEN) BYTE,

    MODE3 BYTE,
    MODE3$STATE BYTE,
    LOW$PTR3 BYTE,
    HIGH$PTR3 BYTE,
    CK$SUM3 BYTE,
    HDG$VEC3 (HDG$VEC$LEN) BYTE,
    DTG3 (DTG$LEN) BYTE,
    FCB3 (FCB$LEN) BYTE,
    ABUF3 (BUF$LEN) BYTE,
    BBUF3 (BUF$LEN) BYTE,

    /*  FOLLOWING DECLARATIONS FOR INDIRECT ADDRESSING
        TO IMPLEMENT MULTIPROGRAMMING  */
    (MODE$B,MODE$STATE$B,LOW$PTR$B,HIGH$PTR$B) ADDRESS,
    (DTG$B,CK$SUM$B) ADDRESS,
    (HDG$VEC$B,FCB$B,ABUF$B,BBUF$B) ADDRESS,
    (MODE BASED MODE$B,MODE$STATE BASED MODE$STATE$B) BYTE,
    (LOW$PTR BASED LOW$PTR$B,HIGH$PTR BASED HIGH$PTR$B) BYTE,
    (DTG BASED DTG$B,CK$SUM BASED CK$SUM$B) BYTE,
    (HDG$VEC BASED HDG$VEC$B,FCB BASED FCB$B) BYTE,
    (ABUF BASED ABUF$B,BBUF BASED BBUF$B) BYTE;


RI: PROCEDURE BYTE;
    /*  READ A CHARACTER FROM PAPER TAPE  */
    DECLARE
      (I,CHAR) BYTE,
      HSI LITERALLY '3',
      HSS LITERALLY '2',
      HSC LITERALLY '2',
      CLK0 LITERALLY 'OUTPUT(HSC)=0F3H',
      CLK1 LITERALLY 'OUTPUT(HSC)=0F9H';
    DO WHILE (ROR(INPUT(HSS),1) AND 11B) <> 0;
      END;
    /*  AVOID SETTING TIMER TOO QUICKLY  */
    CALL TIME(2);
    CLK1;
    CLK0;
    RETURN (NOT INPUT(HSI) AND 07FH);
    END RI;

/*  *****     LIBRARY PROCEDURES FOR SYSTEM ROUTINES     *****  */

DECLARE BDOS LITERALLY '3FFDH';
DECLARE DCNT BYTE;
```

```
MON1: PROCEDURE (F,A);
  /*  MONITOR, INTERFACE TO LIBRARY  */
  DECLARE
    F BYTE,
    A ADDRESS;
  GO TO BDOS;
  END MON1;

MON2: PROCEDURE (F,A) BYTE;
  /*  MONITOR, INTERFACE TO LIBRARY  */
  DECLARE
    F BYTE,
    A ADDRESS;
  GO TO BDOS;
  END MON2;

CSTS: PROCEDURE BYTE;
  /*  CONSOLE STATUS, SYSTEM ROUTINE  */
  RETURN NOT INPUT(1);
  END CSTS;

CI: PROCEDURE BYTE;
  /*  CONSOLE INPUT, SYSTEM ROUTINE  */
  RETURN MON2(1,0);
  END CI;

CO: PROCEDURE (CHAR);
  /*  CONSOLE OUTPUT, SYSTEM ROUTINE  */
  DECLARE CHAR BYTE;
  CALL MON1(2,CHAR);
  END CO;

PRINT: PROCEDURE (VEC$ADR);
  /* OUTPUT DATA FROM ADDRESS VEC$ADR UNTIL '$' IF CONSOLE$LINE  */
  DECLARE VEC$ADR ADDRESS;
  IF LINE$NUM <> CONSOLE$LINE THEN RETURN;
  CALL MON1(9,VEC$ADR);
  END PRINT;

SELECT: PROCEDURE (D);
  /*  DOS PROCEDURE  */
  /*  SELECT DISK NUMBERED D  */
  DECLARE D BYTE;
  CALL MON1(14,D);
  CALL MON1(12,0);
  END SELECT;
```

```
RESET: PROCEDURE;
  /*  DOS PROCEDURE  */
  /*  SELECTS DISK 0  */
  CALL MON1(13,0);
  CALL MON1(12,0);
  END RESET;

MAKE: PROCEDURE (ADDR);
  /*  DOS PROCEDURE  */
  DECLARE ADDR ADDRESS;
  DCNT=MON2(22,ADDR);
  CALL MON1(12,0);
  END MAKE;

OPEN: PROCEDURE (ADDR);
  /*  DOS PROCEDURE  */
  DECLARE ADDR ADDRESS;
  DCNT=MON2(15,ADDR);
  CALL MON1(12,0);
  END OPEN;

CLOSE: PROCEDURE (ADDR);
  /*  DOS PROCEDURE  */
  DECLARE ADDR ADDRESS;
  DCNT=MON2(16,ADDR);
  CALL MON1(12,0);
  END CLOSE;

DELETE: PROCEDURE (ADDR);
  /*  DOS PROCEDURE  */
  DECLARE ADDR ADDRESS;
  CALL MON1(19,ADDR);
  CALL MON1(12,0);
  END DELETE;

READ: PROCEDURE (ADDR) BYTE;
  /*   DOS PROCEDURE  */
  DECLARE
    ADDR ADDRESS,
    ANS BYTE;
  ANS=MON2(20,ADDR);
  CALL MON1(12,0);
  RETURN ANS;
  END READ;

WRITE: PROCEDURE (ADDR) BYTE;
  /*  DOS PROCEDURE  */
  DECLARE
```

```
    ADDR ADDRESS,
    ANS BYTE;
  ANS=MON2(21,ADDR);
  CALL MON1(12,0);
  RETURN ANS;
  END WRITE;

/*  **********    END OF LIBRARY PROCEDURES    **********  */

CO$NBR: PROCEDURE (I);
  /*  CONVERT BINARY NUMBER<10 TO ASCII AND OUTPUT  */
  DECLARE I BYTE;
  CALL CO(I+30H);
  END CO$NBR;

CO$NUM: PROCEDURE (I);
  /*  CONVERT ONE BYTE TO AN ASCII NUMBER AND OUTPUT  */
  /*  USE SHIFTING  */
  DECLARE (I,J,K) BYTE;
  IF LINE$NUM<>CONSOLE$LINE THEN RETURN;
  J=I/100;
  IF J=0
    THEN CALL CO(BLANK);
    ELSE CALL CO$NBR(J);
  I=I-J*100;
  K=I/10;
  IF (K=0) AND (J=0)
    THEN CALL CO(BLANK);
    ELSE CALL CO$NBR(K);
  CALL CO$NBR(I-K*10);
  END CO$NUM;

CRLF: PROCEDURE;
  /*  OUTPUT CARRIAGE RETURN THEN LINE FEED IF LINE$NUM=CONSOLE  */
  IF LINE$NUM<>CONSOLE$LINE THEN RETURN;
  CALL CO(CR);
  CALL CO(LF);
  END CRLF;

INVALID: PROCEDURE (CHAR);
  /*  IF CONSOLE$LINE THEN OUTPUT "INVALID ENTRY"  */
  DECLARE CHAR BYTE;
  IF LINE$NUM<>CONSOLE$LINE THEN RETURN;
  CALL CRLF;
  CALL CRLF;
  CALL PRINT(.OUT05);
  CALL CO(CHAR);
  CALL PRINT(.OUT17);
```

```
   CALL CRLF;
   END INVALID;

Y$OR$N: PROCEDURE (CHAR) BYTE;
   /*  RETURN TRUE IF CHAR=Y OR N  */
   DECLARE CHAR BYTE;
   IF (CHAR='Y') OR (CHAR='N') THEN RETURN TRUE;
   RETURN FALSE;
   END Y$OR$N;

ESCAPE: PROCEDURE BYTE;
   /*  IF ESCAPE KEY HAS BEEN PUSHED, RETURN TRUE  */
   IF CSTS
     THEN IF (CI AND 07FH)=TERM$MARK THEN RETURN TRUE;
   RETURN FALSE;
   END ESCAPE;

CLEAR$FCB: PROCEDURE;
   /*  SET FCB TO ZERO.
       DFCB WILL BE DELETED FROM PROGRAM WHEN DMA
       IMPLEMENTED  */
   DECLARE I BYTE;
   DO I=0 TO FCB$LEN$M;
     DFCB(I),FCB(I)=0;
     END;
   END CLEAR$FCB;

CLEAR$MODE: PROCEDURE;
   /*  CLEAR CURRENT LINE MODE AND STATE.
       RETURN TO THE SUPERVISOR  */
   MODE=BLANK;
   MODE$STATE=0;
   END CLEAR$MODE;

CLEAR$CMD: PROCEDURE;
   /*  SET CURRENT LINE MODE TO COMMAND STATE  */
   MODE='C';
   MODE$STATE=2;
   CALL PRINT(.OUT29);
   END CLEAR$CMD;

CLEAR$ALL: PROCEDURE;
   /*  INITIALIZE ALL CONTROL PARAMETERS  */
   CALL CLEAR$MODE;
   HDG$VEC,ABUF,BBUF=1;
   LOW$PTR,HIGH$PTR=0;
   CALL CLEAR$FCB;
   END CLEAR$ALL;
```

```
FLICKER: PROCEDURE;
  /*  ERASE LAST CHARACTER OUTPUT.
      USED TO FLASH THE LINE$NUM  */
  CALL CO(BACK$SPACE);
  CALL CO(BLANK);
  CALL CO(BACK$SPACE);
  FLICK$ON=FALSE;
  END FLICKER;

HIGH$SPEED: PROCEDURE BYTE;
  /*  SIMULATE FULLY BUFFERED COMM LINE  */
  /*  SEND 24,800 BITS, AVERAGE MAP  */
  HS$CTR=HS$CTR+1;
  DO WHILE HS$CTR<22;
    RETURN HS$VEC(HS$CTR);
    END;
  DO WHILE HS$CTR<3100;
    IF HS$CTR=3099 THEN RETURN EOB$MARK;
    RETURN HS$CHAR;
    END;
  HS$CTR=4000;
  HS$CHAR=HS$CHAR+1;
  NO$DATA$IN=TRUE;
  RETURN BLANK;
  END HIGH$SPEED;

SCAN$TAPE: PROCEDURE BYTE;
  /*  RETURN ONE CHARACTER FROM THE PAPER TAPE
      READER UNLESS READER IS NOT READY.
      IF NOT IN THE RECEIVE STATE AND TERM$MARK
      ENCOUNTERED, CLEAR MODE.  TURNS ON
      THE HIGH SPEED LINE  */
  DECLARE CHAR BYTE;
  IF (INPUT(2) AND 01EH)<>0
    THEN DO;
      NO$DATA$IN=TRUE;
      RETURN BLANK;
      END;
  CHAR=RI AND 07FH;
  IF MODE<>'W'
    THEN DO;
      IF CHAR=TERM$MARK
        THEN DO;
          CALL CLEAR$MODE;
          NO$DATA$IN=TRUE;
          RETURN BLANK;
          END;
```

```
        ELSE IF CHAR=STX$MARK THEN HS$CTR=0;
      END;
  RETURN CHAR;
  END SCAN$TAPE;

SCAN: PROCEDURE BYTE;
  /*  SCAN RETURNS THE NEXT CHARACTER FROM THE COMM LINE
      DETERMINED BY LINE$NUM.  */
  /*  WOULD ACTUALLY SET AN OUTPUT PORT AND
      INTERRUPT THE COMMUNICATIONS MODULE PROCESSOR  */
  DECLARE CHAR BYTE;
  IF LINE$NUM=1 THEN RETURN HIGH$SPEED;
  IF LINE$NUM=2 THEN RETURN SCAN$TAPE;
  IF NOT CSTS
    THEN DO;
      NO$DATA$IN=TRUE;
      RETURN BLANK;
      END;
  /*  SIMULATE THAT THE CALLED LINE$NUM'S BUFFER
      WAS EMPTY AND AN ATTEMPT WAS MADE TO READ FROM IT.
      SET CHAR=BLANK AND EXECUTE A RETURN.  */
  CHAR=CI AND 07FH;     /*  MODIFY DEPENDING ON INPUT DEVICE  */
  /*  IF CHAR IS TERM$MARK TAKE ACTION
      DEPENDING ON COMM LINE MODE  */
  IF (CHAR=TERM$MARK) AND (MODE<>'W')
    THEN DO;
      IF (MODE='C') OR (MODE='G') OR (MODE=BLANK)
        THEN CALL CLEAR$MODE;
        ELSE CALL CLEAR$CMD;
      NO$DATA$IN=TRUE;
      RETURN BLANK;
      END;
   RETURN CHAR;
   END SCAN;

BUF$WAIT: PROCEDURE (BASE);
  /*  TEST FIRST BYTE OF BUFFER.
      IF CLOSED, EQUAL 255, THEN WAIT  */
  DECLARE
    (BASE,TIMER) ADDRESS,
    (BUF BASED BASE) BYTE;
  DO WHILE BUF=255;
    TIMER=TIMER+1;
    CALL TIME(10);  /*  WAIT ONE MILLISECOND  */
    IF TIMER>650   /*  AVERAGE DISK ACCESS TIME  */
      /*  THE TIME TO FILL THE SECOND INTERNAL BUFFER
          PLUS THIS IDLE TIME SHOULD BE ENOUGH FOR
          NORMAL ACCESS  */
```

```
      THEN DO;
        /*  ERROR.  OUTPUT BUFFER ADDRESS.
            OPEN BUFFER AND PROCEED  */
        CALL PRINT(.OUT27);
        /*  SHOULD USE HEX OUTPUT REALLY  */
        CALL CO$NUM(HIGH(BASE));
        CALL CO$NUM(LOW(BASE));
        CALL CRLF;
        BUF=1;
        RETURN;
        END;
    END;
  END BUF$WAIT;

NEDS$MAKE: PROCEDURE;
  CALL BUF$WAIT(ABUF$B);
  CALL BUF$WAIT(BBUF$B);
  ABUF,BBUF=1;
  CALL MAKE(DFCB$B);  /*  FCB$B WHEN DMA  */
  END NEDS$MAKE;

NEDS$OPEN: PROCEDURE;
  CALL BUF$WAIT(ABUF$B);
  CALL BUF$WAIT(BBUF$B);
  ABUF,BBUF=1;
  CALL OPEN(DFCB$B);  /*  FCB$B WHEN DMA  */
  END NEDS$OPEN;

NEDS$CLOSE: PROCEDURE;
  CALL BUF$WAIT(ABUF$B);
  CALL BUF$WAIT(BBUF$B);
  CALL CLOSE(DFCB$B);  /*  FCB$B WHEN DMA  */
  END NEDS$CLOSE;

NEDS$DELETE: PROCEDURE;
  CALL DELETE(DFCB$B);  /*  FCB$B WHEN DMA  */
  END NEDS$DELETE;

NEDS$READ: PROCEDURE (BASE) BYTE;
  /*  TRANSFER DATA FOR PRESENT DOS  */
  DECLARE
    BASE ADDRESS,
    (I,BUF BASED BASE) BYTE;
  IF READ(DFCB$B)=1     /*  FCB$B WITH DMA  */
    THEN DO;
      BUF=1;
      RETURN 1;
      END;
```

```
      /*  WAIT TILL DISK IS DONE  */
      /*  CURRENT DOS WAITS  */
   DO I=0 TO 127;
     BUF(I+1)=DBUF(I);
     END;
   BUF=1;
   /*  INTERRUPT ROUTINE WITH DMA  */
   RETURN 0;
   END NEDS$READ;

NEDS$WRITE: PROCEDURE (BUF$B);
   /*  TRANSFER DATA FOR PRESENT DOS  */
   DECLARE
     BUF$B ADDRESS,
     (I,BUF BASED BUF$B) BYTE;
   DO I=1 TO BUF$LEN$M;
     DBUF(I-1)=BUF(I);
     END;
   /*  DOS WRITE RETURNS 0 IF OK, 1 IF CR>MAXREC, 2 IF NO SPACE  */
   /*  WOULD CALL ERROR ROUTINE ACTUALLY  */
   IF (I:=WRITE(DFCB$B))<>0      /*  FCB$B  */
     THEN DO;
       CALL PRINT(.OUT16);
       CALL CO$NUM(I);
       CALL CRLF;
       END;
   /*  WAIT TILL DISK IS DONE  */
   /*  CURRENT DOS WAITS  */
   BUF=1;
   /*  WRITE IS COMPLETE OPEN BUFFER  */
   /*  INTERRUPT ROUTINE WITH DMA  */
   END NEDS$WRITE;

EOB: PROCEDURE (CHAR) BYTE;
   /*  IF THE END OF BULLETIN, RETURN TRUE  */
   DECLARE CHAR BYTE;
   IF CHAR=EOB$MARK THEN RETURN TRUE;
   /*  ALSO TEST A FLAG  */
   RETURN FALSE;
   END EOB;

SOB: PROCEDURE (CHAR) BYTE;
   /*  IF THE START OF BULLETIN, RETURN TRUE  */
   DECLARE CHAR BYTE;
   IF CHAR=SOB$MARK THEN RETURN TRUE;
   /*  ALSO TEST A FLAG  */
   RETURN FALSE;
   END SOB;
```

```
M$ENTRY: PROCEDURE ADDRESS;
  /*  SHOULD CALCULATE BY SHIFTING  */
  RETURN PTR$TBL(LOW$PTR)*MAIN$TBL$LEN;
  END M$ENTRY;

CLEAR$M$ENTRY: PROCEDURE;
  /*  SET TABLE LINE TO EMPTY AND ZEROS  */
  DECLARE I BYTE;
  DO I=0 TO MAIN$TBL$LEN-1;
    MAIN$TBL(M$ENTRY+I)=CLEAN$M$TBL(I);
    END;
  END CLEAR$M$ENTRY;

IN$GR$LEVEL: PROCEDURE BYTE;
  /*  SEE IF LOW$PTR IN GRAPH AREA  */
  IF (LOW$PTR>=GR$LEVEL) AND (LOW$PTR<MAIN$TBL$TOP-1)
    THEN RETURN TRUE;
  RETURN FALSE;
  END IN$GR$LEVEL;

FILE$TBL: PROCEDURE;
  /*  SEND THE MAIN TABLE TO THE DISK  */
  /*  SPEED UP ROUTINE AS MUCH AS POSSIBLE  */
  DECLARE
    I BYTE,
    MAIN$CTR ADDRESS;

    DONE: PROCEDURE BYTE;
      IF MAIN$CTR>=MAIN$TBL$MAX$AND$END
        THEN DO;
          CALL NEDS$CLOSE;
          RETURN TRUE;
          END;
      RETURN FALSE;
      END DONE;

  CALL CLEAR$FCB;
  /*  DFCB(1),FCS(1) SET TO 0  */
  MAIN$CTR=0;
  CALL NEDS$OPEN;
  DO FOREVER;
    CALL BUF$WAIT(ABUF$B);
    DO I=1 TO BUF$LEN$M;
      ABUF(I)=PTR$TBL(MAIN$CTR);
      MAIN$CTR=MAIN$CTR+1;
      END;
    ABUF=255;
```

```
      CALL NEDS$WRITE(ABUF$B);
      IF DONE THEN RETURN;
      CALL BUF$WAIT(BBUF$B);
      DO I=1 TO BUF$LEN$M;
        BBUF(I)=PTR$TBL(MAIN$CTR);
        MAIN$CTR=MAIN$CTR+1;
        END;
      BBUF=255;
      CALL NEDS$WRITE(BBUF$B);
      IF DONE THEN RETURN;
      END;
    END FILE$TBL;

EOB$HDL: PROCEDURE (BUF$B) BYTE;
    /*  AT THE END OF BULLETIN, BLANK REST OF BUFFER
        AND SEND IT TO THE DISK.  CLOSE FILE AND RETURN TRUE  */
    DECLARE
      I BYTE,
      BUF$B ADDRESS,
      (BUF BASED BUF$B) BYTE;
    DO WHILE BUF<=BUF$LEN$M;
      BUF(BUF)=00H;
      /*  NULL  */
      BUF=BUF+1;
      END;
    BUF=255;
    MODE$STATE=50;
    CALL NEDS$WRITE(BUF$B);
    CALL NEDS$CLOSE;
    RETURN TRUE;
    END EOB$HDL;

MATCH$STA: PROCEDURE BYTE;
    /*  RETURNS NO. BYTES INTO STA$TBL BEFORE MATCH  */
    /*  IF NO MATCH RETURN STA$TBL$MAX  */
    DECLARE (I,J,K) BYTE;
    DO I=0 TO STA$TBL$END-1;
      K=I*STA$TBL$LEN;
      J=0;
      DO WHILE HDG$VEC(J+2)=STA$TBL(K+J);
        IF (J:=J+1)=5 THEN RETURN K;
        END;
      END;
    RETURN STA$TBL$MAX;
    END MATCH$STA;

STA$HDL: PROCEDURE (CHAR,BASE) BYTE;
    /*  PROCEDURE HANDLES HOURLY REPORTS.
```

```
      IF THE STATION REPORT IS TO BE RETAINED, THE
      STA$TBL INDEX WILL BE ENTERED INTO THE BUFFER.
      RETURN ALL DATA UNTIL THE NEXT STX$MARK.
      IF STATION IS NOT RETAINED, DELETE TITLE
      AND DATA AND RETURN BLANK UNTIL THE NEXT STX  */

    ENTER$STA$TITLE: PROCEDURE (I,BASE);
      DECLARE
        BASE ADDRESS,
        (I,J,BUF BASED BASE) BYTE;
      BUF(BUF)=I;
      DO J=1 TO 4;
        BUF(BUF+J)=BLANK;
        END;
      BUF=BUF+5;
      END ENTER$STA$TITLE;

DECLARE
  BASE ADDRESS,
  (CHAR,I,BUF BASED BASE) BYTE;
IF CHAR=STX$MARK
  THEN DO;
    HDG$VEC=2;
    MODE$STATE=40;
    RETURN CHAR;
    END;
IF MODE$STATE=45 THEN RETURN CHAR;
IF MODE$STATE=40
  THEN DO;
    IF HDG$VEC<=6
      THEN DO;
        HDG$VEC(HDG$VEC)=CHAR;
        HDG$VEC=HDG$VEC+1;
        RETURN CHAR;
        END;
    IF (I:=MATCH$STA)<>STA$TBL$MAX
      THEN MODE$STATE=45;
      ELSE DO;
        MODE$STATE=20;
        CHAR=BLANK;
        END;
    IF BUF>=6
      THEN DO;
        BUF=BUF-5;
        CALL ENTER$STA$TITLE(I,BASE);
        END;
      ELSE DO;
        BUF(BUF)=STX$MARK;
```

```
            BUF=BUF+1;
            CALL ENTER$STA$TITLE(I,BASE);
            END;
        RETURN CHAR;
        END;
    IF BUF>1 THEN BUF=BUF-1;
    RETURN BLANK;
    END STA$HDL;

CK$SUM$HDL: PROCEDURE BYTE;
    /*  IF CK$SUM>X THEN RETURN FALSE  */
    /*  ADD ACTUAL ROUTINE  */
    CK$SUM=12;
    RETURN TRUE;
    END CK$SUM$HDL;

BUF$FILL: PROCEDURE (BUF$B) BYTE;
    /*  RETURNS TRUE IF THIS BUFFER FINISHES A BULLETIN  */
    /*  IF SOB FOUND SETS MODE$STATE=99 AND CLEARS FCB  */
    /*  ALSO TAKES CARE OF DISK IF SOB  */
    DECLARE
      BUF$B ADDRESS,
      (BUF BASED BUF$B) BYTE,
      CHAR BYTE;
    CALL BUF$WAIT(BUF$B);
    CHAR=BLANK;
    DO WHILE NOT SOB(CHAR);
      IF BUF<BUF$LEN
        THEN DO;
          CHAR=SCAN;
          IF NO$DATA$IN THEN RETURN FALSE;
          /*  IF CK$SUM TOO HIGH, DROP BULLETIN  */
          IF NOT CK$SUM$HDL THEN CHAR=SOB$MARK;
          IF EOB(CHAR) THEN RETURN EOB$HDL(BUF$B);
          /*  END OF BULLETIN  */
          IF LOW$PTR<SA$LEVEL
            THEN CHAR=STA$HDL(CHAR,BUF$B);
            /*  HANDLES HOURLY REPORTS  */
          BUF(BUF)=CHAR;
          BUF=BUF+1;
          END;
        ELSE DO;
          /*  NOT (EOB OR SOB) BUT BUF IS FULL  */
          BUF=255;
          CALL NEDS$WRITE(BUF$B);
          RETURN FALSE;
          END;
      END;
```

```
   /*  SOB IN THE MIDDLE OF A BULLETIN  */
   MODE$STATE=99;
   /*  CALL COMM MODULE  DUMP  */
   CALL NEDS$CLOSE;
   CALL NEDS$DELETE;
   RETURN TRUE;
   END BUF$FILL;

SHIFT$PTR: PROCEDURE (LOW,HIGH);
   /*  SHIFT MAIN TABLE INDIRECT POINTERS DOWN
       THE TABLE WITHIN DATA LIMITS.  LAST
       POINTER BECOMES THE NEW INDEX, AVAILABLE ENTRY.  */
   DECLARE
     (LOW,HIGH) BYTE,
     (X,Y) BYTE;
   X=PTR$TBL(PTR$INDEX);
   DO WHILE LOW<HIGH;
     Y=PTR$TBL(LOW);
     PTR$TBL(LOW)=X;
     X=Y;
     LOW=LOW+1;
     END;
   PTR$TBL(PTR$INDEX)=X;
   END SHIFT$PTR;

SWAP$PTR: PROCEDURE;
   /*  SWAP PRESENT POINTER AND INDEX  */
   DECLARE TEMP BYTE;
   TEMP=PTR$TBL(PTR$INDEX);
   PTR$TBL(PTR$INDEX)=PTR$TBL(LOW$PTR);
   PTR$TBL(LOW$PTR)=TEMP;
   END SWAP$PTR;

COMPARE$DTG: PROCEDURE (DTG$OLD$B) BYTE;
   /*  RETURNS TRUE IF INCOMING FILE MORE CURRENT
       AND THE ENTIRE DTG IS CHECKED.
       IF JUST THE DAY OR DAY/HOUR
       TESTED AND OLD=NEW, THEN RETURN "2" */
   DECLARE
     DTG$OLD$B ADDRESS,
     (I,J,DTG$OLD BASED DTG$OLD$B) BYTE;
   IF LOW$PTR>SA$LEVEL
     THEN J=5;
     ELSE IF LOW$PTR=SA$LEVEL
       THEN J=1;
       ELSE J=3;
   IF DTG$OLD<DTG
     THEN RETURN TRUE;
```

```
    ELSE IF DTG$OLD=DTG
      THEN DO;
        DO I=1 TO J;
          IF DTG$OLD(I)<DTG(I)
            THEN RETURN TRUE;
            ELSE IF DTG$OLD(I)>DTG(I)
              THEN RETURN FALSE;
          END;
        IF LOW$PTR<=SA$LEVEL THEN RETURN 2;
        RETURN FALSE;
        END;
      ELSE IF DTG='0'
        THEN RETURN TRUE;
        ELSE RETURN FALSE;
  END COMPARE$DTG;

GOOD$FILE: PROCEDURE BYTE;
  /*  RETURNS TRUE IF FILE IS KEPT.  UPDATES PTR$TBL IF NECESSARY  */
  HDG$VEC(20)=CK$SUM;
  DO WHILE LOW$PTR<HIGH$PTR;
    IF MAIN$TBL(M$ENTRY)>=TITLE$TBL$END-2
      THEN DO;
        /*  FOUND EMPTY TABLE LOCATION  */
        CALL SWAP$PTR;
        RETURN TRUE;
        END;
      ELSE IF COMPARE$DTG(M$ENTRY+1+.MAIN$TBL)
        THEN DO;
          /*  NEW DTG FITS HERE, SHIFT  */
          CALL SHIFT$PTR(LOW$PTR,HIGH$PTR);
          RETURN TRUE;
          END;
    LOW$PTR=LOW$PTR+1;
    END;
  /*  NO EMPTY FILE AND ALL FILE DTG EQUAL OR NEWER  */
  /*  NOT A GOOD FILE, DUMP IT  */
  RETURN FALSE;
  END GOOD$FILE;

STORE$TO$MAIN: PROCEDURE;
  DECLARE (I,J) ADDRESS;
  IF GOOD$FILE
    THEN DO;
      /*  FILL MAIN TABLE AT LOW$PTR LOCATION  */
      J=M$ENTRY;
      MAIN$TBL(J)=HDG$VEC(1);
      DO I=1 TO 10;
        MAIN$TBL(J+I)=HDG$VEC(10+I);
```

```
        END;
      CALL FILE$TBL;
      END;
  CALL CLEAR$FCB;
  /*  DELETE THE FILE NOW AT THE INDEX.
      NEXT TABLE LOCATION TO USE FOR INPUT  */
  DFCB(1),FCB(1)=PTR$TBL(PTR$INDEX);
  CALL NEDS$DELETE;
  END STORE$TO$MAIN;

HIGH$PTR$CALC: PROCEDURE (I) BYTE;
  /*  RETURNS UPPER LIMIT IN THE MAIN TABLE FOR THIS DATA TYPE  */
  DECLARE
    I BYTE,
    J ADDRESS;
  J=(I+1)*TAI$TBL$LEN;
  DO WHILE LOW$PTR=TAI$TBL(J+6);
    J=J+TAI$TBL$LEN;
    IF J>=TAI$TBL$MAX
      THEN DO;
        CALL PRINT(.OUT15);
        RETURN FALSE;
        END;
    END;
  HIGH$PTR=TAI$TBL(J+6);
  RETURN TRUE;
  END HIGH$PTR$CALC;

MATCH$TITLE: PROCEDURE BYTE;
  /*  PROCEDURE RETURNS INDEX TO TITLE TBL IF A MATCH
      IS FOUND BETWEEN THE BULLETIN TITLE AND THE TAI$TBL.
      SETS HIGH AND LOW POINTERS IF MATCH.
      RETURN THE VALUE OF TITLE$TBL$END$M IF NO MATCH IS FOUND  */
  DECLARE
    (I,J) BYTE,
    K ADDRESS;
  DO I=0 TO TAI$TBL$END-1;
    K=I*TAI$TBL$LEN;
    J=0;
    DO WHILE HDG$VEC(J+1)=TAI$TBL(K+J);
      J=J+1;
      IF J=6
        THEN DO;
          LOW$PTR=TAI$TBL(K+6);
          IF HIGH$PTR$CALC(I)
            THEN RETURN TAI$TBL(K+7);
            ELSE RETURN TITLE$TBL$END$M;
          END;
```

```
      END;
    END;
  RETURN TITLE$TBL$END$M;
  END MATCH$TITLE;

IN$HDL: PROCEDURE;
  /*  MESSAGE TO BE FILED.  LOAD THE DATA  */
  IF MODE$STATE=10
    THEN DO;
      MODE$STATE=20;
      IF IN$GR$LEVEL
        THEN DO;
          /*  GRAPH, FILE LAST 3 BYTES OF MANOP  */
          ABUF(1)=HDG$VEC(17);
          ABUF(2)=HDG$VEC(18);
          ABUF(3)=HDG$VEC(19);
          ABUF=4;
          END;
      IF LOW$PTR<=SA$LEVEL
        THEN DO;
          /*  HOURLY TYPE REPORT.  LOAD IN DTG  */
          DECLARE I BYTE;
          ABUF(1),ABUF(2),ABUF(3),ABUF(13)=LF;
          ABUF(4),ABUF(14),ABUF(15)=CR;
          ABUF(5),ABUF(6)=STX$MARK;
          DO I=0 TO 5;
            ABUF(7+I)=DTG(I);
            END;
          DTG(4),DTG(5)=30H;
          IF LOW$PTR=SA$LEVEL THEN DTG(2),DTG(3)=30H;
          ABUF=16;
          END;
      END;
  IF (BBUF<>1) AND (BBUF<>255)
    THEN IF BUF$FILL(BBUF$B) THEN RETURN;
  DO FOREVER;
    IF NO$DATA$IN THEN RETURN;
    IF BUF$FILL(ABUF$B) THEN RETURN;
    IF NO$DATA$IN THEN RETURN;
    IF BUF$FILL(BBUF$B) THEN RETURN;
    END;
  END IN$HDL;

SA$DTG$SAME: PROCEDURE BYTE;
  /*  LOCATE DISK FILE TO ENTER HOURLY TYPE REPORTS.
      EITHER DUMPS REPORT (TOO OLD), OPENS AN OLD
      FILE (CURRENT HOUR),
      OR RETURNS TO MAKE A NEW FILE  */
```

```
   DECLARE X BYTE;
   IF MAIN$TBL(M$ENTRY)=TITLE$TBL$END$M THEN RETURN FALSE;
   X=COMPARE$DTG(M$ENTRY+1+.MAIN$TBL);
   IF X=2
     THEN DO:
       /*  OPEN AN OLD FILE  */
       CALL CRLF;
       CALL CLEAR$FCB;
       DFCB(1),FCB(1)=PTR$TBL(LOW$PTR);
       CALL NEDS$OPEN;
       /*  SET CURRENT RECORD TO OLD RECORD-COUNT
           CHANGE DFCB(15) TO FCB(15) WITH DMA  */
       DFCB(32),FCB(32)=DFCB(15);
       RETURN TRUE;
       END;
   IF X=FALSE
     THEN DO;
       /*  BULLETIN TOO OLD.  TAKE NO ACTION  CALL DJMP  */
       MODE$STATE=3;
       RETURN (NO$DATA$IN:=TRUE);
       END;
   RETURN FALSE;
   /*  RETURN AND HAVE MANOP$HDL CREATE NEW FILE  */
   END SA$DTG$SAME;

NEW$BULL$IN: PROCEDURE;
   /*  MAKE A NEW FILE  */
   CALL CLEAR$FCB;
   DFCB(1),FCB(1)=PTR$TBL(PTR$INDEX);
   CALL NEDS$DELETE;
   /*  ENSURE NO FILE EXISTS WITH THE SAME NAME
       WHICH WOULD TIE UP DISK SPACE  */
   CALL NEDS$MAKE;
   END NEW$BULL$IN;

MANOP$HDL: PROCEDURE;
   /*  HANDLES BULLETIN HEADINGS  */
   DECLARE CHAR BYTE;
   CHAR=BLANK;
   DO FOREVER;
     IF MODE$STATE=3
       THEN DO;
         /*  DISCARD INPUT UNTIL SOB  */
         DO WHILE NOT SOB(CHAR);
           CHAR=SCAN;
           IF NO$DATA$IN THEN RETURN;
           END;
         CHAR=BLANK;
```

```
            HDG$VEC=1;
            MODE$STATE=8;
            END;
         DO WHILE NOT (SOB(CHAR) OR EOB(CHAR));
            CHAR=SCAN;
            IF NO$DATA$IN THEN RETURN;
            HDG$VEC(HDG$VEC)=CHAR;
            HDG$VEC=HDG$VEC+1;
            IF HDG$VEC=20
              THEN DO;
                /*  MANOP IS LOADED  */
                MODE$STATE=10;
                /*  IF HOURLY TYPE DO  */
                IF LOW$PTR<=SA$LEVEL
                  THEN IF SA$DTG$SAME THEN RETURN;
                CALL CRLF;
                /*  MAKE A NEW FILE  */
                CALL NEW$BULL$IN;
                RETURN;
                END;
            IF HDG$VEC=7
              THEN DO;
                IF (HDG$VEC(1):=MATCH$TITLE)=TITLE$TBL$END$M
                  /*  IGNORE MSG TILL NEXT SOB  */
                  /*  SEND MSG TO COMM MODULE TO DUMP  */
                  THEN CHAR=EOB$MARK;
                IF HDG$VEC(1)=TITLE$TBL$END
                  THEN DO;
                    /*  THIS STATION'S ADDRESS HAS BEEN CALLED.
                        PUT THIS COMM LINE IN THE SUPERVISOR STATE  */
                    CALL CLEAR$MODE;
                    NO$DATA$IN=TRUE;
                    RETURN;
                    END;
                END;
            END;
         MODE$STATE=3;
         END;
      END MANOP$HDL;

   G$HDL: PROCEDURE;
      /*  DEVELOPMENT ROUTINE TO AID BULLETIN INPUT  */
      IF MODE$STATE<=2
        THEN DO;
          HDG$VEC=1;
          CALL PRINT(.OUT13);
          MODE$STATE=3;
          END;
```

```
IF MODE$STATE=3
  THEN DO;
    DO WHILE HDG$VEC<=6;
      HDG$VEC(HDG$VEC)=SCAN;
      IF NO$DATA$IN THEN RETURN;
      HDG$VEC=HDG$VEC+1;
      END;
    IF (HDG$VEC(1):=MATCH$TITLE)=TITLE$TBL$END$M
      THEN DO;
        CALL PRINT(.OUT12);
        RETURN;
        END;
    IF HDG$VEC(1)=TITLE$TBL$END THEN RETURN;
    CALL PRINT(.OUT11);
    HDG$VEC=11;
    MODE$STATE=5;
    END;
IF MODE$STATE=5
  THEN DO;
    DO WHILE HDG$VEC<=19;
      HDG$VEC(HDG$VEC)=SCAN;
      IF NO$DATA$IN THEN RETURN;
      HDG$VEC=HDG$VEC+1;
      END;
    MODE$STATE=10;
    IF LOW$PTR<=SA$LEVEL
      THEN DO;
        IF SA$DIG$SAME
          THEN DO;
            IF MODE$STATE=3
              THEN DO;
                CALL PRINT(.OUT20);
                NO$DATA$IN=FALSE;
                RETURN;
                END;
            END;
          ELSE CALL NEW$BULLETIN;
        END;
      ELSE CALL NEW$BULLETIN;
    CALL PRINT(.OUT19);
    END;
CALL INSHDL;
IF NO$DATA$IN THEN RETURN;
IF MODE$STATE<>50 THEN RETURN;
IF LOW$PTR<=SA$LEVEL
  THEN DO;
    IF ECH(1)=PTR$TBL(PTR$INDEX) THEN CALL STORE$TO$MAIN;
    END;
```

```
      ELSE CALL STORE$TO$MAIN;
    END G$HDL;

  W$HDL: PROCEDURE;
    /*  RECEIVE STATE  */
    DECLARE CHAR BYTE;
    IF MODE$STATE<=2
      THEN DO;
        IF MODE$STATE=1 THEN CALL PRINT(.OUT23);
        MODE$STATE=3;
        END;
    IF MODE$STATE<=3
      THEN DO;
        CALL MANOP$HDL;
        IF NO$DATA$IN THEN RETURN;
        END;
    CALL IN$HDL;
    IF NO$DATA$IN THEN RETURN;
    /*  MODE$STATE=50 NORMAL EOB  */
    IF MODE$STATE=50
      THEN DO;
        /*  CALL UPDATE MAIN TABLE IF REQUIRED  */
        IF LOW$PTR<=SA$LEVEL
          THEN DO;
            IF ECB(1)=PTR$TBL(PTR$INDEX) THEN CALL STORE$TO$MAIN;
            END;
          ELSE CALL STORE$TO$MAIN;
        MODE$STATE=3;
        END;
      ELSE MODE$STATE=8;
    NO$DATA$IN=TRUE;
    /*  RETURN TO POLLING AFTER BULLETIN END  */
    END W$HDL;

  C$HDL: PROCEDURE;
    /*  COMMAND STATE  */

    TWO$DIGIT$1: PROCEDURE;
      /*  TAKE IN FIRST DIGIT  */
      IF MODE$STATE=5
        THEN DO;
          CALL PRINT(.OUT10);
          MODE$STATE=8;
          END;
      LOW$PTR=SCAN-30H;
      IF NO$DATA$IN THEN RETURN;
      MODE$STATE=10;
      END TWO$DIGIT$1;
```

```
TWO$DIGIT$2: PROCEDURE;
  /*  TAKE IN SECOND DIGIT  */
  DECLARE CHAR BYTE;
    CHAR=SCAN-30H;
    IF NO$DATA$IN THEN RETURN;
    /*  CALCULATE TWO DIGIT VALUE  */
    LOW$PTR=LOW$PTR*10+CHAR;
    IF LOW$PTR>MAIN$TBL$TOP
      THEN DO:
        /*  ENTRY TOO LARGE  */
        CALL PRINT(.OUT09);
        CALL CO$NUM(LOW$PTR);
        CALL CLEAR$CMD;
        NO$DATA$IN=TRUE;
        RETURN;
        END;
    IF MAIN$TBL(M$ENTRY)=TITLE$TBL$END$M
      THEN DO;
        /*  TABLE ENTRY IS EMPTY  */
        CALL PRINT(.OUT08);
        CALL CLEAR$CMD;
        NO$DATA$IN=TRUE;
        RETURN;
        END;
  END TWO$DIGIT$2;

D$HDL: PROCEDURE;
  /*  DISPLAY A FILE  */

  FILE$OUT: PROCEDURE BYTE;
    /*  RETURNS TRUE IF REQUESTED REPORT FOUND AND OUTPUT  */

    OUT$BUF: PROCEDURE (BASE);
      /*  OUTPUTS ONE BUFFER OF DATA  */
      DECLARE
        BASE ADDRESS,
        (I,BUF BASED BASE) BYTE;
      CALL BUF$WAIT(BASE);
      IF LOW$PTR<SA$LEVEL
        THEN DO;
          /*  HANDLES HOURLY REPORTS  */
          DO I=1 TO BUF$LEN$M;
            IF BUF(I)=STX$MARK
              THEN DO;
                IF MODE$STATE=45 THEN MODE$STATE=55;
                IF MODE$STATE=35 THEN MODE$STATE=45;
                CALL CO(004);
```

```
                END;
              ELSE DO;
                IF MODE$STATE=45
                  THEN DO;
                    IF BUF(I)<STA$TBL$MAX
                      THEN DO;
                        /*  REPLACE STATION IDENTIFIER  */
                        CALL CRLF;
                        CALL PRINT(.STA$TBL(BUF(I)));
                        END;
                      ELSE CALL CO(00H);
                    END;
                  ELSE CALL CO(BUF(I));
                MODE$STATE=35;
                END;
            END;
          END;
        ELSE DO;
          /*  OUTPUT FULL BUFFER AS IS  */
          DO I=1 TO BUF$LEN$M;
            CALL CO(BUF(I));
            END;
          END;
      END OUT$BUF;

    CALL BUF$WAIT(ABUF$B);
    ABUF=255;
    /*  IF READ ROUTINE RETURNS FILE COMPLETE,
        RETURN DONE.  */
    IF NEDS$READ(ABUF$B)=1 THEN RETURN TRUE;
    CALL OUT$BUF(ABUF$B);
    /*  IF CANCEL DISPLAY, RETURN DONE  */
    IF ESCAPE THEN RETURN TRUE;
    /*  RETURN TO POLL ROUTINE AFTER OUTPUT OF ONE BUFFER  */
    IF (NO$DATA$IN:=TRUE) THEN RETURN FALSE;
    END FILE$OUT;
    /*  OUTPUT TO DIFFERENT BUFFERS FOR GRAPH  */
    /*  USE SIX BUFFERS FOR GRAPHIC DATA
        AND EXECUTE AT LEAST TWO READS
        BEFORE RETURN TO POLLING ROUTINE  */


DECLARE CHAR BYTE;
IF MODE$STATE<20
  THEN DO;
    /*  DETERMINE FILE  */
    IF MODE$STATE<10 THEN CALL TWO$DIGIT$1;
    IF NO$DATA$IN THEN RETURN;
```

```
      IF MODE$STATE<20 THEN CALL TWO$DIGIT$2;
      IF NO$DATA$IN THEN RETURN;
      CALL CRLF;
      CALL CRLF;
      CALL CLEAR$FCB;
      DFCB(1),FCB(1)=PTR$TBL(LOW$PTR);
      CALL NEDS$OPEN;
      MODE$STATE=35;
      IF IN$GR$LEVEL THEN CALL PRINT(.OUT28);
      END;
    DO FOREVER;
      IF FILE$OUT THEN RETURN;
      IF NO$DATA$IN THEN RETURN;
      END;
    END D$HDL;

  SEE$T: PROCEDURE;
    /*  DISPLAY THE MAIN TABLE  */
    DECLARE
      (K,L,CHAR) BYTE,
      J ADDRESS;
    IF MODE$STATE=5
    THEN DO;
      CALL PRINT(.OUT07);
      MODE$STATE=8;
      LOW$PTR=1;
      END;
      /*  FILE 0 HOLDS THE TABLE  */
    DO WHILE LOW$PTR<=MAIN$TBL$TOP;
      /*  ABOVE MAIN$TBL$TOP FILES FOR COM LINE USE  */
      IF (LOW$PTR=20) OR (LOW$PTR=40) OR (LOW$PTR=MAIN$TBL$TOP)
        THEN DO;
          /*  STOP, A FULL PAGE ON SCREEN  */
          IF MODE$STATE=8
            THEN DO;
              CALL PRINT(.OUT24);
              MODE$STATE=10;
              END;
          CHAR=SCAN;
          IF NO$DATA$IN THEN RETURN;
          IF LOW$PTR=MAIN$TBL$TOP THEN LOW$PTR=1;
          IF CHAR='P'
            THEN CALL PRINT(.OUT07);
              /*  UNLESS P STOP PAGING  */
            ELSE RETURN;
          MODE$STATE=8;
          END;
      IF CSTS
```

```
      THEN IF (CHAR:=CI AND 07FH)=TERM$MARK
        /*  IF ESCAPE KEY ON, STOP  */
        THEN RETURN;
        ELSE IF CHAR='P'
          THEN DO;
            /*  IF "P" HAS BEEN PUSHED, THEN PAGE  */
            IF LOW$PTR<20
              THEN DO;
                LOW$PTR=20;
                CALL PRINT(.OUT07);
                END;
              ELSE IF LOW$PTR<40 THEN DO;
                LOW$PTR=40; CALL PRINT(.OUT07); END;
              ELSE DO; LOW$PTR=1; CALL PRINT(.OUT07); END;
            END;
    CALL CRLF;
    CALL CO$NUM(LOW$PTR);
    CALL PRINT(.OUT05);
    /*  REMOVE.  POINTER OF NO INTEREST TO USER  */
    CALL CO$NUM(PTR$TBL(LOW$PTR));
    CALL PRINT(.OUT05);
    J=M$ENTRY;
    IF (L:=MAIN$TBL(J))>(TITLE$TBL$END$M)
      THEN DO;
        /*  INDEX TOO LARGE, NOTIFY OPERATOR  */
        CALL CO$NUM(L);
        L=TITLE$TBL$END$M;
        CALL PRINT(.OUT09);
        CALL CRLF;
        END;
    DO K=0 TO MAIN$TBL$LEN-2;
      IF K=0
        /*  CONVERT TO ENGLISH TITLE  */
        THEN CALL PRINT(.TITLE$TBL(L*TITLE$TBL$LEN));
        ELSE CALL CO(MAIN$TBL(J+K));
      IF (K=0) OR (K=6) OR (K=8) OR (K=9)
        THEN CALL PRINT(.OUT05);
      END;
    CALL CO$NUM(MAIN$TBL(J+MAIN$TBL$LEN-1));
    CALL PRINT(.OUT05);
    LOW$PTR=LOW$PTR+1;
    /*  RETURN TO POLL AFTER OUTPUT OF ONE LINE  */
    IF (NO$DATA$IN:=TRUE) THEN RETURN;
    END;
  CALL CRLF;
  END SEE$T;
```

```
ERASE$ENTRY: PROCEDURE;

  ERASE$OUT: PROCEDURE;
    /*  DOUBLE CHECK FILE TO DELETE  */
    CALL PRINT(.OUT20);
    CALL CO$NUM(LOW$PTR);
    CALL PRINT(.OUT21);
    END ERASE$OUT;

  DECLARE (I,CHAR) BYTE;
  IF MODE$STATE<10 THEN CALL TWO$DIGIT$1;
  IF NO$DATA$IN THEN RETURN;
  IF MODE$STATE<20
    THEN DO;
      CALL TWO$DIGIT$2;
      IF NO$DATA$IN THEN RETURN;
      /*  LOW$PTR IS SET  */
      CALL ERASE$OUT;
      MODE$STATE=25;
      END;
  CHAR=BLANK;
  DO WHILE NOT Y$OR$N(CHAR);
    CHAR=SCAN;
    IF NO$DATA$IN THEN RETURN;
    IF NOT Y$OR$N(CHAR)
      THEN DO;
        CALL INVALID(CHAR);
        CALL ERASE$OUT;
        END;
    END;
  IF CHAR='N' THEN RETURN;
  I=0;
  DO WHILE LOW$PTR>TAI$TBL(I*TAI$TBL$LEN+5);
    /*  FIND DATA TYPE  */
    I=I+1;
    END;
  IF NOT HIGH$PTR$CALC(I-1)
    THEN DO;
      /*  BRACKET THE DATA AREA  */
      CALL PRINT(.OUT22);
      RETURN;
      END;
  CHAR=PTR$TBL(LOW$PTR);
  DO WHILE LOW$PTR<HIGH$PTR-1;
    /*  SHIFT DOWN  */
    PTR$TBL(LOW$PTR)=PTR$TBL(LOW$PTR+1);
    LOW$PTR=LOW$PTR+1;
    END;
```

```
      PTR$TBL(LOW$PTR)=CHAR;
      CALL CLEAR$M$ENTRY;
      CALL CLEAR$FCB;
      DFCB(1),FCB(1)=CHAR;
      CALL NEDS$DELETE;
      CALL FILE$TBL;
      END ERASE$ENTRY;

   FIND$HDL: PROCEDURE;

      NEED$READ: PROCEDURE BYTE;
        /*  RETURN TRUE IF BUFFER END  */
        ABUF=ABUF+1;
        IF ABUF>=BUF$LEN$M THEN RETURN TRUE;
        RETURN FALSE;
        END NEED$READ;

      MSG$DIVIDER: PROCEDURE BYTE;
        /*  IF THE STX RETURN TRUE  */
        IF ABUF(ABUF)=STX$MARK THEN RETURN TRUE;
        RETURN FALSE;
        END MSG$DIVIDER;

      LOAD$DTG: PROCEDURE BYTE;
        /*  PUT MESSAGE DTG INTO LINE INFO AREA  */
        DO WHILE BBUF<=6;
          DTG(BBUF)=ABUF(ABUF);
          BBUF=BBUF+1;
          IF NEED$READ THEN RETURN FALSE;
          END;
        BBUF=1;
        RETURN TRUE;
        END LOAD$DTG;

      FIND$OUTPUT: PROCEDURE BYTE;
        /*  OUTPUT DATA UNTIL NEXT STX (RETURN TRUE)
            OR UNTIL END OF BUFFER (RETURN FALSE)  */
        DECLARE CHAR BYTE;
        DO WHILE (CHAR:=ABUF(ABUF))<>STX$MARK;
          CALL CO(CHAR);
          IF NEED$READ THEN RETURN FALSE;
          END;
        HDG$VEC=TRUE;
        CALL CRLF;
        RETURN TRUE;
        END FIND$OUTPUT;

      FIND$MSG: PROCEDURE;
```

```
DECLARE I BYTE;
DO FOREVER;
  IF MODE$STATE=50
    THEN DO;
      /*  DISCARD DATA UNTIL STX  */
      DO WHILE NOT MSG$DIVIDER;
        IF NEED$READ THEN RETURN;
        END;
      MODE$STATE=60;
      IF NEED$READ THEN RETURN;
      END;
  IF MODE$STATE=60
    THEN DO;
      DO WHILE ABUF(ABUF)=00H;  /* NULL  */
        /*  SKIP NULL AREA BETWEEN ENTRIES.
            EACH BULLETIN STARTS A NEW BUFFER
            THUS THE NULLS MUST BE SKIPPED
            IF ROUTINE IS BETWEEN TWO BULLETINS  */
        IF NEED$READ THEN RETURN;
        END;
      IF MSG$DIVIDER   /*  SOB  */
        THEN DO;
          /*  TWO STX IN A ROW MAKES A SOB  */
          MODE$STATE=70;
          IF NEED$READ THEN RETURN;
          END;
        ELSE MODE$STATE=80;
      END;
  IF MODE$STATE=70
    THEN DO;
      /*  STORE BULLETIN DTG  */
      IF NOT LOAD$DTG THEN RETURN;
      MODE$STATE=50;
      END;
  IF MODE$STATE=80
    THEN DO;
      IF ABUF(ABUF)=HDG$VEC(I)
        THEN DO;
          /*  MSG WANTED  */
          MODE$STATE=90;
          CALL CRLF;
          DO I=1 TO 6;
            /*  OUTPUT THE DTG  */
            CALL CO(DTG(I));
            END;
          CALL PRINT(.OUT05);
          /*  OUTPUT THE STATION TITLE  */
          CALL PRINT(.STA$TBL(HDG$VEC(1)));
```

```
                CALL CRLF;
                IF NEED$READ THEN RETURN;
                END;
              ELSE MODE$STATE=50;
            END;
        IF MODE$STATE=90
          THEN DO;
            IF FIND$OUTPUT
              /*  OUTPUT THE MSG  */
              THEN MODE$STATE=50;
              ELSE RETURN;
            END;
        END;
      END FIND$MSG;

    FIND$BUFFILL: PROCEDURE;
      DO WHILE LOW$PTR<HIGH$PTR;
        /*  TEST ALL HOURLY REPORTS ONE AT A TIME  */
        IF MODE$STATE=30
          THEN DO;
            IF MAIN$TBL(M$ENTRY)=TITLE$TBL$END$M
              /*  MAIN TABLE ENTRY EMPTY  */
              THEN RETURN;
            CALL CLEAR$FCB;
            DFCB(1),FCB(1)=PTR$TBL(LOW$PTR);
            CALL NEDS$OPEN;
            MODE$STATE=50;
            END;
        DO WHILE NEDS$READ(ABUF$B)<>1;
          /*  READ ONE RECORD AT A TIME, SEARCH FOR STATION MATCH  */
          CALL FIND$MSG;
          /*  QUIT ANYTIME  */
          IF ESCAPE THEN RETURN;
          /*  RETURN TO POLL BETWEEN BUFFERS  */
          IF (NO$DATA$IN:=TRUE) THEN RETURN;
          END;
        MODE$STATE=30;
        LOW$PTR=LOW$PTR+1;
        END;
      END FIND$BUFFILL;

  IF MODE$STATE=5
    THEN DO;
      HDG$VEC=2;
      MODE$STATE=8;
      /*  "ENTER STATION"  */
      CALL PRINT(.OUT01);
      END;
```

```
   IF MODE$STATE<10
     THEN DO;
       DO WHILE HDG$VEC<=6;
         /*  LOAD STATION TO SEARCH FOR  */
         HDG$VEC(HDG$VEC)=SCAN;
           IF HDG$VEC=4
             THEN IF HDG$VEC(4)>='A'
               THEN DO;
                 HDG$VEC(5),HDG$VEC(6)=BLANK;
                 HDG$VEC=6;
                 END;
         IF NO$DATA$IN THEN RETURN;
         HDG$VEC=HDG$VEC+1;
         END;
       CALL CRLF;
       IF (HDG$VEC(1):=MATCH$STA)=STA$TBL$MAX
         THEN DO;
           /*  STATION IS NOT FILED  */
           CALL PRINT(.OUT14);
           RETURN;
           END;
       HDG$VEC=FALSE;  /*  SET TRUE IF STATION FOUND  */
       /*  BRACKET THE HOURLY REPORTS  */
       LOW$PTR=1;
       HIGH$PTR=13;
       MODE$STATE=30;
       END;
   IF MODE$STATE>=30
     THEN DO;
       CALL FIND$BUFFILL;
       IF NO$DATA$IN THEN RETURN;
       IF NOT HDG$VEC THEN CALL PRINT(.OUT06);
       /*  IF HDG$VEC=TRUE AT LEAST ONE MSG REPORT OUTPUT  */
       END;
   END FIND$HDL;

/*      START OF C$HDL MAIN BODY      */
DECLARE CHAR BYTE;
CHAR=BLANK;
IF MODE$STATE<=2
  THEN DO;
    IF MODE$STATE=1
      THEN DO;
        /*  WHAT COMMAND  */
        CALL PRINT(.OUT04);
        MODE$STATE=2;
        END;
    CHAR=SCAN;
```

```
      IF NO$DATA$IN THEN RETURN;
      MODE=CHAR;
      MODE$STATE=5;
      END;
   IF MODE='D'
     THEN CALL D$HDL;
     ELSE IF MODE='A'  /*  FOR ARQ  */
       THEN DO;
         /*  CLEARS THE CRT NOW  */
         DECLARE I BYTE;
         DO I=0 TO 50;
           CALL CO(LF);
         END;
       END;
       ELSE IF MODE='F'
         THEN CALL FIND$HDL;
         ELSE IF MODE='T'
           THEN CALL SEE$T;
           ELSE IF MODE='E'
             THEN CALL ERASE$ENTRY;
             ELSE IF MODE='R'
               THEN GO TO BEGIN$PROGRAM; /*  ONLY "GO TO" IN PROGRAM */
               ELSE IF MODE='C'
                 THEN DO;
                   MODE$STATE=2;
                   NO$DATA$IN=TRUE;
                   CALL PRINT(.OUT04);
                   END;
                 ELSE CALL INVALID(MODE);
   IF NO$DATA$IN THEN RETURN;
   CALL CLEAR$CMD;
   NO$DATA$IN=TRUE;
   END C$HDL;

MODE$NOT$SET: PROCEDURE BYTE;
   IF (MODE<>'G') AND (MODE<>'W') AND (MODE<>'C') THEN RETURN TRUE;
   RETURN FALSE;
   END MODE$NOT$SET;

SCANIT: PROCEDURE;
   /*  SUPERVISOR MODE  */
   /*  FOR TESTING ONLY  */
   IF MODE=BLANK
     THEN DO;
       IF MODE$STATE=0
         THEN DO;
           CALL PRINT(.OUT03);
           CALL PRINT(.OUT02);
```

```
            MODE$STATE=1;
            END;
        DO WHILE MODE$NOT$SET;
          MODE=SCAN;
          IF NO$DATA$IN THEN RETURN;
          IF MODE$NOT$SET
            THEN DO;
              CALL INVALID(MODE);
              CALL PRINT(.OUT02);
              END;
          END;
        CALL CRLF;
        END;
    IF MODE='G'
      THEN CALL G$HDL;
      ELSE IF MODE='W'
        THEN CALL W$HDL;
          ELSE CALL C$HDL;
    IF MODE$STATE=0 THEN NO$DATA$IN=FALSE;
    IF NO$DATA$IN THEN RETURN;
    CALL CLEAR$MODE;
    END SCANIT;

SET$COM$LINE: PROCEDURE (LINE);

    SET$BASES: PROCEDURE (BASE);
      /*  SET INDIRECT ADDRESSING FOR MULTIPROGRAMMING  */
      DECLARE
        I BYTE,
        BASE ADDRESS;
      MODE$B=BASE;
      MODE$STATE$B=BASE+1;
      LOW$PTR$B=BASE+2;
      HIGH$PTR$B=BASE+3;
      CK$SUM$B=BASE+4;
      HDG$VEC$B=BASE+5;
      DTG$B=HDG$VEC$B+HDG$VEC$LEN;
      FCB$B=DTG$B+DTG$LEN;
      DO I=0 TO FCB$LEN$M;
        DFCB(I)=FCB(I);
        END;
      ABUF$B=FCB$B+FCB$LEN;
      BBUF$B=ABUF$B+BUF$LEN;
      END SET$BASES;

    /*  SETS PARAMETERS FOR A PARTICULAR COMM LINE  */
    DECLARE LINE BYTE;
    DO CASE LINE;
```

```
      DO;  /*  LINE=0  */
        /*  NOT AVAILABLE NOW  */
        NO$DATA$IN=TRUE;
        END;
      DO;  /*  LINE=1  */
        PTR$INDEX=61;
        CALL SET$BASES(.MODE1);
        END;
      DO;  /*  LINE=2  */
        PTR$INDEX=62;
        CALL SET$BASES(.MODE2);
        END;
      DO;  /*  LINE=3  CONSOLE  */
        PTR$INDEX=63;
        CALL SET$BASES(.MODE3);
        END;
      END;
    IF FLICK$ON AND (MODE<>'W') THEN CALL FLICKER;
    END SET$COM$LINE;

START$UP: PROCEDURE;

  DISK$TBL: PROCEDURE BYTE;
    /*  BRING TABLE FROM DISK  */
    DECLARE
      (I,J) BYTE,
      MAIN$CTR ADDRESS;
    MAIN$CTR=0;
    CALL CLEAR$FCB;  /*  FCB(1)=0 IS SET  */
    CALL NEDS$OPEN;
    /*  ERROR IF UNABLE TO OPEN  */
    /*  IF ERROR CALL PRINT(.OUT25); RETURN FALSE;  */
    CALL BUF$WAIT(ABUF$B);
    ABUF=255;
    DO WHILE NEDS$READ(ABUF$B)<>1;
      CALL BUF$WAIT(ABUF$B);
      DO I=1 TO BUF$LEN$M;
        PTR$TBL(MAIN$CTR),J=ABUF(I);
        IF (MAIN$CTR:=MAIN$CTR+1)<=MAIN$TBL$END
          THEN IF J>63
            THEN DO;
              /*  ERROR.  TABLE INDEX OUTSIDE LIMITS  */
              CALL PRINT(.OUT09);
              RETURN FALSE;
              END;
        IF MAIN$CTR=MAIN$TBL$MAX$AND$END
          THEN RETURN TRUE;
        END;
```

```
      ABUF=255;
      END;
    /*  ERROR.  NOT ENOUGH RECORDS TO FILL TABLE  */
    CALL PRINT(.OUT25);
    RETURN FALSE;
    END DISK$TBL;

  CLEAR$TABLE: PROCEDURE;
    /*  SHOULD ERASE WHOLE DISK TO CLEAR CONFIDENTIAL  */
    /*  INITIALIZE MAIN TABLE  */
    DECLARE (I,J) BYTE;
    DO J=0 TO MAIN$TBL$END-1;
      CALL CLEAR$FCB;
      DFCB(1),FCB(1)=J;
      /*  ENSURE ALL DISK SPACE DE-ALLOCATED  */
      /*  CALL DELETE WITHOUT HEAD LIFT
          DIRECT CALL FOR SPEED  */
      CALL MONI(19,DFCB$B);
      LOW$PTR,PTR$TBL(J)=J;
      CALL CLEAR$M$ENTRY;
      END;
    CALL CLEAR$FCB;  /*  FCB(1)=0 IS SET  */
    CALL NEDS$MAKE;
    CALL FILE$TBL;
    END CLEAR$TABLE;

  TABLE$SET: PROCEDURE BYTE;
    /*  INITIALIZE PTR$TBL AND MAIN$TBL  */
    DECLARE CHAR BYTE;
    CHAR=BLANK;
    DO WHILE NOT Y$OR$N(CHAR);
      /*  IS DISK CURRENT  */
      CALL PRINT(.OUT18);
      DO WHILE NOT CSTS; END;
      CHAR=SCAN;
      END;
    IF CHAR='N'
      THEN RETURN DISK$TBL;
      ELSE DO;
        CHAR=BLANK;
        DO WHILE NOT Y$OR$N(CHAR);
          /*  DISK WILL BE ERASED  */
          CALL PRINT(.OUT26);
          DO WHILE NOT CSTS; END;
          CHAR=SCAN;
          END;
        IF CHAR='N' THEN RETURN FALSE;
        CALL CLEAR$TABLE;
```

```
        RETURN TRUE;
        END;
    END TABLE$SET;

  CLEAR$MOST: PROCEDURE;
    CALL CLEAR$ALL;
    MODE='W';
    MODE$STATE=3;
    END CLEAR$MOST;

  CALL RESET;    /*  RESET DISK  */
  DFCB$B=5CH;   /*  FOR PRESENT DOS  */
  DBUF$B=80H;   /*  FOR PRESENT DOS  */
  HS$CHAR='A';   /*  HIGH SPEED LINE DATA  */
  /*  RUN SYSTEMS TEST.
      MEMORY TEST AND I/O TESTS.  */
  FLICK$ON=FALSE;
  LINE$NUM=3;
  CALL SET$COM$LINE(1);
  DO WHILE NOT TABLE$SET; END;  /* CALL TABLE SET UNTIL FILLED  */
  CALL CLEAR$MOST;
  CALL SET$COM$LINE(2);
  CALL CLEAR$MOST;
  CALL SET$COM$LINE(3);
  CALL CLEAR$ALL;
  LINE$HAS$DATA=08H;
  HS$CTR=4000;
  /*  DO SET$BASES   CLEAR$MODE FOR EACH COMM LINE;  */
  END START$UP;

FLICKER$SET: PROCEDURE;
  /*  OUTPUT LINE NUMBER BEING PROCESSED  */
  IF LINE$NUM<CONSOLE$LINE
    THEN DO;
      CALL CO$NBR(LINE$NUM);
      FLICK$ON=TRUE;
      END;
  END FLICKER$SET;

FIND$A$LINE: PROCEDURE BYTE;
  /*  WOULD READ AN INPUT PORT FROM
      THE COMMUNICATIONS MODULE  */
  IF CSTS OR MODE$NOT$SET
    /*  MODE<>C,W,G          OR CSTS  */
    THEN LINE$HAS$DATA=LINE$HAS$DATA OR 08H;
  IF HS$CTR<3500 THEN LINE$HAS$DATA=LINE$HAS$DATA OR 02H;
  IF (INPUT(2) AND 01EH)=0 THEN LINE$HAS$DATA=LINE$HAS$DATA OR 04H;
  LINE$NUM=0;
```

```
    CALL FLICKER$SET;
    DO WHILE NOT SHR(LINE$HAS$DATA,LINE$NUM);
      IF FLICK$ON THEN CALL FLICKER;
      /*  IF NO DATA, RUN A SYSTEM TEST  */
      IF (LINE$NUM:=LINE$NUM+1)>CONSOLE$LINE THEN RETURN FALSE;
      CALL FLICKER$SET;
      END;
    RETURN TRUE;
    END FIND$A$LINE;

  HDL$LINE: PROCEDURE;
    DECLARE J BYTE;
    NO$DATA$IN=FALSE;
    CALL SET$COM$LINE(LINE$NUM);
    DO WHILE NOT NO$DATA$IN;
      /*  DO WHILE DATA IN BUFFER  */
      /*  COULD RETURN TO POLLING AFTER HANDLING
          X BYTES OR WHATEVER ROUTINE DESIRED  */
      CALL SCANIT;
      END;
    DO J=0 TO FCB$LEN$M;
      /*  DROP WITH DMA  */
      FCB(J)=DFCB(J);
      END;
    LINE$HAS$DATA=LINE$HAS$DATA AND (NOT SHL(01B,LINE$NUM));
    IF FLICK$ON THEN CALL FLICKER;
    END HDL$LINE;



      /*   MAIN PROGRAM   */

    BEGIN$PROGRAM:
    CALL START$UP;
    DO FOREVER;
      IF FIND$A$LINE THEN CALL HDL$LINE;
      END;

EOF
```

LIST OF REFERENCES

1. Backler, Jordan, " Disk Storage Devices", Digital Design Handbook, p. 59-75, December 1974.

2. Bradford, B. E., LCDR, USN, " Optimum Transmission of Environmental Data", Fleet Numerical Weather Central Technical Memo No. 14, p. 10-25, 1966.

3. Cushman, Robert H., " The Intel 8080: First of the Second-Generation Microprocessors", EDN Magazine, p. 30-36, May 1974.

4. Falk, Howard, " Microcomputers Software Makes Its Debut", IEEE Spectrum, p. 78-84, October 1974.

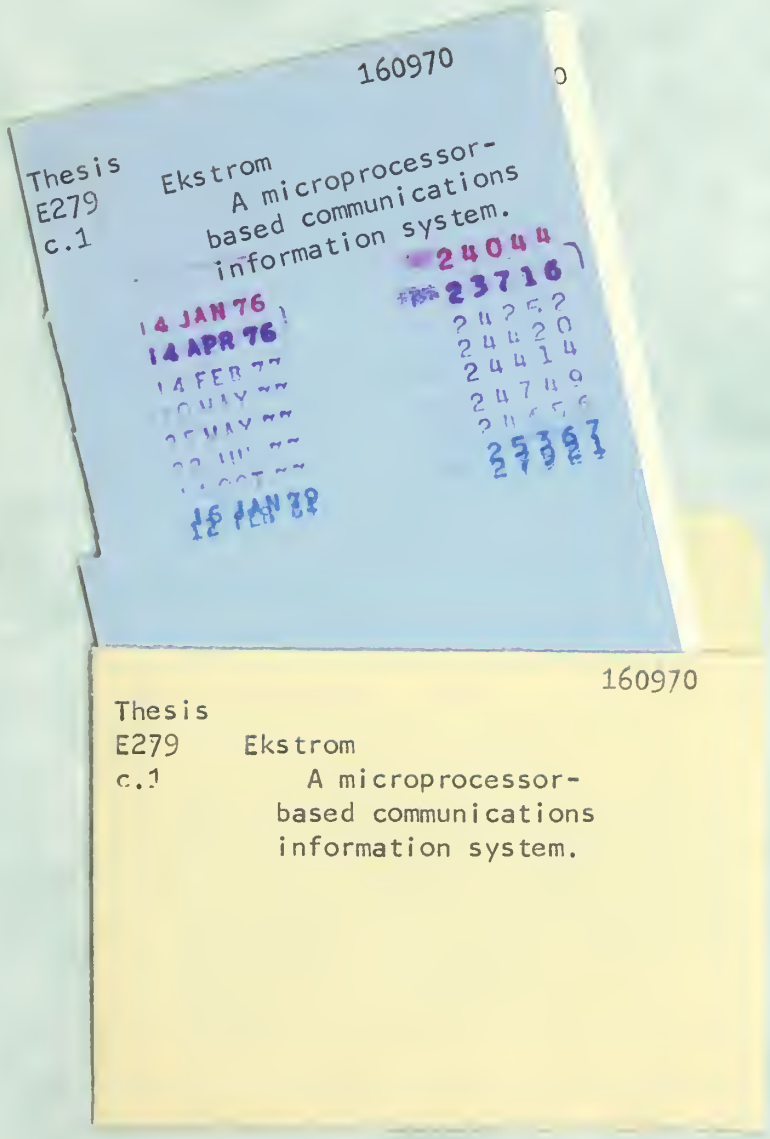
5. Falk, Howard, " Computer Systems: Hardware/Software", IEEE Spectrum, p. 38-43, January 1975.

6. Gray, R. J., LCDR, USN, Specifications Naval Environmental Display Station (NEDS) Operating System (NOS), p. 1-66, Fleet Numerical Central, March 1974.

7. Guide to PL/M Programming, p. 1-48, Intel Corporation, 1973.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Chairman, Code 72 Computer Science Group Naval Postgraduate School Monterey, California 93940	1
4.	Professor V. Michael Powers, Code 72Pw Naval Postgraduate School Monterey, California 93940	2
5.	LT Robert H. Ekstrom, USN 365 Mary Lane Crystal Lake, Illinois 60014	1
6.	LT William H. Reinhardt, USN 12875 Abra Place San Diego, California 92128	1



160970

Thesis
E279
c.1

Ekstrom
A microprocessor-
based communications
information system.