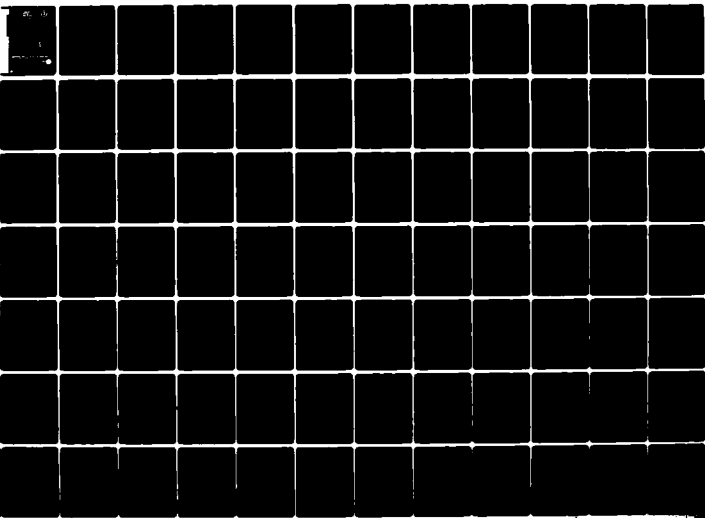
AD-A083 046 GEORGIA INST OF TECH ATLANTA SCHOOL OF ELECTRICAL EN--ETC F/G 9/2
THE FEASIBILITY OF IMPLEMENTING MULTICOMMAND SOFTWARE FUNCTIONS--ETC(U)
OCT 79 T P BARNWELL, J L HAMMOND, J H SCHLAG DAAG29-78-G-0139

UNCLASSIFIED ARO-15900.1-A-EL NL

1 OF 4
ADA
083 046



FINAL REPORT

LEVEL II

THE FEASIBILITY OF IMPLEMENTING MULTICOMMAND SOFTWARE FUNCTIONS ON A MICROCOMPUTER NETWORK

Principal Investigators:

T. P. Barnwell
J. L. Hammond
J. H. Schlag
E. B. Wagstaff

Submitted To:

U. S. ARMY RESEARCH OFFICE

Grant Number:

DAAG29-78-G-0139

Report Period Covering July 1, 1978 to September 30, 1979

October 1979

GEORGIA INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING
ATLANTA, GEORGIA 30332

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER 15900.1-A-EL	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) THE FEASIBILITY OF IMPLEMENTING MULTICOMMAND SOFTWARE FUNCTIONS ON A MICROCOMPUTER NETWORK		5. TYPE OF REPORT & PERIOD COVERED Final Report: 1 Jul 78 - 30 Sep 79 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) T. P. Barnwell, J. L. Hammond, J. H. Schlag, E. B. Wagstaff		8. CONTRACT OR GRANT NUMBER(s) DAAG29 78 G 0139
9. PERFORMING ORGANIZATION NAME AND ADDRESS Georgia Institute of Technology Atlanta, Georgia 30332		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS U. S. Army Research Office P. O. Box 12211 Research Triangle Park, NC 27709		12. REPORT DATE Oct 79 13. NUMBER OF PAGES 306
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

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

18. SUPPLEMENTARY NOTES

The view, opinions, and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

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

microcomputers
computer networks
feasibility studies
monitor systems
distributed processing

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

This report presents the results of a study of design considerations for hybrid monitor systems for distributed microcomputer networks. The objective of the study was to determine the feasibility of such monitor systems and to look at typical designs. A Detailed survey of the literature was carried out and the characteristics of existing monitor systems were established. The report presents a conceptual design for a monitor system for distributed microcomputer (cont.)

next page

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

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. ABSTRACT CONTINUED

networks obtained by adapting certain aspects of existing systems to the specialized requirements of microcomputer networks. Several novel features are incorporated into the design to minimize overhead and enhance useability. A typical implementation of the conceptual design using state-of-the-art hardware is given and its operation on a specific monitoring task is considered in detail. The implementation is recommended for use with the AIRMICS/GEORGIA TECH Experimental Network.

FINAL REP[redacted]T

THE FEASIBILITY OF IMPLEMENTING MULTICOMMAND SOFTWARE FUNCTIONS ON A MICROCOMPUTER NETWORK

PRINCIPLE INVESTIGATORS

T. P. BARNWELL
J. L. HAMMOND
J. H. SCHLAG
E. B. WAGSTAFF

SUBMITTED TO

U. S. ARMY RESEARCH OFFICE

OCT[redacted]79

School of Electrical Engineering,

GEORGIA INSTITUTE OF TECH[redacted]

Atlanta, Georgia 30332

FOREWORD

The work reported herein was performed under a grant from the U. S. Army Research Office in support of work for the U. S. Army Computer System Command Institute for Research in Management Information and Computer Sciences. The study was one task on a project entitled "The Feasibility of Implementing Multicommand Software Functions on a Microcomputer Network".

Principal investigators on the project are Dr's. T. P. Barnwell, J. L. Hammond, J. H. Schlag and E. B. Wagstaff. Dr. J. H. Schlag is the program manager.

ABSTRACT

This report presents the results of a study of design considerations for hybrid monitor systems for distributed micro-computer networks. The objective of the study was to determine the feasibility of such monitor systems and to look at typical designs.

A detailed survey of the literature was carried out and the characteristics of existing monitor systems were established.

The report presents a conceptual design for a monitor system for distributed microcomputer networks obtained by adapting certain aspects of existing systems to the specialized requirements of microcomputer networks. Several novel features are incorporated into the design to minimize overhead and enhance useability.

A typical implementation of the conceptual design using state-of-the-art hardware is given and its operation on a specific monitoring task is considered in detail. The implementation is appropriate for use with the AIRMICS/GEORGIA TECH Experimental Network and its use for this purpose is recommended.

TABLE OF CONTENTS

TABLE OF CONTENTS

(Continued)

TABLE OF CONTENTS

(Continued)

TABLE OF CONTINTS

(Continued)

TABLE OF CONTENTS

(Continued)

TABLE OF CONTENTS

(Continued)

TABLE OF CONTENTS

(Continued)

TABLE OF CONTENTS

(Continued)

TABLE OF CONTENTS

(Continued)

TABLE OF CONTENTS

(Continued)

TABLE OF CONTENTS

(Continued)

TABLE OF CONTENTS

(Continued)

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

This report presents the results of a performance monitor feasibility study performed as one task under a grant entitled "The Feasibility of Implementing Multi-Command Software Functions on a Microcomputer Network" from the United States Army Computer Systems Command Institute for Research in Management Information and Computer Sciences.

The objective of the study was to investigate the feasibility of using combined hardware/software monitors for distributed microcomputer networks.

The field of computers in general, and computer networks in particular, is undergoing explosive growth. Extremely rapid advances in hardware, such as the advent of the microprocessor, have made possible designs for distributed computer systems which could not have been cost effective even a few years ago.

To keep abreast of the rapidly changing state-of-the-art, AIR-MICS is concerned with the potential applicability of distributed database microcomputer networks to their data processing and management information problems. The present grant provides funds to study several aspects of microcomputer networks to assess their applicability to these problems.

This report is concerned with a part of the overall study directed toward monitor systems for distributed microcomputer networks. Monitor requirements for distributed microcomputer networks are developed using monitor systems for existing networks as a guide. Feasible design approaches are developed to satisfy the requirements.

Other parts of the study have indicated that packet switching, as opposed to alternative approaches such as line switching, is the most cost effective switching technique to use with the microcomputer networks for the AIRMICS application. Thus when aspects of the monitor system are impacted by such details, a packet switching network is assumed.

The remainder of the report is divided into seven major parts: Section 2, which gives a detailed survey of the literature on the monitor problem; Section 3, which develops an overall design for a hybrid monitor system for distributed microcomputer networks; Section 4, which is a study of a specific implementation of such a monitor system; Section 5, which is a description of the experimental network; Section 6, which details Netwoık Cobol; and Section 7, Conclusion.

2. REVIEW OF THE LITERATURE

Section 8 of the report contains a bibliography of selected papers under the headings: Hardware Monitors for Stand-Alone Computers, Software Monitors for Stand-Alone Computers, Hardware/ Software Monitors for Computer Networks, Parameters to be Measured for Monitoring, Existing Computer Networks, Analytic and Simulation Models for Computer Networks, Measurements for Determining Parameters for use with Network Models, and Commercial Monitor Equipment.

The purpose of this section is to provide a concise guide to this literature in several areas germane to the major thrust to the study.

2.1 Existing Computer Networks

Specialized computer networks began to appear in the middle and late 1960's and since the early 1970's have been implemented for commercial service. As could be expected, there is a considerable body of literature on all aspects of computer networks.

Computer networks can be classified in a number of ways using, for example, application, type of hosts, geometry or method of switching. The method of switching has a significant effect on certain aspects of the monitoring problem and thus this classification will be used to narrow the scope of the present survey.

Major types of switching for computer networks can be classified as nonswitched or leased-circuit, circuit-switched, packet-switched and multiple access. Halsey, et.al. (1979) (Ref. 35)*

---

*Numbers refer to Bibliography in Section 8.

surveys the public data networks world-wide in the first three categories and enumerates fifteen networks of the leased-circuit type and seventeen of the packet-switched type.

Leased-circuit and circuit-switched networks were the first types to be used and much existing theory and equipment were developed for this type of network. As noted in the Introduction, however, the interest in this study is in packet-switched networks which are a more recent innovation. Wood (1975) (Ref. 34) surveys eight packet-switching networks from countries around the world, including the ARPANET, which is possibly the oldest and best documented U.S. packet switching network. At the time of this survey, the hosts in the networks examined were large computers. The ARPANET, in particular, is well monitored and the equipment is discussed in detail in the literature. See Kleinrock (1974) (Ref. 33).

Minicomputers are a recent innovation and thus the number of papers describing minicomputer networks would be expected to be relatively limited. Five papers describing reasonably general purpose minicomputer networks were found in the literature. Three of these papers, Fraser (1975) (Ref. 31), Aiso, et al. (1975) (Ref. 29) and Kitazawa, et al. (1978) (Ref. 30), describe networks which share a common bus controlled by a switching computer (or computers). Farber (1975) (Ref. 27) describes a network using what he terms a "communication ring" controlled by distributed ring interfaces. Unfortunately, none of these four networks are felt to be an optimum choice for the present application since they do not efficiently handle bursts of traffic between nodes as a packet switched network would.

Labetoulle (1977) (Ref. 26) describes a network which is possibly the best suited to applications of the type of interest in the present study. He gives attention to the bursty nature of communications between nodes and considers packet switching as a possibility. However, from considerations of the hardware costs at the time of his study (before 1977), he chooses a communication loop based on the Newhall-Farmer protocol, rather than using packet switching. Labetoulle does not consider the monitoring problem.

2.2 Hardware Monitors and Software Monitors for Stand-Alone Computers

Hardware and software monitors for stand-alone computers have been in use for a number of years and there is a considerable amount of literature on the subject. The book by Svobodova (1976) (Ref. 24) contains a section on hardware and software monitors and an extensive bibliography. Typical of several earlier survey papers with references is the one by Lucas (1971) (Ref. 10).

To a large extent, hardware and software monitors are complementary in that they have access to different aspects of the computation. There are some activities, however, such as CPU activity, which are observable by both hardware and software.

A software monitor is a special program incorporated into the software of the system under test. Through use of commands, such as interrupts, codes can be written to monitor many parameters of the system.

Hardware monitors are typically some sort of "black box" which measures certain system parameters through direct wired-in connections. A complete hardware monitor also requires control logic, accumulators, and a recording unit.

As pointed out by Svobodova (Ref. 24), a software monitor can observe hardware-related events only if they are accompanied by a control transfer to an instruction at a known logical address or if they store other identifying information.

On the other hand, a hardware monitor can sense software-related events only when they are accompanied by a control transfer to a fixed absolute address. This is possible because hardware monitors can normally monitor the state of any memory element.

Hardware monitors require no system overhead while software monitors can be costly in the use of resources.

A hardware monitor is well suited to the task of counting or timing the duration of events or combinations of events, where the term event is used to denote any occurrence of significance to a unit of work processed by the system. Cockrum and Crockett (Ref. 1) present a good study of the use of hardware monitors for event monitoring. They list events which can be monitored by single sensors under four headings: fourteen events for the Central Processor Unit, seven events for the Direct Access Storage Device, four events for the Control Units and four events for Unit Record Equipment. They also list five types of events which require multiple sensors and comparators and provide examples of how to determine the combined events.

One source of data that can be accessed by a hardware monitor is the memory bus. Fryer (Ref. 8) discusses in some detail what can be found on the memory bus and also gives details of the required monitors. He points out that the memory bus has three types of information, namely: 1) address lines which specify which memory location is to be accessed, 2) data lines carrying the data read or to be written, and 3) control information which includes a read/write line and sometimes a split cycle line for read-modify-write operations. Fryer states that measuring the actual execution time of a section of code is easily accomplished with a bus monitor.

Typical general software monitor tools which have been implemented are the following:

- metering packages for time spent in executing selectable supervisor modules while the system is running other tasks
- packages for obtaining the distribution of segment utilizations
- packages for counting the number of times specified procedures are called
- general event tracing packages.

Some software monitor systems have been tailored to give data for use with specific analytic models. A software monitor for use with a queueing theory multiprogramming model of an IBM 360/65 under OS/MFT using the HASP Execution Task Monitor is described by

Wong and Strauss, (Ref. 14). This monitor system is composed of two programs. The collection program which collects the required data and dumps the information on magnetic tape and the analysis program which processes the data collected. The collection program periodically samples the OS/360 system tables and control blocks by disabling all I/O interrupts, collecting the required data, and then enabling the interrupts again. The data of interest is CPU activity, the priority mapping of certain tasks, I/O queueing activity and I/O activity of the devices on the selector channels.

2.3 Hardware/Software Monitors and Monitors for Computer Networks

The general design characteristics of a hybrid, or hardware/software monitor, for a stand-alone computer are discussed by Svobodova (Ref. 24). A specific design for an elaborate hybrid monitor for computer networks is discussed in detail by Morgan and his coworkers (Refs. 16, 17). The design of a monitor system for a specific computer network is illustrated by the monitor system for the ARPA network (Ref. 18).

Hybrid monitor systems attempt to exploit the desirable features of both hardware and software monitors. Svobodova describes a two level hybrid monitor structure. One level consists of software for detecting software-related events, for controlling which events are monitored and for generating signals detectable by an external hardware monitor. Another level consists of an external hardware monitor which combines signals from the software monitor with hardware probe signals and processes and outputs the results. The interface between the software monitor and the external hardware monitor is provided by an M-register (which is a set of hardware latches) set and reset by the software to providing external connections for the hardware monitor.

Morgan and his coworkers developed the design of a system of hardware and software devices for monitoring the behavior of a computer network. The monitor system is distributed so that each node in the computer network is provided with a "remote controlled hybrid monitor" and a "regional network measurement center". Communication lines couple all of the regional network measurement centers to one "network monitor control".

The *remote controlled hybrid monitor* is a general device containing event detectors and time measuring modules as well as data processing and storage equipment and communication modules. The event detector can detect the following:

1. events defined in terms of data or address ranges
2. events defined in terms of Boolean functions of other events
3. events defined as a sequence of other events
4. characters in bit-serial lines.

The time measuring modules contain four types of devices:

1. time stamp units
2. event times
3. interval times
4. a network clock synchronized with a standard reference clock.

Although the general devices could be adopted to do so, specific attention is not given to measuring features of a packet switched network, such as message delay and traffic.

The monitor system for the ARPA network typifies a system whose major function is to monitor the performance of a packet switched computer network by measuring input traffic, line traffic and message delays. The monitor is limited to determining the behavior of the communication subnetwork which provides the message service to the user-host system. The monitor functions are implemented in software at the switching computers (IMPS) located at each node in the network. All of the monitor equipment is under program control and, upon request, data can be collected at specific nodes and summarized in special measurement messages which are sent to a specific collection Host.

Six measurement tools are implemented for the ARPA system. A Trace tool allows messages to be "traced" as they pass through a sequence of IMPS. A trace block is generated for each marked packet. The trace block contains time stamps which occur when: (a) the last bit of the packet arrives, (b) the packet is put on a queue, (c) the packet starts transmission and (d) the acknowledgement is received.

Another measurement tool is the Accumulated Statistics message which consists of several tables of data summarizing activity at a network node over an interval of time. These statistics include: (a) message size statistics such as histograms of packet lengths in words for large packets, (b) a global traffic matrix containing such data as the number of round-trips sent from a probed site to each site, and (c) channel statistics for channels connected to a probed site.

A Snapshot tool gives an instantaneous look at the operation of an IMP. Snapshot data includes: several queue lengths, the IMP's routing table, lost queue lengths, and data about storage allocations.

An Artificial Message Generation tool is a package built into each IMP giving it the ability to generate artificial messages. The two remaining tools are Status Reports and Control, Collection and Analysis.

2.4 Parameters Measured by Monitoring Systems

In principle, it should be possible to identify a minimal set of states, or parameters, which will completely describe a computer system or computer network. Identification of such a set of parameters, however, has not been found in the literature and apparently is beyond the state of the art at the present time.

Although a minimum set of parameters to be monitored is not identified, several authors, including Svobodova (Ref. 24), Cox (Ref. 20) and Morgan (Ref. 16) identify general sets of parameters and the authors of the papers referenced in Sections 2.2 and 2.3 all identify the variables measured by their monitoring tools. A compilation of the variables from these sources has been made. A similar compilation made by Sutton and Morgan (Ref. 46) contains essentially all of these variables and it is given with minor additions in Table 1.

The parameters have been classified under the three general headings of Computer Network Parameters, Workload Parameters and Miscellaneous Items. The first category refers to those variables internal to any part of the computer network. This category is further subdivided into Utilization of Resources, Throughput, and Response.

Workload Parameters are parameters associated with the external load on the network, while the Miscellaneous category includes those parameters which do not fit into the first two categories.

TABLE 1. PARAMETERS MEASURED BY MONITORING SYSTEMS (Adapted from Sutton and Morgan with minor additions)

1. COMPUTER NETWORK PARAMETERS

Utilization of Resources

a. Frequency of

- Specific software activity. This includes system software, utilities, and a part or whole of the operating systems of nodes or hosts.
- Processor activity
- Line or Link activity
- Channel or controller activity
- Auxiliary or main storage device activity
- Data set activity
- Data set structure activity
- Processor states
- Instruction execution.

b. Quantity of auxiliary or main storage space requested or used.

c. Quantity of data moved to or from specific devices.

Throughput

a. Time required to transmit/handle a message/packet through a network node or other specific resource.

b. Number of messages, packets or jobs handled by a node, network or host.

c. Number of bits transmitted or received by a link, line node, network or host.

d. Raw speed of a resource.

e. Time between dispatch of packets, messages or jobs.

Response

a. Time to set-up or disconnect a logical or physical path through a network or node.

b. Time required to respond to a call for service.

2. WORKLOAD PARAMETERS.

a. User response time (or think time).

b. Time between arrivals of packets, messages or jobs.

c. Frequency and types of requests for service.

d. Reference pattern of software.

e. Size of packet, message or job in characters, lines or cards.

f. Real time on the system.

g. Quantities and types of storage requested and used.

3. MISCELLANEOUS ITEMS.

a. Time for the object system to detect, correct or recover from trouble with data transmission; lines, nodes, hosts or specific devices out of service; software errors, and link problems.

b. Time for the object system to detect saturation of lines links, nodes, hosts or other devices.

c. Number of packets, messages or jobs within the system and the number of jobs active.

d. Size of queue.

2.5 Commercial Monitoring Equipment

In the course of the literature survey the characteristics of general purpose commercial monitoring equipment were examined. This task was facilitated by two survey papers, one by Stiefel (1979) (Ref. 52) concerned with network diagnostic tools and another by Hart, et.al. (1971) (Ref. 51) concerned with monitoring host-controlled resources.

The paper by Stiefel summarizes the properties of thirty-eight different pieces of test equipment ranging in price from twenty-nine dollars to seventeen thousand dollars. This array of equipment tests such things as modem performance, polling, response time, and link quality. There are units to carry out software debugging, fault testing and related tasks. Other units provide an RS-232 status monitor and measurements to test computer terminals.

Most of the test instruments, however, are tailored for leased-line or circuit switched networks. None of the applications listed indicates measurement of packet-switched network parameters such as packet delay, queue length, etc. Thus, one must conclude that, although some specific measurement techniques could be applicable, none of the instruments described could serve, directly, the desired network monitoring function.

The instruments described by Hart for measuring host-controlled resources also cover a variety of costs and complexities. One or another of the instruments would seem to provide all of the types of measurements desired for host-controlled resources. The problem with these instruments, however, is that of interfacing and adapting a general purpose instrument to specific tasks. In almost all cases, the general purpose instruments are tailored for use with large scale, multiprocessing computers, whereas the present application is concerned with microcomputers which perform essentially one task at a time.

3. OVERALL DESIGN CONSIDERATIONS FOR HYBRID MONITORS

3.1 General Requirements for a Monitor System

Section Two contained a summary of the parameters measured by existing monitor systems and the monitoring tools used by certain large scale computer networks. In the light of this information, the problem of conceptual design of monitoring equipment for distributed microcomputer networks would seem to be one of adaption to specialized properties and needs. This section of the report presents general design considerations for a monitoring system specifically tailored to a distributed microcomputer network using packet switching. The network is assumed to contain a relatively small, but arbitrary, number of nodes distributed in space, as indicated in Figure 3.1. The switching computers, which are small scale versions of the ARPA IMPS, are located at each node and control the flow of packets into and out of the nodes over the connecting communication links.

From a consideration of their characteristics, several distinctive properties of microcomputer networks can be identified. These properties translate into the following specific requirements for a distributed microcomputer monitoring system.

1) The host microcomputers at each node perform essentially one operation at a time under control of the CPU. Thus, monitor equipment at each node can be designed to monitor only one operation at a time. Such monitor equipment can be simpler than that required to function in a multiprocessing environment.

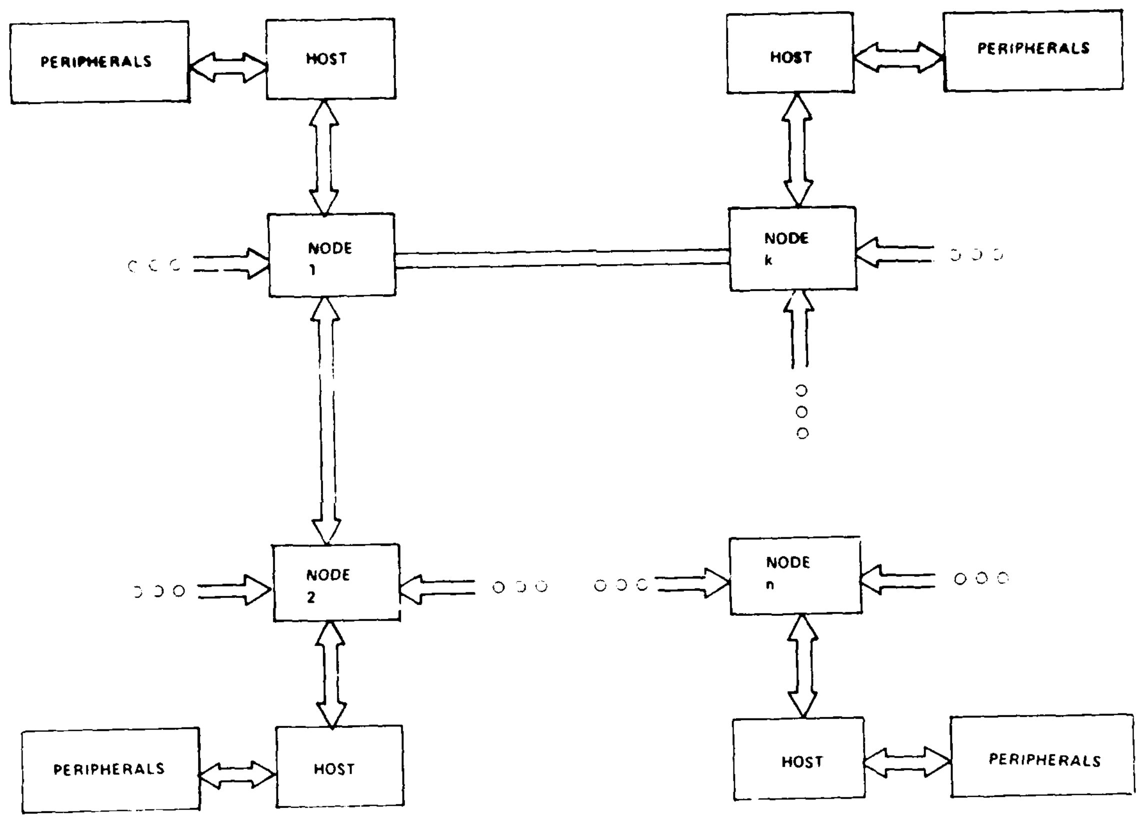


FIGURE 3.1 A DISTRIBUTED MICROCOMPUTER NETWORK

2) Queueing theory models may be useful for describing micro computer networks and data appropriate to such models should be obtained.

3) In applications of microprocessor networks, it is desirable to monitor total resource utilization for each job and for each task of which the job is comprised.

4) Microprocessor equipment is evolving at a rapid rate. Hybrid monitor systems should, therefore, be designed to take advantage of what is currently feasible, such as having a microprocessor as a part of the monitor equipment at each node when this can be useful.

In addition to the specialized properties listed above, monitor systems for distributed microcomputer networks have the following properties in common with other such systems:

5) The monitor system should be controlled from a central location,

6) The monitor system should require a minimal overhead, and

7) Results from monitor measurements should be presented in a form which is as useful as possible to the ultimate user of the network.

Of course, specific implementations of monitor equipment must be tailored to particular hardware and software for each computer network.

3.2 Specific Variables to be Monitored

A consideration of the variables measured by monitor systems reported in the literature and of the specialized requirements for distributed microcomputer networks leads to the following choices for variables to be monitored. The variables are listed on two levels - the variables employed by the end user of the network, and the more basic measured variables from which these are derived.

The variables desired by the user of the computer network are those required to characterize job performance - typically total resource utilization and total computing time on a per task or per job basis. For an experimental network, it is also desirable to measure a set of variables which will characterize the behavior of the network in transmitting data between the host computers.

The basic measured variables for resource utilization involve the total time devoted to each task or job by all of the host microcomputers, the host peripherals and the components of the network. This translates into a measurement of the total time devoted to each task by the following:

At each node

- host cpu
- host disk
- line printers
- terminals

For the network

- all links
- all node cpu's

The total computing time is measured directly from sign-on to sign-off at the appropriate terminal.

To characterize the network, it is necessary to determine the behavior of packets in moving from node to node and waiting in queues to be transmitted. The appropriate variables are random with time and thus the basic measured data is used to construct histograms or averaged to determine such statistics as the mean or variance. The set of variables listed below has been chosen to describe the network functions:

<u>at each node</u>

- packets awaiting service
- packets arriving per unit time
- number of packets transmitted per unit time over each link
- number of transmitted packets not acknowledged.

<u>for the whole network</u>

- packet delay over each path
- number of packets in the network at a particular time.

In addition to the variables noted above, additional measurements, such as time spent in executing portions of the software package, may be required. Some provision for this type of measurement will be made in the proposed monitor system.

3.3 A Proposed Monitor System

A consideration of the general requirements listed in Section 3.1 and the specific variables to be monitored as listed in Section 3.2 has led to the design of a general monitor structure and a philosophy to accomplish the required task. The design centers on five specific types of problems; namely, a general approach to the measurement tasks, nature and physical location of monitor components, communication between the parts of the monitor system, control of the monitor system, and identifying and accounting for specific jobs.

3.3.1 Nature and Physical Location of Monitor Components: The proposed monitor system has a Monitor Control (MC) location at one designated node and Monitor Stations (MS) at each of the other nodes of the network. Each nodal monitor station contains a microprocessor, memory, a serial port connecting to the node switching computer and a collection of sensors interfacing with the host computer at that node to measure the use of the resources controlled by the host. The equipment at a typical node is shown in Figure 3.2.

Each nodal monitor station will also share a two-port memory* with the switching computer to facilitate monitoring the network resources. Appropriate data concerning the operation of the network can be stored in this two-port memory by the switching computer and accessed by the monitor system. By choosing the read-write rate for the two-port memory to be twice the system clock rate, the monitor will require effectively no overhead in this operation.

Each nodal monitor station will collect all necessary data for its node from the host and its peripherals and also from the switching computer. In cases where it is appropriate to do so,

*The idea for this type of sensor was originated by Drs. Barnwell and Schlag.

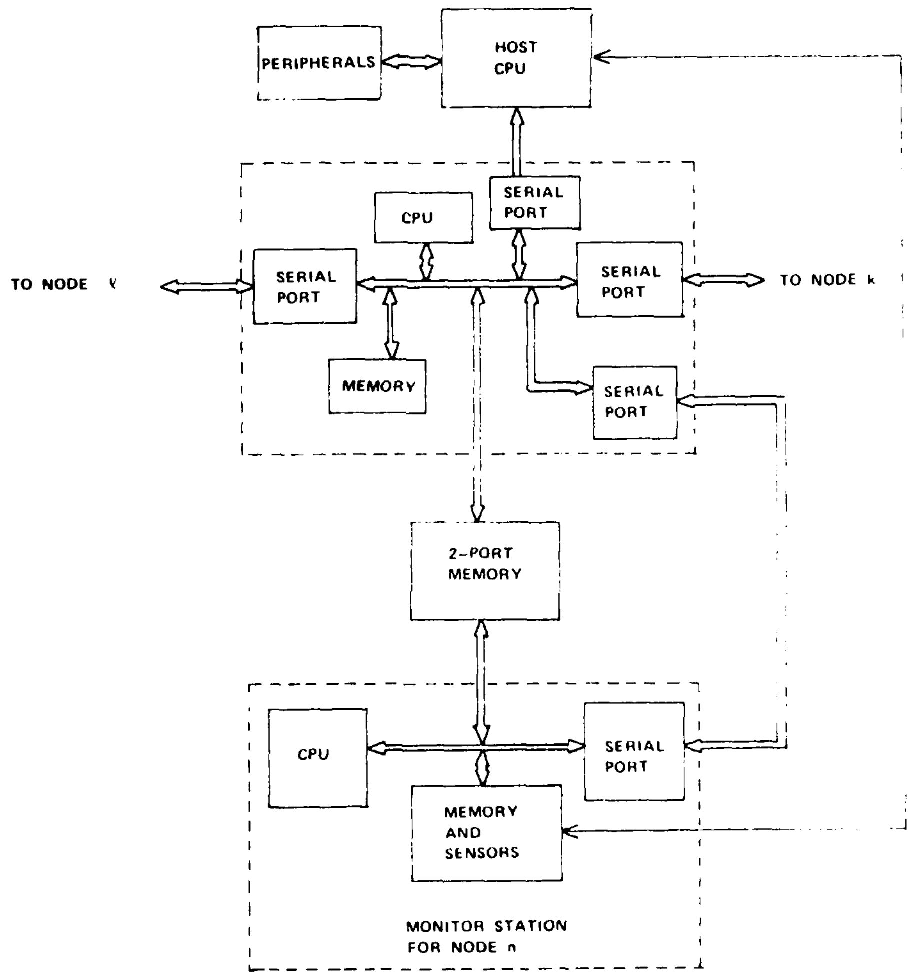


FIGURE 3.2 A TYPICAL NODE IN A COMPUTER NETWORK WITH ASSOCIATED MONITORING EQUIPMENT

preliminary data processing can take place at the node. For example, the mean value of a set of data can be determined. At periodic intervals, data from the nodal monitors will be transmitted to the Monitor Control Location.

3.3.2 General Approach to the Measurement Tasks: The measurement tasks will be treated in two parts, those associated with microprocessor host controlled resources and those associated with network resources.

The host-controlled resource activities at each node will be monitored directly and assigned to the job on which they are used. This is not a difficult task since a microcomputer CPU can control only one task at a time, and hence the resource activities controlled by such CPU's do not overlap.

The network functions are controlled by the CPU's of the switching computers, and therefore, network activity can overlap activity of the host controlled resources. Allocating the use of every resource of the network directly to the specific job on which it is used would be a difficult task. Therefore, it is planned to monitor every network resource but to allocate the cost to jobs on an average basis by measuring the number of packets used per job, the particular node-pair links traversed by the packets and the total traffic load at the time of transmission. A calibration of the network will be made to give the average cost, in terms of resource utilization, of transmitting packets over each node-pair link as a function of total traffic load over that link.

It is felt that this approach will minimize implementation difficulties while providing adequate accuracy.

3.3.3 Communication Between the Parts of the Monitor System: As noted above, the monitor stations are distributed throughout the network to facilitate collecting data at each node. This distribution of the monitor components, while desirable, necessitates transmitting data to the MC location by some means.

One method for data transmission which would not require overhead is that of constructing a monitor communication network to match that of the original computer network. This alternative was discarded as too costly in equipment.

The approach which was chosen is that of transmitting monitor data through the network in the same manner as data is exchanged by the host computers - by packets. This choice requires overhead since the monitor packets compete with the data packets for use of the network. The exact amount of overhead required, however, depends on the frequency of sending monitor packets and it is felt that this frequency can be kept low. A desirable aspect of the use of monitor packets is the fact that these packets can also be used to collect data on packet delay, transit times and other aspects of the operation of the network.

A scheme for generating monitor packets, called pickup packets, could have the packets originate either at the MC location or at the individual nodes. Generation at the MC location has tentatively been chosen as the best alternative.

The pickup packets will contain a data field and addresses structured in the same manner as other packets. The MC will dispatch the pickup packets at regular intervals, routing them so that at least one packet will traverse each link in the network before they all return to the MC. The routing details depend on the structure of the network as well as the specific routing strategies.

As each pickup packet arrives at a node, a real time measurement will be made and the time of arrival will be entered into an appropriate location in the data field of the pickup packet. A similar measurement will be made when the packet leaves the node. This data will be coded as to the pickup packet to which it applies, stored and then transmitted in the data field of the next pickup packet. The timing data collected by the pickup packets will ultimately be processed by the host at the MC to determine average packet delay and related parameters.

Whenever a pickup packet arrives at a node, all monitor data awaiting transmission to the MC will be placed, appropriately coded, into its data field. After traversing its portion of the network, the pickup packet will return to the MC and deliver the monitor data acquired in route. Thus the pickup packet will serve the dual role of transmitting data from the monitor stations to the monitor control and probing the network to determine packet delay and related parameters.

3.3.4 Control of the Monitor System: Control of the monitor system will reside at the MC location. Final data processing and monito data printout will take place at the MC and programs to control the monitor equipment at the various nodes can also originate and be distributed to the monitor stations through the MC.

Each nodal monitor station will contain an EPROM memory which will contain subroutines appropriate to controlling the monitoring equipment for any given task or job. These instructions will apply to all sensors, including the dual-port memory at that node.

In setting up a particular experiment, desired measurements will be specified as inputs to the MC. The MC host computer will then determine what measurements must be carried out at each node to obtain the desired data and will prepare appropriate programs for transmission to RAM memory at each node. The required program will be transmitted from the MC via a preliminary set of pickup packets.

With a small number of nodes in a central location, the RAMs at each node could, alternately, be programmed through a terminal at the node.

In addition to the task of setting up each experiment to be monitored, the MC must collect, process and output all monitor data. Instructions for doing this will be placed in EPROM memory at the MC location.

Note that the programs placed in RAM memory to control particular experiments will consist largely of calls to subroutines stored in EPROM memory. Thus, such programs will be short and easy to prepare.

3.3.5 Identifying and Accounting for Specific Jobs: The distributed microcomputer network will typically be processing a number of jobs concurrently. One requirement of the monitor system is that it be able to determine the cost, in resources used, for each job independently.

As noted above, host controlled resource use will be assigned directly to specific jobs, while network resource use will be assigned on an indirect basis. The accounting procedures are as follows.

Requests for host-controlled resources at each node are assigned an ID number associated with each job. This number is placed in a memory location accessed by the nodal monitor station, such as one in the two-port memory, and it remains there as long as the CPU controls a resource used on this job. The ID number is changed when the CPU or its peripherals perform a task for another job.

The monitor routines can be set up to use the ID number in initiating and ending measurements and in determining the memory locations for storing measured results. The procedure allows the nodal monitor stations at different nodes to monitor the activities associated with different jobs.

To allocate the network resources to various jobs, the job ID number is recorded in an appropriate location on each packet associated with carrying out the job. Monitor equipment is designed to count the number of packets associated with each job and to record the path traversed by each packet and the average traffic load on the path at the time of transmission. This data, along with a calibration of packet processing costs, can be used to allocate network costs to specific jobs.

4. STUDY OF A TYPICAL IMPLEMENTATION OF A HYBRID MONITOR SYSTEM

The objective of the study is to assess the general problem of hybrid monitors for distributed microcomputer networks. It is felt, however, that no general design study is complete without putting design concepts to the test of at least one possible implementation. This section of the report presents an implementation of a hybrid monitor system and an assessment of it in monitoring a typical job assigned to the computer network. This section also contains comments on a monitor structure for the AIRMICS/GEORGIA TECH Experimental Network.

4.1 An Implementation of the Monitor System

The Nodal Monitor Station shown in Figure 3.2 can be implemented with one of several appropriate microprocessor systems. Figure 4.1 shows a possible implementation with components from the American Microsystems S6800 family.

The operation of the Nodal Monitor Station is discussed under three headings: General Operation, Specific Measurement Modules and Representative Specific Measurements.

4.1.1 General Operation: The Nodal Monitor Station receives data in three ways: by reading memory locations in the Dual-Port RAM, through the Serial Communication Port and from the Data Gathering system. The servicing of these inputs and the storing of data into the RAM memory is carried out under the control of the microprocessor.

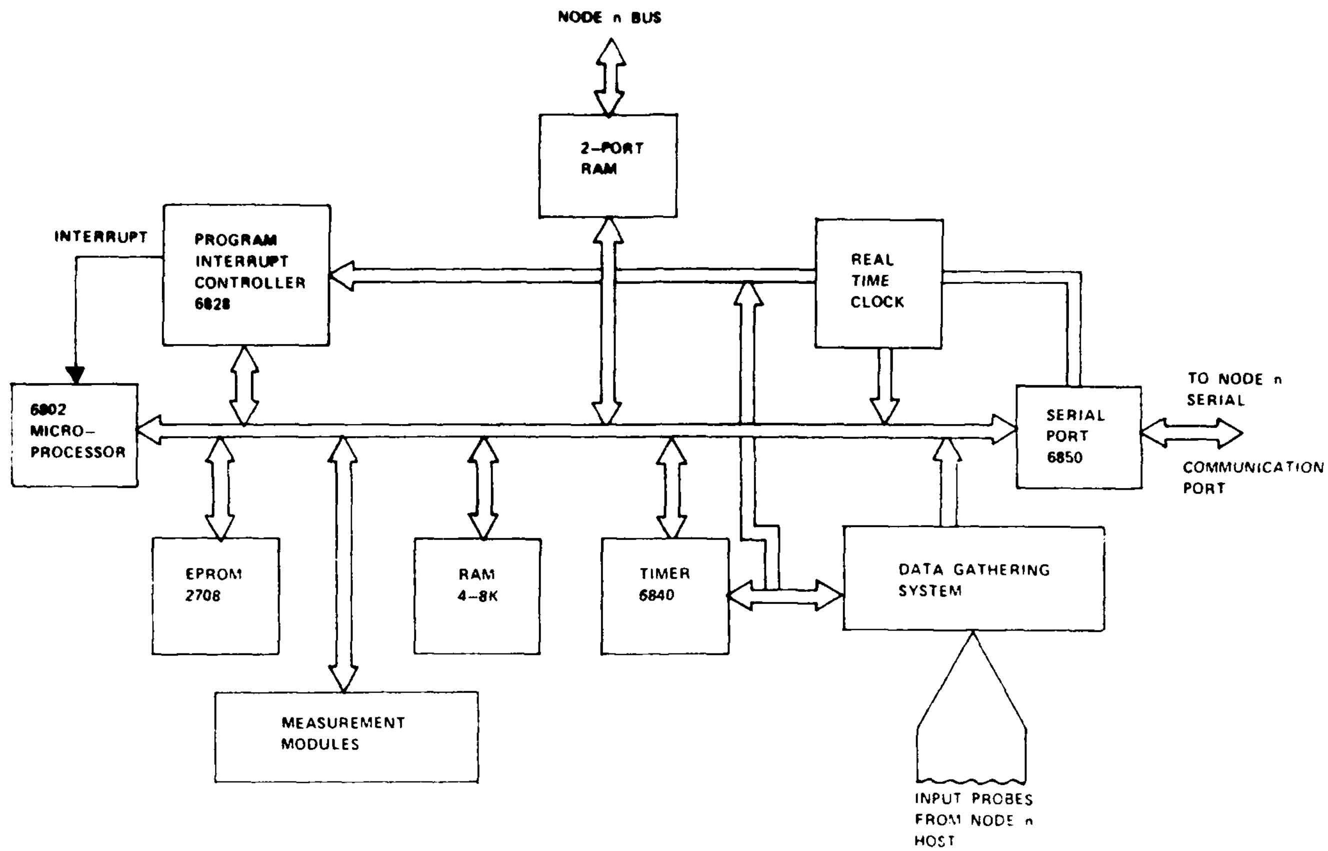


FIGURE 4.1 AN IMPLEMENTATION OF THE BASIC NODAL MONITOR STATIONS

The EPROM contains the basic subroutines which control all of the functions of the Monitor Station. Details of what data is taken and in what sequence measurements are made need to be flexible and read into the system for each particular experiment. This is accomplished by storing, for each experiment, a program in the RAM consisting largely of calls to the subroutines stored in the EPROM. Experiments can be set up from the Monitor Control Location using pickup packets sent out through the network to read in the programs. Alternately, the RAM can be loaded locally through a terminal associated with the local host.

As noted in Figure 4.1, the Nodal Monitor Station has a Data Gathering System which collects data from probes into the Micro-Computer host. These probes provide data on such things as the status of devices and are used with the Timer, the Real Time Clock and several standard Measurement Modules to monitor the host-controlled resources. The Measurement Modules are discussed below in the section on specific modules.

The host-controlled resource measurements are carried out by a program executed by the Monitor Microprocessor which uses specific software from RAM storage and general subroutines from the EPROM. The program, diagrammed in Figure 4.2, runs in an "infinite loop". The program is designed to be interrupted by events associated with the network, namely:

- to read data from the Two-Port RAM at regular intervals, and

- to process data to and from pickup packets.

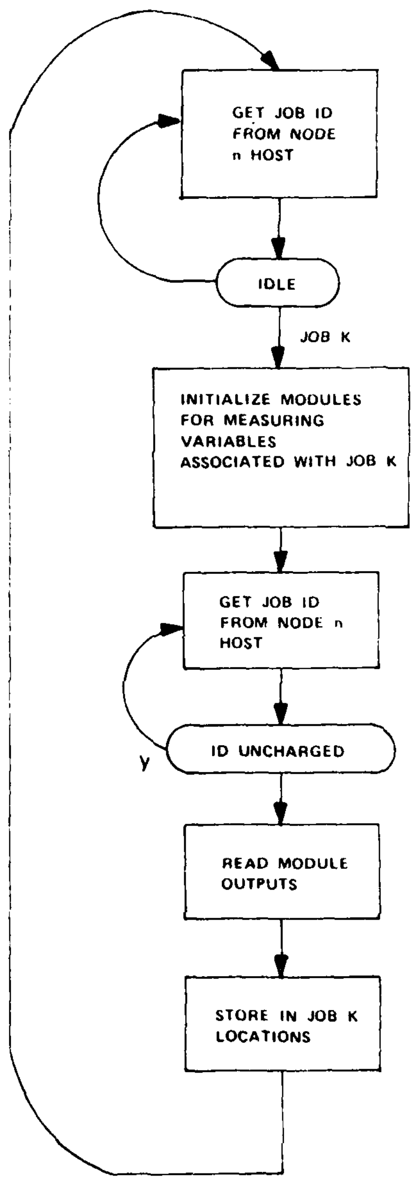


FIGURE 4.2 PROGRAM FOR CARRYING OUT HOST-CONTROLLED RESOURCE MEASUREMENTS

The network resources are monitored though data stored in appropriate locations in the Two-Port RAM. This device is also discussed in the section on specific modules.

Communication between the Monitor Control and the Nodal Monitor Stations will take place using pickup packets. The EPROM will have a basic routine which enables the CPU to communicate data through the Serial Port. Thus, data can be transferred to or from pickup packets which are in buffers at the node corresponding to the nodal monitor station. Arriving pickup packets will cause an interrupt in the monitor microprocessor program to ensure prompt service of the pickup packets.

<u>4.1.2 Specific Measurements Modules</u>: The measurements of the variables required to monitor a distributed microcomputer network can be carried out using several basic types of measurement modules. These modules include counters for time and events, a histogram generator, a masked-word range comparator and a logic combination device. The logical structure of these modules will be given in this subsection. Subsection 4.1.3 discusses how a number of the basic variables are measured using these modules. The Two-Port RAM and a Real-Time Clock will be included as modules in this discussion.

The real time clock and counters for time and events are shown in Figure 4.3. One Real Time Clock is required at each monitor station along with possibly one half dozen time counters and a similar number of event counters.

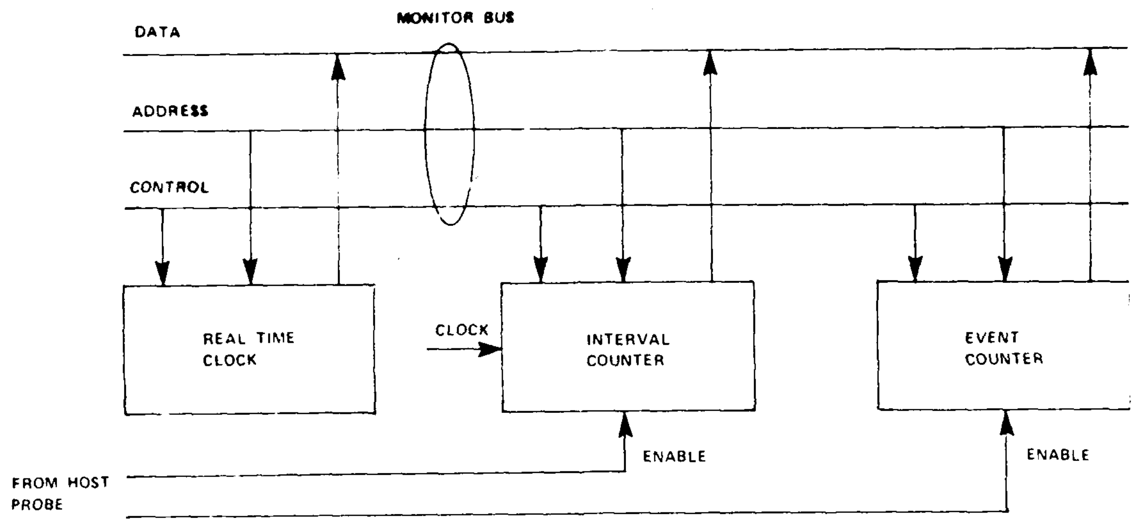


FIGURE 4.3 INTERVAL COUNTER, EVENT COUNTER AND A REAL TIME CLOCK

The Real Time Clocks at all of the monitor stations must be synchronized. Given this basic requirement, the clocks can be addressed with software and commanded to output to the data bus a digital number giving the appropriate time.

Both types of counters can be addressed from the monitor bus. Once put in the proper state by the monitor software, they respond to status signals obtained through probes from the host microcomputers. For example, if a disk status signal is high while the disk is operating, the Time Counter will turn on upon receipt of this signal and continue counting until the signal reverses state, causing the counter to turn off. At an appropriate time after the counter is turned off, a signal indicating the time interval is supplied to the data bus upon command from the monitor software. The Event Counter works in a similar fashion, counting the occurrence of events in a status signal rather than a time interval.

Since most of the network variables are random in nature, it will be efficient to have several histogram generators at each monitor station to reduce the random data to histogram form before transmission to the Monitor Control Location.

A logic diagram of a histogram generator is given in Figure 4.4. The device takes any data signal and quantizes it into a set of magnitude ranges for excitation of appropriate counters. The counters, eight or possibly sixteen in number, are read by appropriate monitor software. The Data Valid Signal, which must be present for the counters to function, is derived from the source of the variable whose histogram is to be generated.

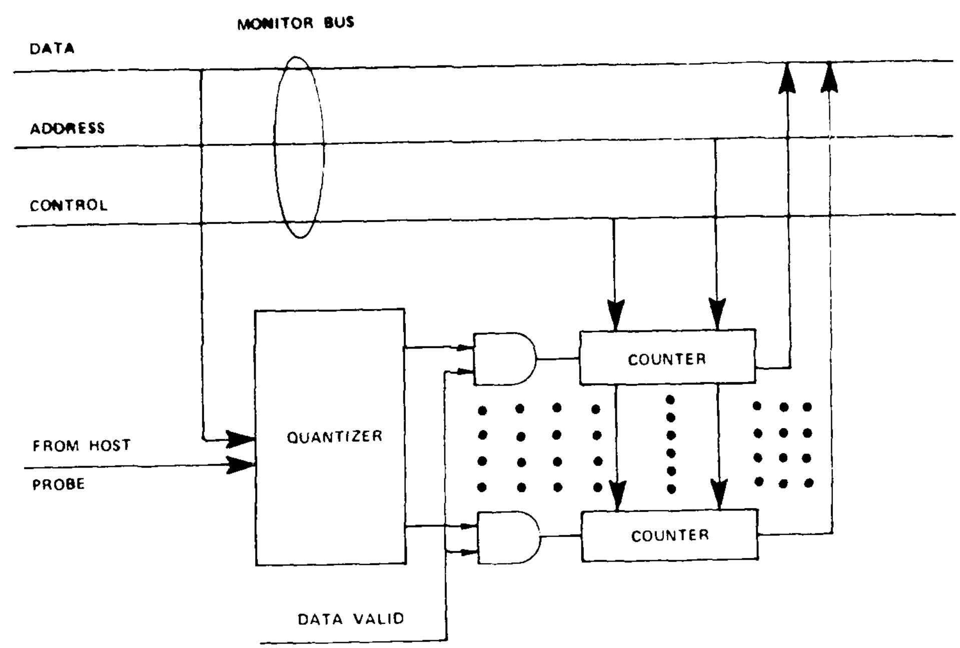


FIGURE 4.4 HISTOGRAM GENERATOR

A masked-word range comparator is used to measure the time the CPU spends executing a particular software region. This is accomplished by monitoring the occurrence of addresses between two specific values.

An implementation of the Masked-Word Range Comparator is shown in Figure 4.5. The 16-bit latches are loaded with the extreme values of the address range to be monitored. Addresses from the host probe are compared to the values stored in the latches in a comparator. Address values in the appropriate range actuate a counter which can be enabled by a signal from another source. The device can be set up and controlled completely with monitor software.

An implementation of a Logic Combination Unit is given in Figure 4.6. Its operation is much like that of the Masked-Word Range Comparator. For this unit, the eight-bit latches can be loaded with appropriate patterns for comparison to, say, the status word of some device. Using the Logic Combination Unit, specific patterns in the status word can be detected. If a counter is connected to the output, the time the device spends in one of its states can thus be measured.

The Two-Port RAM, which is a part of each monitor station, is regarded for purposes of discussion as a measurement module. This RAM permits non-intrusive access to data from the node switching computer. This is accomplished by using a RAM with a read/write rate of twice the clock rate of the node switching computer so that data can be read into the RAM by the switching CPU and read

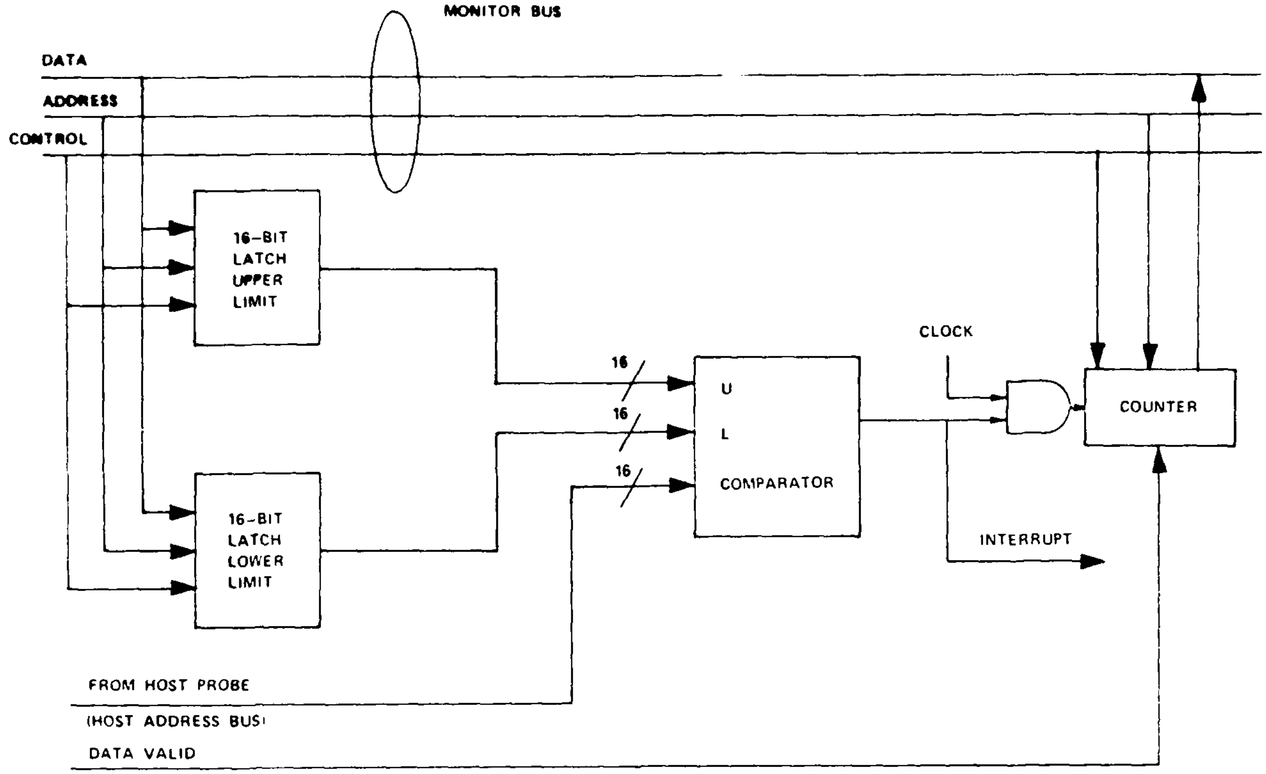


FIGURE 4.5 MASKED-WORD RANGE COMPARATOR

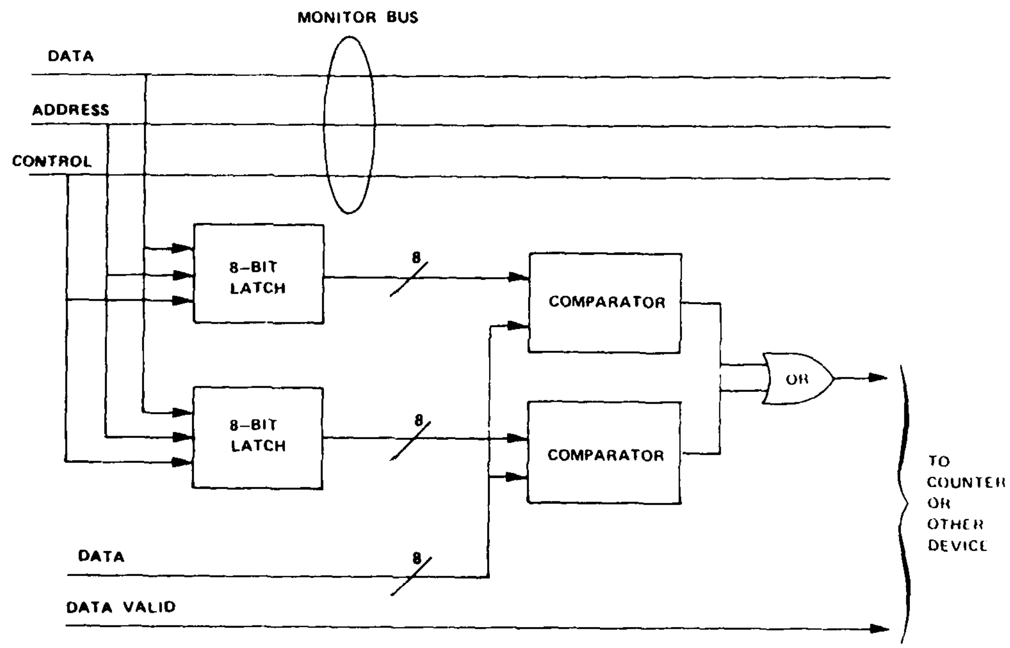


FIGURE 4.6 LOGIC COMBINATION UNIT

out of the RAM by the monitor CPU in one period of the switching CPU clock.

The Monitor Microprocessor and the Node Microprocessor will be identical devices. Thus, all network data which must be monitored can be stored in this RAM for access by the monitor CPU. Job ID number, pointers or other data on packet queues and packet arrival times are typical of the data to be stored in the Two-Port RAM.

4.1.3 Representative Specific Measurements: This subsection indicates in general terms how representative variables are measured with the Measurement Modules. More detail on some measurements will be given in Section 4.2 in the discussion of the monitor operation for a particular example.

In Section 3.2 specific variables to be monitored were classified as pertaining to resource utilization, variables describing the network operation and additional variables. Representative variables from each of these categories will be discussed below.

The activity of host-controlled resources (disks, line printers, terminals, etc.), can all be monitored through use of status signals obtained through the probes connected to the host at each node. A status signal is used as input to an Interval Counter such as shown in Figure 4.3. When the Interval Counter is actuated by its control signal, it will detect a resource active signal and measure the time the resource is in the active state.

The software program for carrying out host-controlled resource measurements is tailored to actuate the counter, through an appropriate control signal, when the ID number of a particular job is stored in the dual-port RAM by the host microcomputer controlling the resource being monitored. When the ID number is changed, indicating another job is being serviced, the software program causes the counter to read out the measured time to a storage location assigned to the particular job.

The activity of a host CPU or a switching computer CPU can be measured by determining when the CPU is executing instructions located in memory outside the wait loop. This measurement can be made using the Masked-Word Range Comparator of Figure 4.5, which requires the appropriate CPU address bus as an input. Use time of a host CPU will be allocated to a particular job in the same manner as described for the CPU controlled resources. Use time of a switching computer CPU will be totalled without allocation to specific jobs.

Most of the measurements involving the network have to do with measuring the parameters of the flow of packets. The proposed monitoring system will determine the average parameters of packet flow using measurements made on the normal data packets complemented with measurements made with pickup packets. Both types of measurements use the Two-Port RAM.

The normal data packets will all be labeled with a job number. Whenever a packet is transmitted from a node, the switching CPU, which controls the transmission, will store a count in the Two-Port

RAM in a storage location corresponding to the link over which the packet was transmitted. Classification as to job as well as to link can also be retained if desired.

The monitor software will cause the Two-Port RAM storage location to be sampled at regular intervals and the increase in the number of packets stored will be the number of packets transmitted over the particular link in the interval between samples. Of course the count in the storage location must be set to zero in initiating an experiment.

A similar procedure, storing a count for incoming packets, can give a measurement of the number of arriving packets per unit time on each link. Summation of either type of count over all links at a node gives the number of packets arriving at or leaving the node.

Several measurements, such as packets awaiting service, number of packets not acknowledged, and the number of packets generated at a particular node can be made by storing a count in an appropriate location in the Two-Port RAM following specific actions controlled by the switching computer CPU. The actions which can initiate a count to produce the above measurements are, respectively: storing an incoming packet in the buffer for receiving packets, retransmitting a packet, and transferring a packet out of the buffer in which it is generated. As with the other measurements noted, the above measurements rely on access of the Two-Port RAM storage locations to the monitor CPU.

Pickup packets will be used to measure packet delay. This measurement will be implemented by giving a pickup packet a special identification number which is read into a location in the Two-Port RAM immediately after the pickup packet is received or transmitted at a node. The monitor software monitors the RAM location and produces an interrupt when a pickup packet ID is received. The Real Time Clock is read following the interrupt and a "time stamp" is recorded, either in the data field of an arriving pickup packet or in storage for insertion in the field of the next pickup packet if the packet is leaving the node.

The Monitor Control Location ultimately receives all of the pickup packets and can extract the time of arrival and departure from each node over each link. This data is adequate to determine the profile of packet delays.

The network variables measured at each node are random, and thus it may prove to be efficient to convert most of these into a histogram before transmitting the data to the Monitor Control location. The Histogram Generator shown in Figure 4.4 can be used to generate the histogram if this option is used.

4.2 Example Illustrating the Use of the Monitor System

The purpose of this section is to define a typical task for the computer network and discuss in detail the functioning of the monitor system in monitoring the network as it performs this task. An inventory type task is chosen, and for such an application it is assumed that the Monitor Control Location is also the site of a large data base containing complete inventory data. The other nodes in the network have smaller data bases containing local data.

<u>4.2.1 Task Definition</u>: The task is defined by the following sequence of operations which could arise in a distributed computer controlled inventory system.

a) A user signs on at a terminal located at Node K and requests the restoration of a portion of his local data base which has been lost (say the Node K inventory of item A).

b) The Monitor Control Location supplies the required data from its large data base over the network.

c) The user at Node K requests a search of the Node K inventory of item A for an item A_i. This item is found to be absent from inventory.

d) The user requests a search of the local listing of the item A inventory at other nodes to determine the number of A_i items located at each node.

e) The user at Node K requests that his needs for A_i be filled from the supply at the node having the largest number of items A_i. (Assume that this is Node L.) The request is granted.

f) Node K updates its inventory of items A.

g) Node K instructs the Monitor Control to update its inventory listing of items A.

h) The Monitor Control instructs all other nodes to update their inventory listings of items A.

i) User signs off.

It is assumed that the inventory listing of items A is substantial so that a thousand or so packets of several hundred bytes each would be required to transmit it across the network. It is also assumed that the network is operating with a background of other tasks being executed.

4.2.2 Computer Network and Corresponding Monitor System Operation on Assigned Task: To illustrate properly the operation of the monitor system, it is necessary to examine the details of monitoring each activity of the computer network in carrying out a typical task, such as that defined above.

Examination of the nine activities listed for the task defined above indicates that they can be segmented into four distinct jobs, as given in Table 2. The Table lists the resources required for each job and it can be noted that Jobs 2 and 3 require only local resources at Node K, while Jobs 1 and 4 require the resources of the network and the resources at more than one node.

A detailed activity study is made for Jobs 1 and 2, since the requirements for these jobs illustrate all characteristics of the computer network and monitor system operation. In the study the computer network is assumed to operate in a specific manner. It should be understood, however, that this operation is intended to be typical and not that of a specific system.

TABLE 2. Segmentation of Illustrative Task

Job Number	Activities	Principal Resources Used
1	a, b	Node K - Host CPU, Node CPU, Disk, Terminal MC Node - Host CPU, Node CPU, Disk Links - K to MC and any alternate
2	c	Node K - Host CPU, Disk, Line Printer, Terminal
3	d	Node K - Host CPU, Disk, Terminal
4	e, f, g, h, i	Node K - Host CPU, Node CPU, Terminal Node L - Host CPU, Node CPU, Disk Node K - Host CPU, Node CPU, Terminal Node I (all I) - Host CPU, Node CPU, Terminal MC Node - Host CPU, Node CPU, Disk Links - MC to each node and alternate

The details of the Job and Job 2 activity, with the corresponding function of the Monitor System are presented in an Appendix in Tables A1 - A5. Tables A1 - A3 list the general monitor system functions and Table 4 enumerates the activities associated with Job 1 and Table 5 lists the activities associated with Job 2.

A summary indication of the functioning of the Monitor System is presented in Figure 4.7, which is a schematic representation of the monitor functions at one node, Node K. As each host-controlled resource is used, the job ID is read into the appropriate memory location in the Two-Port RAM. The software measurement program senses the job ID and actuates an "infinite loop" which allows appropriate modules to measure the active time of the resources. Concurrently, as packets are generated and transmitted, the Node CPU increments the counts in the indicated memory locations in the Two-Port RAM.

The software measurement program is interrupted at regular intervals to allow the Monitor CPU to read the indicated Two-Port RAM memory locations and transfer the readings to output locations in the Two-Port RAM. The data stored in the output locations is transferred to the data field of pickup packets when they arrive periodically. The arrival (and departure) of pickup packets also causes an interrupt to allow the Monitor CPU to read a Real Time Clock and insert this "time stamp data" into the data field of the pickup packets.

The Monitor Stations at the other nodes in the network operate in the same manner as at Node K. For this example, the final output, printed out at the MC location, consists of the following:

- Total time for the computer network to accomplish the task.
- Total host-controlled resource utilization for the task as compiled from the measured active time for each host-controlled resource, segmented by jobs.

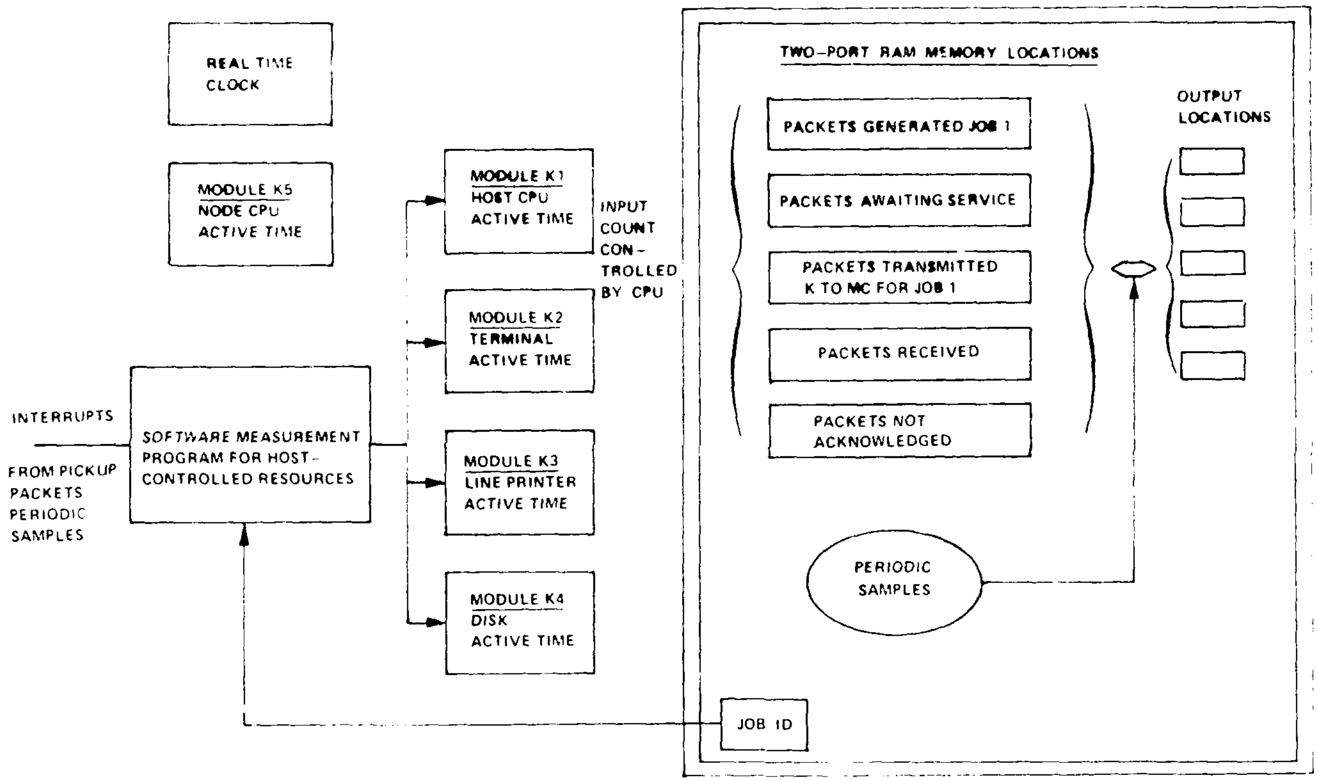


FIGURE 4.7 SCHEMATIC REPRESENTATION OF THE MONITOR FUNCTIONS AT NODE K FOR THE INVENTORY CONTROL PROBLEM

- Total average network resource use determined from a ratio of the count of packets generated on the task to total packets generated, allocating measured node CPU and link times to the task on a pro rata basis accounting for differences with respect to time and to links traversed.
- Total task cost obtained by multiplying resource use time by appropriate resource costs.
- Average or histogram for packet delay time for each link, with time as a parameter if appropriate, computed from the data obtained by pickup packets.
- Average or histogram for queue length at each node computed from the count of packets awaiting service, with time as a parameter if appropriate.
- Average or histogram for the number of packets not acknowledged at each node, with time as a parameter if appropriate.
- Statistics for traffic flow - average or histogram for total packets arriving at each node, average or histogram for packets arriving (and leaving) over each link both with time as a parameter if appropriate.

The tables in Appendix A give details of the computer network and measurement system activities on typical parts of the task. Examination of these details shows that the proposed measurement system structure can be implemented in a feasible manner.

4.3 A Monitor Structure for the AIRMICS/GEORGIA TECH Experimental Network

The implementation of the hybrid monitor system discussed in this section of the report was chosen for its possible applicability with the AIRMICS/GEORGIA TECH Experimental Network. Although the long range plans for the Experimental Network have not been specifically quantified, the monitor system described in Section 4.1 is very flexible and has most of the features which could be required by this network. In addition, the points of entry into the Nodal Monitor Stations are compatible with what is available at the existing nodes of the Experimental Network.

As discussed in general terms for the monitor system, three types of measurements are possible, namely: host-controlled resource measurements, network related measurements, and auxiliary measurements, such as measurement of the execution time of specific pieces of software. A choice of what, and how much monitor equipment to install will depend in detail on the studies to be made with the network. Some general comments can, however, be made.

Of course if resources are available, a complete monitor system with ample equipment for all three types of measurements can be implemented. On the other hand, the following comments are germane if the measurement system budget is limited.

It is felt that emphasis in studies made with the Experimental Network will very likely be on characteristics of the network itself--its geometry, its routing algorithms, etc., rather than on the efficiency of the microcomputer hosts. To the extent that this is true, the network related measurements can be emphasized and implemented completely, with less attention being given to the other two categories.

At the present time, the Experimental Network is distributed over only two locations on the Georgia Tech campus. As long as this is the case, there is no need to use the complexity required by the scheme for setting up experiments completely from the Monitor Control location.

Of course there is the possibility that the Experimental Network could be used to evaluate prototype equipment for measuring the efficiency, or monitoring the proper functioning, of microcomputer hosts. In such a case, the host-controlled and auxiliary measurements can be emphasized and the others deemphasized.

5. EXPERIMENTAL NETWORK

5.1 The Communication Network Philosophy

A major facet of the current system is a packet switched microprocessor based communications network. This network, which far exceeds the requirements of the demonstration system, has error detection and correction capability in addition to its communication functions. The network is wholly package switched and all data and internal communications are handled through a packet switched protocol. The protocol was deliberately made to be open ended so that additional packet classes may be added to the network later. The current network implements three packet classes: data packets, which carry the host to host communication messages; local acknowledgements, which acknowledge adjacent node communications; and source acknowledgements, which acknowledge the final receipt of the message at the destination node. Other classes of messages which might be later implemented include data based request messages, requests for distributing processing capability, and requests for utility processing.

A message transmission scenario through the network can be described as follows: A host initiates a data transfer to another host on the network by transfering to its network node, in a very simple protocol, the destination of the message and the contents of the message. The network node, which we shall call the origination node, takes two specific actions. First, it buffers the message as a safeguard against the errors in the communication process. It will retain this buffered message until it receives a "source acknowledgement" packet from the destination node indicating that the message has been received at its final destination. Second, the origination node forms a data transfer packet addressed to the destination node. Once the packet is formed, the node will attempt to send the packet across the primary route to the destination node. If this communication route is busy, the origination node will try a secondary route. The system supports three possible alternate routings. If any of the appropriate communications links are free, the message will commence transmission immediately. If all of these communications links are busy, the message will be queued for later transmission on the primary link.

When the message is received at the first adjacent node in the transfer path, this intermediate node takes two specific actions. First, this intermediate node forms and transmits a local acknowledge packet back to the origination node. This local acknowledgement informs the origination node that an error free reception of the message has occurred. This fact is noted in the origination node, and the buffered message is marked as having been locally acknowledged. If no local acknowledgement is forthcoming in a fixed amount of time, the buffered message will be retransmitted. This particular error correction technique allows the network to handle all detected errors in a uniform fashion: by discarding and not acknowledging the error packets, they will be automatically retransmitted. The second action taken by the first adjacent node is to retransmit the data packet forward towards its destination. The procedure for doing this is identical to the data transfer procedur described for the origination node.

The intermediate node also holds the data message until it receives a local acknowledgment. Unlike the origination node, however, all intermediate nodes discard the data message when the local acknowledgment is received. The data message thus travels from node to node through the network with local errors being corrected until it reaches the destination node.

At the destination node, three specific actions occur. First, as in all the intermediate nodes, a local acknowledgment is transmitted to the adjacent node from which the message arrived. Second, information as to the message's source and the message itself is transmitted to the host. If the host communication link is busy, this message is queued for later transmission. Third, the source acknowledgement packet is formed and is transmitted to the originat ing node. This source acknowledgement packet travels through the network in a fashion identical to a data message packet until it arrives at the origination node. Upon its arrival, the originating node discards its buffered copy of the original message. If no local acknowledgement is received within a fixed time constant, the data message will be retransmitted from the origination node.

This network is implemented on network nodes of identical hardware. The software which runs within each node is identical to the software that runs on all the other nodes. Routing for this network is originally setup by a predetermined network architecture, but may be changed dynamically by host requests for reroutings. Hence this network may be reconfigured during actual operation, though this feature was not used as part of the study.

The maximum packing length in this network is 256 bytes. The maximum message length is three packets. These parameters cannot be dynamically configured, though they can be changed by minor programming. All communications links have switched selectable baud rates, which may be chosen up to a maximum rate of 19.2 kilobaud. The actual network development work, however, was done at a setting of 1200 baud. This results at a maximum node-through baud rate of 19.2 kilobaud. Since this was an experimental network designed primarily to study networking techniques, the network code was not optimized for maximum communication through-put. In fact, all communication input-output is done through accumulator transfers. This offers maximum flexibility with some loss of speed.

5.2 The Microprocessor Network

The development of a packet switched communication network presents many special and unique programming debugging problems. It is true that only one program is being developed; however, in an operating network this one program runs simultaneously in many network node processors. Within each network node (which are, of course, computers in their own right), there is a separate and unique real time environment. Errors which are associated with the real time nature of the node programs occur as a direct result of the network traffic. Due to the asynchrony of the entire system, this means that in many instances errors which occur and are detected cannot be repeated.

Another characteristic of the network debugging environment is that often it is impossible to determine the source of an error. Errors which are generated in one node processor may be transmitted out of that node without the recognition of the node program itself. Thus, when the error is detected it may be far away from its source.

The apparent statistical nature of the behavior of packets within the communication network forces the programmer into using techniques which are themselves somewhat statistical in nature. Many of these techniques are not only appropriate for the debugging of the network, but are also appropriate for the later testing and measuring of the network performance. The most important of these techniques which was used in this development was the use of a "traffic generator." A traffic generator is a piece of hardware whose task is to simulate the existence of a larger network than the one which is really being tested. The type of traffic generators used in this study were the so-called "constant load" traffic generators. This form of traffic generator forces an ambient condition in the network in which a known number of messages are always present within the operating nodes. Thus, for example, if the number of messages desired were five, the traffic generator would insert five messages into the network. Whenever one of the messages exits the network by returning to the traffic generator, the traffic generator would insert a new message in its place. In this way an approximate load of five bogus messages is kept within the operating network.

The use of the traffic generator represents a Monte Carlo approach to the problem of network debugging. When a traffic generator is allowed to run for long periods of time, a large number of different real time network states are excited. Thus, the network's operation may be checked over many operating conditions beyond the scope of its original architecture.

Two different traffic generators were implemented as part of this study. The first, called the "dummy load" traffic generator, was implemented as the combination of a multi-task Fortran program on the Nova 820 in the digital signal processing laboratory and a modified version of the network node program on a network communication box.

This traffic generator operated in conjunction with a number of dummy routes which were preassigned during network initialization. The dummy routes always started in the traffic node, passed through one or more other nodes in the network, and finally terminated in the traffic node. The traffic generator initiates messages along these dummy routes and receives the messages when they return. Thus, from the view point of the ordinary network host, the network operates normally, but appears to be bearing communication traffic from a larger outside network.

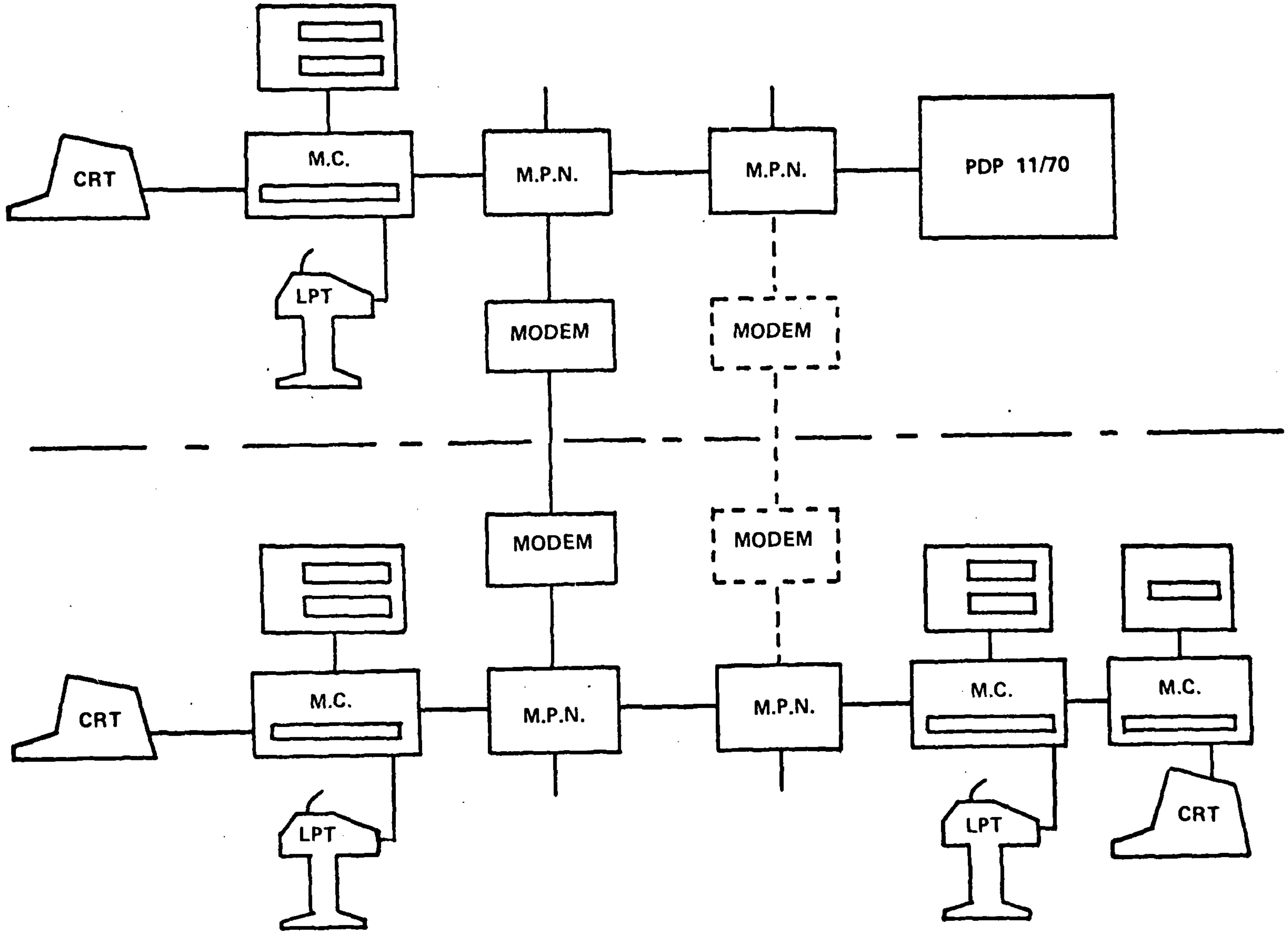
The second traffic generator was the so-called "host involvement" traffic generator. This function was implemented entirely in the Nova 820 and required no special modification of the network nodes. During the operation of this traffic generator, all the network hosts were dedicated to the testing procedures. Each host ran a program which returned an exact copy of the message it received to the source of the message. The traffic generator sent a variety of messages to the various hosts and waited for the message to return. Hence, the use of this traffic generator constituted a test which involved the host's data transfer program as well as a multiple real time environment within the network itself.

5.3 Description of Computer Network Hardware

This section describes the specifications of the computer hardware that was purchased in order to complete the proposed research project.

Figure 5.1 shows a fundamental block diagram of the complete computer network. In the computer network there are four host computers--an Intel microcomputer system, a Data General computer system, a Motorola microcomputer system, and an PDP 11/70 minicomputer system. Associated with each computer host is a microprocessor communications node. The microprocessor node has the responsibility o handling all the network communications sent and is discussed in detail in Appendix C.

Since the Intel and PDP 11/70 host computers are located at the AIRMICS computer site, and the Motorola and Data General hosts are located at the Electrical Engineering laboratory, the connection between the Intel and the Data General microprocessor host is accomplished by means of a standard telephone line and two Universal Data System 1200 baud modems. The following sections will describe all of the commercially available equipment in the computer networks except the PDP 11/70 computer system which was an existing AIRMICS facility and not purchased for the purpose of completing this particular project.



COMPUTER NETWORK

5.1 BLOCK DIAGRAM OF COMPLETE COMPUTER NETWORK

1) The Intel microcomputer system, illustrated in Figure 5.2, is a standard Intel model 230 microcomputer system with an additional Teletype 40 line printer. The computer system contains a central processing unit, random access memory, read only memory, dual floppy disk drives with controller, CRT with keyboard and controller, line printer serial interface, and network serial interface.

2) The Intel central processing unit is a standard Intel microprocessor with an 8080A microprocessor chip, 2.6 MHz processor clock, system controller, multibus priority resolution circuits, multi-bus controlling data drivers, address drivers, system clock generator, and I/O board address decoder.

3) The random access memory is a 64 Kbyte memory used for storing parts of the operating system, user programs, and data. The read-only memory is used to hold the resident portion of the ISIS II monitor, revision 1.2.

4) The complete ISIS II operating system is stored on the floppy disk and is read in automatically from the read only memory portion of the monitor. The dual floppy disk drives are housed in a separate cabinet and interfaced to a floppy disk controller in the central processor unit cabinet. The disk will hold approximately 200 Kbytes of data on each disk and handle double density cassettes.

5) The CRT is housed in an integral part of the central processor console and the CRT screen storage uses a section of memory to store the characters that are being displayed. This means that a character can be displayed on a CRT screen by storing the character

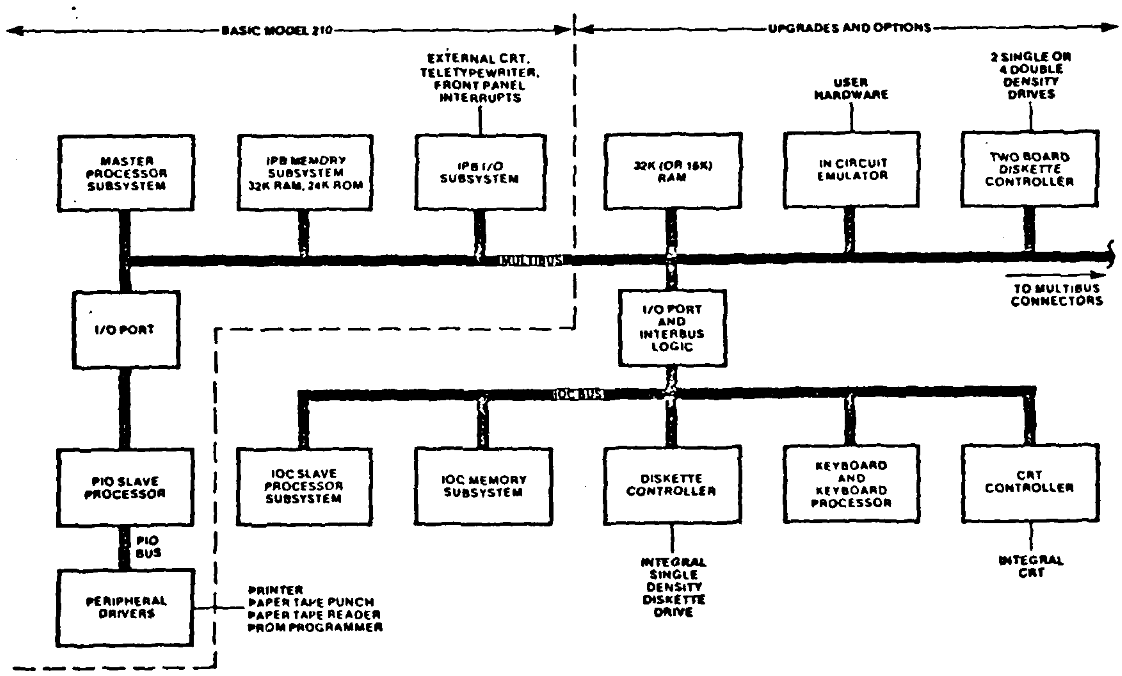


FIGURE 5.2 THE INTEL MICROCOMPUTER SYSTEM

in an appropriate memory location in random access memory. The line printer is interfaced through a standard RS232 EIA level interface with baud rates adjustable from 110 to 9600 baud. The baud rate is programmable under software control, and is currently set at 1200 baud.

6) The network interface is a standard RS232 EIA interface which permits the system to talk to the computer system network.

The Motorala and Data General microcomputer systems have equivalent hardware to the Intel system.

5.4 Network Trafficking Experiments

5.4.1 Introduction: Any reasonably designed computer network will transfer messages from one computer host to another, as does the computer network described in this project. The transfering of messages between a microcomputer host running COBOL inventory programs does not push the computer communication network to a point anywhere close to its limits in order to test additional loading on the network. The data General Nova 820 computer was added as an additional host in the network with its own network processor node (shown in Figure 5.3). Instead of running time consuming inventory programs to send out messages through the network, the Nova 820 sent messages in a controlled manner such that a given number of messages would be kept in the network at one given time. In this manner it was possible to simulate additional network traffic without the expense of adding costly host computer systems. The traffic generator system could, in effect, simulate the effect of many additional hosts on the network.

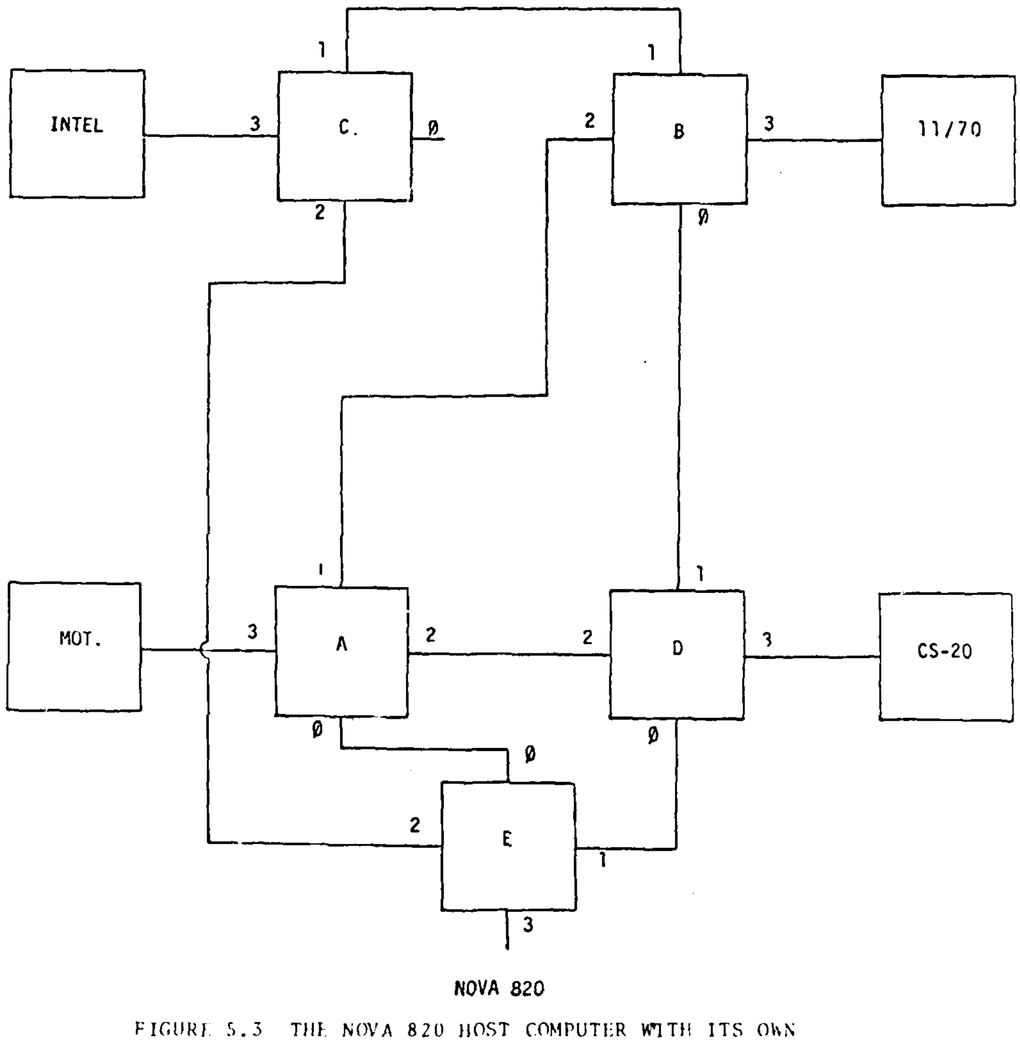


FIGURE 5.3 THE NOVA 820 HOST COMPUTER WITH ITS OWN NETWORK PROCESSOR NODE

5.4.2. Traffic Routes: In order to send messages through the sys it was first necessary to establish particular message routes for the traffic node to send messages through the network and back to itself. The traffic generator would then send the message out to the network on a given route, the message would travel completely through the network route and return to the Nova 820. When the message returned, the message was compared character by character with the transmitted message and any deviation in output to input would be indicated on the Nova 820 main terminal. The Nova 820 also kept track of lost messages, that is, messages that were sent out to the computer network on a given route but were never returned to the 820. The particular traffic routes that were set up in the microprocessor communication system are tabulated in Table 3, and an illustration of each route is shown in Appendix B.

5.4.3. The First Traffic Experiment: The first traffic experiment was a simple two communication processor loop that sent messages from one processor (which acted as a traffic node) through one of the other processors and returned. Traffic could be initiated through any of the node ports and in this manner the hardware could be checked for reliable operation. It was of particular interest to verify that each of the four serial ports on the processors were operating reliably and that a reasonable error rate could be expected between any two communicating ports. The single loop traffic experiment was repeated with each of the communication processors so that the reliability of each processor could be verified. The

TABLE 3. TRAFFIC ROUTES

NAME	NODE
F	E--D--A--E
G	E--A--D--E
H	E--D--B--A--E
I	E--A--B--D--E
J	E--C--B--D--E
K	E--D--B--C--E
L	E--C--B--D--A--E
M	E--A--D--B--C--E
N	E--C--B--A--E
O	E--A--B--C--E
P	E--C--B--A--D--E
Q	E--D--A--B--C--E

result of the single traffic loop experiment were as follows:

1. A number of intermittent and hard failures were found in the serial I/O ports and repaired.

2. A number of hardware problems with the interrupt structure were isolated and repaired.

3. A maintenance log was established on all the components of the microprocessor nodes indicating any failure of any of the components and the fact that the components had passed the single-loop test.

4. The single-loop test proved that the processor node software is a single-input/single-output process.

5. The single-loop traffic test provided us some burn in time for the processor components, which was extremely important since input mortality is one of the most important reliability problems ir integrated circuit technology.

<u>5.4.4 Single Host Traffic Test:</u> In this test the Nova 820 communicated with each microcomputer host through its microprocessor node. Programs were written on each of the three microcomputers to receive a message from the network and return the same message to its originator. The Nova 820 would then send a message through its node to the host node. The host would echo that message back to the 820 where the 820 compared the return message with the transmitted message to detect only errors. The Nova 820 could vary the number of messages sent back to back, the length of each message and the character in the message.

The results of these series of tests were as follows:

1. The test verified the hardware protocol between each host and its communication processor. This hardware protocol included the correct cabling and correct action and polarity of data terminal ready, clear to send, data set ready and request to send.

2. These tests verified the reliability of the host to communication processor serial interface.

3. These tests verified the operation of the communication drivers in both the host computers and the microprocessor nodes.

4. The test produced necessary burn in and reliability time on the microprocessor nodes.

5.4.5 Multiple Host Traffic Test: In the multiple host traffic test, two or more microprocessor hosts with their communication processors were hooked to the traffic communication processor and 820. The 820 would send messages to both of the host microcomputer systems through the microprocessor network. The microcomputer host would echo the messages back to the 820 where they were checked. In this test the 820 could vary the number of messages back to back, the number of characters in each message and the routes through the microprocessor network. This test was designed primarily to check the ability of the nodes to handle multidirectional traffic. With multidirectional traffic, the nodes are forced to queue messages and queue local acknowledgements, and the interrupt handler in the nodes is forced to sort and queue messages in different directions. The results of this test were as follows:

1. This test uncovered several subtle software errors, such as that the software would handle single traffic patterns, but would fail when multiple-loop tests would force certain routines to be used in a re-entry manner.

2. The test validated the basic structure of the message buffers and queueing structure used by the nodes to sort and store multiple messages.

3. This test pointed out several network characteristics that will be covered below in the section on general network characteristics.

5.4.6 Multiple Loop Traffic Test: In this series of tests the microprocessor nodes of the network were connected without host computers to the 820 traffic node. The 820 would send messages through different routes in the network with the 820 selected as the final destination of the message. The 820 was programmed to maintain a certain number of messages running through the network at any one time. That is, the 820 would send out n-messages into the network. As soon as one message was returned from the network, another message was immediately sent back to the network. In this manner, n-messages were always kept running through the network. The 820 could control the number of messages in the network, the length and content of each message and the root for each message. This test was designed to operate the network in a controlled loading manner so that various network limits could be investigated. The result of this series of tests are as follows:

1. It was determined that the microprocessor nodes could send multiple path messages at baud rates of 1200 baud or less.

2. It was determined in 10 hours of continuous testing that the number of CRC errors, loss messages, lost local acknowledgements, and incorrect messages was extremely small. The error rate was less than one error per a million characters.

Additional network characteristics were determined, and will be discussed in the section on general network characteristics.

5.4.7 Network Test with Inventory Control Program and Node Trafficking: In this test the Intel microcomputer and the Motorola microcomputer and PDP 11/70 minicomputers were connected as with their microprocessor nodes as a standard network and each node of the inventory control program was tested for correct operation of the programs as well as the network responses. The CS-20 microcomputer was eliminated from this test because the manufacturer, Data General, would not give us proper information to properly modify the network driver to operate further COBOL in the correct manner. In this test, each of the instructions for the inventory control program (as described in the inventory control program section) was executed on each machine and verified. Operator initiated messages were sent between each pair of host computers and the received message verified. Remote holding of data bases was tested and verified, as well as remote transaction initiation. The results of this series of tests were as follows:

1. This test verified that basically the same COBOL program could be used on the three host computer to correctly operate the inventory control program.

2. This test verified that the COBOL programs could implement the correct protocols to talk to the network and receive messages from the network.

3. This test pointed out the sensitivity of the different host computers to network protocols. These will be discussed in detail in the section on host network characteristics.

4. The sensitivity of the host to the network protocol emphasized the desirability of a communication processor whose host protocol can be tailored to the host machine drivers.

5.4.8 CS-20 Inventory Control Test: In this test, the software inventory program for the CS-20 was tested with a remote terminal acting as the network. Even though the CS-20 would not implement the desired network protocol, the remote terminal was used to verify that the inventory program would operate correctly with a modified protocol. This verified the portability of the COBOL software even though the network protocol could not be implemented.

5.4.9 Inventory Control Program Test with Trafficking: In this series of tests, the full network was connected with the exception of the CS-20 host computer. The CS-20 communication processor was included. The 820 computer was connected to the network through a traffic generator node as shown in Figure 5.3. The 820 generated series of messages through different routes through the microprocessor communication network at the same time that the inventory control program was sending messages between host computers.

5.5 General Characteristics of the Computer Communication Network

The following section describes the characteristics of the computer communication network as determined by the series of inventory control programs and the traffic generator programs as described in the previous section.

It was determined during the series of tests that the network exhibited certain characteristic behaviors in particular situations, and that the network was sensitive to certain types of situations. None of these characteristics made the network unusable, but it is important to understand these limitations as a step to improving future network communication systems.

ITEM 1: Sensitivity of the network to host protocol. Because the host computers were being operated from a higher level language (COBOL), there was very low flexibility in establishing a complex protocol between the host and the microprocessor host. Therefore, the following simple protocol was established for sending a message from the host to the network:

a) The host starts a message by sending the letter of the destination node proceeded by an open parenthesis. In this particular network, the nodes were lettered "a" through "d".

b) The host follows the destination code with a string of message characters. These characters can be any eight bit code asking for data.

c) The message string is terminated by sending an ACSII exclamation point character. If an exclamation point is used as part of the message string, it must be proceeded by an escape character so that the microprocessor node will not take it to be the end of the message. An example of a typical message is given below:

(C THIS IS A TEST MESSAGE!

The protocol for messages from the microprocessor to the host is similar except for the source of the message and the destination. Neither the host nor the communication processor acknowledge any messages and no vertical or longitudinal parity is checked.

Because this protocol is extremely simple, it makes it easy to incorporate into high level languages such as COBOL, but this simple protocol does not provide a means for checking the communications between the host and the microcomputer or the microcomputer and the host. Therefore, all communication errors between the host and microcomputer will be undetected. If an error occurs in the message string, the result would be an erroneous message at the destination, but the network would be unaffected. One of the most serious problems would be the possibility of an error in the destination code at the begining of the message. If this code were wrong, the network would try to send the message to a different destination. This would result in a host getting a wrong message, or if the destination were not part of the network, the message would remain in the network trying to be transmitted. If enough such messages were kept in the network, the processor memory space could be exhausted, causing the network to fail. If the exclamation point were communicated with an error, the result would most likely be two messages packed together as one, which would be an error in the total communication, but would not bother the action of the communication network.

ITEM II: Sensitivity of the network software to buffer overflows. The present network software is sensitive to possible overflow in buffer storage or queue sizes. The present network will try to handle incoming data beyond its capacity. This is not a problem with the normal function of the network, since its capacity is clearly capable of handling the inventory control program without exceeding buffer or queue sizes, but under the extreme condition where the traffic generator is sending excessive traffic through the network, the network can be caused to fail due to excessive traffic. An extremely straightforward method of handling this problem would be to implement the "clear" and "send" lines between the microprocessor nodes and the host. These lines could be used to stop the host computer from putting more traffic on the network than the network can accomodate. The hardware to implement this connection is currently in the network, but the software to support these lines is not included in the present node software package.

ITEM III: Sensitivity of the network to loss of local acknowledgements. In the present network a fairly simple acknowledgement scheme is used for verifying data transmission between two microprocessor nodes. In this process, a packet is formed in one node and sent to another node. The packet is checked for parity and CRC error, and if both of these are correct, the message is acknowledged by sending a local acknowledgement packet back to the transmitting nod If the parity or the CRC does not check, then the receiving node does not send an acknowledgement at all and the transmitting node will wait a given amount of time and then try re-sending the message. This scheme, in effect, uses a time out for a negative acknowledgement. Since there is no verification or check of the local acknowledgement package, some problems can arise from this technique if errors occur in the transmission of local acknowledgement packets. If, for instance, the receiving node receives a message packet and it is correct, then it sends an acknowledgement to the transmitting node. If there is an error in the transmission local acknowledgement, the transmitting node will time out and re-send the message which the receiving node will now take as a second valid message. A second, but less likely possibility is that the transmitting node would receive a packet in error that it thinks is a valid local acknowledgement and would clear its buffer of a packet that has not been correctly received. A third possibility is the reception of a local acknowledgement packet with invalid information in it. The local acknowledgement packet contains information telling the transmitting node which message has been correctly received and, therefore, to clear its buffer of that particular message. If an acknowledgement packet is received and taken to be valid, but has incorrect data concerning which packet was being acknowledged, the transmitting node would clear its buffer of the wrong message and continue to re-transmit the acknowledged message. If a sufficient number of erroneous local acknowledgements are passed through the network, the network could fail due to mis-sent and un-sent messages.

ITEM IV: Sensitivity of the network to messages with improper destinations. One characteristic of the network that is an outgrowth of the time-out and re-transmit scheme for unacknowledged messages is a problem with messages that enter the network with improper destination. If a host sends a message to a microprocessor node with an invalid destination, or the node receives a message with an error in its destination, then the node might transmit a message to a destination that does not exist. If the destination does not exist, then the message can never be acknowledged. The source node will then continue to re-transmit the message on a time-out and re-transmit basis. Therefore, the message will forever be re-transmitted by the source node. If enough messages with no proper destination are put into the network, the network will start to degrade in performance as the false messages are being transmitted, and then finally fail as the node buffers become overflowed with messages with no destinations. One possible solution to this problem without altering the time-out and re-transmit scheme would be to add additional software functions to the node to check for messages with invalid destinations and clean them out of the node buffer. This could be implemented as a table of valid destinations, or by determining that after a message has been retransmitted a certain number of times that it be declared an invalid destination. When an invalid destination message is found, the message should be removed from the buffer so that the buffer space can be returned for active operation and some record should be kept of the number of messages removed.

ITEM V: Availability of network operational status. The present network is not equiped with a means of monitoring the present operational status of the network, so that even detected errors that occur are not recorded on a real time operational basis. There is also no implementation of correctional measures when error rates exceed certain values. Implementing routines to keep track of misoperation of the network would be an extensive task, but would be an excellent extension of the network capability. There would be two possible approaches to this extension. The first would be to handle network operational status on a local basis. That is, each node would keep track of any network misoperation that it detected and would relay that information to its host. Each host would then be programmed to take any corrective actions required, and possibly relay information about network operational status to the host operator. Another approach would be to dedicate a node of the network and possibly a host computer to keep track of the total operational basis of the network. In this scheme, any node that detected a malfunction in the network would form a packet to be sent to the status node regarding the failure and the type of failure. The status node would be responsible for collecting this information and performing corrective actions. Once corrective action was determined, the status mode would then send a packet back to the appropriate nodes telling them to restructur the routing of the network to try to compensate for network failures. This second technique, though more complex, has the advantage that decisions about restructuring the

AD-A083 046 GEORGIA INST OF TECH ATLANTA SCHOOL OF ELECTRICAL EN--ETC F/G 9/2
THE FEASIBILITY OF IMPLEMENTING MULTICOMMAND SOFTWARE FUNCTIONS--ETC(U)
OCT 79 T P BARNWELL, J L HAMMOND, J H SCHLAG DAAG29-78-G-0139

UNCLASSIFIED ARO-15900.1-A-EL NL

2 OF 4
AD A083046

network could be made on status reports from all the nodes instead of just local behavior.

6. NETWORK COBOL

6.1 Introduction

Because the Army uses multi-vendor machines, it is desirable for the Army to have a language subset that is compatible with all of its machines. Such a language subset could provide a single program which would be executed by all the processors in the system. This capability would greatly simplify life cycle management by eliminating the need for different versions of the same program to run on several machines.

The following is a subset of M6800 COBOL, MICROSOFT COBOL, Data General CS-20 COBOL, and DEC PDP-11 COBOL, called NETWORK COBOL. NETWORK COBOL has this important advantage of being compatible with the INTEL, MOTOROLA, DATA GENERAL, and PDP 11/70 machines.

NETWORK COBOL has been tested and is the design language which was used with the AIRMICS/GEORGIA TECH microprocessor network to develop a distributed data-base-management program.

Under the AIRMICS/GEORGIA TECH project, several things were accomplished:

1. A common subset of the COBOL versions available for the INTEL 8080 MDS System, the M6800 based EXOTERM, the DATA GENERAL C520 System, and the PDP 11/70 was generated. This subset is termed NETWORK COBOL.

2. A demonstration distributed and duplicate data base management program was developed to do simple inventory control.

3. Programs were developed to convert between the various COBOL formats and also handle the hardware related differences between the COBOL dialects.

4. Several other programs were developed to rectify isolated differences in the various operating systems.

6.2 Acknowledgement

In compliance with the request of the Executive Committee of the Conference on Data System Languages (CODASYL), and specifically the CODASYL COBOL Committee, the following acknowledgement is extracted from that contained in the publication COBOL, Edition 1974.

"Any organization interested in reproducing the COBOL report and specifications*, in whole or in part, using ideas taken from this report as the basis for an instruction manual or for any other purpose is free to do so. However, all such organizations are requested to reproduce this section as part of the introduction to the document. Those using a short passage, as in a book review , are requested to mention COBOL in acknowledgement of the source, but need not quote this entire section.

"COBOL is an industry language and is not the property of any company or group of companies, or of any organization or group of organizations.

"No warranty, expressed or implied, is made by any contributor or by the COBOL Committee as to the accuracy and functioning of the programming system and language. Moreover, no responsibility is assumed by any contributor, or by the Committee, in connection therewith.

"Procedures have been established for the maintenance of CO[illegible] Inquiries concerning the procedures for proposing changes should be directed to the Executive Committee of the Conference on Data Systems Languages (CODASYL).

"The authors and copyright holders of the copyrighted material used herein have specifically authorized the use of this material, in whole or in part, in the COBOL specifications. Such authorization extends to the reproduction and use of COBOL specifications in programming manuals or similar publications."

*COBOL, Edition 1965, produced by joint efforts of the CODASYL COBOL Committee and the European Computer Manufacturers Association (ECMA).

FLOW-MATIC (Trademark of Sperry Rand Corporation), Programming for the Univac (R) I and II, Data Automation Systems copyrighted 1958, 1959 by Sperry Rand Corporation; IBM Commercial Translator Form No. F 28-8013, copyrighted 1959 by IBM; FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell.

6.3 Preface

M6800 COBOL is based on the specification of the COBOL standard published by the American National Standards Institute (formerly known as the United States of America Standards Institute) and contained in the publication USA Standard COBOL X3.23 - 1974.

As its name implies, COBOL (COmmon Business Oriented Language) is especially efficient in the processing of business problems. Such problems typically involve relatively little algebraic or logical processing; instead, they most often manipulate large files of basically similar records in a relatively simple way. This means that COBOL emphasizes mainly the description and handling of data items and input/output records.

This publication explains NETWORK ANS COBOL which is a compatible subset of American National Standard COBOL. The compiler supports the processing modules defined in the standard. These processing modules include the following:

NUCLEUS defines the permissible character set and the basic elements of the language in each of the four COBOL divisions: IDENTIFICATION DIVISION, ENVIRONMENT DIVISION, DATA DIVISION, PROCEDURE DIVISION.

TABLE HANDLING allows the definition of tables of contiguous data items and accessing these items through subscripts.

SEQUENTIAL ACCESS allows the records of a file to be accessed in an established sequence. It also provides for the specification of rerun points and the sharing of memory area among files.

RANDOM ACCESS allows the records of a mass storage file to be accessed in a random manner specified by the programmer. It also provides for the specification of rerun points and the sharing of memory area among files. Specifically defined keys, supplied by the programmer, control successive references to the file.

LIBRARY allows the programmer to specify text that is to be copied from a library. This feature is different on all machines and so is not used in NETWORK COBOL.

6.4 Organization of Manual

A COBOL source program consists of information in four divisions: the IDENTIFICATION DIVISION, ENVIRONMENT DIVISION, DATA DIVISION, and PROCEDURE DIVISION. Taken together, these divisions constitute the total program (including a description of the configuration needed, the forms of various data files, and the programming steps necessary to perform these procedures), and are presented to the processor for compilation into a corresponding object program.

In this manual, NETWORK COBOL is described as follows:

- Sections 6.6 and Sections 6.7 describe the COBOL language structure. It presents the COBOL theory behind work formation, the use of words to name elements in a program, and a discussion of the syntax of the language.
- Sections 6.8 through Sections 6.15 contain a discussion of the format and organization of data files, together with methods used to remove data from, or place data into, such files.
- Sections 6.16 through Sections 6.19 present a detailed description of the IDENTIFICATION, ENVIRONMENT, DATA, AND PROCEDURE DIVISIONS, respectively.

Appendix E contains a composite list of COBOL reserved words in the NETWORK COBOL.

6.5 Command Syntax Notation

Notation conventions used in command specifications and examples throughout this manual are listed below.

Notation	Description
lowercase letters	lowercase letters identify an element that must be replaced with a user-selected value. CRndd could be entered as CRA03.
CAPITAL LETTERS	Capital letters must be entered as shown for input, and will be printed as shown in output. DPndd means "enter DP followed by the values for ndd."
[ ]	An element inside brackets is optional. Several elements placed one under the other inside a pair of brackets means that the user may select any one or none of those elements. [KEYM] means the term "KEYM" may be entered.
{ }	Elements placed one under the other inside a pair of braces identify a required choice. {A / id} means that either the letter A or the value of id must be entered.
. . .	The horizontal ellipsis indicates that a previous bracketed element may be repeated, or that elements have been omitted. name ,name . . . means that one or more values may be entered, with a comma inserted between each name value.
. . .	The vertical ellipsis indicates that commands or instructions have been omitted.

	OPEN MASTER-FILE. . means that there . are one or more . statements omitted CLOSE MASTER-FILE. between the two commands.
Numbers and special characters	Numbers that appear on the line (i.e., not subscripts), special symbols, and punctuation marks other than dotted lines, brackets, braces, and underlines appear as shown in output messages and must be entered as shown when input. (value) means that the proper value must be entered enclosed in parentheses; e.g., (234).
subscripts	Subscripts indicate a first, second, etc., representation of a parameter that has a different value for each occurrence. $name_1$, $name_2$, $name_3$ means that three successive values for name should be entered, separated by commas.

6.6 COBOL Language Structure

6.6.1 Introduction: COBOL (the COmmon Business Oriented Programming Language) consists of selected English words that impart key meanings to the COBOL compiler. The language is arranged into statements, sentences, and paragraphs in a manner similar to written English. The words of this language are selected English words (called "reserved words" because they cannot be used in any other context and are listed in Appendix E), names of data and procedures, and numeric or non-numeric "literals". Punctuation is permitted, but the only meaningful punctuation symbol is the period.

COBOL words are arranged into statements using the formats described in this manual in the separate discussion of each statement. One or more statements compose a sentence, which is terminated by a period. One or more sentences, in turn, constitute a paragraph, which can be given a name so that control can pass to the paragraph by referencing its name elsewhere in the program. Similarly, several paragraphs make up a section that can also have a name and, in addition, can be loaded as an "overlay". Several sections constitute a division. There are four divisions in a COBOL program, each describing a different, important part of the program.

Structural hierarchy of the COBOL programming language and the purpose of each level therein are:

. The COBOL Program	Contains all the information required to perform a given task on the computer.
. Division	Describes a specific category of information essential to the compiler; or, in the case of the PROCEDURE DIVISION, specifies processing steps.
. Section	In the PROCEDURE DIVISION, defines the smallest block of the program that can be loaded at one time or as an overlay, in other divisions, groups a particular type of information within a division.
. Paragraph	Comprises one or more sentences forming the smallest block of the program that can be referenced by name.
. Sentence	Consists of one or more statements terminated by a period.

- Statement — Consists of a group of words that perform only one operation or function in the program.
- Word — Consists of a group of characters and/or symbols that provide the structural basis of a statement.

<u>6.6.2 Character Set</u>: The complete character set for NETWORK ANS COBOL consists of the 51 characters listed below:

Character	Meaning
0-9	digits
A-Z	letters
	space (blank)
+	plus sign
-	minus sign (hyphen)
*	asterisk
/	stroke (virgule, slash)
=	equals sign
$	currency sign
,	comma (decimal point)
;	semicolon
.	period (decimal point)
"	double quotation mark
(	left parenthesis
)	right parenthesis
>	greater than sign

<	less than sign
'	single quotation mark

6.6.3 Characters Used for Punctuation: The following characters are used for punctuation:

Character	Meaning
	space
,	comma
;	semicolon
.	period
"	quotation mark
(	left parenthesis
)	right parenthesis

The following general rules of punctuation apply in writing a COBOL source program:

1. When any punctuation mark is indicated in a format in this publication, it is required in the program.

2. At least one space must appear between two successive words and/or parenthetical expressions and/or literals. Two or more successive spaces are treated as a single space, except within nonnumeric literals.

3. An arithmetic operator or an equal sign must be preceded by a space and followed by a space. A unary operator may be preceded by a left parenthesis.

4. A comma may be used as a separator between successive operands

of a statement. An operand of a statement is shown in a format as a lower-case word.

5. In the procedure division, a semicolon may be used to separate a series of clauses. An example: DATA RECORD IS TRANSACTION; RECORD CONTAINS 80 CHARACTERS.

6.6.4 Characters Used for Editing: Editing characters are single characters or specific two-character combinations belonging to the following set:

Character	Meaning
B	space
0	zero
+	plus
-	minus
CR	credit (not verified)
DB	debit (not verified)
Z	zero suppression (not verified)
*	check protection (not verified)
$	currency sign (not verified)
,	comma (not verified)
.	period (decimal point) (not verified)

(For applications, see the discussion of alphanumeric edited and numeric edited data items in "Data Division", Sections 6.18.6.4 and 6.18.6.5).

6.6.5 Characters Used for Relation Conditions: A relation character is a character that belongs to the following set:

Character	Meaning
>	greater than
<	less than
=	equal to

Relation characters are used in relation conditions (discussed in "Procedure Division" Section 6.19.4.1). The word NOT may precede the relation character.

6.7 Words

6.7.1 Definition and Application: The character set for words comprises 37 characters: the letters A through Z, the digits 0 through 9, and the hyphen. A word is composed of a combination of not more than 30 such characters chosen from this set with the following exceptions:

1. A word cannot begin or end with a hyphen.

2. The space (blank) is not an allowable character in a word and is used as a word separator. Where a space (blank) is required, more than one may be used except for the restrictions stated in Section 6.14, "Reference Format". A word is ended by a space, period, right parenthesis, comma, or semicolon.

Rules for using punctuation characters in connection with words are:

1. If ANS-68 compatibility is desired, a space should follow a

period, comma, or semicolon when one of these punctuation characters is used to terminate a word, and a space should not immediately follow a left parenthesis or immediately precede a right parenthesis.

2. A space must not immediately follow a beginning quotation mark or precede an ending quotation mark unless a space is desired in the literal (which is enclosed in quotation marks).

6.7.2 Data-Name: A data-name is a word with at least one non-numeric character that names a data item in the DATA DIVISION. A space (blank) is not allowed within a data-name, and ANS COBOL reserved words must not be used. (See appendix E, "NETWORK ANS COBOL Reserved Words".)

6.7.3 Procedure-Name: A procedure-name is either a paragraph-name or a section-name. A procedure-name may be composed solely of numeric characters. However, two numeric procedure-names are equivalent only when they are composed of the same number of digits and have the same value: for example, 0023 is not equivalent to 23.

6.7.4 Literal: A literal is a string of characters whose value is defined by the set of characters composing the literal. Every literal is one of two types: non-numeric or numeric.

A non-numeric literal is a string of any allowable ASCII characters (including reserved words, but excluding the quotation mark character) up to 120 characters in length, bounded by quotation marks. The double quotation mark (") is used. The value of a non-numeric literal is the string of characters itself, excluding the quotation marks. Any spaces enclosed in the quotation marks are part of the literal and therefore part of the value. All non-numeric literals are classed as alphanumeric.

A numeric literal is a string of characters selected from digits 0 through 9 (to a maximum of 15 digits), the plus sign, minus sign, and decimal point. The value of a numeric literal is the algebraic quantity represented by the characters in the literal. Every numeric literal is classed as numeric.

Rules for the formation of numeric literals are:

1. The literal must contain at least one digit.

2. The literal must not contain more than one sign character. If a sign is used, it must appear as the leftmost character of the literal. If the literal is unsigned, it is positive.

3. The literal must not contain more than one decimal point. If the literal contains no decimal point, it is an integer.

If a literal conforms to the rules for formation of numeric literals but is enclosed in quotation marks, it is a non-numeric literal, i.e., alphanumeric, and is treated as such by the compiler.

6.7.5 Figurative-Constants: Figurative-constants are certain constants to which fixed data-names are assigned. Such data-names must not be bounded by quotation marks when used as figurative-constants. Singular and plural forms of figurative-constants are equivalent and may be used interchangeably.

Fixed data-names and their meanings:

ZERO-----------------------------	Represents the value 0, or one or
ZEROS	more of the character 0, depending
ZEROES	on context.

SPACE SPACES	Represents one or more blank spaces
HIGH-VALUE HIGH-VALUES	Represents one or more characters that have the highest value in the ASCII collating sequence. NOTE: All machines except Intel use 8 bit characters. Intel uses 7 bit characters.
LOW-VALUE LOW-VALUES	Represents one or more characters that have the lowest value in the ASCII collating sequence.
QUOTE QUOTES	Represents one or more occurrences of the quotation mark character. The word QUOTE cannot be used in place of a quotation mark in a source program to bound a non-numeric literal.
ALL literal	Represents one or more of the string of characters comprising the literals. The literal must be either a non-numeric literal or a figurative-constant other than ALL literal. When a figurative—constant is used, the word ALL is redundant and is used for readability only.

When a figurative-constant represents a string of one or more characters, the compiler determines the length of the string from context in accordance with the following rules:

1. When a figurative-constant is associated with another data item, that is, when the figurative-constant is moved to or compared with another data item, the string of characters specified by the figurative-constant is repeated--character by character on the right--until the size of the resultant string is equal to the size (in characters) of the associated data item.

2. When a figurative-constant is not associated with another data item, that is, when the figurative-constant appears in a DISPLAY or STOP statement, the length of the string is one character. The figurative-constant ALL literal may not be used with DISPLAY or STOP.

A figurative-constant can be used wherever a literal appears in the format, except that whenever the literal is restricted to having only numeric characters.

6.7.6 Reserved Words: Reserved words are used for syntactical purposes and cannot appear as user-defined words. (See Appendix E, "NETWORK ANS COBOL Reserved Words.") The three types of reserved words are key words, optional words, and connectives.

6.7.7 Key Words: A key word is required when the format in which the word appears is used in a source program. Within each format such words are uppercase and underlined. The three types of key words are:

1. Verbs such as ADD, READ, and PERFORM.

2. Required words (in statement and entry formats) such as TO and GIVING.

3. Words that have a specific functional meaning such as NUMERIC, and SECTION.

6.7.8 Optional Words: Within each format, uppercase words that are not underlined are called optional words and can appear at user discretion. The presence or absence of each optional word within a format does not alter compiler translation. Misspelling an optional word or its replacement by another word of any kind is not allowed.

<u>6.7.9 Connectives</u>: The two types of connectives are:

1. Qualifier connectives (used to associate a data-name or a paragraph-name with its qualifier) such as OF and IN.

2. Logical connectives (used in the formation of conditions) such as AND, OR, AND NOT, and OR NOT.

6.8 Concept of Computer-Independent Data Description

To make data as computer independent as possible, characteristics or properties of the data are described in relation to a Standard Data Format rather than an equipment orientated format. This Standard Data Format is oriented to general data processing applications; it uses the decimal system to represent numbers (regardless of the radix used by the computer) and the remaining characters in the COBOL character set to describe non-numeric data items.

6.9 Logical Record and File Concept

The following discussion defines file information by distinguishing between the physical aspects of the file and the conceptual characteristics of the data contained within the file.

<u>6.9.1 Physical Aspects of a File</u>: The physical aspects of a file describe data as it appears on the input or output media and include such features as:

1. The mode in which the data file is recorded on the external medium.

— 2. The grouping of logical records within the physical limitations of the file medium.

3. Means by which the file can be identified.

<u>6.9.2 Conceptual Characteristics of a File</u>: The conceptual characteristics of a file are the explicit definition of each logical entity within the file itself. In a COBOL program, the input or output statements refer to one logical record.

It is important to distinguish between a logical record and a physical record. A COBOL logical record is a group of related information, uniquely identifiable and treated as a unit. A physical record is a physical unit of information whose size and recording mode is convenient to a particular computer for the storage of data on an input or output device. The size of a physical record is hardware-dependent and bears no direct relationship to the size of the file contained on a device.

A logical record can be contained within a single physical unit or it may require more than one physical unit to contain it. There are several source language methods available for describing the relationship between logical records and physical units. Once the relationship is established, control of accessibility of logical records as related to the physical unit is the responsibility of the object program. In this manual, references to records are to logical records unless the term "physical record" is specified.

The concept of a logical record is not restricted to file data but applies also to the definition of working-storage and linkage section. Thus, working-storage and linkage section items may be grouped into logical records and defined by a series of Record Description entries.

6.9.3 Record Concepts: The Record Description entry consists of a set of Data Description entries that describe the characteristics of a particular record. Each Data Description entry comprises a level-number followed by a data-name (if required) and a series of independent clauses (as required).

6.9.4 Concept of Levels: A level concept is inherent in the structure of a logical record. This concept arises from the need to specify sub-divisions of a record for the purpose of data reference. Once a subdivision is specified, it may be sub-divided further to permit more detailed data referencing.

The most basic subdivisions of a record - that is, those not further sub-divided - are called elementary items; consequently, a record consists of a sequence of elementary items, or the record itself may be an elementary item.

For ease of reference, a set of elementary items is combined into a group. Each group consists of a named sequence of one or more elementary items. These groups, in turn, may be combined into multiples of two or more, thus, an elementary item may belong to more than one group.

6.9.5 Level-Numbers: A system of level-numbers shows the organization of elementary items and group items. Since records are the most inclusive data items, level-numbers for records start at 01. Less inclusive data items are assigned higher (not necessarily successive) level-numbers to a maximum of 15. Special level-number 77, is an exception to this rule (see below). Separate entries are written in the source program for each level-number used.

A group includes all group and elementary items following it until a level-number less than or equal to the level-number of that group is encountered. The level-number of an item (either an elementary or a group item) immediately following the last elementary item of the previous group must be the same as that of one of the groups to which the prior elementary item belongs.

Noncontiguous working-storage and linkage section items that are not sub-divisions of other items and are not themselves subdivided are assigned the special level-number 77.

6.9.6 Initial Values of Tables: In the WORKING-STORAGE SECTION, initial values of elements within tables are specified in the following way:

The table may be described as a record by a set of contiguous Data Description entries, each of which specifies the "value" of an element, or part of an element, of the table. In defining the record and its element any Data Description clause (USAGE, PICTURE, etc.) may be used to complete the definition, where required. This form is necessary when the elements of the table require separate handling. The hierarchical structure of the table is then shown by the use of the REDEFINES entry and its associated subordinate entries; these subordinate entries, which are repeated due to OCCURS clauses, must not contain VALUE clauses.

6.10 Algebraic Signs

Algebraic signs are used (1) to show whether the value of an item involved in an operation is positive or negative, and (2) to identify the value of an item as positive or negative on an edited report for external use.

Most forms of representation have a standard or normal manner of depicting an operational sign. Thus, an indication that an operational sign is associated with an item is usually sufficient. Since some forms of representation allow alternative methods for depicting operational signs, it is possible to describe certain types of operational signs that deviate from the normal method. Editing sign control characters are used to display the sign of an item and are not operational signs. These editing characters are available only through the use of the PICTURE clause.

6.11 Uniqueness of Data Reference

Every name used in a COBOL source program must be unique, that is, no other name may have the identical spelling.

6.12 Indexing

References can be made to individual elements within a table of like elements by specifying indexing for that reference. An index is assigned to that level of the table by using the INDEXED BY clause in the definition of a table. A name given by the INDEXED BY clause is known as an index-name and is used to refer to the assigned index. An index-name must be initialized by a SET statement before it is used as a table reference. (See "Table-Handling Statements", Section 6.19.9)

The index can be represented by a numeric literal that is an integer or by an index-name. The lowest permissible index value is 1. The highest permissible index value in any particular case is the number of maximum occurrences of the item as specified in the OCCURS clause.

The indices, or set of indices, that identifies the table element is enclosed in parentheses immediately following the table element data-name. The table element data-name appended with a subscript is called a subscripted data-name or an identifier. When more than one subscript appears within a pair of parentheses, the subscripts must be separated by commas.

The composite format of a subscripted data-name is:

```
data-name (subscript-1 [,subscript-2 [,subscript-3]])
```

The composite format of a subscript is:

```
integer-1
index-name-1
```

The following are the restrictions on indexing and subscripting. Tables may have one, two, or three dimensions. Therefore, references to an element in a table may require up to three subscripts or indexes.

An index can be modified only by the SET, SEARCH, and PERFORM statements. Data items described by the USAGE IS INDEX clause permit storage of the values of the index-names as data without conversion; such data items are called index data items.

6.13 Format Notation

The format of a COBOL statement is described in this manual using the uniform notations itemized below. (See also Command Syntax Notation, Section 6.5)

1. A COBOL reserved word, printed entirely in capital letters, is a word that is assigned specific meaning in the COBOL system. It must not be used in any context or position other than that shown in the format description. SUBTRACT, FROM and ROUNDED in the example below are reserved words.

2. One or more COBOL elements vertically stacked and enclosed in a set of square brackets indicate that this portion of the syntax is optional and may be included or omitted at the discretion of the programmer.

3. A pair of braces is used to enclose vertically stacked COBOL elements when one, and only one, of the elements is required; the others are to be omitted. Refer to the example below.

4. The ellipsis . . . denotes a succession of operands of repeated COBOL elements that may be used in the same particular statement, even though the operands or elements are omitted in the text. An ellipsis is associated with the last complete element preceding it, i.e., if a group of operands and key words are enclosed within brackets and the right bracket is followed by the ellipsis, the group (and not merely the last operand) may be repeated in its entirety.

5. An underlined word is required unless the part of the format containing it is itself optional (enclosed in brackets). If a required word is omitted or incorrectly spelled, it causes an error in the interpretation of the program.

6. All COBOL words that are optional words (not underlined) may be included or omitted at the option of the programmer. These words are used only for the sake of readability; misspelling, however, constitutes an error.

7. Lowercase words represent information that is supplied by the programmer. The nature of the information required is indicated in each case. In most instances the programmer is required to provide an appropriate data-name, procedure-name, literal, etc. Refer to the example below.

8. The period is the only required punctuation. Other punctuation, where shown, is optional.

9. Special characters (such as the equal sign) are essential where shown, although they may not be underlined.

10. The notation ▲ indicates the position of an assumed decimal point in an item.

11. A numeric character with a plus or minus sign above it ($\overset{\pm}{n}$) indicates that the value of the item has an operational sign that is stored in combination with the numeric character.

12. Character positions in storage are shown by boxes $\boxed{A}\boxed{B}\boxed{C}\boxed{D}$ An empty box means an unpredictable result.

13. The symbol △ indicates a space (blank).

The following example shows a typical COBOL statement and use of the notation described above.

SUBTRACT {identifier-1 / literal-1} [, identifier-2 / , literal-2] ...FROM identifier-m [ROUNDED]

6.14 Reference Format

6.14.1 General Description: The reference format, which provides a method for describing COBOL source programs, is described in terms of character positions or columns on a CRT line. The line may be up to 80 characters in length. Rules for spacing given in the discussion of the reference format take precedence over all other rules for spacing. Division of a source program is ordered as follows: the IDENTIFICATION DIVISION, then the ENVIRONMENT DIVISION, then the DATA DIVISION, then the PROCEDURE DIVISION. Each division must be written according to the rules for the reference format.

The standard COBOL line format is as follows:

Columns 1-6	six digit sequence number
Column 7	continuation area
Columns 8-11	area A
Columns 12-72	area B
Columns 73-80	identification area

The MICROSOFT COBOL uses this format. The Data General COBOL may use this card format, but the preferred format, called CRT format, eliminates the sequence number field and uses free format for the remaining fields:

Column 1	Area A, Continuation (hyphen character), comment indicator (*).
Columns 2-80	Area B.

The M6800 COBOL programs use the format:

Columns 1-4	four digit line number
Column 6	continuation area
Columns 7-10	area A
Columns 11-71	area B

Conversion programs between these formats have been written and are available.

6.14.2 Reference Format Representation:

Margin L	designates the line number area.
Margin C	represents the continuation column. An * (asterisk) in margin C causes the compiler to treat the entire line as a comment line. A / (slash) in Margin C will cause the compiler to start printing the source program on the top of a new page. The remainder of the line is treated as a comment. A - (hyphen) in Margin C is used to continue a non-numeric literal from one line to the next.
Margin A or Area A	represents the first column in the coding area.
Margin B or Area B	represents the second area in coding portion of the line.

6.14.3 Continuation of Non-Numeric Literals: When a non-numeric literal is continued from one line to another, a hyphen is placed in Margin C of the continuation line and a quotation mark is placed in Area B following the hyphen. All spaces at the end of the continued line and any spaces following the quotation mark of the continuation line and preceding the final quotation mark of the literal are considered part of the literal. Note that each line in this system is terminated by a carriage return. If it is desired that additional spaces are to be included at the end of the continued line, they must actually be typed in.

6.14.4 Division, Section, and Paragraph Formats:

Division Header. The division header must be the first line of a division reference format. The division header starts in Margin A with the division-name followed by a space, the word DIVISION, and a period. No other text may appear on the same line as the division header.

Section Header. The section header begins on any line except the first line of a division reference format. The section header starts in Area A with the section-name followed by a space, the word SECTION, and a period followed by a space. No other text may appear on the same line as the section header.

A section consists of paragraphs in the ENVIRONMENT and PROCEDURE DIVISION, and Data Description entries in the DATA DIVISION. Paragraph-names but not section-names are permitted in the IDENTIFICATION DIVISION.

Paragraph-Name and Paragraphs. The name of a paragraph starts in Area A of any line following the first line of a division reference format (or section header if sections are used) and ends with a period followed by a space.

A paragraph consists of one or more successive sentences. The first sentence in a paragraph begins in Area B of either the same line as the paragraph-name or the line immediately following. Successive sentences begin either in Area B of the same line as the preceding sentence or in Area B of the next line.

A sentence consists of one or more statements followed by a period and a space. When the sentences of a paragraph require more than one line, they may be continued on successive lines.

6.14.5 DATA DIVISION Entries: Each DATA DIVISION entry begins with a level indicator or a level-number followed by at least one space, the name of a data item, and a sequence of independent clauses describing the data. The last clause of an entry is always terminated by a period followed by a space.

There are two types of DATA DIVISION entries: those that begin with a level indicator and those that begin with a level-number.

FD is a level indicator. In DATA DIVISION entries that begin with a level indicator, the level indicator begins in Area A, followed by its associated file-name and appropriate descriptive information in Area B.

DATA DIVISION entries that begin with level-numbers are called Data Description entries. A level-number may be one of the following set: 1 through 15, 77. Level-numbers less than 10 are written as zero followed by a digit. At least one space must separate a level-number from the word succeeding it. In DATA DIVISION entries that begin with a Data Description entry, the first Data Description entry starts with a level-number in Area A, followed by the descriptive information in Area B.

6.15 COBOL Input/Output Processing

6.15.1 COBOL Files: NETWORK ANS COBOL supports sequential and indexed sequential file organizations and all access methods appropriate for these organizations.

6.15.2 File Organization:

6.15.2.1 Indexed File Organization: Indexed files are those in which each record is associated with an identifying key. Indexed files may be accessed directly or sequentially; however, they must be assigned to input/output devices capable of direct access. Indexed file organization is indicated in the COBOL language by the statement ORGANIZATION IS INDEXED in the FILE-CONTROL paragraph of the ENVIRONMENT DIVISION.

6.15.2.2 Sequential File Organization: A sequential file is one whose records are organized in a consecutive manner. There is no identifying key associated with each record; therefore, records can be accessed sequentially only. Consecutive files may be assigned to any type of input/output device. Consecutive file organization is indicated when ORGANIZATION IS SEQUENTIAL is written or when the ORGANIZATION clause is omitted altogether.

6.15.3 File Access: The three methods of accessing files are sequential, random, and dynamic.

6.15.3.1 Sequential Access: Sequential access is the technique of referencing records serially within a file. The order in which records are read or written is determined implicitly by relative physical position within the file. This access method is specified by the ACCESS MODE IS SEQUENTIAL clause or it is implied by the omission of that clause.

6.15.3.2 Random Access: Random access is the technique of reading and writing records of a file in an order dictated by the programmer. It may only be used with ORGANIZATION IS INDEXED files. The record to be referenced is indicated by the value of a key at the time that the input/output command is issued. This access method is specified by the ACCESS MODE IS RANDOM clause. The RECORD KEY clause specifies the key.

6.15.3.3 Dynamic Access: Dynamic access mode allows the file to be accessed either sequentially or randomly depending upon the I/O statement. It may only be used with files having ORGANIZATION IS INDEXED. This access mode is specified by the ACCESS IS DYNAMIC clause. The RECORD KEY clause is also required.

6.15.4 Record Keys: Files having an indexed organization may access their records both sequentially and by a user specified key. The variable used as the key is specified by the RECORD KEY clause. The format of this clause is:

RECORD KEY IS data-name-1

where data-name-1 is an alphanumeric data item with no more than 8 characters. If data-name-1 has fewer than 8 characters, it should be followed by a filler data item with enough characters such that the number of characters in the filler and data-name-1 sum to 8. This restriction is entirely the result of the M6800 file management system.

6.15.5 File-Handling Methods: A file-handling method is the effect of the combination of access technique, file organization, and the manner in which the file is opened.

6.15.5.1 Sequential Access:

1. OPEN OUTPUT. This combination creates a consecutive file. The new records replace any previous contents of the file.

2. OPEN EXTEND. New records will be added to the end of a consecutive file.

3. OPEN INPUT. If the file organization is sequential, READ statements obtain records serially in the order in which they were originally written. If the file organization is indexed, READ statements obtain records serially in key value order (not necessarily in the order in which they were written).

6.15.5.2 Random Access:

1. OPEN OUTPUT. This combination creates an indexed file. A RECORD KEY must be specified and its contents consulted upon each WRITE statement.

2. OPEN INPUT. Organization of the file must be indexed. A RECORD KEY must be specified and the contents consulted for each READ statement to locate the desired record within the file.

3. OPEN INPUT-OUTPUT. The sole essential difference between OPEN INPUT and OPEN INPUT-OUTPUT is that the latter permits the file to be updated instead of merely referenced; thus, WRITE statements are allowed to address the file.

6.15.6 Input/Output Processing Summary: Table 4 summarizes the COBOL language file manipulation statements. Each file must be named in an ENVIRONMENT DIVISION SELECT sentence and defined by an FD entry in the DATA DIVISION. Each of the language elements concerned is described fully in succeeding chapters of this manual.

6.16 IDENTIFICATION DIVISION

6.16.1 General Description: The format of the IDENTIFICATION DIVISION is:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. program-name.
 AUTHOR. comment-sentences.
 INSTALLATION. comment-sentences.
 DATE-WRITTEN. comment-sentences.
 DATE-COMPILED. comment-sentences.
 SECURITY. comment-sentences.
```

The IDENTIFICATION DIVISION specifies information essential to identification such as the name of the program, the date the program was written, programmer's name, security, etc. The listing contains all information specified in this division, but the specified information in no way affects the object program. Allowable information is presented in seven separate paragraphs: one mandatory, the others optional. If the opticnal paragraphs are included in the program, they must be in the order indicated above.

6.16.2 Organization: The IDENTIFICATION DIVISION header is always the first line in a source program and appears as shown above, including the punctuation. This header and the fixed paragraph-name(s) must conform to COBOL Coding Sheet specifications. Only the PROGRAM-ID paragraph is mandatory; all others are optional. Comment-sentences for the optional paragraphs consist of any sentence or group of sentences.

6.16.3 PROGRAM-ID Paragraph: The PROGRAM-ID paragraph must always appear as the first paragraph in the IDENTIFICATION DIVISION. This paragraph permits the programmer to declare the name of the source program.

6.16.4 DATE-COMPILED Paragraph: The DATE-COMPILED paragraph should be used to provide the compilation data in the source program listing.

Example: The IDENTIFICATION DIVISION of a typical program might be written:

```
IDENTIFICATION DIVISION
PROGRAM-ID. Inventory
AUTHOR. John Smith
DATE-WRITTEN. October 15, 1977.
DATE-COMPILED. November 1, 1977.
```

REMARKS. This program prints the inventory report.

Table 4. File Manipulation Statements

File Organization	ACCESS MODE IS	type of OPEN STATEMENT	PERMISSIBLE I/O Statement	RECORD KEY Required
Sequential	SEQUENTIAL (or unspecified)	INPUT	READ. . . AT END	No
		OUTPUT	WRITE. . . BEFORE AFTER ADVANCING	No
		EXTEND	WRITE. . .	No
Indexed	SEQUENTIAL (or unspecified)	INPUT	START. . . INVALID KEY READ . . . AT END	Yes
		OUTPUT	WRITE. . . INVALID KEY	Yes
		I/O	START. . . VALID KEY READ. . . AT END WRITE. . . INVALID KEY REWRITE. . . INVALID KEY DELETE. . . INVALID KEY	Yes

Table 4. (Continued)

File Organization	ACCESS MODE IS	Type of OPEN STATEMENT	PERMISSIBLE I/O Statement	RECORD KEY Required
Indexed	RANDOM	INPUT	READ. .INVALID KEY	Yes
		OUTPUT	WRITE .INVALID KEY	Yes
		I/O	READ. .INVALID KEY WRITE .INVALID KEY REWRITE .INVALID KEY DELETE. .INVALID KEY	Yes
Indexed	DYNAMIC	INPUT	START .INVALID KEY READ. .INVALID KEY READ NEXT. .AT END	Yes
		OUTPUT	WRITE .INVALID KEY	Yes
		I/O	START .INVALID KEY READ. .INVALID KEY READ NEXT. .AT END WRITE .INVALID KEY REWRITE .INVALID KEY DELETE. .INVALID KEY	Yes

6.17 ENVIRONMENT DIVISION

6.17.1 General Description: The format of the ENVIRONMENT DIVISION is:

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. source-computer entry.

OBJECT-COMPUTER. object-computer entry.

INPUT-OUTPUT SECTION.

FILE-CONTROL. file-control entry.

I-O-CONTROL. input/output control entry.

The ENVIRONMENT DIVISION describes those aspects of the data processing program that depend on the physical characteristics of a specific computer. The information presented in this division enables the compiler to link the operations indicated in the DATA and PROCEDURE DIVISIONs to the physical aspects of computer hardware and the executive system that is to execute the object program. Thus, the ENVIRONMENT DIVISION is entirely computer-oriented and changes for each of the machines on the network.

The ENVIRONMENT DIVISION is divided into the CONFIGURATION SECTION and the INPUT-OUTPUT SECTION.

The CONFIGURATION SECTION deals with the characteristics of the computing system on which the source program is to be compiled and on which the object program is to operate. This section is divided into two paragraphs: the SOURCE-COMPUTER paragraph describing the computer on which the COBOL compiler is to run and the OBJECT-COMPUTER paragraph defining the computer on which the translated program is to run.

The INPUT-OUTPUT SECTION provides information needed to control transmission and handling of data between external media and the object program. There are two fixed paragraph names in this section: the FILE-CONTROL paragraph, naming and associating the files with external media and the I/O CONTROL paragraph specifying certain other file information.

6.17.2 Configuration Section:

6.17.2.1 SOURCE-COMPUTER Paragraph: The format of this paragraph is:

SOURCE-COMPUTER. computer name.

The SOURCE-COMPUTER paragraph enables the programmer to describe to the compiler the computing system on which source program translation is to take place. The rules for computer-name are:

MACHINE	COMPUTER-NAME ENTRY
M6800	Treated as comment. M6800 recommended
Intel MICROSOFT	Treated as comment. Intel 8080 recommended.
Data General CS-20	CS-20

6.17.2.2 OBJECT-COMPUTER Paragraph: The format of this paragraph is:

OBJECT-COMPUTER

computer-name (MEMORY SIZE integer CHARACTERS)

The rules for the contents of the OBJECT-COMPUTER paragraph are the same as for the SOURCE-COMPUTER paragraph.

6.17.3 The INPUT/OUTPUT Section:

The INPUT-OUTPUT section consists of the FILE-CONTROL and I/O CONTROL paragraphs.

6.17.3.1 File Control Paragraph: The format of the File Control paragraph is:

FILE-CONTROL.

SELECT sentences

The format and meaning of the SELECT sentence varies among the machines.

6.17.3.1.1 SELECT Sentence for M6800 COBOL:

SELECT file-name-1 {(ASSIGN-clause) (ASSIGN-clause)} {(ORGANIZATION-clause (RECORD-KEY-clause)}

Each file defined in the FILE SECTION of the DATA DIVISION must be named once and only once as file-name-1 in a SELECT sentence. Each select file must have a File Description entry in the DATA DIVISION.

The following clauses that compose the SELECT sentence are all optional; except for the ASSIGN clause, they may be written in any order.

ASSIGN Clause. The format of this required clause is:

(ASSIGN TO implementor-name-1)

The ASSIGN clause permits a file to be associated with a particular type of hardware device.

Acceptable implementor-names are:

PRINTER

DISK diskid:number

Where: diskid--represents an eight character disk file.

identification number--represents the file number for the suffix for the diskid.

(Refer to the COBOL operations reference manual for an explanation of the meaning of diskid: number as related to different disk types.)

ORGANIZATION Clause. The format of this clause is:

```
                    SEQUENTIAL
ACCESS MODE IS      RANDOM
                    DYNAMIC
```

SEQUENTIAL denotes that records are obtained or placed equentially: that is, the next logical record is available from the file on a READ statement execution, or a specific logical record is placed in the next position in the file on a WRITE statement execution.

If RANDOM or DYNAMIC is specified, the RECORD KEY clause (see below) must also be specified and the file must be assigned to a direct-access device. In this case, the specified logical record (located using RECORD KEY data-name contents) is made available from the file on a READ statement execution, or is placed in a specific location on the file (located using RECORD KEY data-name contents) on a WRITE statement execution. DYNAMIC access mode differs from

RANDOM access mode in that the file may be accessed sequentially or randomly, depending on the I/O statement. That is, after a record is located by a random read, the records following it can be read sequentially. Another random read can then be issued to switch back to random access.

Sequential access is assumed when these clauses are omitted.

RECORD KEY clause. The format of this clause is:

RECORD KEY IS data-name WITH DUPLICATES

The RECORD KEY clause must be specified if INDEXED organization is specified; it is not meaningful to SEQUENTIAL organization. Data-name must be contained within the record. In addition, it must conform to the rules for the file management system outlined in the COBOL operations reference manual.

The contents of data-name are used by the READ and WRITE statements to locate a specific record in a mass storage file. The symbolic identity of the record to be read or written must be placed in data-name before the appropriate input/output statement is executed

The optional WITH DUPLICATES clause specifies that records with duplicate keys are to be permitted in the file.

6.17.3.1.2 SELECT Sentence MICROSOFT Intel 8080 COBOL:

6.17.3.1.2.1 Sequential Files: For each file having records described in the Data Division's File Section, a Sentence-Entry (beginning with the reserved word SELECT) is required in the FILE-CONTROL paragraph. The format of a Select Sentence-Entry for a sequential file is:

> SELECT file-name ASSIGN TO DISK | PRINTER
> (RESERVE integer AREAS | AREA)
> (FILE STATUS IS data-name-1)
> (ACCESS MODE IS SEQUENTIAL) (ORGANIZATION IS SEQUENTIAL).

All phrases after "SELECT file-name" can be in any order. Both the ACCESS and ORGANIZATION clauses are optional for sequential input-output processing. For Indexed or Relative files, alternate formats are available for this section, and are explained in the sections on Indexed and Relative files (6.12-6.14).

If the RESERVE clause is not present, the compiler assigns buffer areas. An integer number of buffers specified by the Reserve clause may be from 1 to 7, but any number over 2 is treated as 2.

In the FILE STATUS entry, data-name-1 must refer to a two-character Working-Storage or Linkage item of category alphanumeric into which the run-time data management facility places status information after an I/O statement. The left-hand character of data-name-1 assumes the values:

- '0' for successful completion
- '1' for End-of-File condition
- '2' for Invalid Key (only for Indexed and Relative files)
- '3' for a non-recoverable (I/O) error
- '9' for implementor-related errors (see User's Guide)

The right-hand character of data-name-1 is set to '0' if no further status information exists for the previous I/O operation. The following combinations of values are possible:

File Status Left	File Status Right	Meaning
'0'	'0'	O-K.
'1'	'0'	EOF
'3'	'0'	Permanent error
'3'	'4'	Disk space full

For values of status-right when status-left has a value of '2', see the Sections on Indexed or Relative files (6.12-6.14).

6.17.3.1.2.2 Indexed Sequential Files: For an Indexed file organization, the SELECT entry must specify ORGANIZATION IS INDEXED, and the ACCESS clause format is:

ACCESS MODE IS SEQUENTIAL RANDOM DYNAMIC

A file whose organization is indexed can be accessed either sequentially, dynamically or randomly.

Sequential access provides access to data records in ascending order of RECORD KEY values.

In the random access mode, the order of access to records is controlled by the programmer. Each record desired is accessed by placing the value of its key in a key data item prior to an access statement.

In the dynamic access mode, the programmer's logic may change from sequential access to random access, and vice versa, at will.

6.17.3.1.2.3 RECORD KEY Clause: The general format of this clause, when required, is:

RECORD KEY IS data-name-1

where data-name-1 is an item defined within the record descriptions of the associated file description, and is a group item, an elementary alphanumeric item or a decimal field. A decimal key must have no P characters in its PICTURE, and it may not have a SEPARATE sign, No record key may be subscripted.

If random access mode is specified, the value of data-name-1 designates the record to be accessed by the next DELETE, READ, REWRITE or WRITE statement. Each record must have a unique record key value.

6.17.3.1.2.4 File Status Reporting: If a FILE STATUS clause appears in the ENVIRONMENT DIVISION for an Indexed organization file, the designated two-character data item is set after every I/O statement. The following table summarizes the possible settings.

Status Data Item LEFT Character	Status Data Item RIGHT Character				
	No Further Description (0)	Sequence Error (1)	Duplicate Key (2)	No Record Found (3)	Disk Space Full (4)
Successful Completion (0)	X				
At End (1)	X				

Invalid Key (2)		X	X	X	X
Permanent Error (3)	X				

Sequence error arises if access mode is sequential when WRITEs do not occur in ascending sequence for an Indexed file, or the key is altered prior to REWRITE or an unsuccessful READ preceded a DELETE or REWRITE. The other settings are self-explanatory. The left character may also be '9' for implementor-defined errors; see the User's Guide for an explanation of these.

Note that "Disk Space Full" occurs with Invalid Key (2) for Indexed and Relative file handling, whereas it occurred with "Permanent Error" (3) for sequential files.

If an error occurs at execution time and no AT END or INVALID KEY statements are given <u>and</u> no appropriate declarative ERROR section is supplied and no FILE STATUS is specified, the error will be displayed on the console and the program will terminate.

<u>6.17.3.1.3 SELECT Sentence for Data General CS-20</u>: SELECT names internal program files and associates each one with a given hardware device and external file name. Also, logical file organization, access method, I/O status and keys may be defined if required by the program. Refer to Figure 6.1 for examples of the SELECT statement.

If the external file-name option is omitted from the SELECT statement, the system file-names are supplied by default. Refer to the following table for a list of the default file-names.

SYSTEM FILE-NAMES

Device		File-name
	PRINTER	$LPT
	PRINTER-1	$SLPT1
Terminal	KEYBOARD	$TTI
Terminal	DISPLAY	$TTO
	DISK	The first ten characters of the internal (COBOL) file-name with "-" deleted.

6.17.3.1.3.1 Sequential SELECT:

```
SELECT file-name ASSIGN TO  { DISK      }
                            { PRINTER   }
                            { PRINTER-1 }   (.id-lit)
                            { DISPLAY   }
                            { KEYBOARD  }
(;ORGANIZATION IS SEQUENTIAL)
(;ACCESS MODE IS SEQUENTIAL)
(;FILE STATUS IS data-name)
(;DATA SIZE IS integer).
```

6.17.3.1.3.2 Indexed SELECT:

```
SELECT file-name ASSIGN TO DISK(.id-lit)

;ORGANIZATION IS INDEXED

(ACCESS MODE IS   { SEQUENTIAL }
                  { RANDOM     }
                  { DYNAMIC    }   )

;RECORD KEY IS data-name
(;FILE STATUS IS data-name)
(;INDEX SIZE IS integer)
(;DATA SIZE IS integer)
```

FIGURE 6.1 CS-20 SELECT SENTENCE FORMATS

EXAMPLES OF THE SELECT STATEMENT:

(SELECT for a randomly allocated indexed file)

```
SELECT CFILE ASSIGN TO DISK,"DP1F:CFILED";
ORGANIZATION IS INDEXED;
ACCESS MODE IS DYNAMIC;
RECORD KEY IS C-KEY;
FILE STATUS IS CFSTAT.
```

(SELECT for a contiguously allocated indexed file)

```
SELECT CFILE ASSIGN TO DISK,EX-FILE-NAME;
ORGANIZATION IS INDEXED;
ACCESS MODE IS RANDOM;
RECORD KEY IS C-KEY;
FILE STATUS IS CFSTAT;
INDEX SIZE IS 20;
DATA SIZE IS 105.
```

6.17.3.1.3.3 Rules for use: External (System) File Specification--the "id-lit" following the file device type in the SELECT statements is an Interactive COBOL extension. It allows specification of a program external file name. Also, if the device is a disk, an optional device specifier may be used to associate the external file name with a particular disk drive.

An external file-name for an indexed file must not have an extension.

If a data-name is used for the external file-name, the full value of the data-name must be a valid file-name or the file-name must be left justified in the data-item and terminated by a null (LOW-VALUE).

When the external file-name is omitted, file-names are supplied by default. Refer to the following table for a list of these system file-names.

SYSTEM FILE-NAMES

Device		File-name
	PRINTER	$LPT
	PRINTER-1	$LPT1
Terminal	DISPLAY	$TTO
Terminal	KEYBOARD	$TT1
	DISK	The first ten characters of the internal (COBOL) file-name with "S" replacing "-".

The FILE STATUS item must be described as a two character alphanumeric item.

Record keys must be alphanumeric and may be a maximum of 100 characters long.

The DUPLICATES phrase specifies that the value of the associated alternate record key may be duplicated within any of the records in the file. Further, CS interactive COBOL phrase is not specified.

INDEX SIZE specifies the number of 512-byte blocks of contiguous disk storage space to be reserved for the data portion of a sequential, indexed, or relative file when the file is created.

The file device names DISK, PRINTER, DISPLAY, and KEYBOARD are reserved words.

Files assigned to PRINTER or DISPLAY must be sequential and opened in OUTPUT or EXTEND mode only.

Files assigned to KEYBOARD must be sequential and opened in INPUT mode only.

6.17.3.2 I/O CONTROL Paragraph: The format of this paragraph is:

(SAME AREA FOR file-name-1 (,file-name-2) . . . .)

Where the format of the SAME AREA clause is the same for all machines.

When SAME AREA is written, the data areas for all of the files mentioned overlap. Thus, only one of the list of files may be open at the same time. More than one SAME AREA clause may appear in a COBOL program, but no one file-name may appear in more than one such clause.

6.18 DATA DIVISION

6.18.1 General Description: The DATA DIVISION describes data that the object program accepts as input in order to manipulate, create, or produce output. Data to be processed falls into three categories:

1. Data that is contained in files and enters or leaves the internal memory of the computer from a specified area or areas.
2. Data that is developed internally and placed into intermediate or working storage, or into specific format for output reporting purposes.
3. Constants that are defined by the use.

6.18.2 Physical and Logical Aspects of Data Description:

6.18.2.1 DATA DIVISION Organization: The DATA DIVISION is subdivided into the FILE, and WORKING-STORAGE SECTIONS.

The FILE SECTION defines the contents of data files stored on an external medium. Each file is defined by a file description followed by a record description or a series of record descriptions.

The WORKING-STORAGE SECTION describes records and noncontiguous data items that are not part of external data files but are developed and processed internally.

6.18.2.2 DATA DIVISION Structure: The DATA DIVISION is identified by and must begin with the header:

DATA DIVISION.

Each of the sections of the DATA DIVISIONS (except the WORKING-STORAGE SECTION) is optional and may be omitted from the source program. The fixed names of these sections in their required order of appearance as section headers in the DATA DIVISION are:

FILE SECTION.
WORKING-STORAGE SECTION.

Section headers for the FILE SECTION are followed by one or more sets of entries composed of file clauses, followed by associated Record Description entries. WORKING-STORAGE SECTION headers are followed by Data Description entries for noncontiguous items, followed by Record Description entries. See Figure 6.2.

6.18.3 File Section: In a COBOL program the File Description (FD) entry represents the highest level of organization in the FILE SECTION. The FILE SECTION is composed of the section header FILE SECTION and a period, followed by a File Description entry consisting of a level indicator (FD), a data-name, and a series of independent clauses. These clauses specify the size of the physical records, and the names

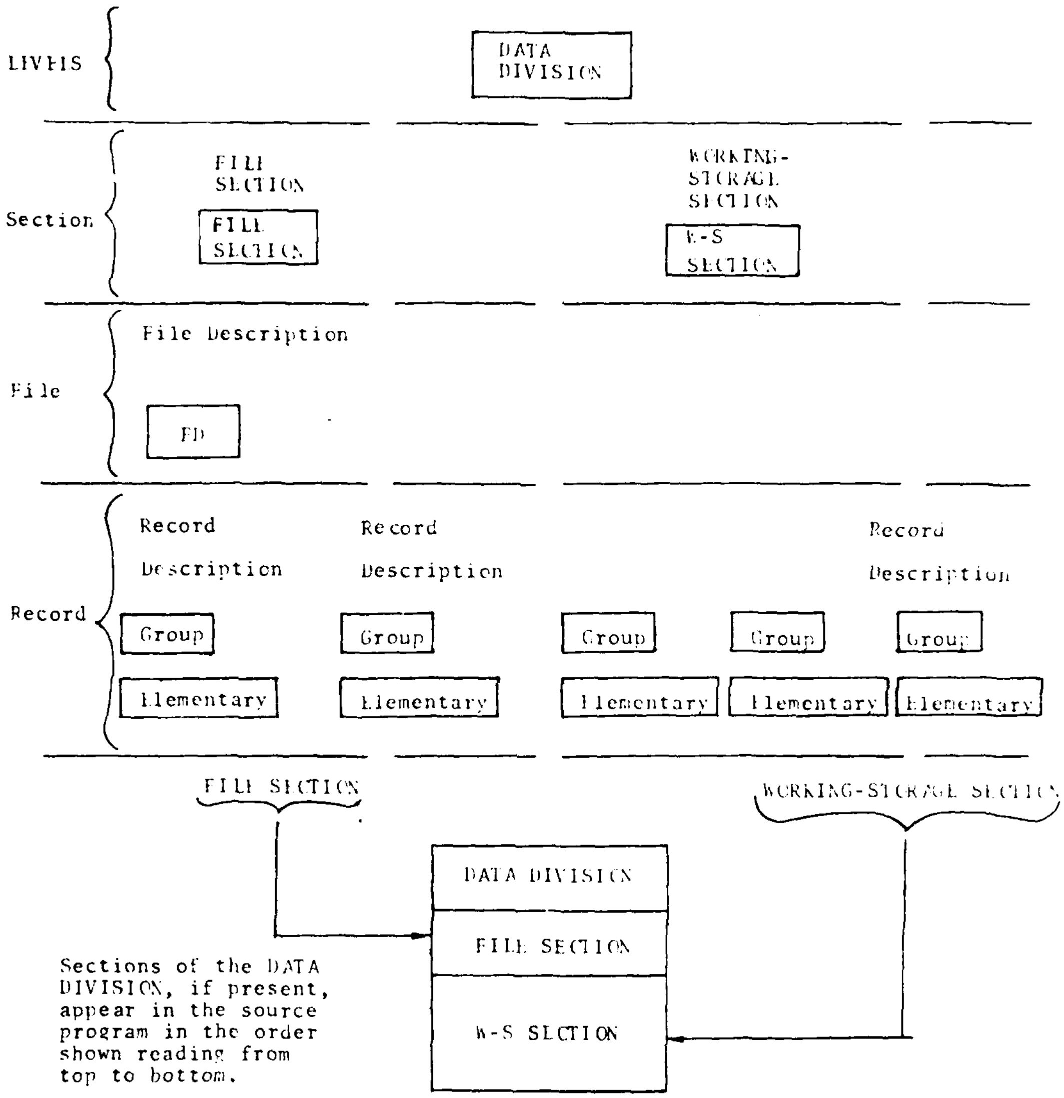


FIGURE 6.2 DATA DIVISION Structure

of the data records and reports that compose the file. The entry itself is terminated by a period. For the Intel 8080 MICROSOFT COBOL, the File Description (FD) Entry also specifies the name of the file as needed by the operating system.

Record Description Structure. A record description consists of a set of Data Description entries that describe the characteristics of a particular record. Each Data Description entry consists of a level-number followed by a data-name, followed by a series of independent clauses, as required. A record description has a hierarchical structure; therefore, the clauses used with an entry may vary considerably, depending upon whether or not it is followed by subordinate entries. The structure of a record description is defined in "Concepts of Levels" in Section 6.9.4; elements allowed in a record description are specified in "Data Description Entries" later in this section (Section 6.18.6).

6.18.4 WORKING-STORAGE SECTION: The WORKING-STORAGE SECTION is composed of the section header WORKING-STORAGE SECTION and a period, followed by Data Description entries for noncontiguous working-storage items and Record Description entries (in that order).

6.18.4.1 Noncontiguous Working-Storage: Items in working-storage that bear no relationship to one another need not be grouped into records provided they do not need to be further subdivided; instead, they are classified and defined as noncontiguous elementary items. Each of these items is defined in a separate Data Description entry that begins with the special level-number 77.

Data clauses required in each Data Description entry are:

1. Level-number.
2. Data-name.
3. The PICTURE clause.

Other record description clauses are optional and can be used to complete the description of the item if necessary.

6.18.4.2 Working-Storage Records: Data elements in working-storage that bear a definite relationship to one another must be grouped into records according to the rules for formation of record description. All clauses that are used in normal input or output record descriptions can be used in a working-storage record description.

6.18.4.3 Initial Values: *The initial value of any item in the* WORKING-STORAGE SECTION except an index data item is specified by using the VALUE clause of the record description. The initial value of any index data item is determined at compilation time.

6.18.5 File Description-Complete Entry Skeleton: The general format of this entry is:

```
FD file-name

LABEL {RECORD IS  } {STANDARD}
      {RECORDS ARE} {OMITTED }

[DATA {RECORD IS  }   data-name-7 [data-name-8]...]
[     {RECORDS ARE}                              ]
```

The File Description entry furnishes information concerning the physical structure, identification, and record names pertaining to a given file.

6.18.5.1 LABEL RECORDS Clause: The format of this clause is:

```
LABEL { RECORD IS      STANDARD }
      { RECORDS ARE    OMITTED  }
```

The OMITTED option specifies that no explicit labels exist for the file or the device to which the file is assigned.

The STANDARD option specifies that standard system labels exist for the file or the device to which the file is assigned. Such labels are written when the file is opened for output and checked automatically by the operating system when the file is opened for input or input/output.

For disk files, the LABEL RECORDS clause varies depending on the machine. For the M6800 COBOL the LABEL RECORDS clause is optional, and if present is treated as a comment. For the Intel 8080 MICROSOFT COBOL and the Data General CS-20 COBOL, LABEL RECORDS are standard.

6.18.5.2 DATA RECORDS Clause: The format of this clause is:

```
[ DATA { RECORD IS   }    data-name-7   (data-name-8). . . ]
[      { RECORDS ARE }                                     ]
```

The DATA RECORDS clause cross-references the description of data records with their associated file description. Each logical record in the file may be named in this clause; the order in which they are listed in the clause is not important. It must be remembered that no two records of the same file are available for processing at the same time; in other words, if one record is read from a file and then another record is read from the same file, the second record replaces the first.

6.18.6 Data Description Entries:

6.18.6.1 General Format:

Level-number { data-name / FILLER } (REDEFINES-clause) (COPY statement)

(PICTURE-clause) (USAGE-clause)

(BLANK-clause) (JUSTIFIED-clause)

(VALUE-clause) (OCCURS-clause)

A Data Description entry (see Table 4) describes characteristics of each item within a data record. Each item is accorded a separate entry that must appear in the order in which the item occurs in the record, since the relative location of each entry is communicated to the compiler by its position in the record description. Each entry consists of a level-number, data-name, and series of clauses terminated by a period.

The reserved word FILLER may be substituted for a programmer-defined data-name when an unused portion of a logical record or data item that is not referenced directly is defined.

Specific formats for individual types of data items are shown below. In each of these formats, clauses that do not appear are categorically forbidden in that data type, while clauses that are mandatory are depicted without brackets.

6.18.6.2 Detailed Formats of Data Items:

Group Item

```
Level-number    {data-name}  [REDEFINES-clause]  [OCCURS-clause]
                {FILLER   }

                [USAGE-clause]

                [VALUE is non-numeric-literal].
```

Example:

```
     01      GROUP-ITEM.
             02     FIELD01 PICTURE X.
             02     FIELD-2 PICTURE X.
```

TABLE 5 VARIOUS DATA DESCRIPTION ENTRIES LISTING

```
01  VARIOUS-DATA-DESC.

    02 ALPHABETIC-TYPES.

       03 A1     PICTURE AAAAAAAA.
       03 A2     REDEFINES A1 PICTURE A(8).
       03 A3     PICTURE A(4) OCCURS 4 TIMES.
       03 A4     PICTURE A(6) VALUE IS 'XYZ A'.
       03 A5     PICTURE A(2) USAGE IS DISPLAY.
       03 A6     PICTURE A(8).
       03 A7     REDEFINES A6 PICTURE A(2) USAGE DISPLAY
                 OCCURS 4 TIMES.

    02 ALPHANUMERIC-TYPES REDEFINES ALPHABETIC-TYPES.

       03 AN1    OCCURS 8 TIMES PICTURE IS X9A.
       03 AN2    PICTURE X(16) USAGE IS DISPLAY.
       03 AN3    REDEFINES AN2 PICTURE X(4) OCCURS 4 TIMES.
```

```
02  ALPHA-EDITED-TYPES.

    03 AE1    PICTURE XXBXXBXX.
    03 AE2    PIC IS XXXXBXX99BOOBXXX.
    03 AE3    REDEFINES AE2 PIC X(10)B09AAX DISPLAY.

02  NUMERIC-EDITED-TYPES.

    03 NE1    PICTURE IS ZZ,999+.
    03 NE2    REDEFINES NE1  PICTURE **,**9-.
    03 NE3    OCCURS 4 TIMES PICTURE ZZZ9.

02  NUMERIC-TYPE.

    03 N1     PICTURE 9999 OCCURS 5 TIMES USAGE DISPLAY.
    03 N2     PIC  S9999 VALUE IS -1234.
    03 N3     REDEFINES N2 PICTURE S99V99.
```

6.18.6.3 Alphanumeric Elementary Item:

level-number { data-name / FILLER } [REDEFINES-clause] [OCCURS-clause]

{ PICTURE / PIC } IS an-type [USAGE IS DISPLAY]

[VALUE IS non-numeric-literal] [{ JUSTIFIED / JUST } RIGHT]

Example:

```
02        CUST-NAME PICTURE X(21) DISPLAY

02        CUST-ADR PIC X (45)
```

6.18.6.4 Alphanumeric Edited Elementary Item :

Level-number { data-name / FILLER } [REDEFINES-clause] [OCCURS-clause]

{ PICTURE / PIC } IS ae-type [USAGE IS DISPLAY]

[VALUE IS non-numeric-literal] [{JUSTIFIED / JUST} RIGHT]

Example:

02 DATE PICTURE XXBXXXBXXXX VALUE '15 DEC 1977'.

6.18.6.5 Numeric Edited Elementary Item:

Level-number {data-name / FILLER} [REDEFINES-clause] [OCCURS-clause]

{PICTURE / PIC} IS {numeric-type BLANK WHEN ZERO / ne-type BLANK WHEN ZERO}

[USAGE IS DISPLAY].

Example:

02 DEPT-NO PIC ZZ999.

02 GROSS-SALES PICTURE SZ,ZZZ,ZZZ,ZZZ.99-.

6.18.6.6 Alphabetic Elementary Item:

Level-number {data-name / FILLER} [REDEFINES-clause] [OCCURS-clause]

{PICTURE / PIC} IS alpha-type [USAGE IS DISPLAY]

[VALUE IS non-numeric-literal]

Example:

02 COUNTY-NAME PICTURE A(35) USAGE IS DISPLAY.

6.18.6.7 ASCII Decimal Elementary Item:

```
Level-number  {data-name}  [REDEFINES-clause]  [OCCURS-clause]
              {FILLER   }

              {PICTURE}    IS numeric-type [USAGE IS DISPLAY]
              {PIC    }

              [VALUE  IS numeric-literal].
```

Example:

```
    02        COST PIC 999V99 VALUE 10.39.
```

6.18.6.8 Packed Decimal Elementary Item:

```
Level-number  {data-name}  [REDEFINES-clause]  [OCCURS-clause]
              {FILLER   }
              {PICTURE}
              {PIC    }    IS numeric-type USAGE IS {COMPUTATIONAL}
                                                    {COMP         }
              [VALUE  IS numeric-literal].
```

Example:

```
    02        TOTAL-RECORDS PIC 9(4) COMPUTATIONAL.
```

6.18.6.9 Index Item:

77 index-name USAGE IS INDEX.

Example:

```
    77 X 1 INDEX
```

6.18.6.10 REDEFINES Clause: The format of this clause is:

Level-number data-name-1 REDEFINES data-name-2

The REDEFINES clause overlaps items in storage (allocates the same storage space for different items at different times) or provides an alternate grouping or description of the same data (redefines an elementary item or a group item.)

The level-numbers of data-name-1 and data-name-2 must be identical.

The REDEFINES clause is not used at the record 01 level in the FILE SECTION. The DATA RECORDS clause in the FD entry indicates the existence of more than one type of record; thus, an implied redefinition exists at the 01 level.

Redefinition begins at data-name-2 and continues until a level-number whose value is equal to or less than data-name-2 is encountered; therefore, between data-names-1 and -2 there must not be a level-number lower than that of data-names-1 and -2. Data-name-1 must follow data-name-2 such that, if data-name-2 is a group entry, the entry for data-name-1 must appear immediately after the entries for all items in that group. However, additional entries that redefine the same area may intervene.

Data-name-1 may be a group or an elementary item irrespective of the nature of the data-name-2 item. If it is a group, the data-name-2 entry is followed by all the entries in that group, since such entries are part of the redefinition; if it is an elementary item, it completely redefines data-name-2. A REDEFINES clause may be specified for an item within the scope of an area being redefined; that is, REDEFINES clauses may be specified for items subordinate to items containing

REDEFINES clauses.

When the REDEFINES clause is used with certain other clauses, entries (except for condition-name entries) containing or subordinate to the REDEFINES clause must not contain VALUE clauses.

When one area is redefined in three or more ways, differences among the COBOL versions exist. If A, B, C and D are all to refer to the same area, the M6800 COBOL and the Intel 8080 MICROSOFT COBOL require that the following sequential structure be used:

Define A
B redefines A
C redefines B
D redefines C

The Data General CS-20 COBOL requires that the structure be:

Define A
B redefines A
C redefines A
D redefines A

When an area is redefined, all descriptions of that area remain in effect for the entire program. The one that is selected depends on the particular reference made to the area. For example, if items A and B share the same area, MOVE X TO A moves X to the area according to the description of A, MOVE Y TO B moves Y to the same area according to the description of B. These statements could be executed anywhere in a program; final contents of the area depend on the order in which they are executed. A table of constant items is redefined so that any item in the table can be referenced by position rather than by individual name. This does not redefine the area according to different patterns, but simply permits the same pattern of items to be considered in a different way.

6.18.6.11 PICTURE Clause: The format of this clause is:

$\left\{\begin{array}{l}\underline{\text{PICTURE}}\\ \underline{\text{PIC}}\end{array}\right\}$ IS character-string

The PICTURE clause describes the general characteristics and editing requirements of elementary items.

The character-string consists of certain allowable combinations of characters in the COBOL character set used as symbols. These allowable combinations determine the category of the item. The five categories of data that can be described with a PICTURE clause are:

1. Alphabetic
2. Alphanumeric
3. Numeric
4. Alphanumeric Edited
5. Numeric Edited

The following rules apply to the use of the PICTURE clause:

1. GENERAL: The number of occurrences of any of the characters indicates the size of an item described by the PICTURE clause. The size may be indicated either by repeating the character or, in a shorthand way, by writing the character once and putting the number of its occurrences in parentheses. Thus, Z (10)9(2) is equivalent to ZZZZZZZZZZ99. A maximum of 30 characters is allowed in a PICTURE clause. This limit does not refer to the number of characters in the item itself, but only to the number of characters (including paren-

theses) used in the PICTURE specifying the item. For example, the same item may be described by a PICTURE containing 12 characters, or by a PICTURE containing only 9 characters, Z(10)9(2). In either case, the actual size of the item is 12 characters. An item containing 75 alphabetic characters may be specified by the PICTURE A(75), which uses only 5 characters, but the same item may not be specified by a PICTURE in which A is repeated 75 times. The size of an alphabetic or alphanumeric item described by the PICTURE is limited to a maximum of 255 characters except for numeric display items, which are limited to 15 digits. The size of an entire Group Item is also limited to 4095 characters.

2. Categories of Data

a. Alphabetic (alpha-type): The PICTURE of an alphabetic item contains only the character A. The number of A's in the character-string denotes the size of the data item, and each A represents one character that at execution time may contain one of the twenty-six letters of the English alphabet or the space character.

b. Alphanumeric (an-type): The PICTURE of an alphanumeric item may contain only the Character X or a combination of the characters X, A, and 9. An X indicates that the corresponding character position of the data item may contain any one of the characters in the ASCII set. When the PICTURE is described with a combination of characters, each character is treated as though it were an X, since no examination of the data placed in the item is made at execution time. Thus, this type of PICTURE description may have documentary significance only to the programmer.

c. Numeric (numeric-type): The PICTURE of a numeric data item may contain only the characters 9, S, and V.

The character 9 represents a digit position containing a numeral and is counted in the size of the item.

The character S indicates the presence of an operational sign and must be written as the leftmost character in the PICTURE.

The character V indicates the position of the assumed decimal point and may occur only once in the character-string. The V does not represent a digit position and therefore is not counted in the size of the item. When a V is written as the last (rightmost) character in the PICTURE, it is redundant.

d. Alphanumeric Edited (ae-type): The PICTURE of an alphanumeric edited item contains any combination of the characters X, A, and 9 together with one or more occurrences of the insertion characters 0 (zero) or B. Each 0 represents a character position into which the character 0 is to be inserted; each b represents a character position into which the space character is to be inserted. Thus, an alphanumeric edited field is one that contains certain character positions into which insertion characters are forced whenever data is stored in the item at execution time.

e. Numeric Edited (ne-type): Editing alters the format and punctuation of data in an item; characters can be suppressed or added. Editing is accomplished by moving a data item to an item described as containing editing symbols. Movement may be direct or indirect: The programmer can specify a MOVE statement or arithmetic statement in which the result of computation is stored in such an item.

Characters that may be used in a PICTURE of a numeric edited item are

9 V $ + - . , 0 B / CR DB Z *

The characters 9 and V are discussed above: their use is exactly the same as in numeric items. The remainder are insertion and replacement characters (see below).

3. Insertion Characters: When an insertion character is specified in the PICTURE, it appears in the edited data item; therefore, the size of the item must reflect these additional characters. Insertion characters and their characteristics are:

$ When a single dollar sign is specified as the leftmost symbol, it appears as the leftmost character in the size of the item.

\+ When a plus sign is specified as the first or last symbol, a plus sign is inserted in the indicated character position of the edited data item provided the data is positive (contains a positive operational sign) or is unsigned. If the data is negative, a minus sign is inserted in the indicated character position. This sign is counted in the size of the item.

\- When a minus sign is specified as the first or last symbol, a minus sign is inserted in the indicated character position of the edited data item provided the data is negative (contains a negative operational sign). If the data is not negative, a blank is inserted in the indicated character position. This sign or blank is counted in the size of the item.

. The period character represents an actual decimal point, as differentiated from an assumed decimal point. When used, a decimal point appears in the edited data item as a character in the indicated character position; therefore, the decimal point is counted in the size of the item. A PICTURE can never contain more than one decimal point, actual or assumed.

, When a comma is used, a comma is inserted in the corresponding character position of the edited data item. It is counted in the size of the item.

0 When a zero is used, a zero is inserted in the corresponding character position in the edited data item. It is counted in the size of the item.

B When a character B is used, a space is inserted in the corresponding character position in the edited data item. It is counted in the size of the item.

/ When the slash character is used, a slash character is inserted in the corresponding character position in the edited data item. It is counted in the size of the item.

CR The credit symbol CR may be specified only at the right end of the PICTURE character-string. It is inserted in the last two character positions of the edited data item provided the value of the data is negative; if the data is positive or unsigned, these last two character positions are set to spaces. Since this symbol always results in two characters (CR or spaces), it is included as two characters in the size of the item.

<u>Table 6: Examples of Insertion Characters</u>

Source Data	Editing PICTURE	Edited Item
4 8	$99	$ 4 8
4 8 ▲ 3 4	$99.99	$ 4 8 . 3 4
4 8 3 4	9,999	4 , 8 3 4
2 9 2	+999	+ 2 9 2
2 9 $\overset{+}{2}$	+999	+ 2 9 2
2 9 $\overset{-}{2}$	+999	- 2 9 2
2 9 $\overset{+}{2}$	-999	- 2 9 2
2 9 $\overset{-}{2}$	999-	2 9 2 -
2 9 2	999-	2 9 2 △
2 4 3 ▲ 2 1	$BB999.99	$ △ △ 2 4 3 . 2 1
2 4 3 ▲ 2 1	$00999.99	$ 0 0 2 4 3 . 2 1
1 1 ▲ 3 $\bar{4}$	99.99CR	1 1 . 3 4 C R
1 1 ▲ 3 4	99.99CR	1 1 . 3 4 △ △
2 3 ▲ 7 6	99.99DB	2 3 . 7 6 D B
2 3 ▲ 7 $\bar{6}$	99.99DB	2 3 . 7 6 △ △
1 2 3 4 5 6	99/99/99	12/34/56

DB The debit symbol DB may be specified only at the right end of the PICTURE. It functions in the same manner as the credit symbol.

4. Replacement Characters: A replacement character suppresses leading zeros in data and replaces them with other characters in the edited data item. Only one replacement character may be used in a PICTURE, although Z or * may be used with any one of the insertion characters. Replacement characters and their characteristics are:

Z One character Z is specified at the left end of the PICTURE character string for each leading zero that is to be suppressed and replaced by blanks in the edited data item. Z's may be preceded by one of the insertion characters $ + or - and interspersed with any of the . , 0 or B insertion characters.

Only the leading zeros that occupy a position specified by Z are suppressed and replaced with blanks. No zeros are suppressed to the right of the first non zero digit whether or not a Z is present, nor are any zeros to the right of an assumed or actual decimal point suppressed unless the value of the data is zero and all the character positions in the item are described by a Z. In this special case, even an actual decimal point is suppressed and the edited item consists of all blanks.

If a $ + or - is present preceding the Z's, it is inserted in the far left character positon of the item even if succeeding zeros in the item are suppressed. In the special case where the value of the data is zero and all the character positions following the $ + or - are specified by Z's, the $ + or - is replaced by a blank.

If a 0 or B or , in the PICTURE is encountered before zero suppression terminates, the character is not inserted in the edited data item but is suppressed, and a blank inserted in its place.

*   The asterisk replaces the leading zeros it edits by an asterisk instead of a blank. It is specified in the same way as the editing character Z and follows the same rules, except that an actual decimal point is never replaced.

$ When the dollar sign is used as a replacement character to suppress leading zeros, it acts as a floating dollar sign and is inserted directly preceding the first nonsuppressed character. One more dollar sign must be specified than the number of zeros to be suppressed. This dollar sign is always present in the edited data whether or not any zero suppression occurs. The remaining dollar signs act in the same way as Z to effect the suppression of leading zeros. No other editing character may precede the initial dollar sign. Each dollar sign specified in a PICTURE is counted in determining the size of the report item.

+   When a plus sign is used as a replacement character, it is a floating plus sign. The plus sign is specified one more time than the number of leading zeros to be suppressed. It functions in the same way as the floating dollar sign: a plus sign is placed directly preceding the first nonsuppressed character if the edited data is positive or unsigned, and a minus sign is placed in this position if the edited data is negative.

-    When a minus sign is used as a replacement character, it is

Table 7: Examples of Replacement Characters

Source Data	Editing PICTURE	Edited Item
0 0 9 2 3	ZZ999	Δ Δ 9 2 3
0 0 9 2 3	ZZZ99	Δ Δ 9 2 3
0 0 0 0 Δ 0 0	ZZZZ.99	Δ Δ Δ Δ . 0 0
0 0 9 Δ 2 3	$***.99	$ * * 9 . 2 3
0 0 0 8 Δ 2 4	$$$$9.99	Δ Δ Δ $ 8 . 2 4
0 0 5 Δ 2 $\bar{6}$	---9.99	Δ Δ - 5 . 2 6
3 2 Δ 6 5	$$$.99	$ 3 2 . 6 5

Table 8: Examples of PICTURE Editing

DATA to be Edited	PICTURE of Report Item	Edited Item
0 1 2 3 4 5	ZZZ,999.99	Δ 1 2 , 3 4 5 . 0 0
0 0 1 2 3 4	Z99,999.99	Δ 0 0 , 0 1 2 . 3 4
0 0 0 1 2 3	$ZZZ,ZZ9.99	$ Δ Δ Δ Δ Δ Δ 1 . 2 3
0 0 0 0 1 2	$ZZZ,ZZZ.99	$ Δ Δ Δ Δ . 1 2
0 0 1 2 3 4	$***,**9.99	$ * * 1 , 2 3 4 . 0 0
1 2 3 4 5 6	$***,***.99	$ 1 2 3 , 4 5 6 . 0 0
1 2 3 4 5 6	$***,***.99	$ * * * * * * 1 . 2 3
0 0 0 0 1 $\overset{+}{2}$	+999,999	+ 0 0 0 , 0 1 2
0 0 0 0 1 $\overset{-}{2}$	-ZZZ,ZZZ	Δ Δ Δ Δ Δ 1 2
1 2 3 4 5 $\overset{-}{6}$	$ZZZ,ZZ9.99CR	$ 1 2 3 , 4 5 6 . 0
0 0 0 1 2 $\overset{+}{3}$	$ZZZ,ZZ9.99DB	$ Δ Δ Δ Δ Δ Δ 1 . 2 3
0 0 1 2 3 4	$(4),$$9.99	Δ Δ Δ Δ $ 1 2 3 . 4 0
0 0 0 0 0 0	$(4),$$$.99	Δ Δ Δ Δ Δ Δ Δ $. 0 0
0 0 0 0 1 $\overset{-}{2}$	----,---.99	Δ Δ Δ Δ Δ Δ Δ - . 1 2
0 0 0 0 1 $\overset{+}{2}$	----,---.99	Δ Δ Δ Δ Δ Δ Δ Δ . 1 2
0 0 0 0 0 1	$$$$,$ZZ.99	Illegal PICTURE

a floating minus sign. The minus sign is specified one more time than the number of leading zeros to be suppressed. It functions in the same way as the floating plus sign, except that a blank is placed directly preceding the first nonsuppressed character if the edited data is positive or unsigned.

5. Summary:

a. Only one of the characters of the set Z * $ + and - can be used within a single PICTURE as a replacement character, although it may be specified more than once.

b. If one of the replacement characters Z or * is used with one of the insertion characters $ + or -, the plus or minus signs may be specified as either the leftmost or rightmost character in the PICTURE.

c. A plus sign and a minus sign may not be included in the same PICTURE.

d. A leftmost plus sign and a dollar sign may not be included in the same PICTURE.

e. A leftmost minus sign and a dollar sign may not be included in the same PICTURE.

f. The character 9 may not be specified to the left of a replacement character.

g. Symbols that may appear only once are V S . CR and DB.

h. The decimal point may not be the rightmost character in a PICTURE.

6.18.6.12 USAGE Clause: The format of this clause is:

$$\underline{\text{USAGE}}\ \text{IS}\quad \left\{ \begin{array}{l} \underline{\text{DISPLAY}} \\ \left[\begin{array}{l} \underline{\text{COMPUTATIONAL}} \\ \underline{\text{COMP}} \end{array} \right] \\ \underline{\text{INDEX}} \end{array} \right\}$$

The USAGE clause specifies the form in which data is represented in the computer. It can be written at any level. If the USAGE clause is written at a group level, it applies to each elementary item in the group in addition, the USAGE clause of an elementary item cannot contradict the USAGE clause of a group to which the item belongs.

This clause specifies the manner in which a data item is represented in the storage of the computer. It does not affect the use of the data item, although the specifications for some statements in the PROCEDURE DIVISION may restrict the USAGE clause of the referent operands.

DISPLAY denotes that the item is carried in the ASCII format. DISPLAY mode is assumed when a USAGE clause is not written. One character is stored in each byte of the item; if the item is numeric, the leftmost byte can contain an operational sign in addition to a digit.

COMPUTATIONAL defines a packed decimal data item whose length is specified by the accompanying PICTURE clause.

INDEX defines an item that is called an index data item and will contain a value that corresponds to an occurrance number of a table element. Index data items must be elementary data items. Since USAGE IS INDEX totally defines the internal representation of the data, a PICTURE clause is not used with an index data item. The VALUE IS clause may not be used with a USAGE IS index data item.

6.18.6.13 BLANK WHEN ZERO Clause: The format of this clause is:

BLANK WHEN ZERO

The BLANK WHEN ZERO clause may be supplied only in conjunction with a numeric edited item. It specifies that when the source item has a value of zero, the edited data item is to contain all spaces.

6.18.6.14 JUSTIFIED Clause: The format of this clause is:

$$\left\{\begin{matrix}\underline{\text{JUSTIFIED}} \\ \underline{\text{JUST}}\end{matrix}\right\} \quad \underline{\text{RIGHT}}$$

This clause is applicable only to alphabetic or alphanumeric items. Normally, when data is moved into an alphabetic or alphanumeric field, the source data is aligned at the leftmost character position of the receiving data item and moved with space fill or truncation on the right.

When the receiving data item is described with the JUSTIFIED clause and the sending data item is larger than the receiving data item, the leftmost characters are truncated. When the receiving data item is described with the JUSTIFIED clause and is larger than the sending data item, the data is aligned at the rightmost character position in the data item with other characters space-filled.

6.18.6.15 VALUE Clause. The format of this clause is:

Value IS literal

The VALUE clause defines the value of constants, or the initial value of working-storage items. This clause must not conflict with other clauses in the data description of the item or in the data description within the hierarchy of the form. The following rules apply:

1. General

a. If the category of the item is numeric, the literal is aligned according to the alignment rules except that the literal must not have a value requiring truncation of digits.

b. If the category of the item is alphabetic or alphanumeric the literal in the VALUE clause must be a nonnumeric literal. The literal is aligned according to the alignment rules except that the number of characters in the literal must not exceed the size of the item.

c. The numeric literal in a VALUE clause of an item must have a value within the range of values indicated by the USAGE or PICTURE clause.

d. The function of any editing clauses or editing characters in a PICTURE clause is ignored in determining the initial appearance of the item described. However, editing characters are included in determining the size of the item.

2. Data Description Entries

a. Rules governing the use of the VALUE clause differ with the respective section of the DATA DIVISION:

(1) In the FILE SECTION, the VALUE clause is not allowed.

(2) In the WORKING-STORAGE the VALUE clause may be used to specify the initial value of any data item. It causes the item to assume the specified value at the start of the object program. If the VALUE clause is not used in an item description, the initial value may be unpredictable.

b. The VALUE clause must not be stated in a Record Description entry containing an OCCURS clause or in an entry subordinate to an entry containing an OCCURS clause.

c. The VALUE clause must not be stated in a Record Description entry containing a REDEFINES clause or in an entry subordinate to an entry containing a REDEFINES clause. This rule does not apply to condition-name entries.

d. The VALUE clause may not be used in an entry at the group level.

e. The VALUE clause may not be used with a USAGE IS Index data item.

<u>6.18.6.16 OCCURS Clause:</u> The format of this clause is:

<u>OCCURS</u> integer-1 TIMES

<u>INDEXED</u> BY index-name-1 [,index-name-2] . . .]

The OCCURS clause eliminates the need for separate entries of repeated data and supplies information required for the application of subscripts.

The OCCURS clause is used in defining tables and other homogeneous sets of repeated data; when it is used, the data-name that is the subject of this entry must either be subscripted whenever it is referenced in a statement. Furthermore, if the subject of this entr is the name of a group item, all data-names belonging to the group must be subscripted whenever they are used as operands.

The data description clauses associated with an item whose description includes an OCCURS clause apply to each repetition of the item described. Also the VALUE clause must not be stated in a data description entry that contains an OCCURS clause or in an entry that is subordinate to an entry containing an OCCURS clause.

An INDEXED BY clause is required if the subject of this entry, or an item within it if it is a group item, is to be referenced by indexing. The index-name identified by this clause is not defined elsewhere; the compiler allocates storage for it unassociated with any data hierarchy.

6.19 PROCEDURE DIVISION

6.19.1 General Description: The PROCEDURE DIVISION of a COBOL source program specifies the procedures--the precise sequence of processing operations--needed to solve a given problem. These operations (computations, logical decisions, input/output, etc.) are expressed in meaningful statements, similar to English.

6.19.2 Procedure Division Elements:

6.19.2.1 Statements: A statement consists of a COBOL verb followed by appropriate operands (data-names or literals) and reserved words. The three types of statements are:

1. Compiler directing

2. Imperative

3. Conditional

6.19.2.1.1 Compiler Directing Statement: A Compiler Directing statement directs the compiler to take certain actions at compilation time. Compiler Directing statements are: COPY. This statement is not in NETWORK COBOL.

6.19.2.1.2 Imperative Statement: An imperative statement specifies an action to be taken unconditionally by the object program. An imperative statement may consist of a series of imperative statements.

6.19.2.1.3 Conditional Statement: A conditional statement describes a condition that is tested to determine which of alternate paths of programmed processing flow is to be taken. Conditional statements are:

1. READ and RETURN statements that have the AT END or INVALID KEY options.

2. WRITE statements with the INVALID KEY option.

3. Arithmetic statements with the SIZE ERROR option.

4. IF statements.

6.19.2.2 Sentences: A sentence is a single statement or series of statements terminated by a period. A single semicolon may be used as a separator between statements within a sentence.

6.19.2.3 Paragraphs: A paragraph consists of one or more sentences identified by a beginning paragraph-name.

6.19.2.4 Sections: A section comprises one or more successive paragraphs, and must begin with a section header. A section header consists of a section-name followed by the word SECTION and a period.

6.19.2.5 Paragraph and Section Naming: Every paragraph or section has a programmer-supplied name that is given in the header entry. This name is used for reference (as, for example, when specifying a GO TO paragraph-name or a GO TO section-name.)

6.19.3 Procedure Division Structure: The formats of the PROCEDURE DIVISION are:

Format 1:

PROCEDURE DIVISION

{section-name SECTION.}

{paragraph-name. {sentence.} . . .} . . .} . . .

Format 2:

PROCEDURE DIVISION

{paragraph-name. sentence.} . . .} . . .

Execution of the program begins at the first statement of the first section.

6.19.4 Conditional Statements: A conditional statement describes a condition that is tested to determine selection of alternate paths of programmed processing flow. The programmer can accomplish this branching using the following types of statements:

1. The GO TO . . . DEPENDING ON . . ., which branches to one of

several procedure-names.

2. Statements with exception branches: AT END, INVALID KEY, and ON SIZE ERROR.

3. The IF, and PERFORM, in which the condition is explicitly stated.

6.19.4.1 Relations: Relational-operators in the COBOL language are:

IS [NOT] {GREATER THAN | $\geq$}

IS [NOT] {LESS THAN | $\leq$}

IS [NOT] {EQUAL TO | =}

EQUALS

Underlined words in the above list must be present when the relational-operator is used. Words not underlined may be omitted if the programmer desires, with no effect on the meaning of the relational-operator.

Relational-operators are combined with identifiers or literals to create relation conditions. The general format is:

{identifier-1 | literal-1 | arithmetic-expression} {relational-operator} {identifier-2 | literal-2 | arithmetic-expression}

6.19.4.2 Logical Operators (AND, OR and NOT): The three logical operators are AND, OR, and NOT. AND and OR are used to create a "compound condition" when two or more tests are specified in the same expression. NOT is used to specify the negation of a condition. NOTE: Compound conditions must be enclosed in parentheses if they are to work correctly. The MICROSOFT COBOL will flag this as an error but generate the correct code. Consider the following example:

IF (CODE IS ZERO AND AGE NOT GREATER THAN 21) ADD A TO B.

Notice how AND and NOT are used to augment the two basic tests. Because the tests are connected by AND, they both must be true for A to be added to B.

Consider the following:

IF (CODE IS NOT ZERO OR AGE GREATER THAN 21) ADD C TO D.

This time the logical operator OR specifies that C is to be added to D if either or both conditions are fulfilled.

NOT can be used in two ways with a simple relational condition: in the relational-operator as in AGE NOT GREATER THAN 21, or preceding the entire condition as in NOT AGE GREATER THAN 21. AGE NOT GREATER THAN 21 and NOT AGE GREATER THAN 21 are exactly equivalent in meaning. If NOT precedes a simple relational condition that contains NOT in the relational-operator, a double negative results and causes an error.

6.19.4.3 Other Condition Tests:

6.19.4.3.1 Sign Test: The format of this test is:

IF {data-name | arithmetic-expression} IS [NOT] {POSITIVE | ZERO | NEGATIVE}

The sign test is also effectively a special case of relation testing equivalent to testing whether an expression is GREATER THAN, LESS THAN, or EQUAL TO ZERO. The data-name must be a numeric value that, if unsigned and not equal to zero is assumed to be positive. The value zero is considered neither positive nor negative. The statement GROSS IS NEGATIVE is equivalent to GROSS IS LESS THAN 0; GROSS IS POSITIVE is equivalent to GROSS IS GREATER THAN 0. Any condition that can be expressed as a sign condition can be expressed as a simple relational condition; the sign condition is merely a convenient way of expressing certain situations.

6.19.4.3.2 Class Test: The format of this test is:

IF data-name IS [NOT] {NUMERIC | ALPHABETIC}

The data-name must be defined in the DATA DIVISION as USAGE DISPLAY. Table 9 lists cases where the class test is valid and meaning of the results.

6.19.4.3.3 Comparison of Numeric Items: For numeric items a relation test determines that the value of one of several items is less than, equal to, or greater than the others, regardless of the length. Numeric items are compared algebraically after alignment of decimal points. Zero is considered a unique value regardless of

Table 9. Valid Class Tests

PICTURE Must Contain	PICTURE May Contain	Allowable Characters	Valid Tests	Meaning
A	B	Alphabetic (A-Z and space)	[NOT] ALPHABETIC	(Not) only characters A-Z and space appear
A 9 X	X B 0 A9 B 0	Alphanumeric (any character)	[NOT] ALPHABETIC [NOT] NUMERIC	(Not) only characters A-Z and space appear (Not) only characters 0-9 appear
S 9	0 V P	Zoned decimal with operational sign	[NOT] NUMERIC	(Not) only characters 0-9 appear in all position, which can contain zone bit.
9	0 V P	Zoned decimal without sign	[NOT] NUMERIC	(Not) only characters 0-9 appear.

length, sign, or implied decimal-point location of an item.

6.19.3.4 Comparison of Non-Numeric Items: For non-numeric items a comparison determines that one of the items is less than, equal to or greater than the other with respect to the binary collating sequence of characters in the ASCII character set. If the non-numeric items are of equal length, the comparison proceeds by comparing characters in corresponding character positions starting from the high-order position and continuing until either a pair of unequal characters or the low-order position of the item is compared. If the non-numeric items are of unequal length, comparison proceeds as described for items of equal length. If this process exhausts the characters of the shorter item, the shorter item is less than the longer unless the remainder of the longer item consists solely of spaces, in which case the items are equal.

Table 10 indicates characteristics of the compared items and the type of comparison made.

6.19.4.4 Conditional Statements with Exception Branches: The format of these statements is:

```
{AT END          }
{INVALID KEY     }     {Imperative-statements}  . . .
{ON SIZE ERROR   }
```

The READ, RETURN, WRITE, REWRITE, DELETE, ADD, SUBTRACT, MULTIPLY, and DIVIDE verbs specify the exception branch as either an optional or a required part of the statement. When the exception branch is present, the verb in whose format it is written is considered to be a conditional statement. Normally, control bypasses the exception branch to the

Table 10 Permissible Comparisons

Item Characteristics		GR	X	ND
Group Item	GR	A	A	A
Alphabetic, Alphanumeric, and Edited	X	A	A	A
Numeric Display	ND	A	A	9

A. Alphanumeric or byte comparison, byte-by-byte from left to right.

9. Numeric comparison.

first statement in the next sentence or the first statement beyond the next ELSE (within an IF statement), but when the exception condition is met, control is given to the imperative-statement following the AT END, INVALID KEY, or SIZE ERROR. None of the statements up to the next period or ELSE (within an IF statement) may be a conditional statement: thus "nesting" of exception branches is not allowed.

6.19.4.5 Nested Conditional Statements: The IF statement may have conditional statements in either of the branches taken because of the outcome of the condition test. Furthermore, the conditional statement can be another IF, thus it is possible to "nest" IFs (in other words, IFs may be contained within IFs). Refer to the "IF Statement" discussion (Section 6.19.8.10).

6.19.5 Input/Output Statements:

6.19.5.1 OPEN Statement: The general format of this statement is:

```
OPEN   [INPUT   [file-name] . . .]
       [OUTPUT  [file-name] . . .]
       [EXTEND  [file-name] . . .]
       [I-O     [file-name] . . .]
```

The OPEN statement initiates processing of the files named in the statement.

One of the INPUT, OUTPUT, EXTEND or I-O options must be specified. The I-O option pertains only to files on direct access media used when ACCESS IS RANDOM is specified.

The EXTEND option means that the file is to be opened for output and that new records are to be added after the last record currently in the file.

An OPEN statement must be executed prior to any other input/output statement. A second OPEN statement for a given file cannot be executed prior to the execution of a CLOSE statement for that file. The OPEN statement itself does not obtain or dispatch data; a READ or WRITE statement must execute to obtain or release, respectively, the first data record.

6.19.5.2 START Statement: The START statement provides a means for logical positioning within an indexed file for subsequent sequential retrieval of records.

Format:

```
                                 ⎧ EQUAL TO           ⎫
                                 ⎪ =                  ⎪
START file-name [KEY IS          ⎨ GREATER THAN       ⎬  data-name]
                                 ⎪ >                  ⎪
                                 ⎩ NOT {LESS} THAN    ⎭
                                       { <  }

          [INVALID KEY imperative-statement]
```

When the START statement is executed, the associated file must be open in INPUT or I-O mode.

File-name must name an indexed file with sequential or dynamic access. File-name must be defined in an FD entry in the Data Division.

When the KEY option is not specified, the EQUAL TO relational operator is implied. When the START statement is executed, the EQUAL TO comparison is made between the current value in the RECORD KEY and the corresponding key field in the file's records. The Current Record Pointer is positioned to the logical record in the file whose key field satisfies the comparison.

When the KEY option is specified, data-name may be either:

- The RECORD KEY for this file, or

- Any alphanumeric data item subordinate to the RECORD

KEY whose leftmost character position corresponds to the leftmost character position of the RECORD KEY (that is, a generic key).

When the START statement is executed, the comparison specified in the KEY relational operator is made between data-name and the key field in the file's records. The Current Record Pointer is positioned to the first logical record in the file whose key field satisfies the comparison.

If the comparison is not satisfied by any record in the file, an INVALID KEY condition exists, and the position of the Current Record Pointer is undefined.

6.19.5.3 READ Statement: For sequential access, the READ statement makes available the next logical record from file. For random access, the READ statement makes available a specified record from a file.

The formats of this statement are:

Format 1:

READ file-name [NEXT] RECORD [INTO identifier]

[AT END imperative-statement]

Format 2:

READ file-name RECORD [INTO identifier]; INVALID KEY imperative-statement

Functions of the READ verb are:

1. Sequential file processing (Format 1) makes available the next logical record from an input file and allows execution of a specified series of imperative-statements when the end-of-file is detected.

2. Random file processing (Format 2) makes available a specific record from an indexed file and allows execution of a specified series of imperative-statements if the contents of the associated RECORD KEY data item are found to be invalid.

When the READ statement is executed, the associated file must be open in INPUT or I-O mode.

File-name must be defined in an FD entry in the Data Division.

Format 1: When ACCESS MODE SEQUENTIAL is specified or assumed for a file, this format must be used. For such files the statement makes available the next logical record from the file. For indexed files, the NEXT option need not be specified; for sequential files, the NEXT option must not be specified.

When ACCESS MODE DYNAMIC is specified for indexed files, the NEXT option must be specified for sequential retrieval. For such files, the READ NEXT statement makes available the next logical record from the file.

Before a Format 1 READ statement is executed, the Current Record Pointer must be positioned by the successful prior execution of an OPEN, START, or READ statement. When the Format 1 READ statement is executed the record indicated by the Current Record Pointer is made available. For sequential files, the next record is the succeeding record in logical sequence. For a sequentially accessed indexed file, the next record is that one having the next higher RECORD KEY in collating sequence.

Format 2: This format must be used for indexed files in random access mode, and for random record retrieval in the dynamic access mode.

Execution of a Format 2 READ statement causes the value in the RECORD KEY to be compared with the values contained in the corresponding key field in the file's records until a record having an equal value is found. The Current Record Pointer is positioned to this record, which is then made available.

If no record can be so identified, an INVALID KEY condition exists, and execution of the READ statement is unsuccessful.

Immediately following execution of a READ statement, the next logical record in the file is accessible in the logical record area associated with the file as defined by the Record Description entry. When multiple record descriptions follow a File Description (FD) entry, it is the responsibility of the programmer to recognize which record is present in the area at any given time. The record is available in the logical record area until another READ statement or a CLOSE statement for that file is executed.

The INTO option is equivalent to a READ statement followed by a MOVE, and results in the record obtained by execution of the READ becoming available in both the record area for the file and in the location indicated by the identifier. The record is moved from the record area into the identifier in accordance with the rules for the MOVE statement.

In the case where the file contains records of varying lengths, the size of the longest record is assumed for the input record for the purpose of executing the MOVE.

The AT END clause is required for files that are accessed sequentially. The statements introduced by this clause are executed when end-of-file is encountered.

For files with SEQUENTIAL organization, when the AT END condition has been recognized, a READ statement for this file must not be executed until a successful CLOSE statement followed by a successful OPEN statement has been executed for this file.

For files with INDEXED organization, when the AT END condition is recognized, a Format 1 READ statement for this file must not be executed until one of the following has been successfully executed:

- A CLOSE statement followed by an OPEN statement
- A Format 2 READ statement (dynamic access)
- A START statement

The INVALID KEY clause must be written for files for which ACCESS IS RANDOM is specified. The imperative-statements are executed if a record corresponding to the contents of the RECORD KEY cannot be located in the file.

The contents of the RECORD KEY data item must be appropriately established prior to execution of the READ statement itself.

6.19.5.4 WRITE Statement: The formats of this statement are:

Format 1:

```
WRITE record-name [FROM identifier-1] [{BEFORE} ADVANCING
                                       {AFTER }
        {identifier-2 LINES}]
        {integer-1 LINES   }
        {PAGE              }
```

Format 2:

```
WRITE record-name [FROM identifier-1]; INVALID KEY
        imperative statement
```

The WRITE statement releases a logical record to an output file. For random access files the statement also allows execution of a specified series of imperative-statements if the contents of the associated RECORD KEY data item are found invalid.

An OPEN OUTPUT, OPEN EXTEND, or OPEN INPUT-OUTPUT must be executed before a WRITE statement can be executed for a file. Once the WRITE is executed there is no guarantee that the logical record released thereby still exists in the logical record area for the file.

A WRITE statement bearing the FROM option is equivalent to a MOVE identifier-1 TO record-name statement followed by WRITE record-name. Moving takes place in accordance with rules for the MOVE statement.

Format 1 relates to files opened for sequential access. The ADVANCING option applies to files containing output destined to be printed. Integer-1 should be an unsigned integer, and identifier-2, similarly, should contain a non-negative integer. The line is printed BEFORE or AFTER the specified number of lines is spaced.

Format 2 is used for mass storage files. Statements following the INVALID KEY clause are executed when:

1. No space exists on the file media to accommodate the record.

2. The file is open for OUTPUT or I-O and a record corresponding to the contents of the RECORD KEY already exists in the file.

6.19.5.5 REWRITE Statement: The format of this statement is:

REWRITE record-name [FROM identifier-1]; INVALID KEY
    imperative-statement.

The REWRITE statement rewrites a previously read logical record to the output file. The statement also allows execution of a specified series of imperative-statements if the contents of the associated RECORD KEY data item are found invalid.

An OPEN I-O must be executed before a REWRITE statement can be executed for a file. Once the REWRITE is executed there is no guarantee that the logical record rewritten still exists in the logical record area for the file.

The statements following the INVALID KEY clause are executed when the record corresponding to the contents of the RECORD KEY clause was not previously read.

6.19.5.6 DELETE Statement: The format of this statement is:

DELETE file-name; INVALID KEY imperative-statement

The DELETE statement deletes a logical record from the output file. The statement also allows execution of a specified series of imperative-statements if the contents of the associated RECORD KEY data item are found invalid.

An OPEN I-O must be executed before a DELETE statement can be executed for a file.

The statements following the INVALID KEY clause are executed when the record corresponding to the contents of the RECORD KEY clause is not found in the file.

6.19.5.7 CLOSE Statement: The format of this statement is:

CLOSE [file-name] [WITH DELETE] . . .

The CLOSE statement terminates the processing of files. Execution of a CLOSE statement causes the standard closing procedures to be carried out on the file named. An OPEN statement must be executed before a CLOSE can be honored for a file; once closed, a file may not be referenced again until another OPEN statement is executed for that file.

If the DELETE option is specified, all records in the file will be deleted.

6.19.5.8 ACCEPT Statement: The format of this statement is:

ACCEPT identifier-1 [, identifier-2] . . .

The ACCEPT statement specifies acceptance of data from the CRT. It is normally used to read unprotected CRT fields.

The identifier must be an unedited DISPLAY data item or a group item. Refer to the operations manual for additional information on reading unprotected fields from the CRT.

6.19.5.9 DISPLAY Statement: The format of this statement is:

```
DISPLAY  { identifier-1 }    [ { identifier-2 } ]  . . .
         { literal-1    }    [ { literal-2    } ]
```

The DISPLAY statement enables data to be written to the CRT. When a DISPLAY statement contains more than one operand, the characters comprising the items named and any literals specified in the statement are displayed consecutively, with no spaces between char acters unless specified.

Any remaining positions on a line at the end of the data transfer are left unchanged. Any number of literals or data names may be specified. The data-name may be that of a group or an elementary item and may also be subscripted. A literal in a DISPLAY statement may be numeric or non-numeric and may be a hexadecimal constant to specify CRT or field attributes.

Example:

DISPLAY PRINT-LINE.

6.19.6 ARITHMETIC Statements: The basic arithmetic operations are specified by the four verbs ADD, SUBTRACT, MULTIPLY, and DIVIDE.

6.19.6.1 Rules for Arithmetic Verbs: The following general rules apply to all arithmetic verbs:

AD-A083 046 GEORGIA INST OF TECH ATLANTA SCHOOL OF ELECTRICAL EN--ETC F/G 9/2
THE FEASIBILITY OF IMPLEMENTING MULTICOMMAND SOFTWARE FUNCTIONS--ETC(U)
OCT 79 T P BARNWELL, J L HAMMOND, J H SCHLAG DAAG29-78-G-0139

UNCLASSIFIED ARO-15900.1-A-EL NL

3 OF 4
ADA
083046

1. All literals specified in arithmetic statements must be numeric.

An identifier used in an arithmetic statement must be an elementary item and must be numeric.

2. The maximum size of an operand is 15 decimal digits. If the entry for an operand in the DATA DIVISION specifies a size greater than 15 digits or if a literal contains more than 15 digits, an error is indicated at compilation time.

3. The items in an arithmetic statement may be mixed sizes as long as they are all numeric. Any necessary decimal-point alignment is supplied automatically throughout computations.

4. No item used in computations may contain editing symbols. If such an item is used, a compilation-time diagnostic results. Operational signs and assumed decimal points are not editing symbols. An item used to receive results may contain editing symbols if it is not used in subsequent computations as an operant. When an item used to receive results contains editing symbols, the result is edited according to editing specifications before it is moved to the item.

ROUNDED, GIVING and SIZE ERROR options apply to all arithmetic statements.

6.19.6.2 GIVING Option: If the GIVING option is written, the value of the identifier that follows the word GIVING is made equal to the calculated result of the arithmetic operation.

If the GIVING option is not written, each operand following the words TO, FROM, BY, and INTO in the ADD, SUBTRACT, MULTIPLY, and

DIVIDE statements, respectively, must be an identifier (not a literal). Each identifier is used in the computation, and also receives the result.

6.19.6.3 ROUNDED Option: If the ROUNDED option is not specified, truncation occurs when the number of places calculated (after decimal -point alignment) for the result is greater than the number of places in the data item that is to be set equal to the calculated result. When the ROUNDED option is specified, the least significant digit of the resultant data-name increases in value by 1 whenever the most significant digit of the excess is greater than or equal to 5.

Rounding of a computed negative result is performed by rounding the absolute value of the computed result and then making the final result negative.

Table 11 illustrates the relationship between a calculated result and the value stored in an item that is to receive the calculated result.

6.19.6.4 SIZE ERROR Option: An arithmetic statement, if written with a SIZE ERROR option, is not an imperative-statement. Rather, it is a conditional statement and is prohibited in contexts where only imperative-statements are allowed.

Whenever the number of integer places in the calculated result exceeds the number of integer places specified for the resultant item, a size error condition arises. If the SIZE ERROR option is specified and a size error condition arises, the value of the resultant item is not altered and the series of imperative-statements specified for the

Table 11. Rounding or Truncation of Calculations

CALCULATED RESULT	PICTURE	VALUE AFTER ROUNDING	VALUE AFTER TRUNCATING
-12.36	S99V9	-12.4	-12.3
8.432	9V9	8.4	8.4
35.6	99V9	35.6	35.6
65.6	99V	66	65
0.0055	V999	0.006	0.005

condition is executed.

If the SIZE ERROR option is not specified and a size error condition arises, no assumption should be made about the correctness of the final result even though the program flow is not interrupted.

6.19.6.5 ADD Statement: The formats of this statement are:

Format 1:

ADD {identifier-1 | literal-1} [, identifier-2 | , literal-2] . . . , identifier-n [ROUNDED] [ON SIZE ERROR imperative-statement]

Format 2:

ADD {identifier-1 | literal-1} [, identifier-2 | , literal-2] . . . TO identifier-m [ROUNDED] [ON SIZE ERROR imperative-statement]

Format 3:

```
ADD  {identifier-1}  ,  {identifier-2}  [, identifier-3] . . .
     {literal-1   }     {literal-2   }  [, literal-3   ]

GIVING identifier-m [ROUNDED] [ON SIZE ERROR imperative-
statement]
```

The ADD statement sums the values of two or more numeric items and/or literals and sets one or several items equal to the resultant value. Operands used in an ADD statement must conform to "Rules for Arithmetic Verbs" (Section 6.19.6.1) in addition to specific rules applying to this individual statement. Use of the SIZE ERROR and ROUNDED options is also discussed in the referenced paragraph.

When Format 1 is used the values of all the operands including identifier-n are added together and the result is stored as the new value of identifier-n, the resultant-identifier.

Example: Given the statement ADD A, B, C, the values of A, B, and C before and after execution are:

	A	B	C
Before	5	6	8
After	5	6	19

Note that the value of A and B do not change as the result of the addition.

Format 2 adds the values of the operands (identifier-1 or literal-1 and identifier-2 or literal-2) preceding the reserved word TO, and this intermediate result is added to the data items specified by identifier-m, identifier-n, etc.

Example: Given the statement ADD W, X, Y to Z, the values of W, X, Y and Z before and after execution are:

	W	X	Y	Z
Before	2	7	8	12
After	2	7	8	29

Note that the value of all operands participates in the addition.

Format 3 adds the values of the operands (identifier-1 or literal-1 and identifier-2 or literal-2, etc.) preceding the reserved word GIVING, and this intermediate result is placed in identifier-m, identifier-n, etc.

Example: Given the statement ADD A, B, C, GIVING D, the values of A, B, C, and D before and after execution are:

	A	B	C	D
Before	1	2	3	5
After	1	2	3	6

Note that the intermediate result replaces the value of D and is not added to D.

<u>6.19.6.6 SUBTRACT Statement</u>: The formats of this statement are:

<u>Format 1</u>:

<u>SUBTRACT</u> {identifier-1 / literal-1} [, identifier-2 / , literal-2] . . .

<u>FROM</u> identifier-m [<u>ROUNDED</u>] [ON <u>SIZE</u> <u>ERROR</u> imperative-statement]

Format 2:

```
SUBTRACT {identifier-1}   [, identifier-2] . . . FROM
         {literal-1   }   [, literal-2   ]

         {identifier-m}  GIVING identifier-n [ROUNDED]
         {literal-m   }

    [ON SIZE ERROR imperative-statement]
```

The SUBTRACT statement subtracts the value of a numeric item from another item and stores the result in a third item.

Format 1 subtracts the operands preceding the word FROM from identifier-m placing the result in identifier-m.

Format 2 subtracts the operands preceding the word FROM from identifier-m (literal-m) without changing the contents of identifier-m, placing the result in the item following GIVING.

Example: Given the statement SUBTRACT A FROM B GIVING C the values of the operands before and after execution are:

	A	B	C
Before	10	80	90
After	10	80	70

6.19.6.7 MULTIPLY Statement: The formats of this statement are:

Format 1:

```
MULTIPLY {identifier-1}    BY identifier-2 [ROUNDED]
         {literal-1   }

    [ON SIZE ERROR imperative-statement]
```

Format 2:

MULTIPLY {identifier-1 | literal-1} BY {identifier-2 | literal-2} GIVING

identifier-3 [ROUNDED]

The MULTIPLY statement can be used to multiply two items with the value of a third item being set to the product. Operands used in a MULTIPLY statement must conform to "Rules for Arithmetic Verbs", (Section 6.19.6.1), in which the SIZE ERROR and ROUNDED options are also discussed.

Format 1 allows the multiplicand (identifier-1 or literal-1) to be multiplied by the multiplier (identifier-2) and the value of identifier-2 to be set to the product. A literal cannot be used in place of identifier-2.

Example: Given the statement MULTIPLY A BY B the values of the operands before and after execution are:

	A	B
Before	10	20
After	10	200

Note that the values of operand B change to reflect the multiplication.

Format 2 allows the multiplicand (identifier-1 or literal-1) to be multiplied by the multiplier (identifier-2 or literal-2).

Example: Given the statement MULTIPLY A BY B GIVING C the values of the operands before and after execution are:

	A	B	C
Before	5	10	20
After	5	10	50

Note that the values of operands A and B remain the same, while the value of operand C changes.

6.19.6.8 DIVIDE Statement: The formats of this statement are:

Format 1:

```
DIVIDE {identifier-1}  INTO  identifier-2 [ROUNDED]
       {literal-1   }
  [ON SIZE ERROR  imperative-statement]
```

Format 2:

```
DIVIDE {identifier-1}  INTO  {identifier-2}  GIVING
       {literal-1   }        {literal-2   }
  identifier-3 [ROUNDED] [ON SIZE ERROR imperative-statement]
```

Format 3:

```
DIVIDE {identifier-1}  BY  {identifier-2}  GIVING
       {literal-1   }      {literal-2   }
  identifier-3 [ROUNDED] [ON SIZE ERROR imperative-statement]
```

The DIVIDE statement divides the value of one numeric item into the value of one or more numeric items and sets the value of one or more items to the quotient. Operands used in a DIVIDE statement must conform to "Rules for Arithmetic Verbs", Section 6.19.6.1, in addition to specific rules applying only to this individual statement. Use of the SIZE ERROR and ROUNDED options is also discussed in the reference paragraph.

Format 1 allows one division, with the quotients stored as the value of the item following INTO. The dividend (identifier-2) divided by the divisor (identifier-1 or literal-1) and the value of the dividend set to the value of the associated quotient. literals cannot be used in place of identifiers-2. The size error condition results when the divisor is zero or the quotient contains more integer positions than are available.

Example: Given the statement DIVIDE A INTO B the values of the operands before and after execution are:

	A	B
Before	5	10
After	5	2

Format 2 allows the single quotient resulting from a division to be stored in a third item. If Format 2 is used, the dividend (identifier-2 or literal-2) is divided by the divisor (identifier-1 or literal-1), and the value of the resultant quotient becomes the new value of identifiers-3.

Example: Given the statement DIVIDE A INTO B GIVING C the values of the operands before and after execution are:

	A	B	C
Before	5	10	15
After	5	10	2

6.19.7 Data Manipulation Statements:

6.19.7.1 MOVE Statement: The format of this statement is:

```
MOVE  {identifier-1}  TO   identifier-2   (, identifier-3)
      {literal-1   }
. . . (ON SIZE ERROR imperative-statement)
```

The MOVE statement moves data from one area of main storage to another. It edits the data (inserts, deletes, or replaces characters) if the PICTURE of the receiving item so requires.

[illegible] literal-1 may be a numeric literal, [illegible] figurative-constant. [illegible] information may be [illegible] identifier-2, [illegible] the original data in identifier-1 [illegible] items. Identifier-1 or literal-1 is the [illegible], identifier-3, etc., are the receiving [illegible]. [illegible] source and receiving items can be elementary or [illegible] items. [illegible] the MOVE statement, a literal is [illegible] elementary item.) The manner in which the MOVE is [illegible] on the type of source and receiving items [illegible] also of their classes.

The imperative-statement [illegible] clause will be executed whenever significant characters (non-blank or non-zero) are [illegible] facilitate [illegible].

The types of MOVE statements are discussed in the following paragraphs.

6.19.7.1.1 Alphanumeric Moves: Source data is stored left-justified in the receiving area. If the receiving area is not completely filled by data, remaining positions are filled with spaces. If the receiving item is alphabetic, it is treated as alphanumeric.

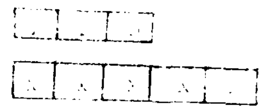
Examples:

Source Data	PICTURE of Receiving Item	Receiving Item
A B C D	A(4) or X(4)	A B C D
A B C D	A(5) or X(5)	A B C D Δ
A B C D 1 2 3	X(8)	A B C D 1 2 3 Δ
1 2 3	X(8)	1 2 3 Δ Δ Δ Δ Δ
A B C D	A(3) or X(3)	A B C

If the receiving item is alphanumeric, the literal may be any literal or figurative-constant. If the figurative-constant takes the form of ALL any-literal, the literal must be enclosed in quotation marks and is considered an alphanumeric item. The size of an ALL any-literal item is determined by the size of the receiving item, with characters repeated from left to right.

Examples:

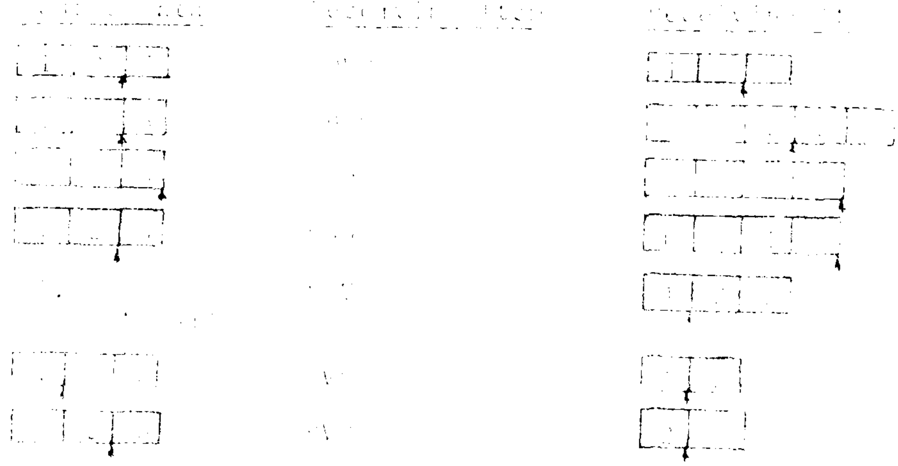
Source Data	PICTURE of Receiving Item	Receiving Item
'ABCD'	X(4)	A B C D
'123'	X(3)	1 2



[illegible] Numeric Moves: When the source data [illegible] [illegible] the receiving area, it is aligned according to its decimal point and the decimal point in the receiving area. [illegible] source or receiving item, [illegible] [illegible]

[illegible] position could [illegible]

[illegible]



[illegible]

Editing occurs after decimal point alignment. Editing symbols in the receiving item (currency signs, commas, etc.), make this item alphanumeric; if it is subsequently referenced as a source item in a MOVE statement, it is moved in accordance with the rules for alphanumeric items.

Examples:

Source Data	PICTURE of Receiving Item	Receiving Item
1 2 3 4 5	$**9.99	$ 1 2 3 . 4 5
1 2 3 4 5	999.9	1 2 3 . 4
0 0 0 1 2	$**9.99	$ * * 0 . 1 2

If the receiving item is numeric or numeric edited, the literal can be any numeric literal . The point location and size of the literal are determined by the actual literal in the source statement. Further examples of editing are given in "PICTURE Clause" under "DATA DIVISION Structure" in Section 6.18.6.11.

Examples:

Source Data	PICTURE of Receiving Item	Receiving Item
+1.23	S9V99	1 2 $\overset{+}{3}$
+1.23	S9V9	1 2
123	9(5)	0 0 1 2 3
+37	S999V99	0 3 7 0 $\overset{+}{0}$
03737.3	$***9.9	$ 3 7 3 7 . 3

Table 12. Permissible Moves

Source Item		Receiving Field		
		GR	A	NP
Group	GR	A	A	A
[illegible]	A	[illegible]	[illegible]	[illegible][1]
[illegible]	NP	[illegible]	A[1]	[illegible]
Numeric literal		A	A[1]	[illegible]
[illegible] literal		[illegible]	A	[illegible][1]

[illegible] byte [illegible], byte-[illegible]-byte from left to right [illegible]

[illegible] in an inte[illegible]. In this case the [illegible] [illegible] with [illegible].

[illegible]

[illegible] characters in the source field cause unpredict-able data. [illegible]-editing is performed.

[illegible] INSPECT Statement: The INSPECT statement provides the [illegible] replace occurrences of characters in a data item.

Format:

INSPECT identifier-1 REPLACING {ALL / LEADING / FIRST} {identifier-2 / literal-1}

BY {identifier-3 / literal-2}

Identifier-1 must reference either a group item or any category of an elementary item, described implicitly or explicitly as USAGE IS DISPLAY. Identifier-2 through identifier-3 must reference a one byte elementary alphabetic, alphanumeric, or numeric item described implicitly or explicitly as USAGE IS DISPLAY. Literals must be non-numeric and may be any figurative constant except ALL.

Rules Applicable to All Formats: Inspection begins at the leftmost position of the data referenced by identifier-1, regardless of its class, and proceeds on a character-by-character basis to the rightmost character position. The contents of the data item referenced by identifier-1 is treated subject to whether the identifier is discribed as alphanumeric, unsigned numeric, or signed numeric:

1. Alphanumeric - identifier treated as a character string.

2. Unsigned numeric - inspected as though it had been redefined as alphanumeric and the INSPECT statement had been written to reference the redefined data.

3. Signed numeric - inspected as though the data item had been moved to an unsigned numeric data item of the same length, subject to the rules set forth above.

4. The rules for replacement are as follows:

a. When literal-1 is a figurative-constant, each character in the data referenced by identifier-1 that is equal to the figurative-constant is replaced by the single character referenced by literal-2 or identifier-3.

b. When literal-2 is a figurative-constant, each character in the data referenced by identifier-1 that is equal to the character referenced by literal-1 or identifier-2 is replaced by the character referenced by the figurative-constant.

5. The required words ALL, LEADING, and FIRST are adjectives that apply to the succeeding BY phrase:

a. If ALL identifier-2/literal-1s are to be replaced, this [illegible] done according to the replacement rules specified in paragraph 4.

b. If the adjective LEADING is used, all occurrences of the character string referenced by literal-1 or identifier-2 are re[illegible] by the character string referenced by literal-[illegible] or identifi[illegible] [illegible] the leftmost occurrence, in the data referenced [illegible] replacement begins.

[illegible], the leftmost occurrence, [illegible] the character string [illegible] identifier-2 [illegible], is replaced, in the [illegible], by the character string referenced [illegible].

[illegible]

[illegible] VALUE 12345678.

[illegible] PIC 999/99/9999.

[illegible] TO [illegible]-SS-NUMBER.

[illegible]-SS-NUMBER REPLACING [illegible] BY [illegible].

[illegible] value of [illegible]-SS-NUMBER will [illegible].

[illegible] [illegible] that control the [illegible] in which statements [illegible] are executed :

1. GO TO permanently releases control to the first statement in the procedure named.

2. PERFORM causes statements in a remote procedure to be executed and control returns to the statement following the PERFORM.

3. STOP allows the program to terminate in an orderly manner.

4. IF causes control to branch into either a "true" or "false" path, depending on the outcome of a condition test written in the program. The paths rejoin at the beginning of the next sentence unless a GO TO branch is used in one or both paths.

5. EXIT merely declares that the paragraph in which it is contained is a transfer point that may be referenced by other sequence control statements.

6.19.8.1 Normal Sequence Control: The starting location for the program is at the first statement of the PROCEDURE DIVISION. Control then proceeds to subsequent successive statements until the end of the paragraph or section is reached. Unless the paragraph or section is executed under control of a PERFORM statement, control then passes to the first statement in the next paragraph or section.

Execution of a sequence control statement, of course, alters the normal sequence of control.

6.19.8.2 GO TO Statement: The format of this statement is:

Format 1:

GO TO [procedure-name-1]

[illegible]

[illegible] procedure-name-n} . . .

[illegible]

[illegible] control, conditionally or unconditionally, to another point in a program.

[illegible] statement [illegible] control [illegible] of the [illegible] [illegible] can appear as [illegible] of segments.

[illegible]

[illegible] may be permutations of [illegible] etc. Since the search [illegible] data [illegible] the statement [illegible]

[illegible] control is transferred to [illegible] procedure-name-1, -2, or -n, [illegible] value [illegible] etc. [illegible]

[illegible]

passes to the next statement following the GO TO statement. A maximum of 16 procedure-names may be used in one GO TO statement.

Example:

GO TO FEDERAL-TAX, STATE-TAX, LOCAL-TAX DEPENDING ON GROSS-SALARY-CODE.

6.19.8.3 PERFORM Statement: The formats of this statement are:

Format 1:

```
PERFORM procedure-name-1 [THRU procedure-name-2]
```

Format 2:

```
PERFORM procedure-name-1 [THRU procedure-name-2]
{identifier-1}  TIMES
{integer-1   }
```

Format 3:

```
PERFORM procedure-name-1 [THRU procedure-name-2]
UNTIL condition-1
```

Format 4:

```
PERFORM procedure-name-1 [THRU procedure-name-2] VARYING
{index-name-1}  FROM  {index-name-2}  BY  {identifier-3}
{identifier-1}        {identifier-2}      {literal-3   }
                      {literal-2   }
UNTIL condition-1
[AFTER {index-name-4}  FROM  {index-name-5}  BY
[      {identifier-4}        {identifier-5}
[                            {literal-5   }
```

```
{identifier-6}      UNTIL      condition-2
{literal-6   }
[AFTER   {index-name-7}   FROM   {index-name-8}    BY
         {identifier-7}          {identifier-8}
                                 {literal-8   }
{identifier-9}      UNTIL      condition-3]]
{literal-9   }
```

The PERFORM statement causes a departure and return from normal procedures execution to another part of the program to execute one or more procedures. These procedures are executed a predetermined number of times or until a specified condition is satisfied, after which normal procedures execution resumes. In its simplest format the PERFORM provides a branch, execution of the procedure, and a return; in the more complex formats a branch is made, but the number of executions is contingent upon a condition controlled and tested by the statement. Thus, the PERFORM statement permits repetitive execution or looping using one statement; that is, it initializes and maintains loop criterion (variable), tests the criterion and performs operations.

The return point for the PERFORM statement is determined by whether the procedure to which it branches is a paragraph or section. When the instructions compiled from a PERFORM Statement are executed, they transfer control to the first statement of the specified procedure. Instructions that provide return to the statement following PERFORM are set up as follows:

1. If procedure-name-1 is a paragraph-name and a procedure-name-2 is not specified, control is returned after the last statement of

the procedure-name-1 paragraph.

2. If procedure-name-1 is a section and a procedure-name-2 is not specified, control is returned after the last statement of the last paragraph of the procedure-name-1 section.

3. If procedure-name-2 is specified and is a paragraph-name, control is returned after the last statement of the procedure-name-2 paragraph.

4. If procedure-name-2 is specified and is a section-name, control is returned after the last statement of the last paragraph of the procedure-name-2 section.

Note: The "last statement" referenced in each of the above cases must not be an unconditional GO TO statement.

When procedure-name-2 is specified, the only required relationship between procedure-name-1 and procedure-name-2 is that of logical sequence, that is, execution sequence must proceed from procedure-name-1 to the last statement of the procedure-name-2 paragraph or section. GO TO statements and other PERFORM statements are permitted between procedure-name-1 and the last statement of procedure-name-2 provided that the sequence ultimately returns to the final statement of procedure-name-2.

If the logic of a procedure requires a conditional branch prior to the final sentence, the EXIT statement may be used to satisfy the foregoing requirements. In this case, procedure-name-2 must be the name of a paragraph consisting solely of the EXIT statement; all paths must eventually lead to this point. (See the "EXIT Statement" discussion, Section 6.19.8.9)

It is not necessary for procedures to be referenced by a PERFORM statement before they can be executed. Procedures can also be executed in normal sequence from the preceding statement, in which case return of control does not apply after execution of the last sentence in a particular procedure.

6.19.8.4 "Nested" PERFORM Statement: If a sequence of statements referred to by a PERFORM statement includes another PERFORM statement, the sequence of procedures associated with the included PERFORM must itself be either totally included in, or totally excluded from the logical sequence referred to by the first PERFORM. Thus, an active PERFORM statement whose execution point begins within the range of another PERFORM must not contain within its range the exit point of the other active PERFORM statement.

6.19.8.5 TIMES Option: In Format 2, the procedure is executed repetitively a certain number of times. The number of executions may be specified explicitly as an integer or implicitly as the value of an elementary data item.

If an identifier is used it may be of any numeric usage, and it may be subscripted. When this option is included, a counter is set up with a value equal to the value of the identifier-1 item or integer 1. Before each execution of the specified procedure, the counter is tested to see if it is negative or zero. If it is neither negative nor zero, the procedure is executed and the value of the counter decreased by one; when the value of the counter is negative or zero, the procedure is executed and the value of the counter decreased by one; when the value of the counter is negative or zero, the procedure has been executed the specific number of times and control transfers to the statement following the PERFORM statement.

6.19.9.6 UNTIL Option: In Format 3, the number of times the procedure is executed is dependent on the truth or falsity of a condition (condition-1) rather than a stated value. Condition-1 can be any simple or compound conditional expression that is evaluated before the specified procedure is executed. If it is found to be false, the procedure is executed and the expression is evaluated again (values of the items may be altered by execution of the procedure) and tested for truth or falsity; this process is repeated until the conditional expression is found to be true, at which point control transfers to the statement following the PERFORM statement. If the conditional expression is found to be true when the PERFORM statement is first encountered, the specified procedure is not executed. (Refer to "Conditional Statements", Section 6.19.4).

6.19.8.7 VARYING Option: In Format 4 the VARYING option makes it possible to PERFORM a procedure repetitively, increasing or decreasing the value of one to three data items once for each execution until one to three conditional expressions are satisfied.

The flowcharts in Figure 6-3 illustrate the logic of the PERFORM statement when one, two, or three identifiers are varied. Let

1. Each d_i represent an identifier or index-name.
2. Each l_i represent a literal.

3. Each c_i represent a condition.

4. Each p_i represent a procedure-name.

Example: To help clarify use of the VARYING subscript-name option, assume that a rate table is employed in a billing procedure and that the table requires periodic updating. This hypothetical rate table is three-dimensional: divided into five regions, each of which includes ten states, each of which contains rates for twelve cities. It is assumed further that an appropriate rate-updating procedure is available elsewhere in the program. Such a procedure might appear as

```
RATE-UPDATING. MULTIPLY RATE (REGION, STATE, CITY) BY ADJUST-
FACTOR GIVING RATE (REGION, STATE, CITY).
```

It is desired to execute this RATE-UPDATING procedure once for each city of each state in each region, using the current rate for a given city and producing an adjusted rate for that city. Accordingly, the programmer employs a *PERFORM* statement *varying these items:*

```
PERFORM RATE-UPDATING VARYING REGION FROM 1 BY 1 UNTIL REGION
IS GREATER THAN 5 AFTER STATE FROM 1 BY 1 UNTIL STATE EQUALS
11 AFTER CITY FROM 1 BY 1 UNTIL CITY IS GREATER THAN 12.
```

When the PERFORM is executed at object time, the RATE-UPDATING procedure is executed for the first city of the first state in the first region, then for the next city, etc. The PERFORM is complete when the procedure is executed for the twelfth city of the tenth state of the fifth region, by which time the procedure has been executed 600 times.

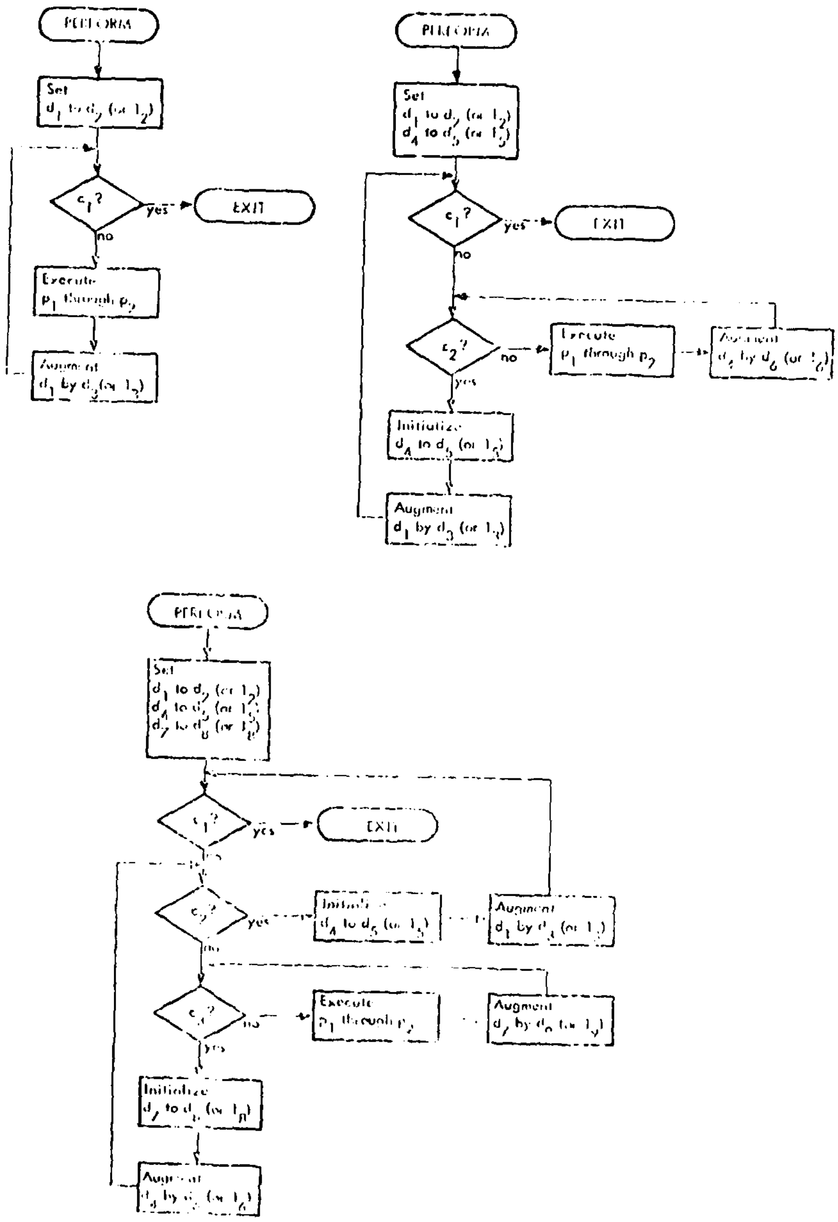


FIGURE 6.3 PERFORM Statement (VARYING Option)

6.19.8.8 STOP Statement: The format of this statement is:

```
STOP  {literal}
      {RUN    }
```

The STOP statement permanently suspends execution of the object program. STOP RUN generates an end-of-program exit to the Monitor that terminates program execution permanently. If STOP is followed by a literal, the literal is typed out and execution is suspended. Any literal may be used.

6.19.8.9 EXIT Statement: The format of this statement is:

```
paragraph-name.  EXIT
```

The EXIT statement ends a procedure to be executed by a PERFORM statement. EXIT must be the only statement in a paragraph; it is equivalent to a paragraph with no sentences and generates no code.

6.19.8.10 IF Statement: The format of this statement is:

```
IF condition THEN {statement-1  }  [ELSE {statement-2  }]
                  {NEXT SENTENCE}  [     {NEXT SENTENCE}]
```

The IF statement causes alternate sequences of operations to be followed, depending on whether the description of a data condition is found to be true or false when the data is evaluated. IF is followed by the description of the condition, then by the actions to be taken if the description of the condition is true. The word ELSE may be used, followed by the operations to be performed if the description of the condition is false.

The condition may be a simple condition as presented by the format below or a compound condition as described under "Conditional Statements", Section 6.19.4. The format of a simple condition is:

```
{ {identifier-1}         { IS [NOT] {GREATER THAN} } {identifier-2} }
{ {literal-1   }         {          {     >      } } {literal-2   } }
{ {formula-1   }         {          {LESS THAN   } } {formula-2   } }
{                        {          {     <      } }                }
{                        {          {EQUAL TO    } }                }
{                        {          {     =      } }                }
{ {identifier-3}           IS [NOT] {POSITIVE}                      }
{ {formula-3   }                    {NEGATIVE}                      }
{                                   {ZERO    }                      }
{  [identifier-4]          IS [NOT] {NUMERIC   }                    }
{                                   {ALPHABETIC}                    }
```

6.19.8.11 Evaluation of the Condition: The condition is evaluated before any action is taken. If the condition is true, either statement-1 or NEXT SENTENCE is executed. When NEXT SENTENCE is specified, control is transfered to the next sentence, and the ELSE part of the statement is ignored. If the condition is false, either statement-2 or NEXT SENTENCE is executed. Control is transferred to the succeeding sentence when NEXT SENTENCE is specified. Statement-1 or statement-2 may be a series of statements and each may be terminated by a period of ELSE.

6.19.8.12 Nested Conditional Statements: Statements-1 and -2 can be imperative-statements or imperative-statements followed by a conditional statement. When either statement-1 or statement-2 or both contain a conditional statement, the conditional statement becomes nested. Nested conditional statements may also contain conditional statements. Nested conditional statements are analogous to the use of parentheses for combining subordinate arithmetic-expressions so that the expressions become part of a larger arithmetic unit.

6.19.8.13 Evaluation of Nested IF Statements: Conditional statements contained within conditional statements (IFs within IFs) must be considered as paired IF and ELSE combinations, proceeding from left to right. Therefore, any ELSE encountered applies to the immediately preceding IF that is not already paired with an ELSE.

In essence, the number of occurrences of ELSE in any conditional statement must be equal to the number of occurrences of IF, regardless of the complexity caused by nesting, with the following exception: when ELSE or NEXT SENTENCE directly precedes the terminal period of a sentence, the entire phrase may be omitted and the period specified at the end of the previous phrase. This rule is extended to resulting sentences, etc. For each ELSE, the associated statement is executed only when the conditional expression in the corresponding IF is found to be false. If there are more IFs than ELSES in a statement, it is assumed that ELSE NEXT SENTENCE phrases at the end of the sentence are omitted.

Example: The sentence in the following paragraph contains two independent nests of conditional statements. The first nest ends after the statement PERFORM procedure-name-2; the second nest consists of the remainder of the sentence and has an implied ELSE NEXT SENTENCE before the period. Each upper-case letter of the alphabet corresponds to a conditional expression.

```
IF A IF B PERFORM procedure-name-1 ELSE NEXT SENTENCE ELSE
IF C NEXT SENTENCE ELSE PERFORM
procedure-name-2 IF D PERFORM procedure-name-3 IF E PERFORM
procedure-name-4 IF F PERFORM procedure-name-5 ELSE PERFORM
procedure-name-6 ELSE STOP RUN.
```

6.19.9 Table-Handling Statements: The structure of a table is defined by the use of an OCCURS clause (refer to "OCCURS clause" Section 6.18.6.16). Entries in a table may be referenced by a subscript or index, which identifies a particular element within a table.

Indexing has the advantage in efficiency that no address computation is involved; an index contains a direct pointer to an individual element in a table rather than a mere occurrence number. The SET statement facilitates the correct setting of indexes.

The formats of the SET statement are:

Format 1:

$$\underline{\text{SET}} \quad \left\{\begin{array}{l}\text{index-name-1}\\ \text{identifier-1}\end{array}\right\} \quad \text{TO} \quad \left\{\begin{array}{l}\text{index-name-2}\\ \text{identifier-2}\\ \text{literal-1}\end{array}\right\}$$

Format 2:

$$\underline{\text{SET}} \quad \text{index-name-3} \quad \left\{\begin{array}{l}\underline{\text{UP BY}}\\ \underline{\text{DOWN BY}}\end{array}\right\} \quad \left\{\begin{array}{l}\text{identifier-3}\\ \text{literal-2}\end{array}\right\}$$

The SET statement establishes reference points for table-handling operations by setting index-names associated with table elements.

All identifiers must be either index data items or numeric elementary items described without any positions to the right of the assumed decimal point, except that identifier-3 must not be an index data item. When a literal is used, it must be a positive integer. Index-names are considered related to a given table and are defined by specification in the INDEXED BY clause.

In Format 1 the following action occurs:

1. Index-name-1 is set to a value corresponding to the same occurrence number to which either index-name-2, identifier-2 or literal-1 corresponds. If identifier-2 is an index data item or if index-name-2 is related to the same table as index-name-1, no conversion takes place.

2. If identifier-1 is an index data item, it may be set equal to either the contents of index-name-2 or identifier-2 where the latter is also an index data item; literal-1 cannot be used.

3. If identifier-1 is not an index data item, it may be set only to an occurrence number corresponding to the value of index-name-2; neither identifier-2 nor literal-1 can be used.

In Format 2 the value of index-name-3 is incremented (UP BY) or decremented (DOWN BY) by a value corresponding to the number of occurrences represented by the value of literal-2 or identifier-3.

7. DISCUSSION & CONCLUSIONS

7.1 Design Conclusions

An examination of the variables in computer networks indicates that these variables can be classified either as host-controlled resource variables or as network variables. Obviously, the former are determined by the nature of the hosts and the latter by the nature of the network.

Two distinctive aspects of distributed microcomputer networks are the facts that the hosts, being microprocessors, can control only one operation at a time and that packet-switching has been chosen for the network. Further considerations for the monitor system is the desire to require a minimal overhead for the monitor system and to acquire it for a cost comparable to that of the (inexpensive) microcomputers.

The review of the literature indicates that the quantities to be measured for microcomputer networks can, in fact, be a subset of the variables measured for larger networks and computers. A particular set of variables, which are felt to be sufficient, is listed in Section 3.

With respect to measuring the desired variables, the host-controlled resource variables can be measured in the same manner as described in the literature for large computers with some simplification due to the limited flexibility of microcomputer hosts. The problems here are interfacing with specific equipment and achieving an integrated monitor system with convenient user access.

With respect to measuring network variables, there is limited discussion in the literature of monitor systems for packet-switching networks. Monitoring for the ARPANET is, of course, discussed in considerable detail. This network, however, differs from those being considered in significant respects such as scale and age, to mention only two. No monitoring system for recently designed packet-switching mini or microcomputer networks was found discussed in the literature.

With this background, the need was felt to adapt existing monitor strategies to the characteristics of distributed microcomputer networks and design a complete monitor system structure for such networks. The design is discussed in Section 3.

The low overhead for the monitor system is felt to be especially attractive. The host-controlled resources are monitored without the use of software and hence, require no overhead. The use of the Two-Port RAM's at each node provides data on the network operation, also without overhead. The pickup packets, which probe packet delay and convey information between the Monitor Stations and the Monitor Control, are the only aspects of the monitor system which require overhead. Such overhead is determined by the ratio of the number of pickup packets to the total number of packets in the network over some reference time interval.

By processing data at each node and storing, for example, histograms or random variables, the need for frequent communication between the Monitor Stations and Monitor Control is minimized. Thus, the limiting factor in pickup packet overhead would seem to be the frequency with which packet delays need to be sampled. This frequency will, of course, depend on the use of the monitor system.

With respect to this point, it is likely that packet delay will be of significant importance in studies for improving the network design. On the other hand, in an operating network, where efficiency is important, frequent measurement of packet delay may not be necessary and hence the number of pickup packets can be kept small.

An implementation of the general monitor system design is given in Section 4. The implementation is chosen to be compatible with the AIRMICS/GEORGIA TECH Experimental Network and thus it has the potential of being used with that network. This point is discussed in Section 4.3.

An example of a typical use of the distributed microcomputer network is formulated in Section 4.2. For this example, the monitor system is studied on a step-by-step basis. As indicated by detailed activity tables in Appendix A, the proposed monitor system can apparently function properly for this test case.

7.2 Network Experimental Conclusions

The results of the Inventory Control Program Test with Trafficking series of tests are as follows:

1. It was possible to overload the network and cause it to fail by trafficking nodes that were also receiving large bursts of data from the host computers.

2. The communication network was more likely to reorder messages during a traffic situation than with no traffic.

3. These tests helped point out some of the characteristics of the network that are described in detail in the section of network characteristics.

4. The test helped point out characteristics of the inventory control problem that are detailed in the section on inventory control program characteristics.

8. BIBLIOGRAPHY

I. HARDWARE MONITORS

1) J. S. Cockrum, E. D. Crockett, "Interpreting the Results of a Hardware Systems Monitor", AFIPS Proc., SJCC, 1971, pp. 23-28.

2) G. Estrin, D. Hopkins, B. Coggan, S. D. Crocker, "SNUPER COMPUTER, A Computer in Instrumentation Automation", AFIPS Proc., SJCC 1967, pp. 645-656.

3) R. W. Murphy, "The System Logic and Usage Recorder", AFIPS Proc., FJCC, 1969, pp. 219-229.

4) R. Aschenbrenner, L. Amiot, N. K. Natarajan, "The Neurotron Monitor System", AFIPS Proc., FJCC, 1971, pp. 31-37.

5) F. Schulman, "Hardware Measurement Device for IBM System /360 Time Sharing Evaluation", Proc. of the 22nd ACM Nat. Conf., Aug. 1967, pp. 163-199.

6) J. Noe, "Acquiring and Using a Hardware Monitor", Datamation, April, 1974, pp. 89-95.

7) L. Svobodova, "Computer Systems Measurability", Computer, May/June, 1976, pp. 9-17.

8) R. E. Fryer, "The Memory Bus Monitor - A New Device for Developing Real-Time Systems", AFIPS Conf. Proc., 1973, NCC, pp. 75-79.

9) F. Arnolt, G. M. Oliver, "Hardware Monitoring of Real-Time Computer Systems Performance", Computer, July/Aug., 1972, pp. 25-29.

10) H. C. Lucas, "Performance Evaluation and Monitoring", Computing Surveys, V. 3, No. 3, Sept. 1971, pp. 79-91.

II. SOFTWARE MONITORS

11) Y. Bard, "The VM/370 Performance Predictor", Computing Surveys, V. 10, No. 3, September 1978, pp. 333-341.

12) P. Balcom, G. Cranson, "USACSC Software Computer System Performance Monitor: SHERLOCK.", Proc. of the 8th Meeting of CPEUG, Sept. 1974, pp. 37-43.

II. SOFTWARE MONITORS (cont'd)

13) R. Castle, "Performance Measurement of USACSC", Proc. of 8th Meeting of CPEUG, September 1974, pp. 55-62.

14) K. Wong, J. C. Strauss, "Use of a Software Monitor in the Validation of an Analytical Computer System Model", Software-Practice and Experience, Vol. 4, 1974, pp. 255-263.

15) J. C. Strauss, "An Analytic Model of the Hasp Execution Task Monitor", Communications of the ACM, Dec. 1974, Vol. 17, No. 12, pp. 679-685.

III. HARDWARE/SOFTWARE MONITORS FOR COMPUTER NETWORKS

16) D. E. Morgan, W. Banks, D. Goodspeed, R. Kolanko, "A Computer Network Monitoring System", Trans. on Software Engineering, Vol. SE-1, September 1975, pp. 299-311.

17) D. E. Morgan, W. Banks, D. Sutton, W. Calvin, "A Performance Measurement System for Computer Networks", Proc. IFIP, Congr., 1974, pp. [illegible].

18) L. Kleinrock, W. E. Naylor, "On Measured Behavior of the ARPA Network", AFIPS. Proc. NCC, 1974, Vol. 43, pp. 767-780.

19) S. Kitazawa, T. Sakai, "Performance Evaluation of the Fujinet Computer Network", Computer Communications, Vol. 1, No. 3, June 1978, pp. 149-155.

IV. PARAMETERS MEASURED BY MONITORING SYSTEMS

20) S. W. Cox, "Interpretive Analysis of Computer System Performance", ACM Performance Evaluation Review, Vol. 2, No. 4, Dec. [illegible], pp. 140-155.

21) C. A. Rose, "A Measurement Procedure for Queueing Network Models of Computer Systems", Computing Surveys, Vol. 10, No. [illegible], pp. 263-275.

22) J. Bear, T. Reeves, "Workload Characterization and Performance Measurement for a CDC Cyber 74 Computer System", 13th Meeting of the CPEUS, NBS Special Publication 500-18, pp. 39-67.

IV. PARAMETERS MEASURED BY MONITORING SYSTEMS (cont'd)

23) D. C. Wood, E. H. Forman, "Throughput Measurement Using a Synthetic Job Stream", AFIPS Proc. FJCC, 1971, pp. 51-55.

GENERAL REFERENCE ITEMS I-IV

24) L. Svobodova, Computer Performance Measurements and Evaluation Methods: Analysis and Applications, Elsevier, North Holland, 1976.

V. EXISTING COMPUTER NETWORKS

25) E. Manning, R. W. Peebles, "A Homogeneous Network for Data Sharing Communications", Computer Networks, 1977, pp. 211-224.

26) J. Labetoulle, E. G. Manning, R. W. Peebles, "A Homogeneous Computer Network", Computer Networks I, (1977), pp. 225-240.

27) D. J. Farber, "A Ring Network", Datamation, Feb. 1975, pp. 45-46.

28) J. McQuillan, W. R. Crowther, B. P. Cosell, D. C. Walden, "Improvements in the Design and Performance of the ARPA Network", AFIPS Proc. FJCC, 1972, pp. 741-754.

29) H. Aiso, Y. Matsushita, et.al., "A Minicomputer Complex - KOCOS", IEEE/ACM Fourth Data Communications Symposium - Quebec City, Oct. 1975, pp. 5-7 to 5-12.

30) Kitazawa, "Performance Evaluation of KUIPNET Computer Network", Computer Communications, Vol. 1, No. 3, June, 1978.

31) A. G. Fraser, "A Virtual Channel Network", Datamation, Vol. 21, No. 2, 1975, pp. 51-56.

32) D. L. Mills, "An Overview of the Distributed Computer Network", AFIPS National Computer Conference Proceedings, Vol. 45, 1976, pp. 523-531.

33) L. Kleinrock, W. Naylor, "On Measured Behavior of the ARPA Network", National Computer Conf., 1974, pp. 767-780.

34) David C. Wood, "A Survey of the Capabilities of 8 Packet Switching Networks", Computer Networks: Trends and Applications, June 1975, pp. 1-7.

V. EXISTING COMPUTER NETWORKS (cont'd)

35) J. R. Halsey, L. E. Hardy, L. F. Powning, "Public Data Networks: Their Evolution, Interfaces and Status", IBM Systems J., Vol. 8, No. 2, Nov. 1979, pp. 223-243.

VI. ANALYTIC AND SIMULATION MODELS FOR COMPUTER NETWORKS

36) S. R. Kimbleton, "A Heuristic Approach to Computer Systems Performance Improvement. I - A Fast Performance Prediction Tool", AFIPS NCC, 1975, pp. 839-845.

37) J. W. Boyse, D. R. Warn, "A Straight-Forward Model for Computer Performance Prediction", Computer Surveys, Vol. 7, No. 2, June 1975, pp. 73-93.

38) K. M. Chandy, U. Herzog, L. Woo, "Approximate Analysis of General Queueing Networks", IBM Journal Research & Development, Jan. 1975, pp. 43-49.

39) F. Baskett, K. M. Chandy, R. Muntz, F. G. Palacios, "Open, Closed, and Mixed Networks of Queues with Different Classes of Customers", J. of the ACM, Vol. 22, No. 2, April 1975, pp. 248-260.

40) M. Reiser, "Interactive Modeling of Computer Systems", IBM System Journal, No. 4, 1976, pp. 309-327.

41) P. J. Denning, J. P. Buzen, "The Operational Analysis of Queueing Networks Models", Computing Surveys, Vol. 10, No. 3, Sept. 1978, pp. 225-261.

42) J. W. Wong, "Queueing Network Modeling of Computer Communication Networks", Computing Surveys, Vol. 10, No. 3, Sept. 1978, pp. 343-351.

43) M. Irland, "Queueing Analysis of a Buffer Allocation Scheme for a Packet Switch", National Telecommunications Conf. Record, 1975, pp. 24-8 through 24-13.

44) F. A. Tobagi, M. Gerla, R. W. Peebles, E. G. Manning, "Modeling and Measurement Techniques in Packet Communication Networks", Proc. of IEEE, Vol. 66, No. 11, Nov. 1978, pp. 1423-1447.

45) L. Kleinrock, Queueing Systems, Volume II: Computer Applications, New York, Wiley Interscience, 1976.

VII. MEASUREMENTS FOR DETERMINING PARAMETERS FOR USE WITH NETWORK MODELS

46) D. Sutton, D. Morgan, "The Monitoring of Computer Systems and Networks: A Summary and Proposal", University of Waterloo Computer Communications, Network Group Report, E-22, May, 1974.

47) F. Tobagi, et al., "On the Measurement Facilities in Packet Radio Systems", Nat. Computer Conf. Proc. (New York), June 1976.

48) S. A. Mamrak, S. R. Kimbleton, "Comparing Equivalent Network Services Through Dynamic Processing Time Prediction", AFIPS Nat. Comp. Conf., 1977, pp. 455-460.

49) F. Tobagi, S. Lieberson, L. Kleinrock, "On Measurement Facilities in Packet Radio Systems", AFIPS Proc. NCC, 1976, pp. 589-596.

50) G. Estrin, L. Kleinrock, "Measures, Models and Measurements in Time-Shared Computer Utilities", Proc. ACM Nat. Meeting, 1967, pp. 85-96.

VIII. COMMERCIAL MONITOR EQUIPMENT

51) L. E. Hart, G. J. Lipovich, "Choosing a System Stethoscope", Computer Decisions, Nov. 1971, pp. 20-23.

52) M. L. Stiefel, "Network Diagnostic Tools", Mini-Micro Systems, March 1979, pp. 62-76.

9. APPENDIX A

TABLES GIVING COMPUTER NETWORK AND MONITOR SYSTEM ACTIVITY FOR INVENTORY CONTROL EXAMPLE

TABLE A1. General Monitor System Functions: Initial Set Up for Complete Problem

- Set up Masked-Word Range Comparators to record the activity of the Host CPU at Node K (Module K1) and the Host CPU at the MC Node (Module MC1). (Other modules are required for Jobs 3 and 4.)
- Set up Interval Counters to record the activity of the Terminal, the Line Printer and the Disk at Node K (Modules K2, K3, and K4) and the Disk at the MC Node (Module MC2). (Other modules are required for Jobs 3 and 4.)
- Initialize the modules for monitoring the Node CPU activity of each node. Modules K5 and MC3 are used with Jobs 1 and 2. Other modules are required for activities associated with Jobs 3 and 4 and with possible alternate routing used in Job 1.
- Initialize the count in all Two-Port RAM Counter locations at each node.
- Set the Job ID numbers to zero in the Two-Port RAM at each node.
- Identify Two-Port RAM memory locations for data to be transmitted to the MC Node (for this example, assume all memory locations fall in this category).
- Identify Two-Port RAM memory locations for variables from which histograms will be generated.
- Activate total problem time counter.

TABLE A2. General Monitor System Functions:

Periodic Monitor Functions

- Transmit pickup packets from MC Node.
- Sample all Two-Port RAM counter locations and store the values read along with the time in appropriate Two-Port RAM memory locations for periodic transfer to the MC Node and/or input certain values to Histogram Generators.

Discussion: The pickup packets cause the following activities to take place.

At each node the ID of the pickup packet is read by the Node CPU, an interrupt is generated, the Node CPU causes the Real Time Clock to be read, the resulting number is recorded in the data field of the pickup packet.

Data from specific Two-Port RAM memory locations and from modules is read into the data field of the pickup packet to be transmitted to the MC Node.

As each pickup packet is transmitted from a node, an interrupt is generated and a time value is read into a storage location identified with the departing pickup packet number.

TABLE A3. General Monitor System Functions:

Monitor Functions at Problem Completion

- Stop total problem line counter.
- Transmit pickup packets to all nodes.
- Read data fields of returning pickup packets at MC Node.
- Compute all desired functions of accumualted data.
- Output all desired data from MC Node Host.

Discussion: As an example of a desired function of the accumulated data at the MC Node, the total number of packets transmitted from K to MC in a short time interval can be computed and divided into the Node CPU and communication channel costs for this interval to obtain the cost per packet over this path during the time interval. The resulting number can be multiplied by the number of Job 1 packets transmitted from K to MC to give the network cost to be used with Job 1 in the same time interval.

TABLE A4. Activities in Execution of Job 1 with Corresponding Monitor System Readings

Activity Number	Computer Network	Monitor System
1	Host CPU at Node K reads input instructions.	° Host CPU identifies the users request for restoration of the Node K data base as Job 1. The job number is communicated through the Serial Port to a storage location in the Two-Port Memory. ° The Host-Controlled Resource measurements program detects the job ID and initializes modules K1, K2, K3 and K4. ° Module K1 begins to time the activity of the Node K Host CPU. ° Module K2 times the activity of the Node K Terminal.
2	The Host CPU at Node K generates a packet addressed to the MC Node.	° The packet addressed to the MC Node is identified as packet number 1 of Job 1.
3	The Node CPU at Node K transfers Packet 1 to the Node Buffer	° Node K CPU stores one count in the Two-Port RAM location for Job 1 "packets generated count" and one count in the location for "packets awaiting service". ° At the beginning of Activity 3, Module K5 begins to time activity of Node K CPU. ° At the end of Activity 3 Module K1 ceases to count.
4	After any required delay, Packet 1 is transmitted by Node K CPU.	° Node K CPU stores a count in the "packets transmitted K to MC" storage location in the Two-Port RAM for Job 1. ° Node K CPU decreases the "packets awaiting service" count by one. ° At the end of Activity 4, Module K5 ceases to count.

TABLE A4. (Cont'd) Activities in Execution of Job 1 with Corresponding Monitor System Readings

Activity Number	Computer Network	Monitor System
5	The MC Node CPU receives Packet 1 into its buffers.	° Module MC3 begins to count Node CPU activity. ° MC Node CPU stores one count in "packets received" location of the Two-Port RAM. ° MC Node CPU stores one count in "packets awaiting service" storage location.
6	The MC Node CPU transmits an acknowledgement to Node K.	° Module K5 begins to count Node K CPU activity.
7	Node K CPU receives the acknowledgement.	° Module K5 ceases to count Node K CPU activity.
8	The MC Node CPU transfers packet 1 to the MC Host.	° At the end of Activity 8, Module MC3 ceases to count MC Node CPU activity. ° The MC Node CPU decreases the "packets awaiting service" count by one.
9	MC Host reads instructions from Packet 1.	° The MC Host identifies the requested service as a part of Job 1 from the job number in the packet. ° ID number is recorded in Two-Port RAM. ° Modules MC1 and MC2 are initialized. ° Module MC1 begins to count Host CPU activity.
10	MC Host accesses its Disk to obtain the Item A inventory data.	° An Active Status Signal for the MC Disk is detected by Module MC2 which begins to count Disk activity.

TABLE A4. (Cont'd) Activities in Execution of Job 1 with Corresponding Monitor System Readings

Activity Number	Computer Network	Monitor System
11	The MC Host begins generating packets and with the MC Node CPU stores the packets in the MC Node Buffers continuing until the Buffers are filled with packets from Job 1 and other jobs. After delays, when the MC Node Buffers are full, all required packets are generated and stored in the MC Node Buffers.	° As each packet is generated and transferred into the MC Node Buffers, the MC Node CPU increments the Two-Port RAM packets generated count on Job 1. ° At the beginning of Activity 11, Module MC3 begins to count MC Node CPU activity. ° The MC Node CPU increments the "packets awaiting service" count as each packet is stored in the Node Buffer. ° When all packets are generated, Modules MC1 and 2 (Host and Disk activity) cease their count.
12	With intermitent delays due to unavailability of communication links, the MC Node CPU transmits the packets from the Node Buffer in the order in which they are stored.	° The MC Node CPU increments the packets transmitted MC to K count with every packet transmitted. ° The MC Node CPU decreases the packets awaiting service count by one as each packet is transmitted. ° All MC Node CPU active time is recorded by Module MC3. ° If alternate routing is required, the count in packets transmitted MC to X is appropriately incremented.
13	For every packet, an acknowledgement is transmitted by the MC Node CPU and received by the Node K CPU. Any packet not acknowledged is retransmitted.	° Module K and MC Node CPU active times are recorded by Modules K5 and MC3. ° The Node K CPU stores a count in "packets not acknowledged" for every packet retransmitted.

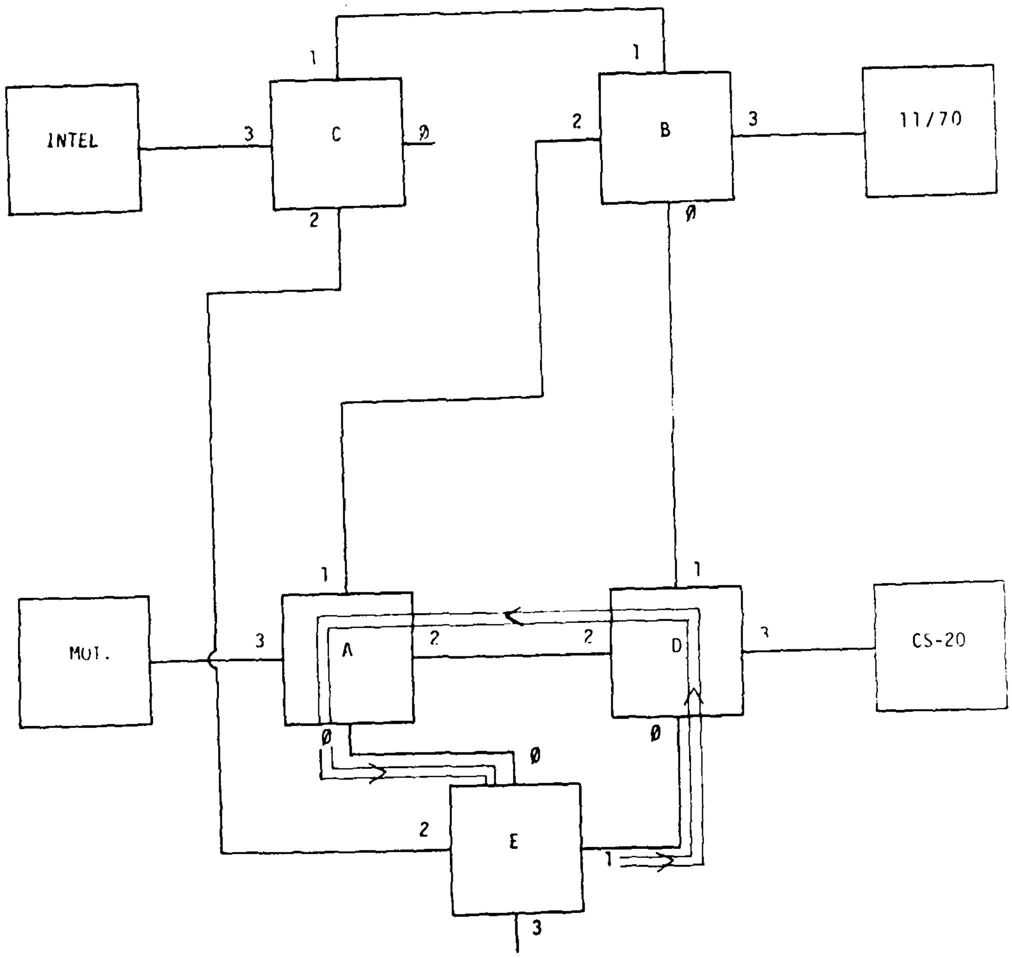
TABLE A4. (Cont'd) Activities in Execution of Job 1 with Corresponding Monitor System Readings

Activity Number	Computer Network	Monitor System
14	The packets correctly received at Node K are transferred from the Node K input buffer to the Node K Host and the data from the packets is stored by the Host on its Disk.	° Module K5 records all Node CPU activity. ° The job number from the arriving packets is read into the Two-Port ID memory location to activate Modules K1 and K4. These modules read all Host CPU and Disk activity. ° Every arriving packet increments the Node K "packets received" count. ° The "packets awaiting service count" is incremented up with every packet received into the Node K Buffers and incremented down when the packet is transferred to the Node K Host.
15	After all packet information has been read into the Node K Disk, Job 1 is complete.	° Any new request for Node K Host or Disk will carry a new ID number, therefore, the count of Job 1 activity concludes at the end of Activity 14.

TABLE A5. Activities in Execution of Job 2 with Corresponding Monitor System Readings

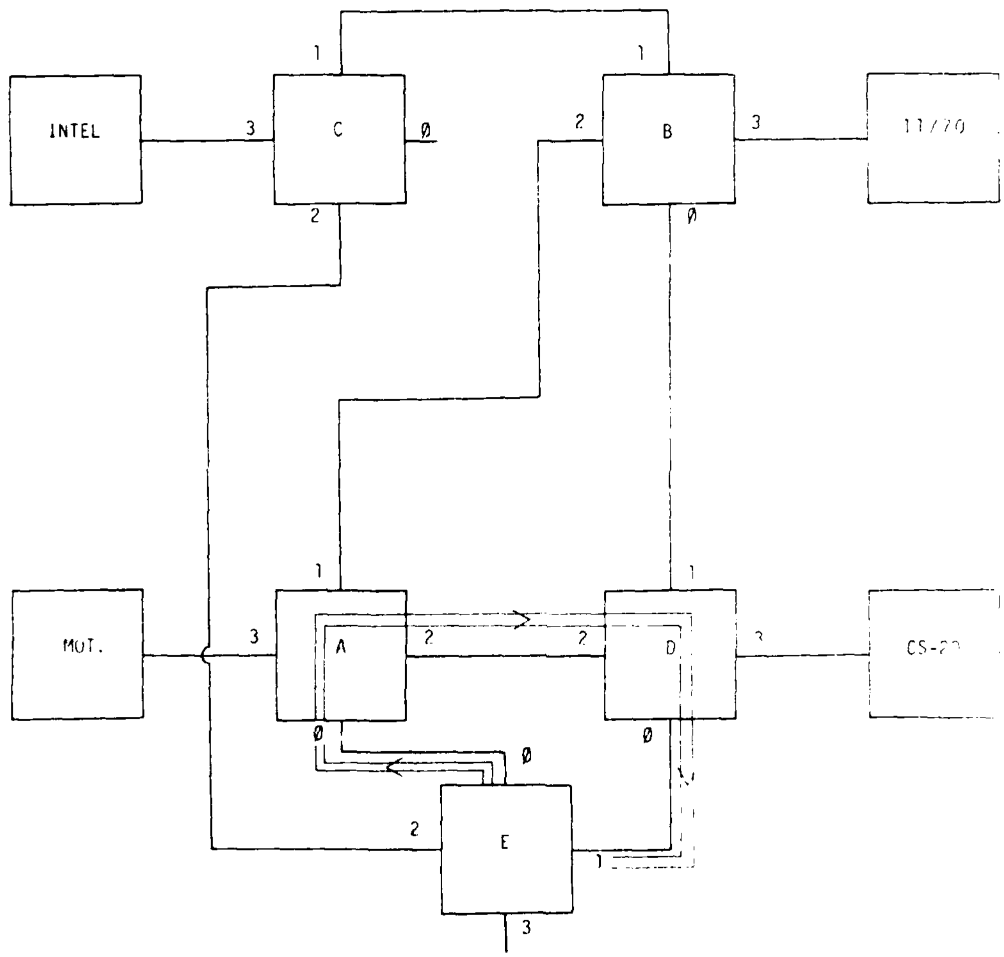
Activity Number	Computer Network	Monitor System
1	Node K Host CPU reads user input instructions.	° Host identifies search for A_i items in Node K inventory as Job 2. ° Host-Controlled Resource Measurement Program detects the job ID and initializes Modules K1, K2, K3, and K4. ° Module K1 begins to time the activity of the Node K Host CPU. ° Module K2 times the activity of the Node K Terminal.
2	The Node K Host CPU transfers the data file on item A from Disk to Memory.	° An "active status" signal from the Disk is detected by Module K4, which times disk activity until it is over.
3	The Node K Host CPU carries out a search of memory for items A_i.	° Module K1 continues to time Node K Host CPU activity.
4	The Node K Host CPU outputs the information that no items A_i are present in inventory.	° Module K3 times the activity of the Line Printer. ° Module K1 terminates its count when Activity 4 is over.

10. APPENDIX B--TRAFFIC ROUTES



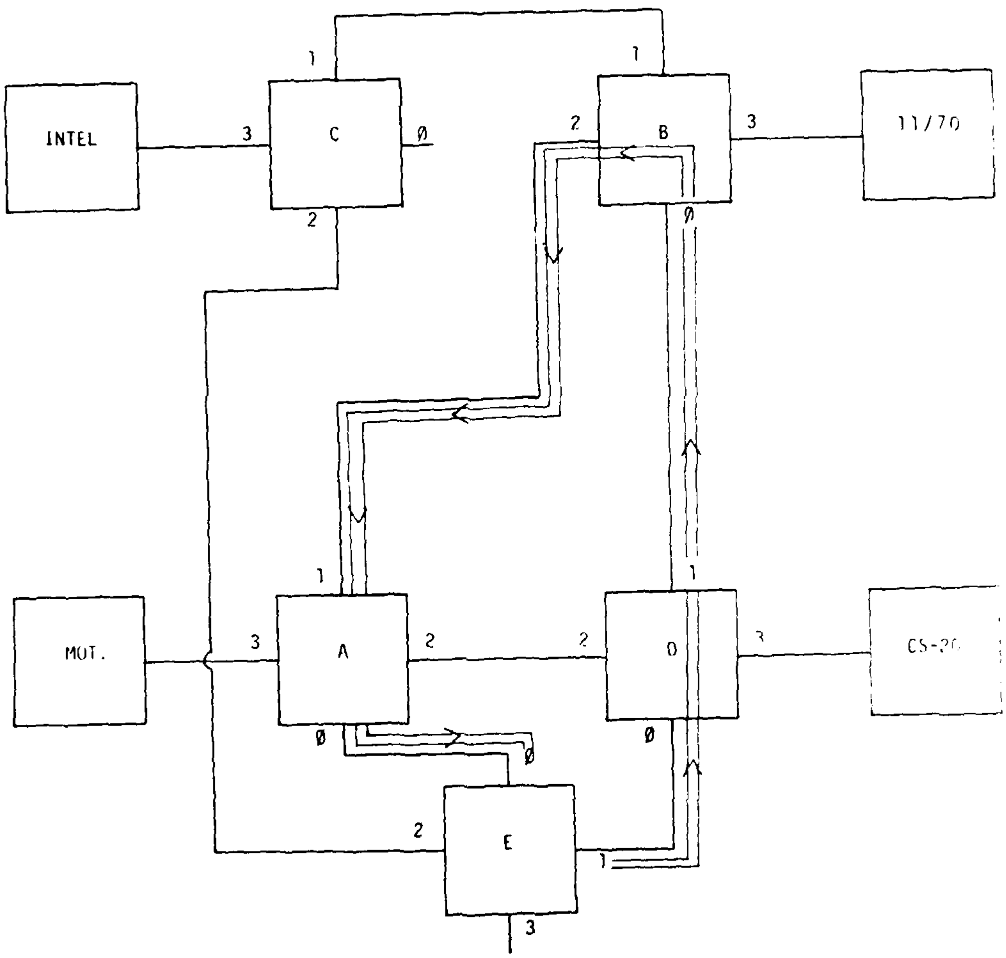
NOVA 820

TRAFFIC ROUTE F



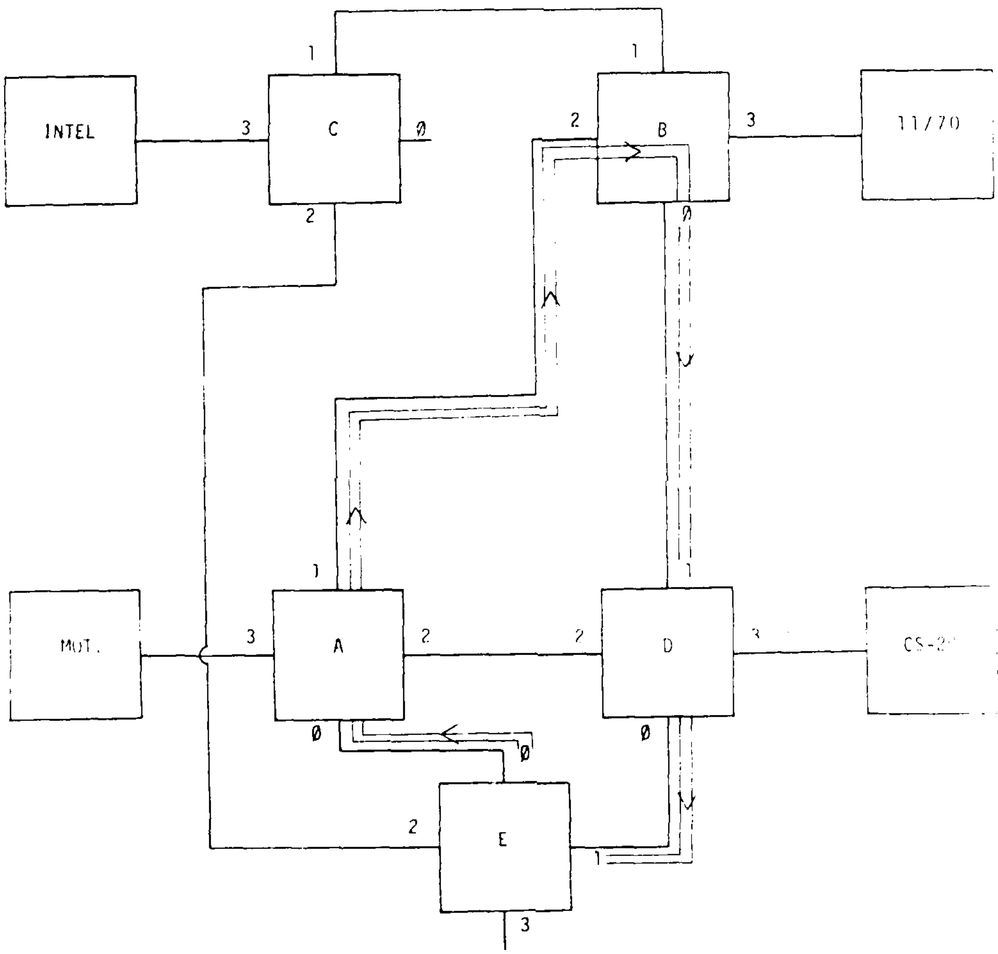
NOVA 820

TRAFFIC ROUTE G



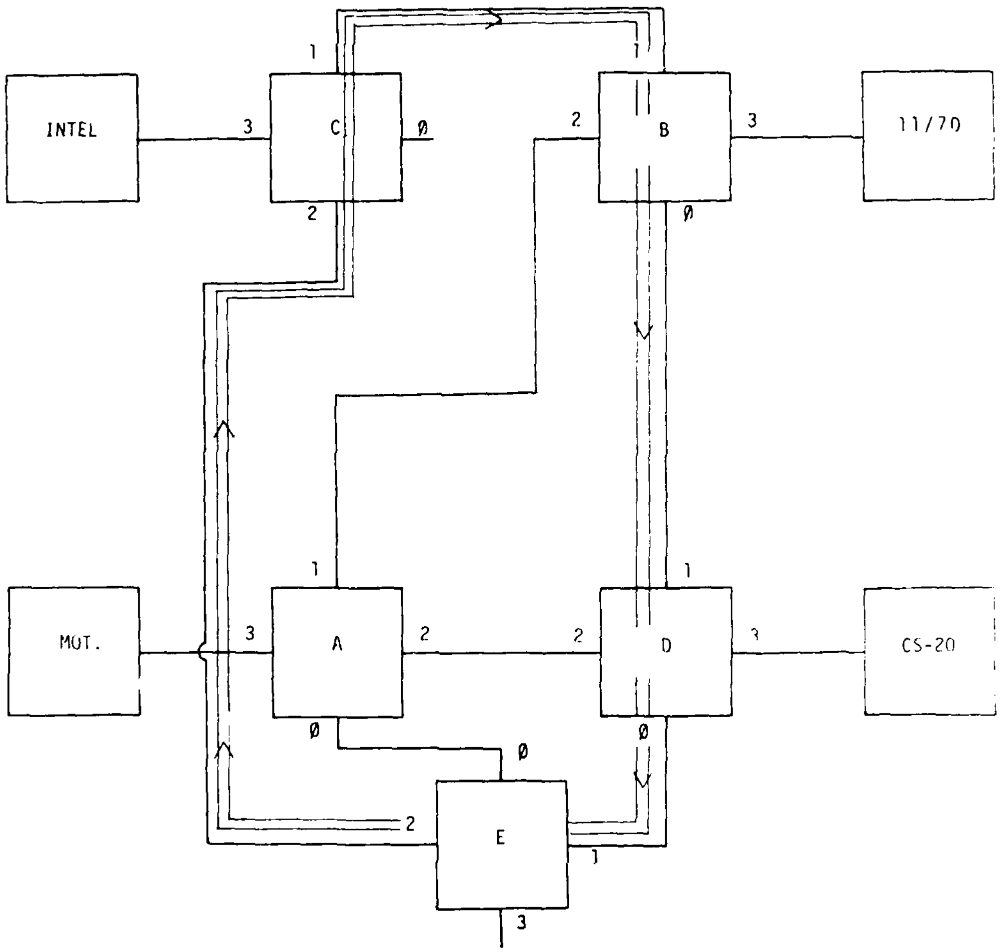
NOVA 820

TRAFFIC ROUTE H



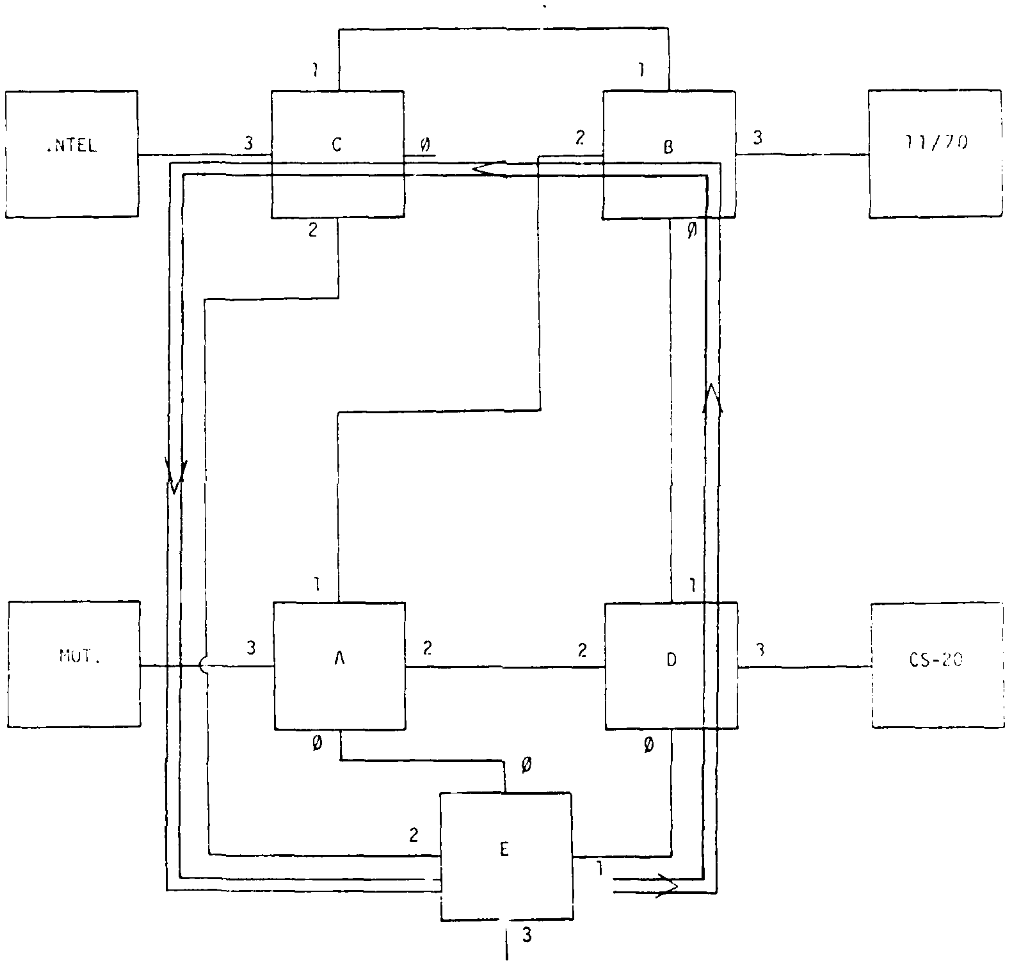
NOVA 820

TRAFFIC ROUTE I



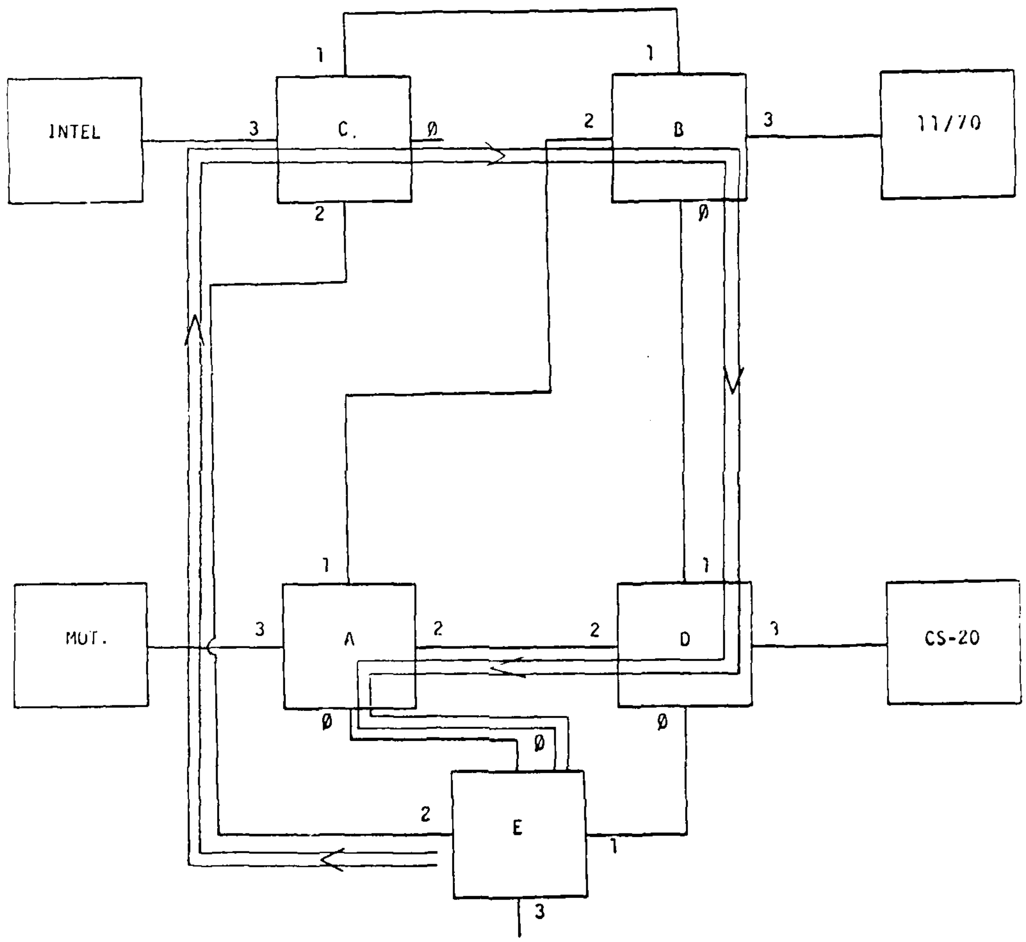
NOVA 820

TRAFFIC ROUTE J



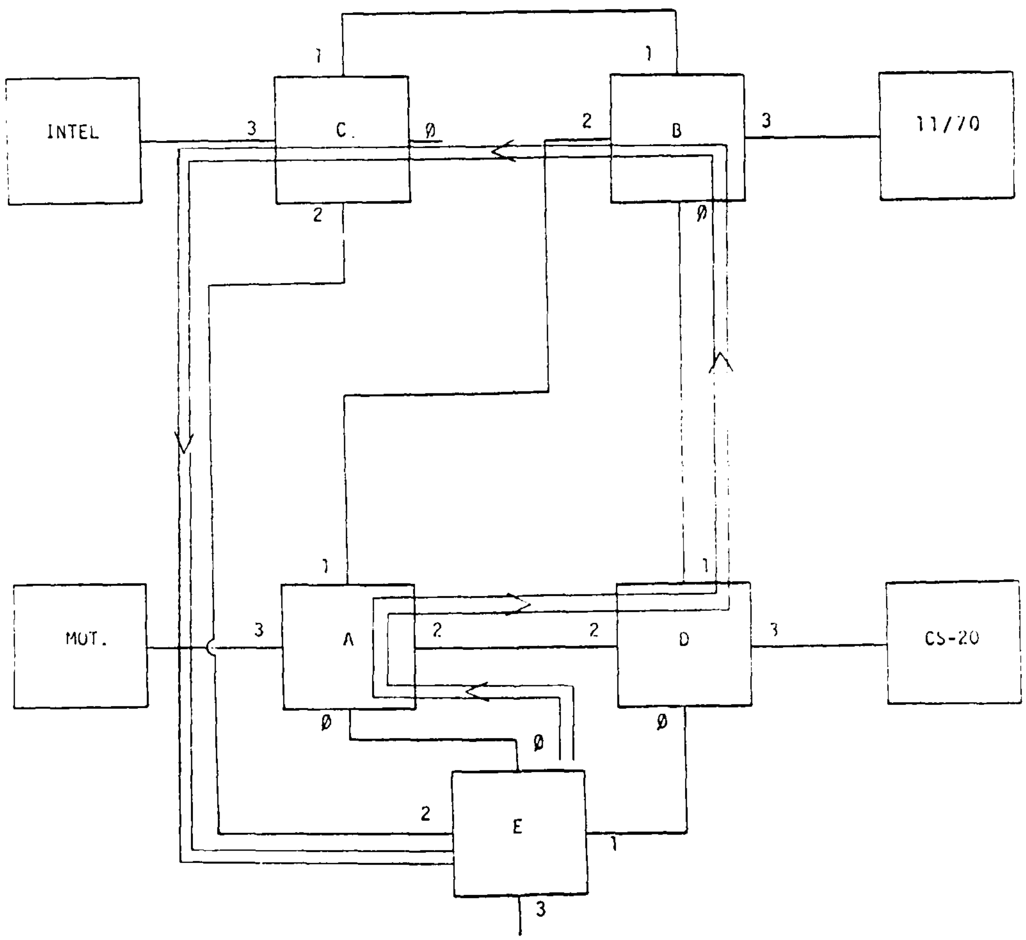
NOVA 820

TRAFFIC ROUTE K



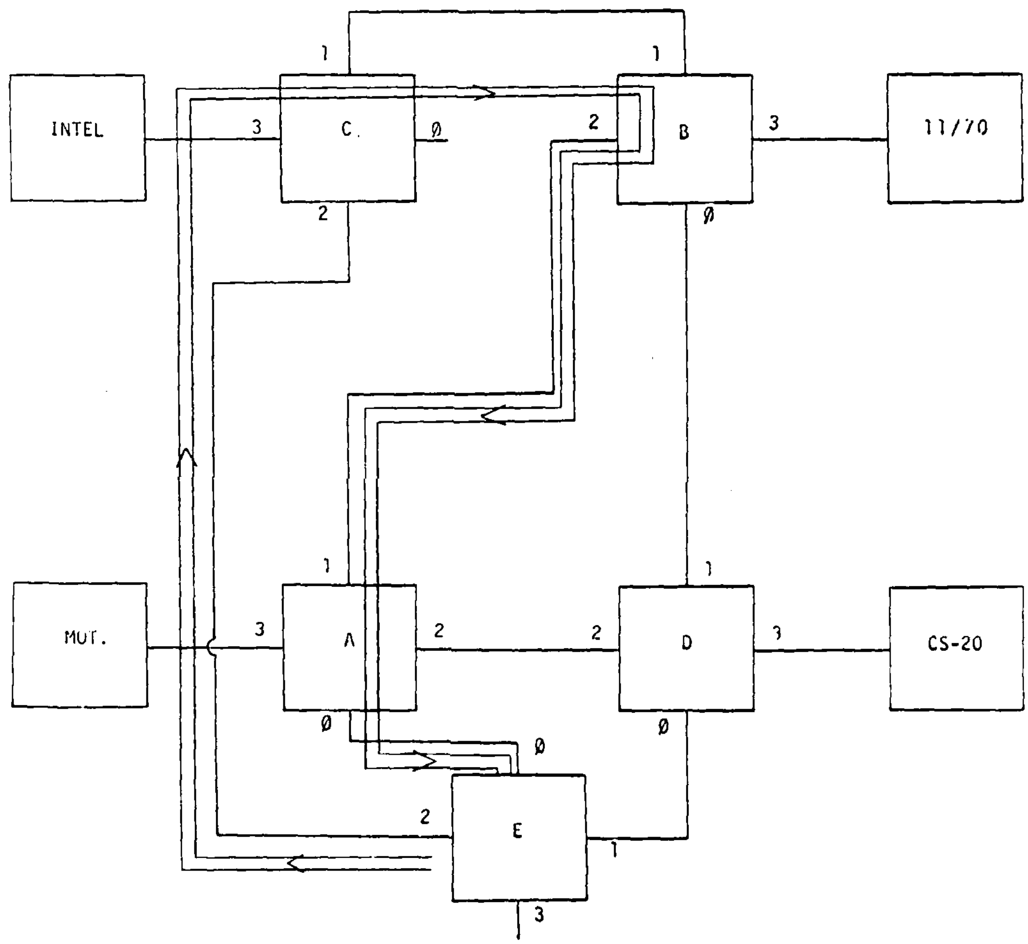
NOVA 820

TRAFFIC ROUTE L



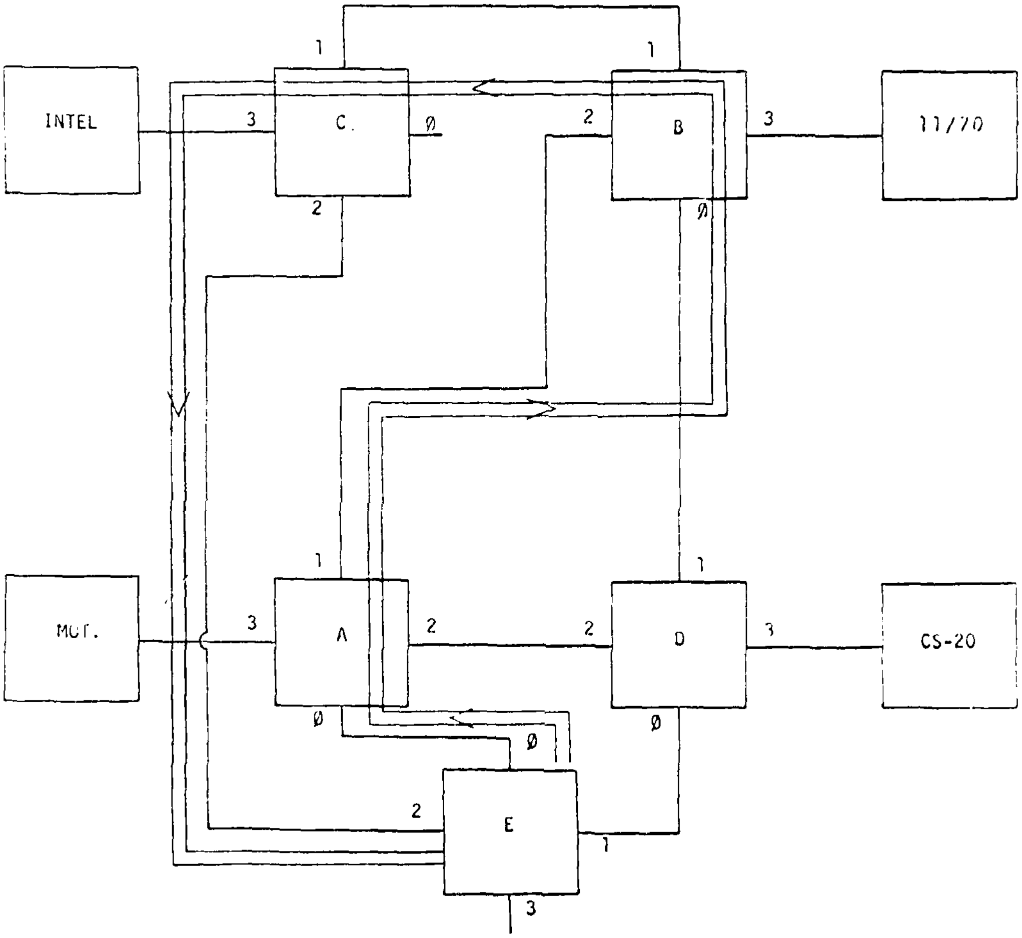
NOVA 820

TRAFFIC ROUTE M



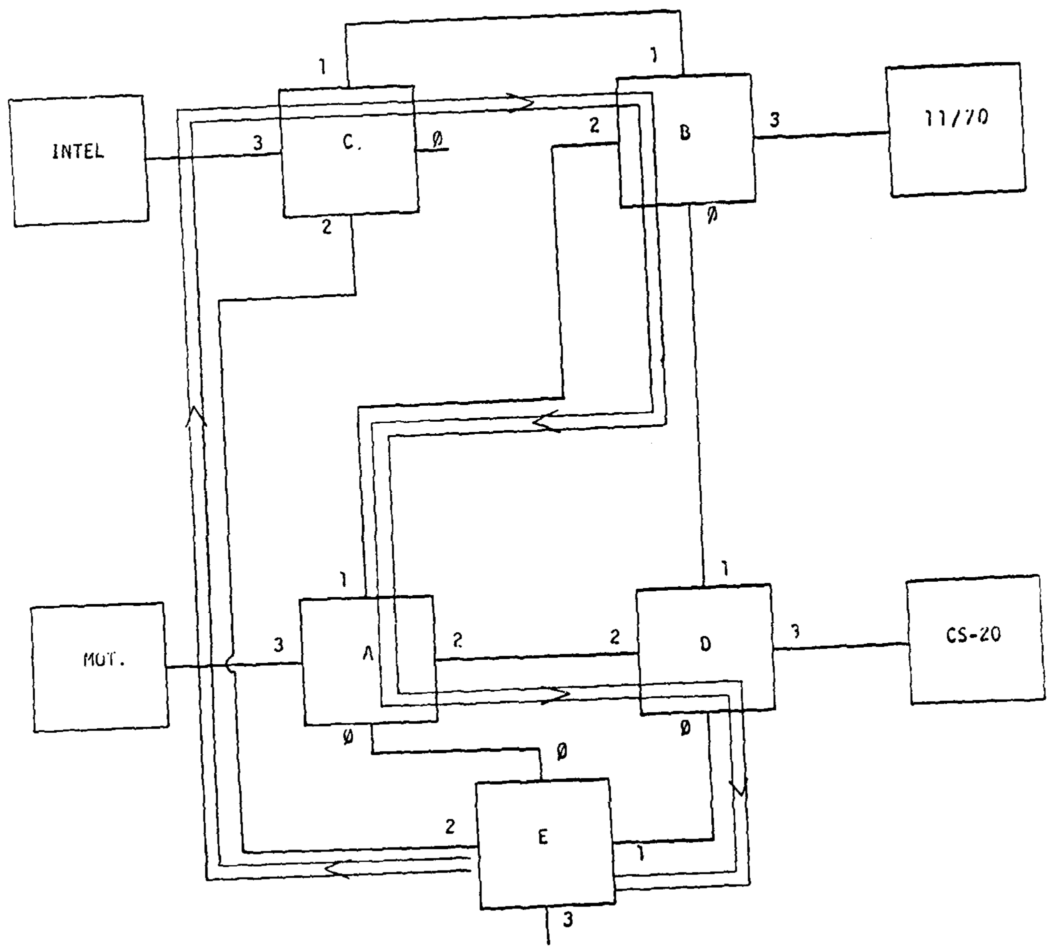
NOVA 820

TRAFFIC ROUTE N



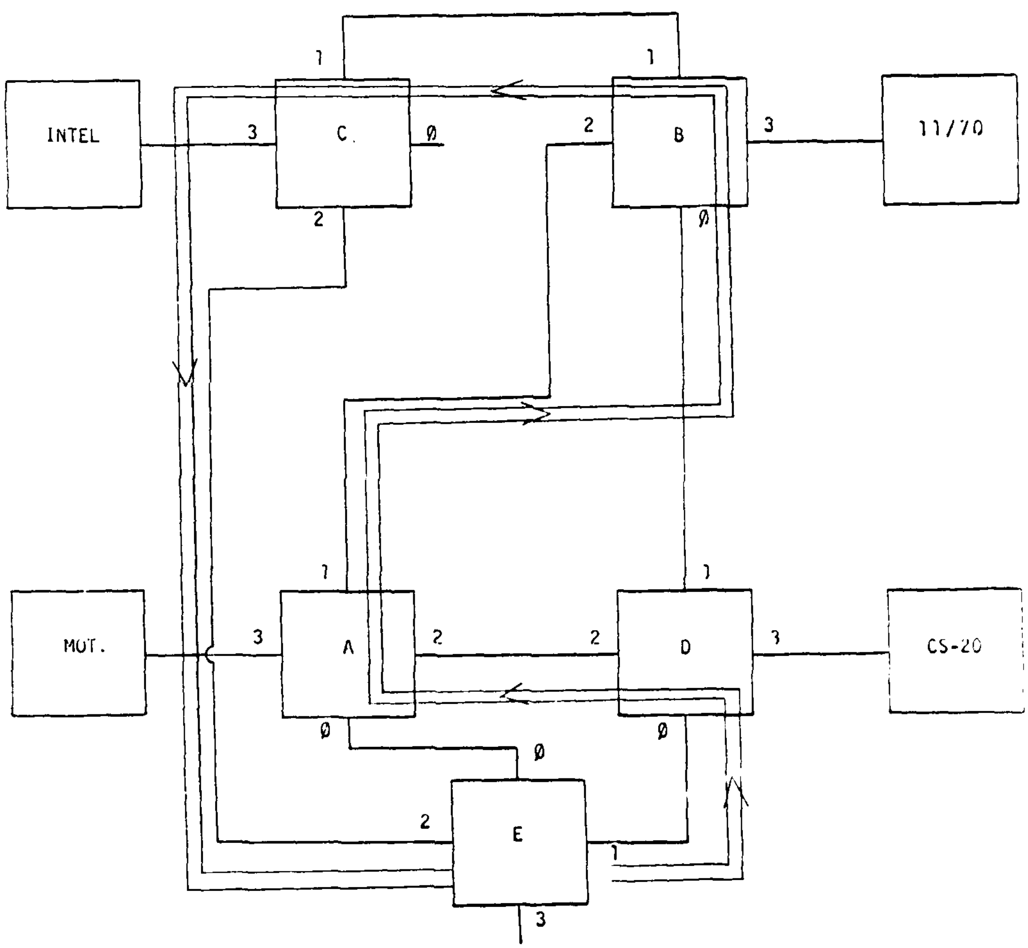
NOVA 820

TRAFFIC ROUTE 0



NOVA 820

TRAFFIC ROUTE P



NOVA 820

TRAFFIC ROUTE Q

11. APPENDIX C

11.1 Introduction

This appendix attempts to explain as clearly as possible the communications control software. It assumes little knowledge of M6800 microprocessor code, but does assume a knowledge of the general configuration of the network. It is presented in five parts: 1) Explanation of the different types of messages, 2) short description of the method of error detecting being used, 3) the general message handling process, 4) a description of important buffers that the program uses, and 5) a flowchart that shows much of the program detail.

11.2 Messages

Essentially there are three types of messages: 1) Data, 2) Source Acknowledgement, and 3) Local Acknowledgement. The first is of prime importance to the system, and the second two nsure the error free transmission of the first.

11.2.1 Data Message: Any communication between two elements in the network is done with a data message. After a data message is sent out from its origin, it remains stored in the origin's RAM until it has been received at its destination. This is when a source acknowledgement should be received by the origin from the destination to indicate safe arrival of the data message.

11.2.2 Source Acknowledgement: The source acknowledgement, as just mentioned, is to indicate to the original sender that a message has reached its destination error free. After receiving this acknowledgement, the message stored in the source's buffers can be cleared.

11.2.3 Local Acknowledgement: The local acknowledgement is one of the steps along the way to an eventual source acknowledgement. If an element in a network is part of the path of a message from the source to the destination, it must receive the message, report to who sent it that the transmission was error free, and send this message back out along its way. This is done through the local acknowledgement. If this local acknowledgement is not received by the sender after a certain period of time, the data message is retransmitted. If the local acknowledgement is received properly, the message can be cleared from the buffers of the intermediate handler. See Figure C.1.

11.3 Message Handling

The way in which a message is processed will be described in an attempt to become more detailed in the discussion of the total system. The reader is referred to Flowchart C.1.

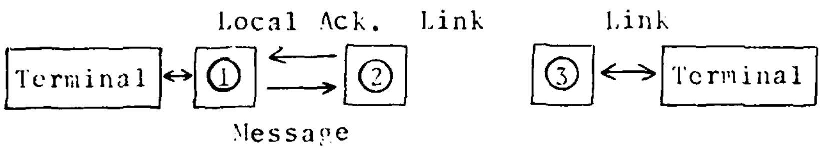
Essentially, the steps are as follows:

1. A message is put in through an interrupt routine that will input one word at a time. This is done to take advantage of the relative speed that the central processing unit possesses compared to the speed of serial data transmission. This interrupt I10 scheme will be discussed in more detail later.

FIGURE C.1

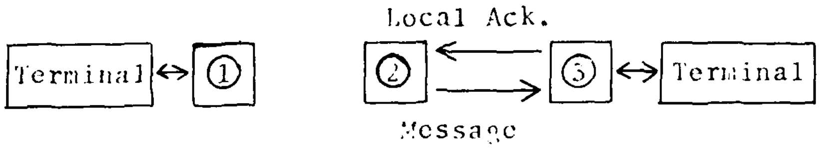
An example of data message transmission with acknowledgements:

(1)



Message is transmitted to first link of its path . Local Ack. is sent by (2) to (1), safe arrival. (1) still waits for a source ack.

(2)

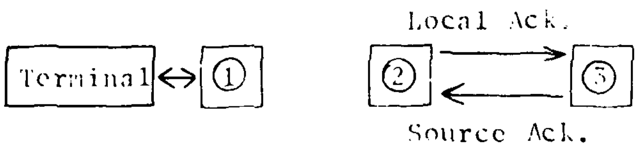


Message transmitted to (3)

(2) Clears message from its buffers after local ack. is received.

(1) Still waiting for source ack.

(3)

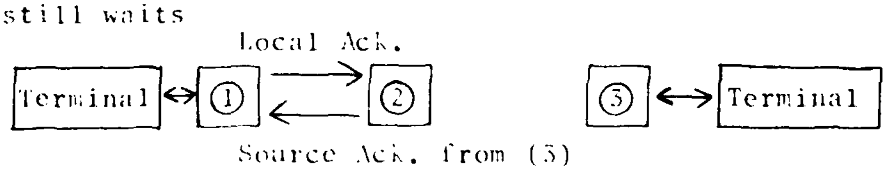


(3) Transmits source ack. for message

(2) Sends (3) a local ack. for the source ack, (3) clears the source ack.

(1) still waits

(4)



2. The message is classed as either a data message, local acknowledgement, or as a source acknowledgement

If the message is a data message, it must be distinguished between a message that has reached its destination and one that needs to be put back out into the network. If the data message is still in the network, a local acknowledgement must be sent to the last node that held the message so that it can clear its buffer, and then the message must be put back into the network to continue towards its destination.

If the message needs to go back out into the system, it is sent to an ACIA for output. If the ACIA is busy, this being the proper ACIA, the message is sent to an ACIA that is linked to the proper one for output. If output is not possible after all the links are tried, the message is to be placed in a queue for output at a more convenient time. (See Figure C.2)

If the data message is at its destination, a local acknowledgement must be sent to the last node that held the message so that it can be cleared from the buffer, and a source acknowledgement must be sent to the message's source to acknowledge the completion of the transfer of information.

If the message is a local acknowledgement, the receiving node knows that the message was received error-free and that its buffer can be cleared.

If the message is a source acknowledgement, the origination node knows that the message was received error-free and that its buffer

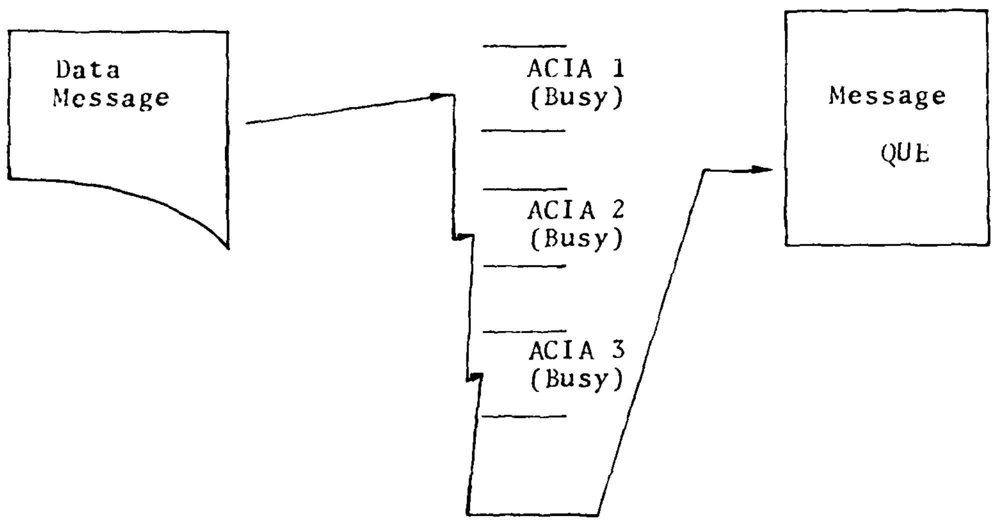


FIGURE C.2: ALL ACIA's BUSY, MESSAGE IS QUEUED can also be cleared.

3. Data Transmission Complete.

11.4 Input/Output

The input and output routines are called in the interrupt portion of the program. A message is outputted one word at a time. After each word is sent to an ACIA for output the program continues to perform its normal process of processing message until output of word is completed, at that time another word is sent to the ACIA and program excecutions resumes again like normal.

In the input case, the inputting ACIA will interrupt normal program flow to input to a buffer one word after completely receiving the word. After each input normal program execution can continue.

Reintrant RAM is used primarily to achieve input and output to the proper buffers in this interrupt scheme. (See next section.)

Note: The input/output flowchart will help this description greatly.

11.5 Headers for the Three Message Types

At the beginning of each message is a header telling the receiver how to treat the message.

11.5.1 Data Message:

00 Message Class
01 Number of Buffers
02 Number of Words in Last Buffer
03 Origin
04 Destination
05 Message Number

06 Sequence Number
07 Local Sequence Number

11.5.2 Source Acknowledgement:

00 Class
01 Destination
02 Message Number
03 Local Sequence Number

FIGURE C.3

11.5.3 Local Acknowledgement:

00 Class

Note: The Local Acknowledgement message only contains this header and a local sequence number.

11.6 Definitions

Message Class - Each message can be classed according to the type of information in its bits. The three types of message classes are: 1) source acknowledgements, 2) local acknowledgements, and 3) data messages.

Number of Buffers - This is the number of buffers the message is sent in. Maximum buffer length is 255 words including the Cyclic Redundancy Code (CRC) and the header.

Number of Words in the Last Buffer - This provides a means of quickly finding the CRC which is located in the last two words of the message. (See Error Detecting, Section 11.7).

Origin - The origin is where the message orginated. It tells which ACIA should get a source acknowledgement. Each element of the network is assigned a number.

<u>Destination</u> - This is where the message is going.

<u>Message Number</u> - This is the name of the first buffer where a message is stored in the source. Used for source acknowledgement purposes.

<u>Sequence Number</u> - The sequence number is the packet number of the message. Presently, a message may be three packets long. Sequence number is used because of the necessity to receive packets in order.

<u>Local Sequence Number</u> - This is the name of the first buffer where a source acknowledgement or a data message is stored in a link. See Figure C.3.

11.7 Error Detecting

The end of each message contains the CRC. The primary concern of the network is data routing and transmission, but accuracy is also a major concern. The CRC is simply a check-sum of all the words contained in the message.

A CRC is on the message, but is also used for comparison when a message is input. Unfavorable comparison results in the message being discarded and retransmitted.

The MC6850 also performs an error check of each word input. It checks for framing errors, receiver overrun, and proper parity. The reader is referred to the MC6850 SPEC sheet.

11.8 Buffer Definitions

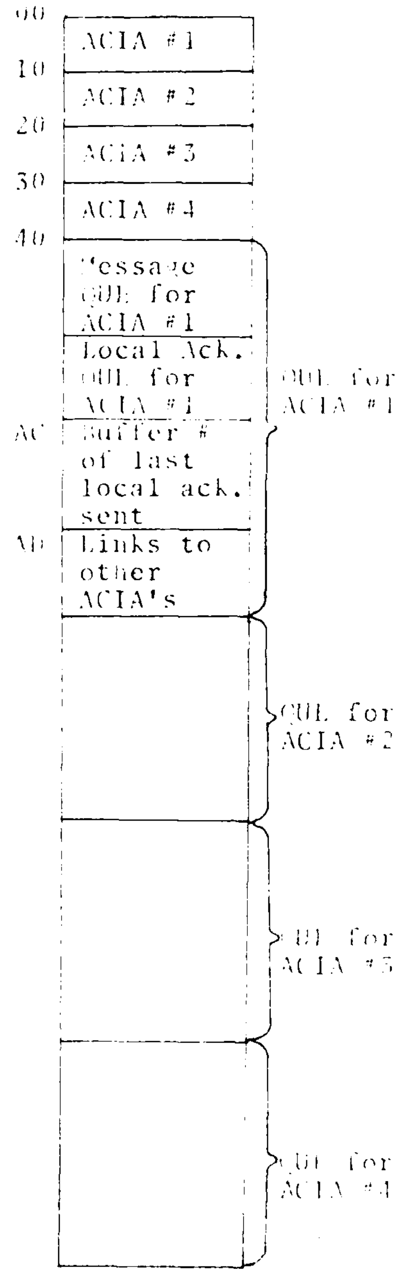
Certain data arrays are used for the temporary storage of information needed to process messages. These so called "Buffers" are explained here.

REAC BUFFER - Reintrant RAM

The upper portion of reintrant RAM consists of space for storing data to input and output a message. See figure.

A typical buffer looks like this:

00	First Input Buffer	Input Section
01	Number of Input Buffers	
02	Present Input Buffer	
03	Location in Present Buffer	
04	I-Input	
05	High Order CRC	Calculated
06	Low Order CRC	
07	ACIA High Order Address	
08	ACIA Low Order Address	
09	Number of Output Buffers Left	Input Section
0A	Number of Words in Last Buffer	
0B	Present Buffer	
0C	Location in Present Buffer	
0D	F0Output	



Each of the ACIA's have a buffer exactly like this. This buffer is used by the input and output interrupt routines for storing a message or receiving a message from data storage buffers.

The next portion of reintrant RAM is the QUE, [illegible] it for local acknowledgements or for data messages. (See Figure) Each ACIA also has associated with it a QUE. Described as it is in the figure. Locations 40-45 are the message QUE. Two messages can be placed in the QUE for a particular ACIA. Locations 46-49 are for local acknowledgements. They contain all that is needed for the sending of a local acknowledgement, the location of the message in the sender, buffers so it can be cleared from there. (See local acknowledgement scetion) Location 4C, the buffer number of the last local acknowledgement sent is used so that this local ack. message may be cleared. Location 4D contains a constant; when operated on will reveal the linked ACIA's that provide an alternate route for sending a message.

40	Number of First Buffer
41	Number of Buffers
42	Number of Words in Last Buffer
43	# of First Buffer
44	# of Buffers
45	# of Words in Last Buffer
46	Local Sequence # 1st Ack.
47	Local Sequence # 2nd Ack.
48	Local Sequence # 3rd Ack.
49	Local Sequence # 4th Ack.
4A	
4B	
4C	Buffer # of Last Ack. Sent
4D	Links to Other ACIA's

BEGSTR BUFFER

The input section of reintrant RAM will be transferred to a Begstr after the message has been totally received. Also when the output of a message is started. Begstr is transferred to the output of reintrant RAM. The purpose of Begstr is to provide a means of finding a message when the time has come for it to be processed. The processing status of a message can be any of the following:

00	Location of the First Buffer
01	Number of Buffers
02	Number of Words in Last Buffer
03	Processing Status
04	High Order CRC Calculated
05	Low Order CRC

(0) Message Not Processed
(1) Processed But No Local Ack.
(2) Home Message Not Processed
(3) Home Message Processed No Local Ack.
(4) Home Message Processed But No Source Ack.

Each Message processed has a Begstr associated with it.

BEGBUF BUFFER

Begbuf contains an array of buffer names and the ACIA that is presently using them. A general description of Begbuf is presented in Figure K. The acia using buffer word is called the "key" word.

BEGBUFF

Buffer Name	
F∅	ACIA using F∅
F1	ACIA using F1
F2	ACIA using F2
FF	ACIA using FF

FIGURE K

XDIREC BUFFER

The directory is an array used to determine the proper ACIA for the output of a message. The low order address for the directors pointer, points to an element that has in it an address of an ACIA. So if the destination is known, the proper acia can be gotten by placing this destination in the low order address of the directors pointer, XDIREC.

ZZ BUFFER

ZZ is the acknowledgement needed QUE. When a message is outputted, its first buffer number is placed in this QUE so when an acknowledgement comes, a quick location of the message and subsequent incrementing of its processing status is achieved. Also when a message is outputted, the time of output is recorded in ZZ, so if too much time has progressed and a local ack. has not been received, the message can be retransmitted.

HOMEREC BUFFER

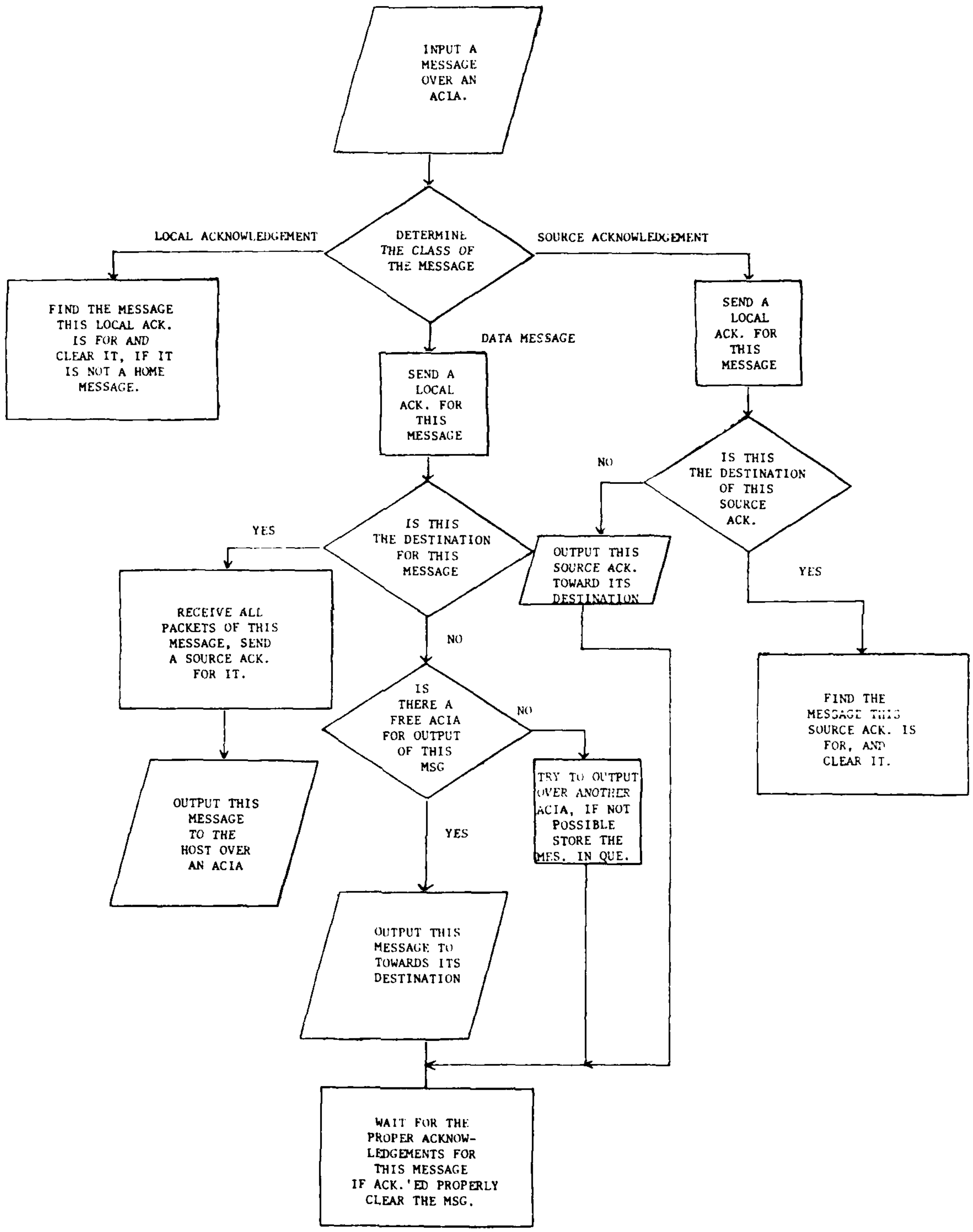
Homerec is used for processing home messages received from the network. It keeps track of the incoming packets and where they are stored in RAM. Homerec may be better understood when it is viewed in its proper context in the flow chart.

00	Message Number
01	Origin
02	# of Last Packet Received
03	Location of Packet 1
04	Location of Packet 2
05	MLGSTR Location of 1st Packet
06	Low Order MLGSTR Address

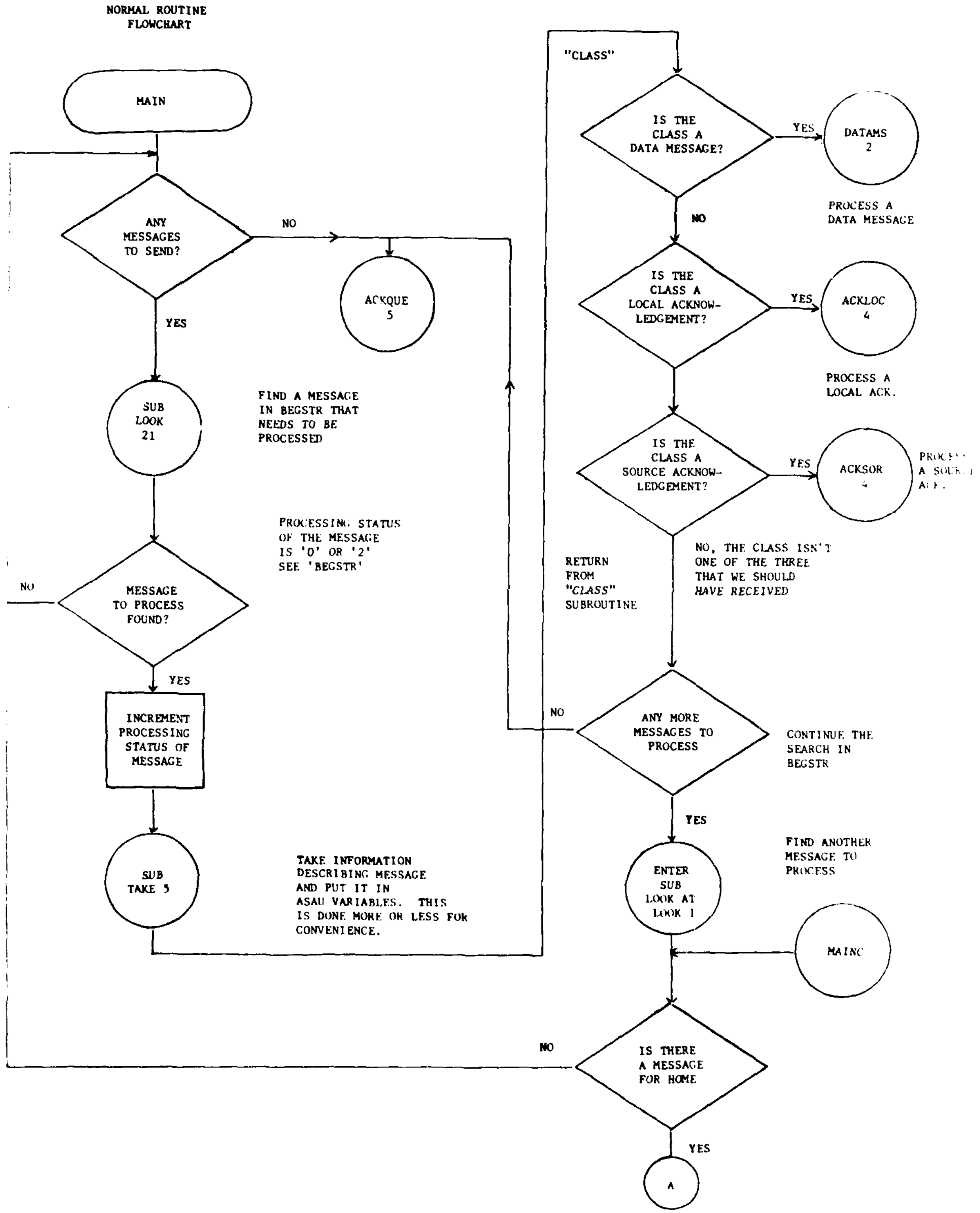
11.9 Flow Charts

The flow diagram that makes up the rest of this paper is intended to tell, in real words, the process that is taking place. For this reason, it may not be word for word corresponding to the program listing.

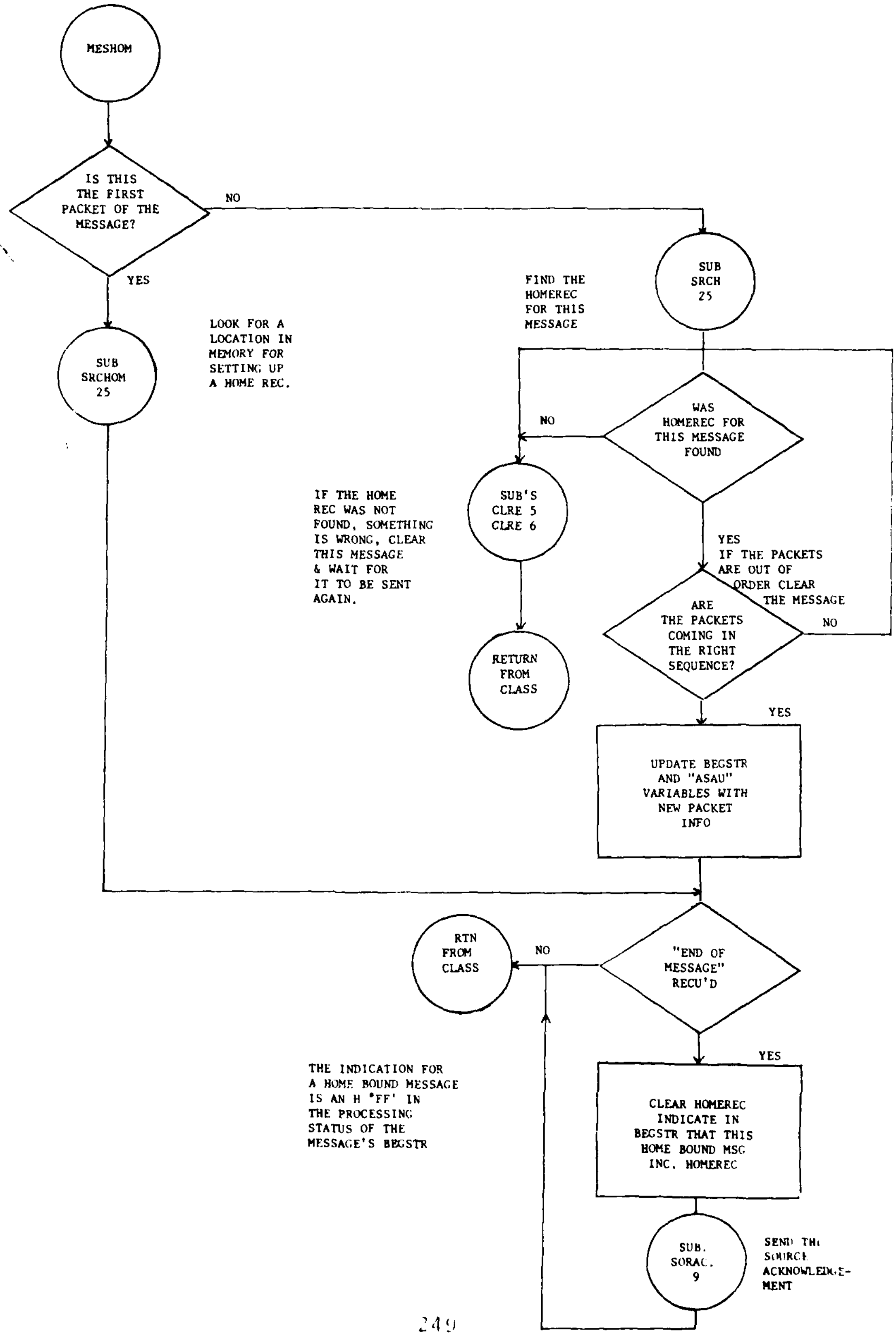
Also, one level deeper in detail could have been done.

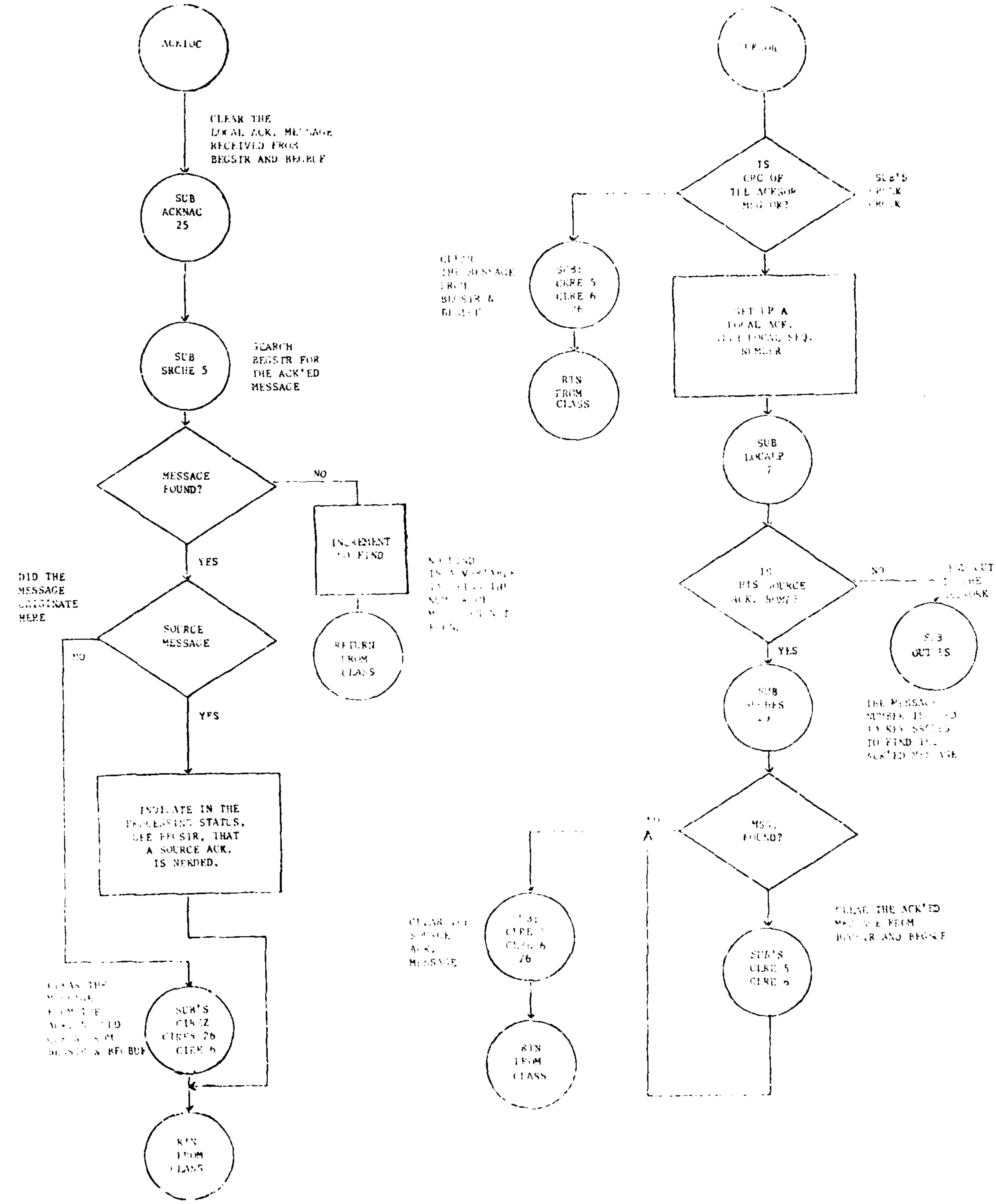


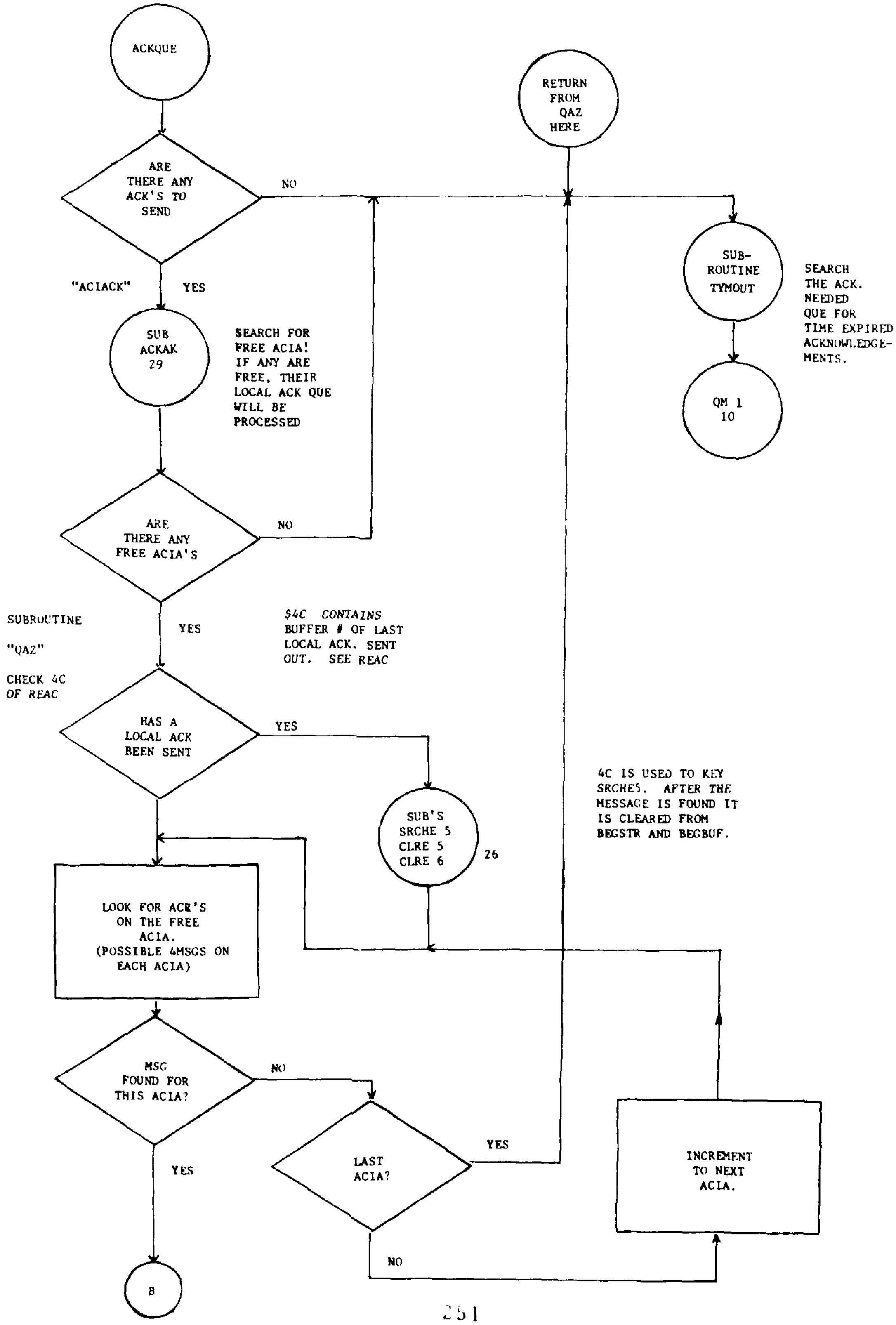
FLOWCHART C.1 MESSAGE HANDLING



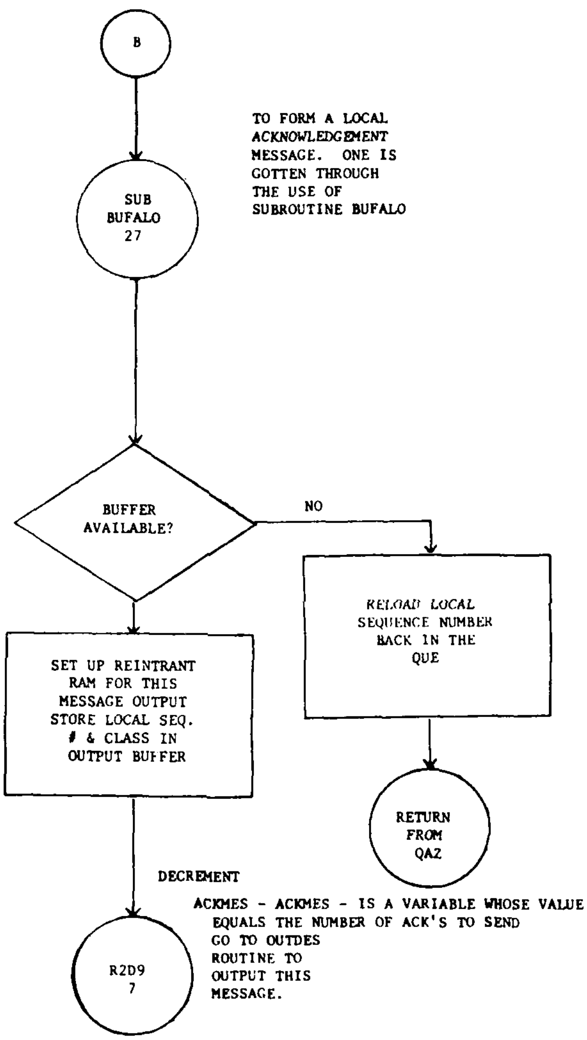
NORMAL ROUTINE FLOWCHART
MAIN
ANY MESSAGES TO SEND?
NO
YES
ACKQUE 5
SUB LOOK 21
FIND A MESSAGE IN BEGSTR THAT NEEDS TO BE PROCESSED
PROCESSING STATUS OF THE MESSAGE IS '0' OR '2' SEE 'BEGSTR'
MESSAGE TO PROCESS FOUND?
NO
YES
INCREMENT PROCESSING STATUS OF MESSAGE
SUB TAKE 5
TAKE INFORMATION DESCRIBING MESSAGE AND PUT IT IN ASAU VARIABLES. THIS IS DONE MORE OR LESS FOR CONVENIENCE.
"CLASS"
IS THE CLASS A DATA MESSAGE?
YES
DATAMS 2
PROCESS A DATA MESSAGE
NO
IS THE CLASS A LOCAL ACKNOWLEDGEMENT?
YES
ACKLOC 4
PROCESS A LOCAL ACK.
IS THE CLASS A SOURCE ACKNOWLEDGEMENT?
YES
ACKSOR 4
NO, THE CLASS ISN'T ONE OF THE THREE THAT WE SHOULD HAVE RECEIVED
RETURN FROM "CLASS" SUBROUTINE
ANY MORE MESSAGES TO PROCESS
NO
CONTINUE THE SEARCH IN BEGSTR
YES
ENTER SUB LOOK AT LOOK 1
FIND ANOTHER MESSAGE TO PROCESS
MAINC
IS THERE A MESSAGE FOR HOME
NO
YES
A

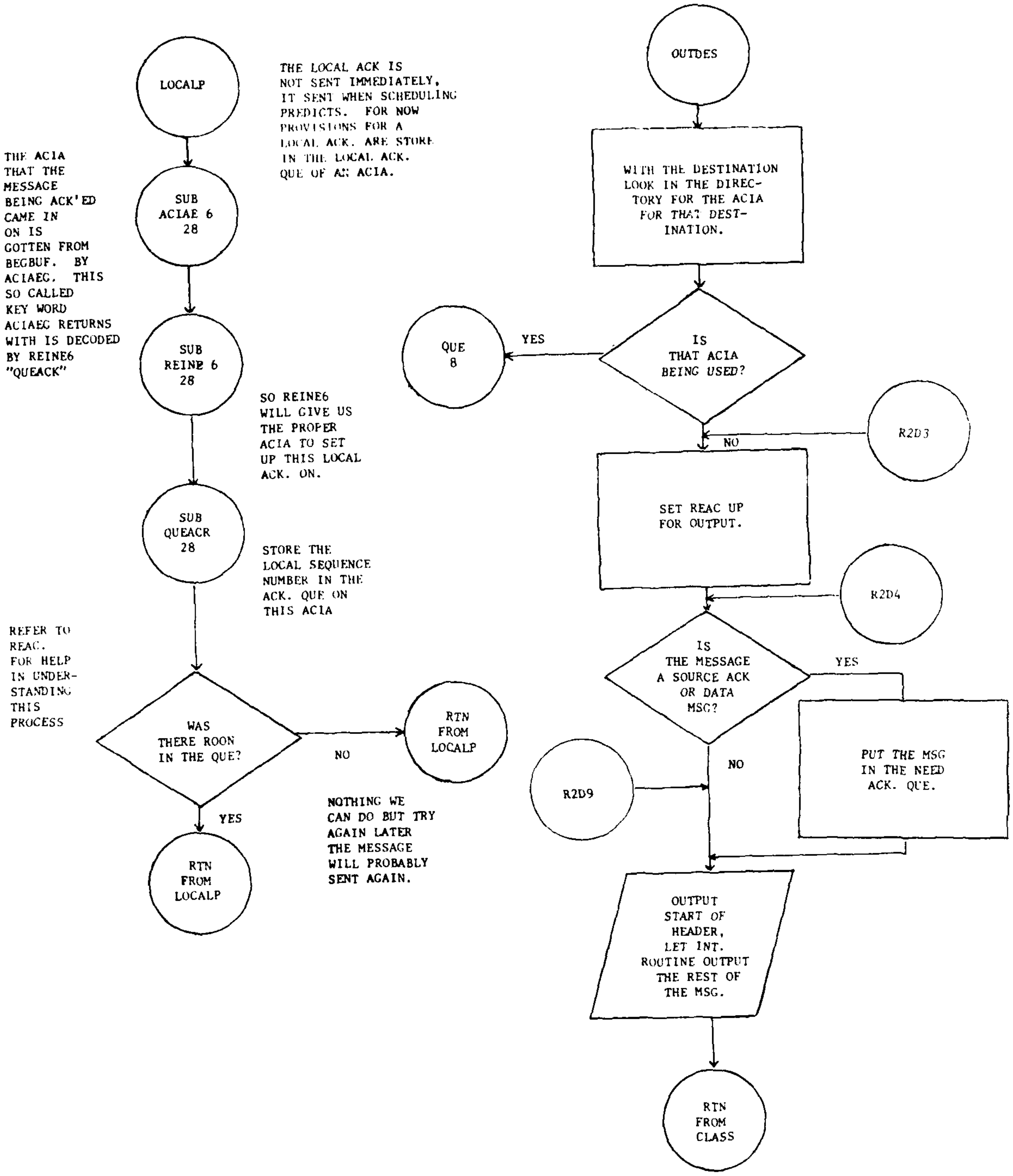

MESHOM
IS THIS THE FIRST PACKET OF THE MESSAGE?
NO
YES
SUB SRCHOM 25
LOOK FOR A LOCATION IN MEMORY FOR SETTING UP A HOME REC.
SUB SRCH 25
FIND THE HOMEREC FOR THIS MESSAGE
WAS HOMEREC FOR THIS MESSAGE FOUND
NO
SUB'S CLRE 5 CLRE 6
IF THE HOME REC WAS NOT FOUND, SOMETHING IS WRONG, CLEAR THIS MESSAGE & WAIT FOR IT TO BE SENT AGAIN.
RETURN FROM CLASS
YES
IF THE PACKETS ARE OUT OF ORDER CLEAR THE MESSAGE
ARE THE PACKETS COMING IN THE RIGHT SEQUENCE?
NO
YES
UPDATE BEGSTR AND "ASAU" VARIABLES WITH NEW PACKET INFO
"END OF MESSAGE" RECU'D
NO
RTN FROM CLASS
YES
CLEAR HOMEREC INDICATE IN BEGSTR THAT THIS HOME BOUND MSG INC. HOMEREC
THE INDICATION FOR A HOME BOUND MESSAGE IS AN H 'FF' IN THE PROCESSING STATUS OF THE MESSAGE'S BEGSTR
SUB. SORAC. 9
SEND THE SOURCE ACKNOWLEDGE-MENT


CLEAR THE
SUB
ACKNAC
25
SEARCH
BEGSTR FOR
THE ACK'ED
MESSAGE
SUB
SRCHE 5
MESSAGE
FOUND?
NO
YES
INCREMENT
DID THE
MESSAGE
HERE
SOURCE
MESSAGE
YES
INDICATE IN THE
STATUS,
THAT
A SOURCE ACK.
IS NEEDED.
SUB'S
RTN
FROM
IS
CRC OF
SUB
SET UP A
LOCAL ACK.
RTN
FROM
CLASS
SUB
NO
YES
SUB
FOUND?
CLEAR THE ACK'ED
SUB'S
CLRE 5
CLRE 6
RTN
FROM
CLASS

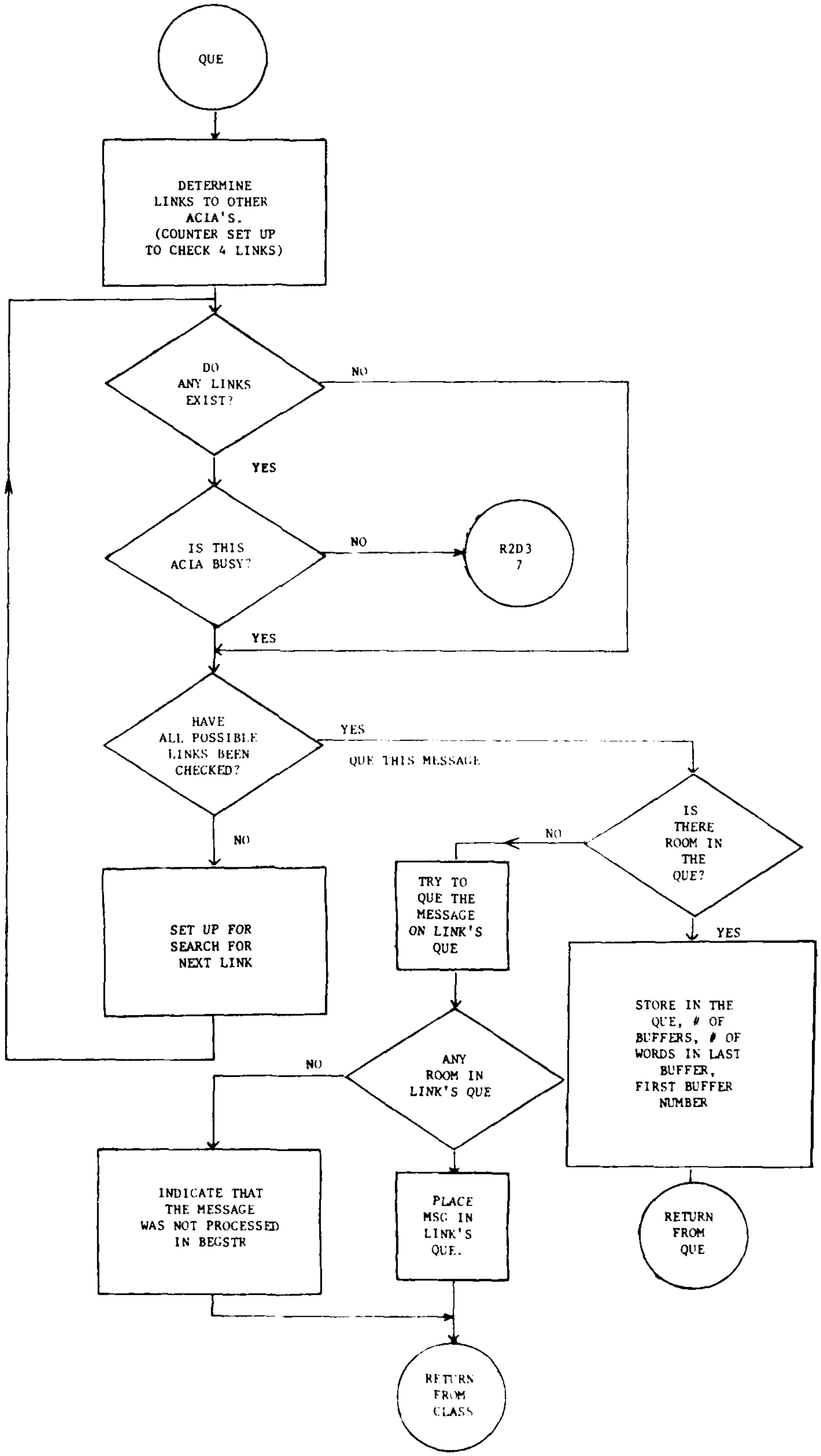


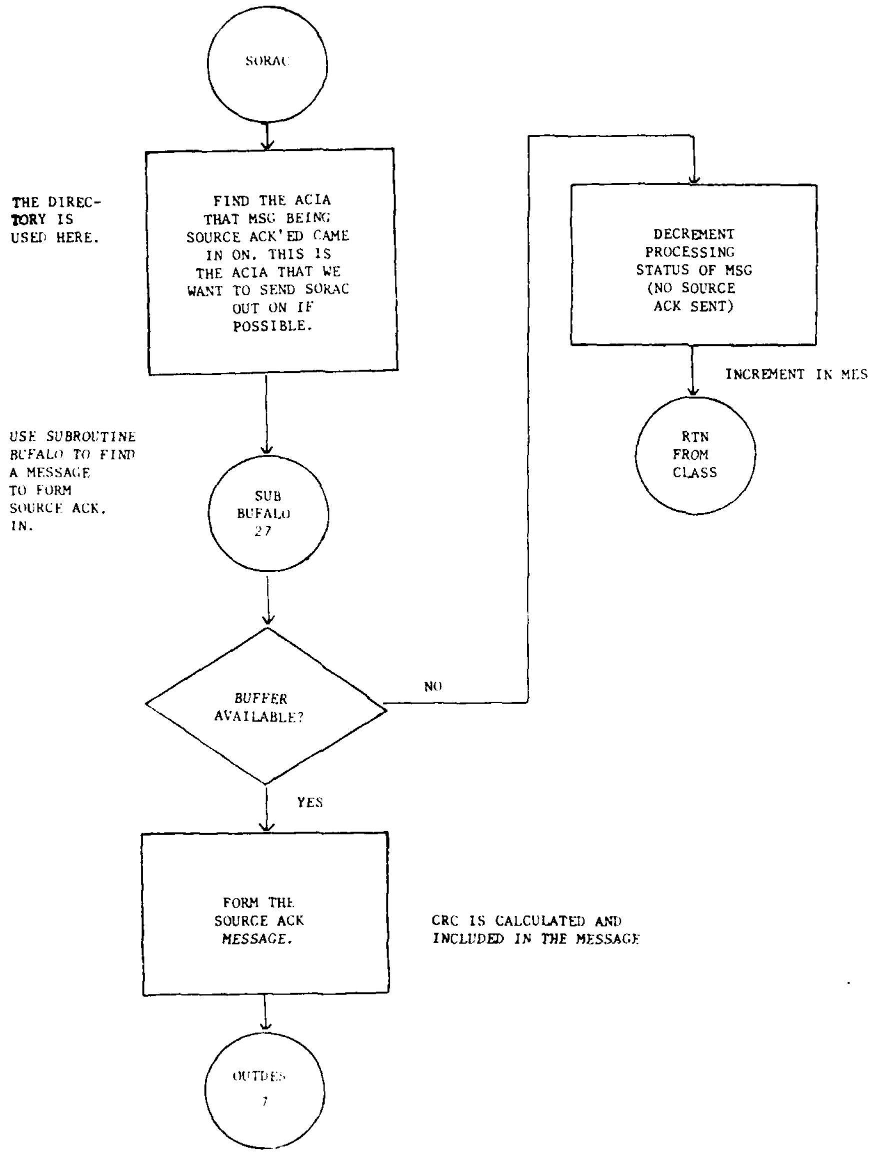
ACKQUE
RETURN FROM QAZ HERE
ARE THERE ANY ACK'S TO SEND
NO
"ACIACK"
YES
SUB-ROUTINE TYMOUT
SEARCH THE ACK. NEEDED QUE FOR TIME EXPIRED ACKNOWLEDGE-MENTS.
SUB ACKAK 29
SEARCH FOR FREE ACIA! IF ANY ARE FREE, THEIR LOCAL ACK QUE WILL BE PROCESSED
QM 1 10
ARE THERE ANY FREE ACIA'S
NO
SUBROUTINE
"QAZ"
CHECK 4C OF REAC
YES
$4C CONTAINS BUFFER # OF LAST LOCAL ACK. SENT OUT. SEE REAC
HAS A LOCAL ACK BEEN SENT
YES
4C IS USED TO KEY SRCHE5. AFTER THE MESSAGE IS FOUND IT IS CLEARED FROM BEGSTR AND BEGBUF.
SUB'S SRCHE 5 CLRE 5 CLRE 6
26
LOOK FOR ACK'S ON THE FREE ACIA. (POSSIBLE 4MSGS ON EACH ACIA)
MSG FOUND FOR THIS ACIA?
NO
YES
LAST ACIA?
INCREMENT TO NEXT ACIA.
YES
NO
B

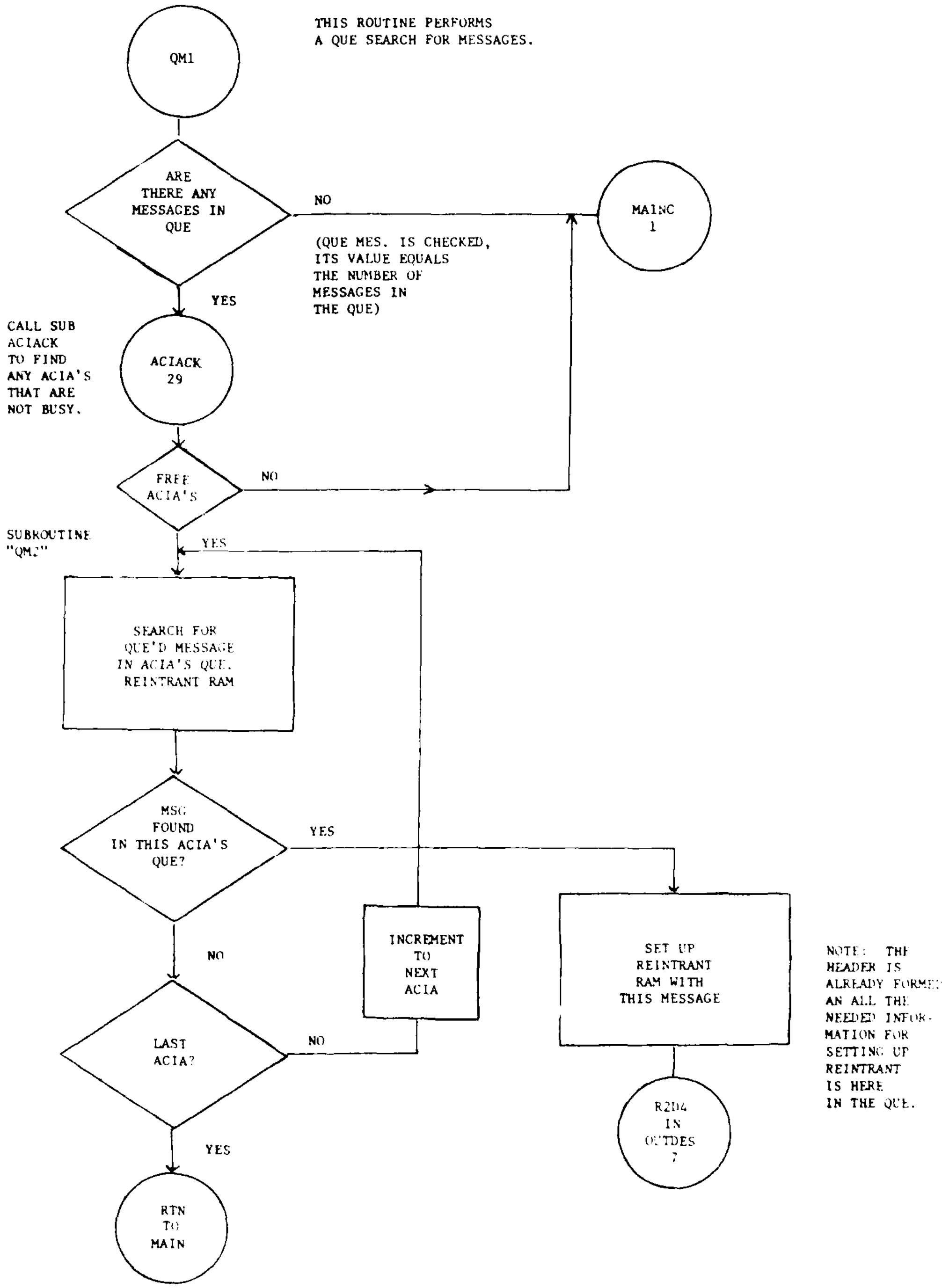

B
SUB
BUFALO
27
TO FORM A LOCAL
ACKNOWLEDGEMENT
MESSAGE. ONE IS
GOTTEN THROUGH
THE USE OF
SUBROUTINE BUFALO
BUFFER
AVAILABLE?
NO
RELOAD LOCAL
SEQUENCE NUMBER
BACK IN THE
QUE
RETURN
FROM
QAZ
SET UP REINTRANT
RAM FOR THIS
MESSAGE OUTPUT
STORE LOCAL SEQ.
& CLASS IN
OUTPUT BUFFER
DECREMENT
ACKMES - ACKMES - IS A VARIABLE WHOSE VALUE
EQUALS THE NUMBER OF ACK'S TO SEND
GO TO OUTDES
ROUTINE TO
OUTPUT THIS
MESSAGE.
R2D9
7

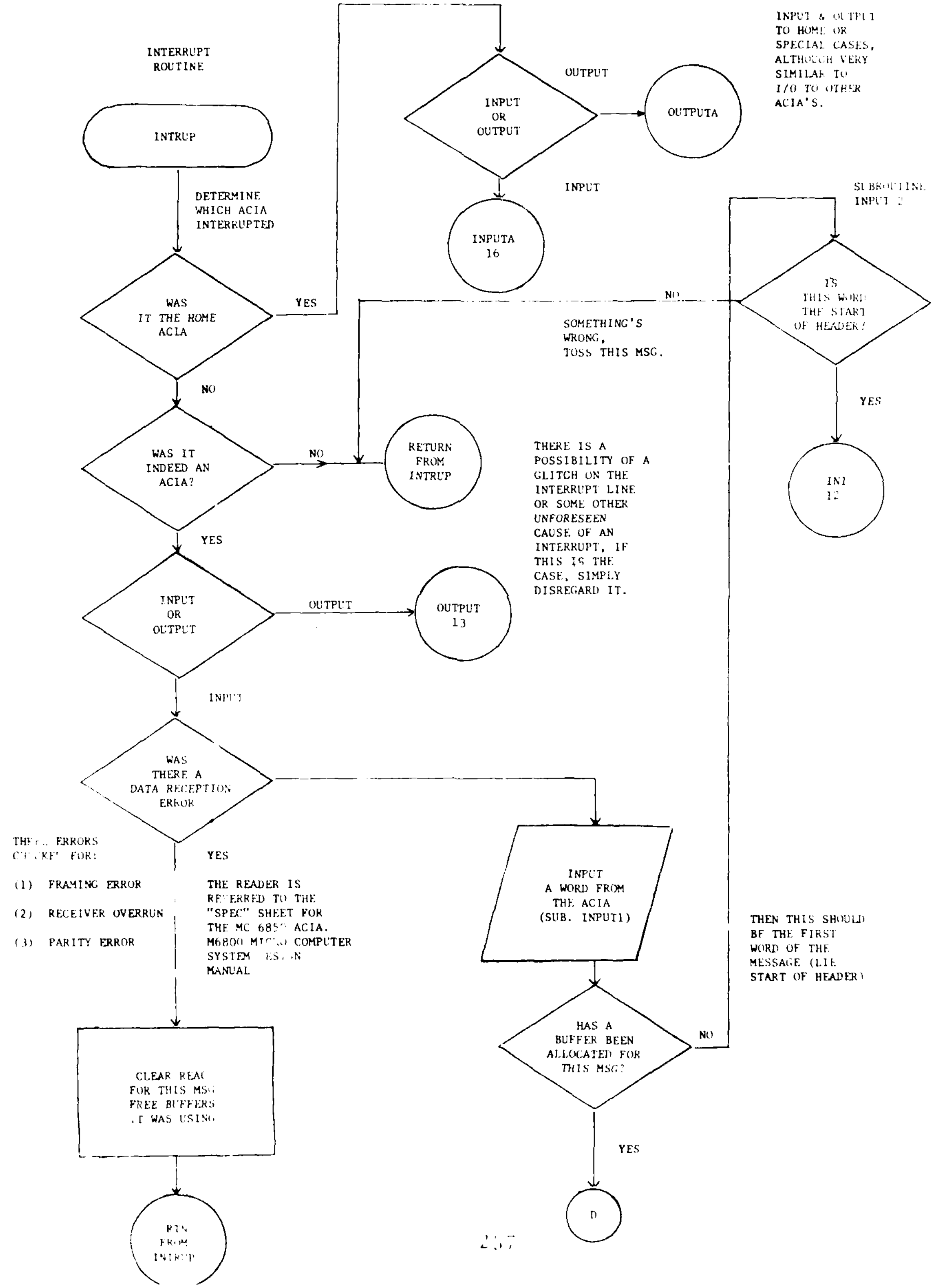


LOCALP
THE LOCAL ACK IS NOT SENT IMMEDIATELY, IT SENT WHEN SCHEDULING PREDICTS. FOR NOW PROVISIONS FOR A LOCAL ACK. ARE STORE IN THE LOCAL ACK. QUE OF AN ACIA.
THE ACIA THAT THE MESSAGE BEING ACK'ED CAME IN ON IS GOTTEN FROM BEGBUF. BY ACIAEG. THIS SO CALLED KEY WORD ACIAEG RETURNS WITH IS DECODED BY REINE6 "QUEACK"
SUB ACIAE 6 28
SUB REINE 6 28
SO REINE6 WILL GIVE US THE PROPER ACIA TO SET UP THIS LOCAL ACK. ON.
SUB QUEACR 28
STORE THE LOCAL SEQUENCE NUMBER IN THE ACK. QUE ON THIS ACIA
REFER TO REAC. FOR HELP IN UNDER-STANDING THIS PROCESS
WAS THERE ROON IN THE QUE?
NO
RTN FROM LOCALP
YES
RTN FROM LOCALP
NOTHING WE CAN DO BUT TRY AGAIN LATER THE MESSAGE WILL PROBABLY SENT AGAIN.
OUTDES
WITH THE DESTINATION LOOK IN THE DIREC-TORY FOR THE ACIA FOR THAT DEST-INATION.
IS THAT ACIA BEING USED?
YES
QUE 8
NO
R2D3
SET REAC UP FOR OUTPUT.
R2D4
IS THE MESSAGE A SOURCE ACK OR DATA MSG?
YES
PUT THE MSG IN THE NEED ACK. QUE.
NO
R2D9
OUTPUT START OF HEADER, LET INT. ROUTINE OUTPUT THE REST OF THE MSG.
RTN FROM CLASS

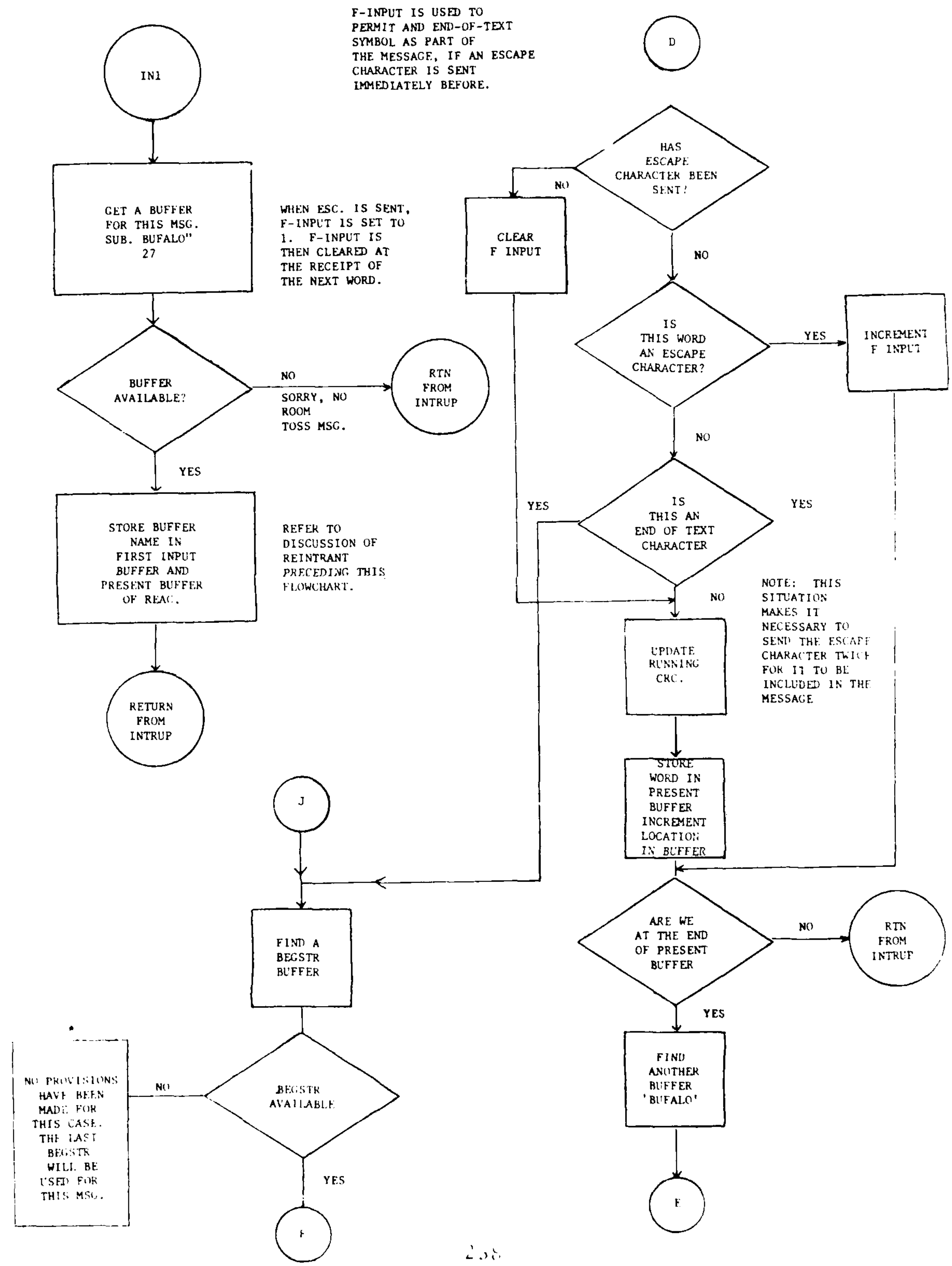

QUE
DETERMINE LINKS TO OTHER ACIA'S. (COUNTER SET UP TO CHECK 4 LINKS)
DO ANY LINKS EXIST?
NO
YES
IS THIS ACIA BUSY?
NO
R2D3 7
YES
HAVE ALL POSSIBLE LINKS BEEN CHECKED?
YES
QUE THIS MESSAGE
NO
SET UP FOR SEARCH FOR NEXT LINK
IS THERE ROOM IN THE QUE?
NO
TRY TO QUE THE MESSAGE ON LINK'S QUE
YES
STORE IN THE QUE, # OF BUFFERS, # OF WORDS IN LAST BUFFER, FIRST BUFFER NUMBER
ANY ROOM IN LINK'S QUE
NO
INDICATE THAT THE MESSAGE WAS NOT PROCESSED IN BEGSTR
PLACE MSG IN LINK'S QUE.
RETURN FROM QUE
RETURN FROM CLASS

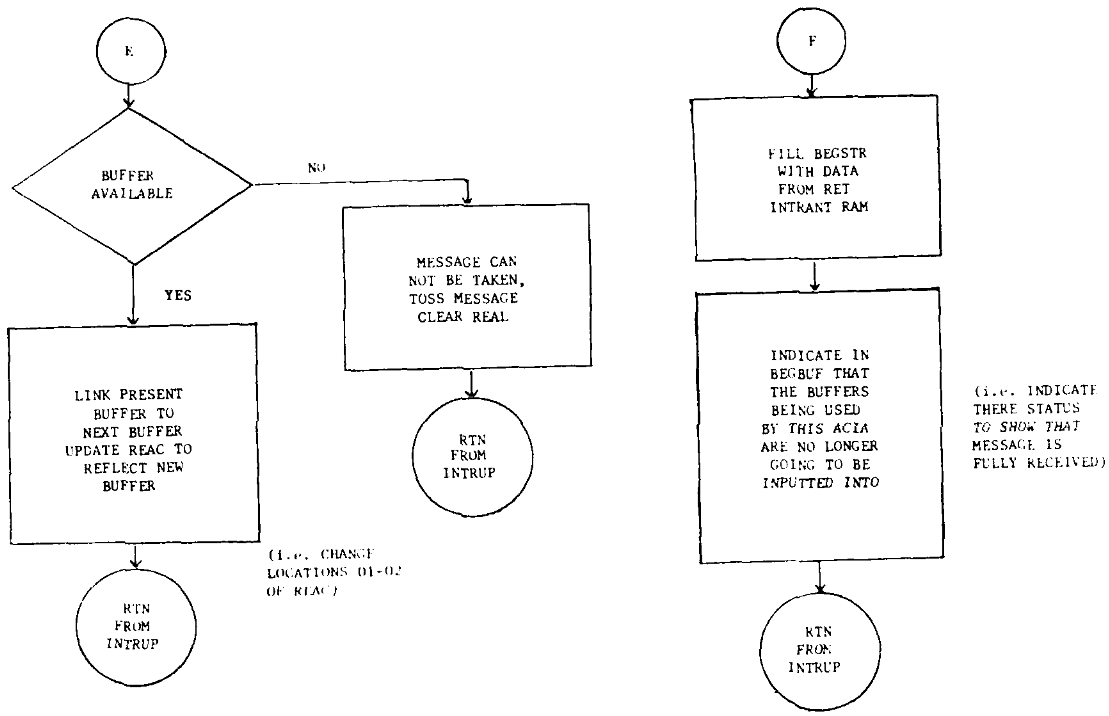

SORAC
THE DIRECTORY IS USED HERE.
FIND THE ACIA THAT MSG BEING SOURCE ACK'ED CAME IN ON. THIS IS THE ACIA THAT WE WANT TO SEND SORAC OUT ON IF POSSIBLE.
DECREMENT PROCESSING STATUS OF MSG (NO SOURCE ACK SENT)
INCREMENT IN MES.
RTN FROM CLASS
USE SUBROUTINE BUFALO TO FIND A MESSAGE TO FORM SOURCE ACK. IN.
SUB BUFALO 27
BUFFER AVAILABLE?
NO
YES
FORM THE SOURCE ACK MESSAGE.
CRC IS CALCULATED AND INCLUDED IN THE MESSAGE
OUTDES 7

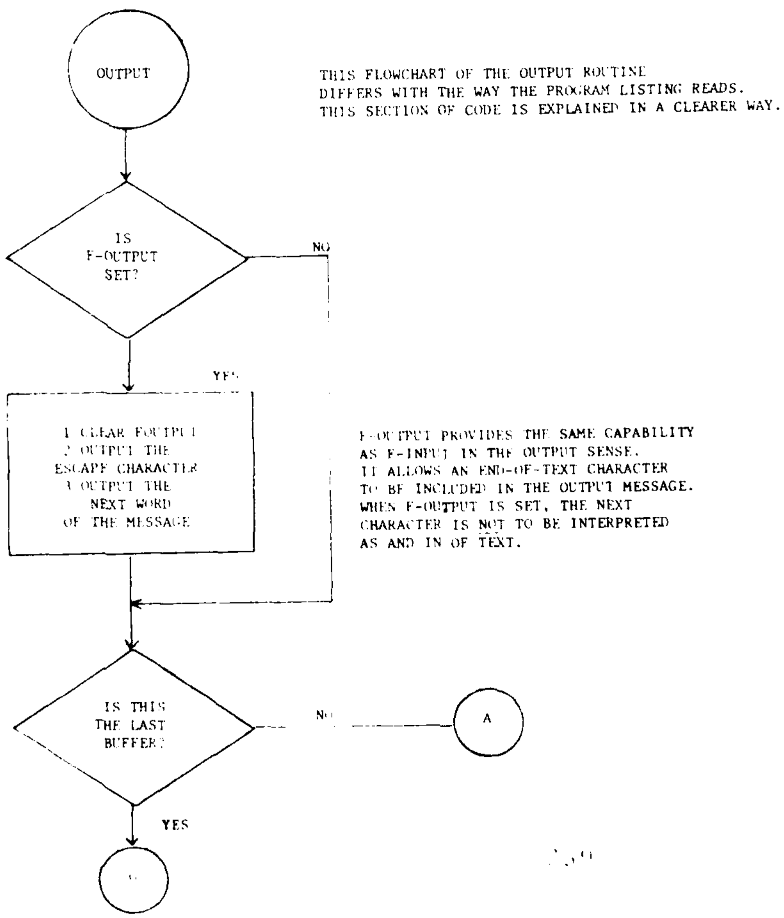

THIS ROUTINE PERFORMS A QUE SEARCH FOR MESSAGES.
QM1
ARE THERE ANY MESSAGES IN QUE
NO
MAINC 1
(QUE MES. IS CHECKED, ITS VALUE EQUALS THE NUMBER OF MESSAGES IN THE QUE)
YES
CALL SUB ACIACK TO FIND ANY ACIA'S THAT ARE NOT BUSY.
ACIACK 29
FREE ACIA'S
NO
SUBROUTINE "QM2"
YES
SEARCH FOR QUE'D MESSAGE IN ACIA'S QUE. REINTRANT RAM
MSG FOUND IN THIS ACIA'S QUE?
YES
INCREMENT TO NEXT ACIA
SET UP REINTRANT RAM WITH THIS MESSAGE
NOTE: THE HEADER IS ALREADY FORMED AN ALL THE NEEDED INFORMATION FOR SETTING UP REINTRANT IS HERE IN THE QUE.
NO
LAST ACIA?
NO
R2D4 IN OUTDES 7
YES
RTN TO MAIN

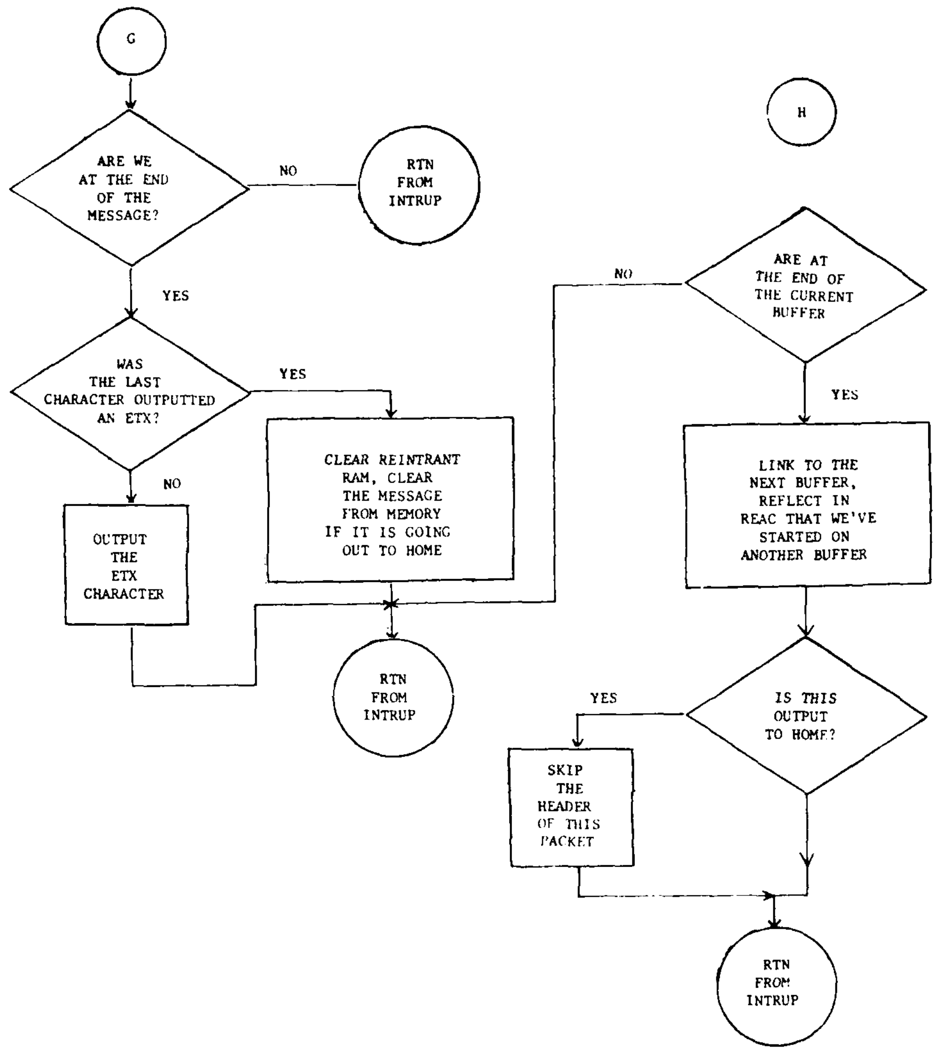


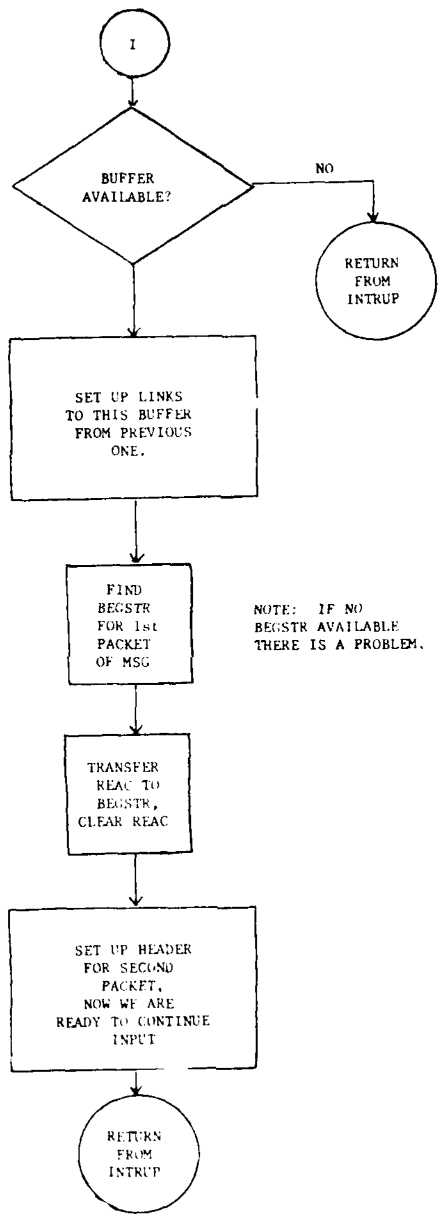
INTERRUPT ROUTINE
INTRUP
DETERMINE WHICH ACIA INTERRUPTED
WAS IT THE HOME ACIA
YES
NO
INPUT OR OUTPUT
OUTPUT
OUTPUTA
INPUT
INPUTA 16
INPUT & OUTPUT TO HOME OR SPECIAL CASES, ALTHOUGH VERY SIMILAR TO I/O TO OTHER ACIA'S.
WAS IT INDEED AN ACIA?
NO
RETURN FROM INTRUP
THERE IS A POSSIBILITY OF A GLITCH ON THE INTERRUPT LINE OR SOME OTHER UNFORESEEN CAUSE OF AN INTERRUPT, IF THIS IS THE CASE, SIMPLY DISREGARD IT.
YES
INPUT OR OUTPUT
OUTPUT
OUTPUT 13
INPUT
WAS THERE A DATA RECEPTION ERROR
YES
(1) FRAMING ERROR
(2) RECEIVER OVERRUN
(3) PARITY ERROR
CLEAR READ FOR THIS MSG FREE BUFFERS IT WAS USING
SUBROUTINE INPUT 2
IS THIS WORD THE START OF HEADER?
NO
SOMETHING'S WRONG, TOSS THIS MSG.
YES
INPUT A WORD FROM THE ACIA (SUB. INPUT1)
HAS A BUFFER BEEN ALLOCATED FOR THIS MSG?
NO
YES
D
THEN THIS SHOULD BE THE FIRST WORD OF THE MESSAGE (I.E. START OF HEADER)

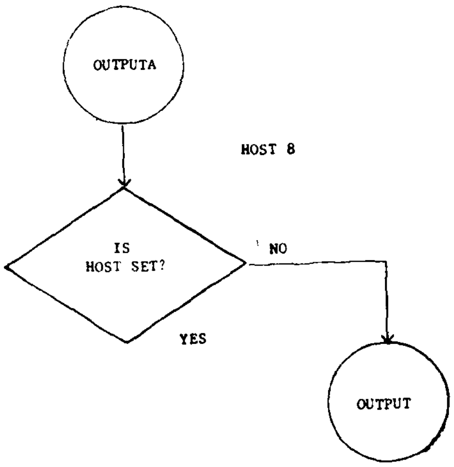

F-INPUT IS USED TO PERMIT AND END-OF-TEXT SYMBOL AS PART OF THE MESSAGE, IF AN ESCAPE CHARACTER IS SENT IMMEDIATELY BEFORE.
IN1
GET A BUFFER FOR THIS MSG. SUB. BUFALO" 27
WHEN ESC. IS SENT, F-INPUT IS SET TO 1. F-INPUT IS THEN CLEARED AT THE RECEIPT OF THE NEXT WORD.
BUFFER AVAILABLE?
NO
SORRY, NO ROOM TOSS MSG.
RTN FROM INTRUP
YES
STORE BUFFER NAME IN FIRST INPUT BUFFER AND PRESENT BUFFER OF REAC.
REFER TO DISCUSSION OF REINTRANT PRECEDING THIS FLOWCHART.
RETURN FROM INTRUP
D
HAS ESCAPE CHARACTER BEEN SENT?
NO
CLEAR F INPUT
NO
IS THIS WORD AN ESCAPE CHARACTER?
YES
INCREMENT F INPUT
NO
IS THIS AN END OF TEXT CHARACTER
YES
YES
NO
NOTE: THIS SITUATION MAKES IT NECESSARY TO SEND THE ESCAPE CHARACTER TWICE FOR IT TO BE INCLUDED IN THE MESSAGE
UPDATE RUNNING CRC.
STORE WORD IN PRESENT BUFFER INCREMENT LOCATION IN BUFFER
ARE WE AT THE END OF PRESENT BUFFER
NO
RTN FROM INTRUP
YES
FIND ANOTHER BUFFER 'BUFALO'
F
J
FIND A BEGSTR BUFFER
BEGSTR AVAILABLE
NO
NO PROVISIONS HAVE BEEN MADE FOR THIS CASE. THE LAST BEGSTR WILL BE USED FOR THIS MSG.
YES
F


E
BUFFER AVAILABLE
NO
YES
MESSAGE CAN NOT BE TAKEN, TOSS MESSAGE CLEAR REAL
RTN FROM INTRUP
LINK PRESENT BUFFER TO NEXT BUFFER UPDATE REAC TO REFLECT NEW BUFFER
(i.e. CHANGE LOCATIONS 01-02 OF REAC)
RTN FROM INTRUP
F
FILL BEGSTR WITH DATA FROM RET INTRANT RAM
INDICATE IN BEGBUF THAT THE BUFFERS BEING USED BY THIS ACIA ARE NO LONGER GOING TO BE INPUTTED INTO
(i.e. INDICATE THERE STATUS TO SHOW THAT MESSAGE IS FULLY RECEIVED)
RTN FROM INTRUP

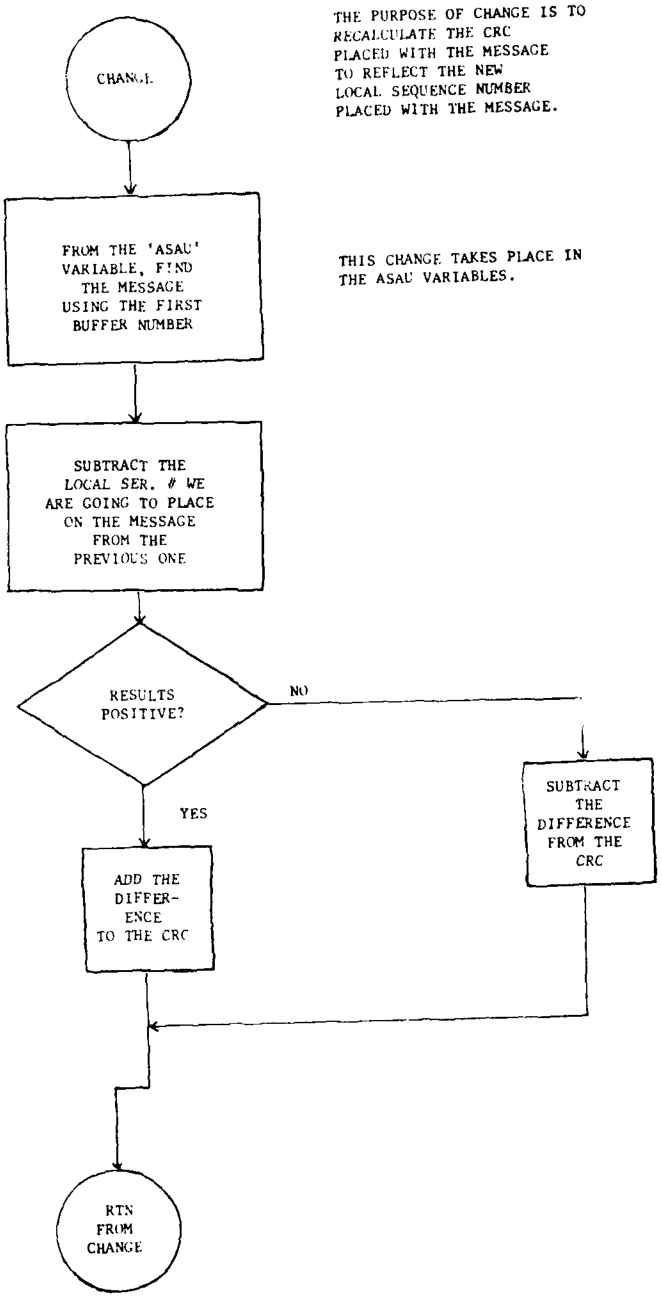

OUTPUT
THIS FLOWCHART OF THE OUTPUT ROUTINE DIFFERS WITH THE WAY THE PROGRAM LISTING READS. THIS SECTION OF CODE IS EXPLAINED IN A CLEARER WAY.
IS F-OUTPUT SET?
NO
YES
1 CLEAR FOUTPUT
2 OUTPUT THE ESCAPE CHARACTER
3 OUTPUT THE NEXT WORD OF THE MESSAGE
F-OUTPUT PROVIDES THE SAME CAPABILITY AS F-INPUT IN THE OUTPUT SENSE. IT ALLOWS AN END-OF-TEXT CHARACTER TO BE INCLUDED IN THE OUTPUT MESSAGE. WHEN F-OUTPUT IS SET, THE NEXT CHARACTER IS NOT TO BE INTERPRETED AS AND IN OF TEXT.
IS THIS THE LAST BUFFER?
NO
A
YES


G
ARE WE AT THE END OF THE MESSAGE?
NO
RTN FROM INTRUP
YES
WAS THE LAST CHARACTER OUTPUTTED AN ETX?
YES
CLEAR REINTRANT RAM, CLEAR THE MESSAGE FROM MEMORY IF IT IS GOING OUT TO HOME
NO
OUTPUT THE ETX CHARACTER
RTN FROM INTRUP
H
ARE AT THE END OF THE CURRENT BUFFER
NO
YES
LINK TO THE NEXT BUFFER, REFLECT IN REAC THAT WE'VE STARTED ON ANOTHER BUFFER
IS THIS OUTPUT TO HOME?
YES
SKIP THE HEADER OF THIS PACKET
RTN FROM INTRUP


I
BUFFER AVAILABLE?
NO
RETURN FROM INTRUP
SET UP LINKS TO THIS BUFFER FROM PREVIOUS ONE.
FIND BEGSTR FOR 1st PACKET OF MSG
NOTE: IF NO BEGSTR AVAILABLE THERE IS A PROBLEM.
TRANSFER REAC TO BEGSTR, CLEAR REAC
SET UP HEADER FOR SECOND PACKET, NOW WE ARE READY TO CONTINUE INPUT
RETURN FROM INTRUP



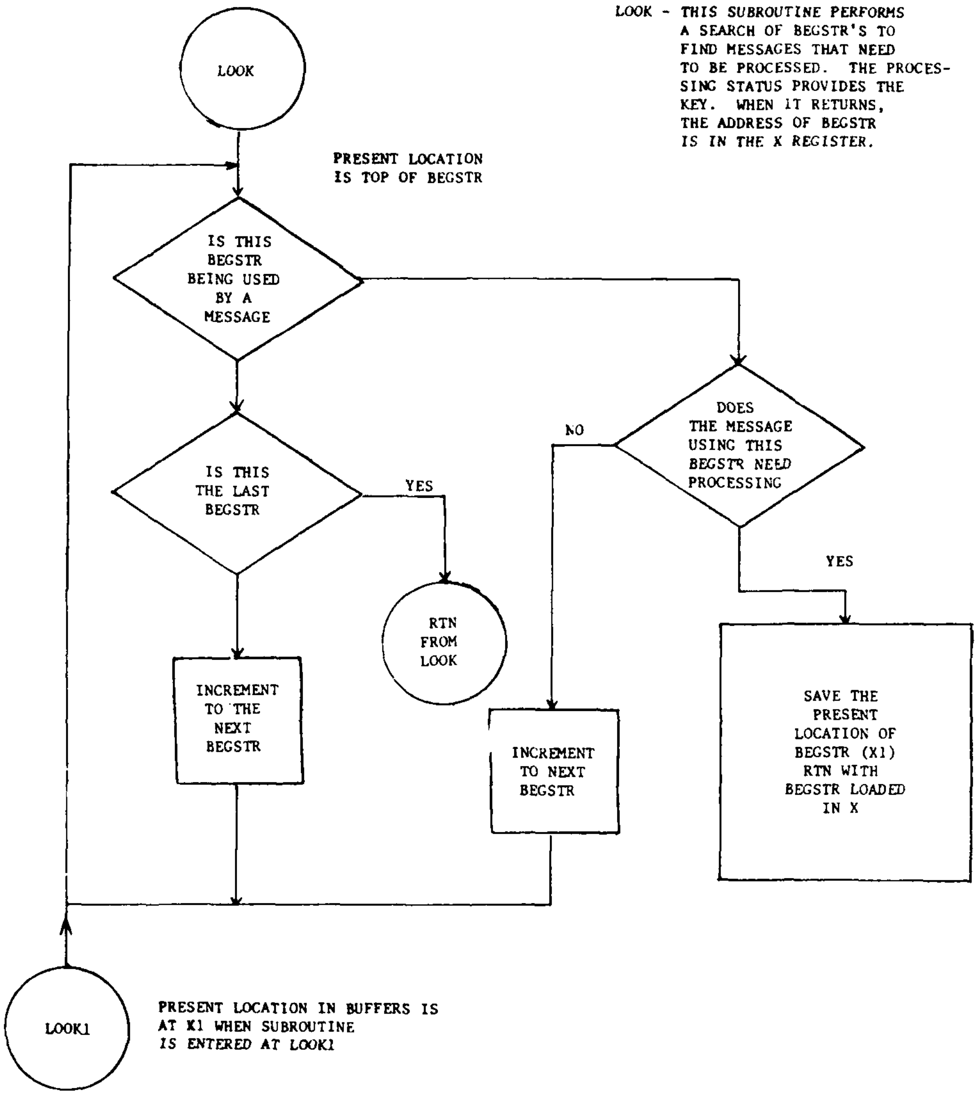
OUTPUTA ACCOMPLISHES THE OUTPUT TO HOME. WITH A FEW EXCEPTIONS IT RESEMBLES VERY CLOSELY THE REGULAR OUTPUT ROUTINE, THIS FLOWCHART IS NOT COMPLETE.

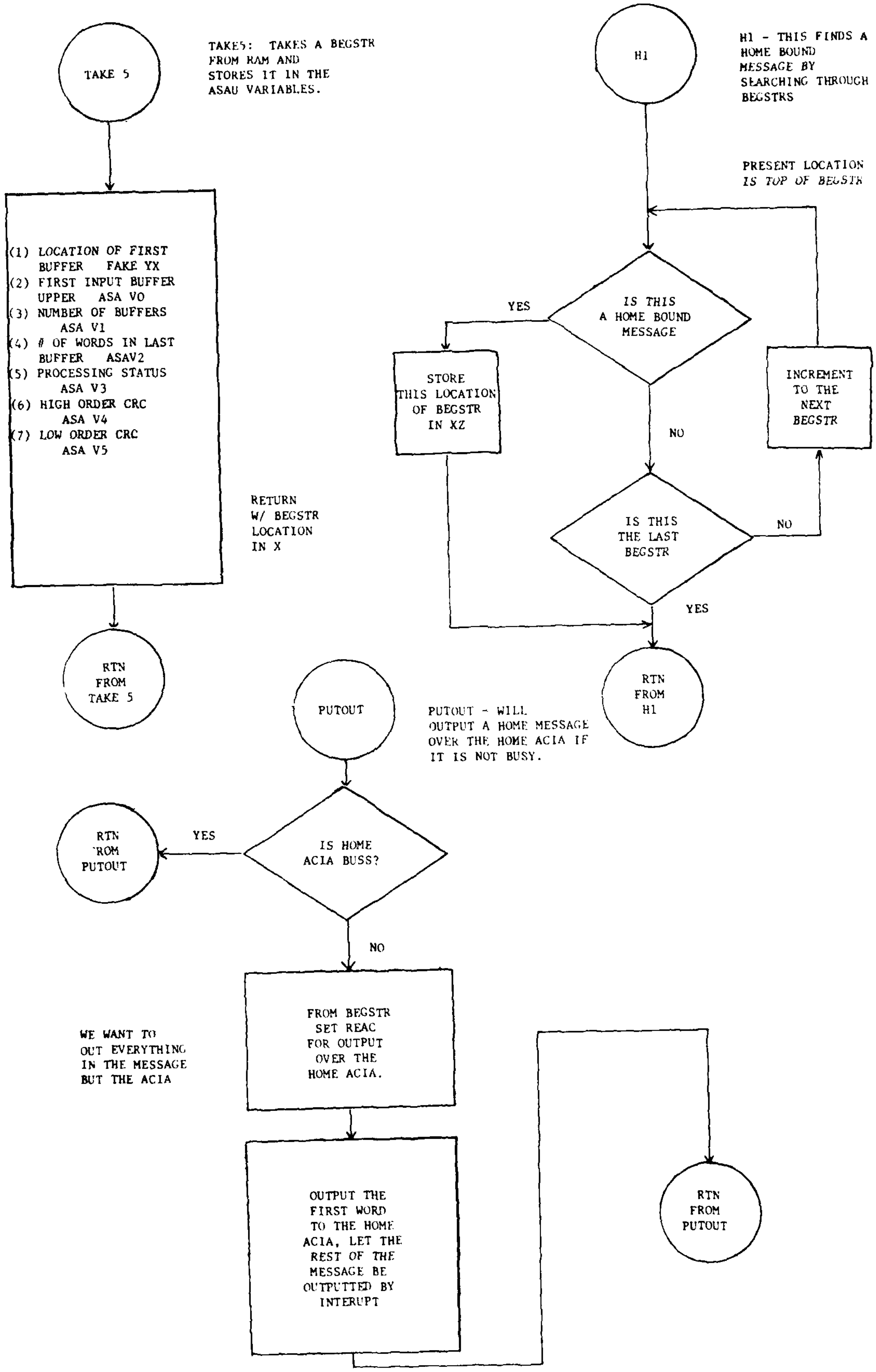


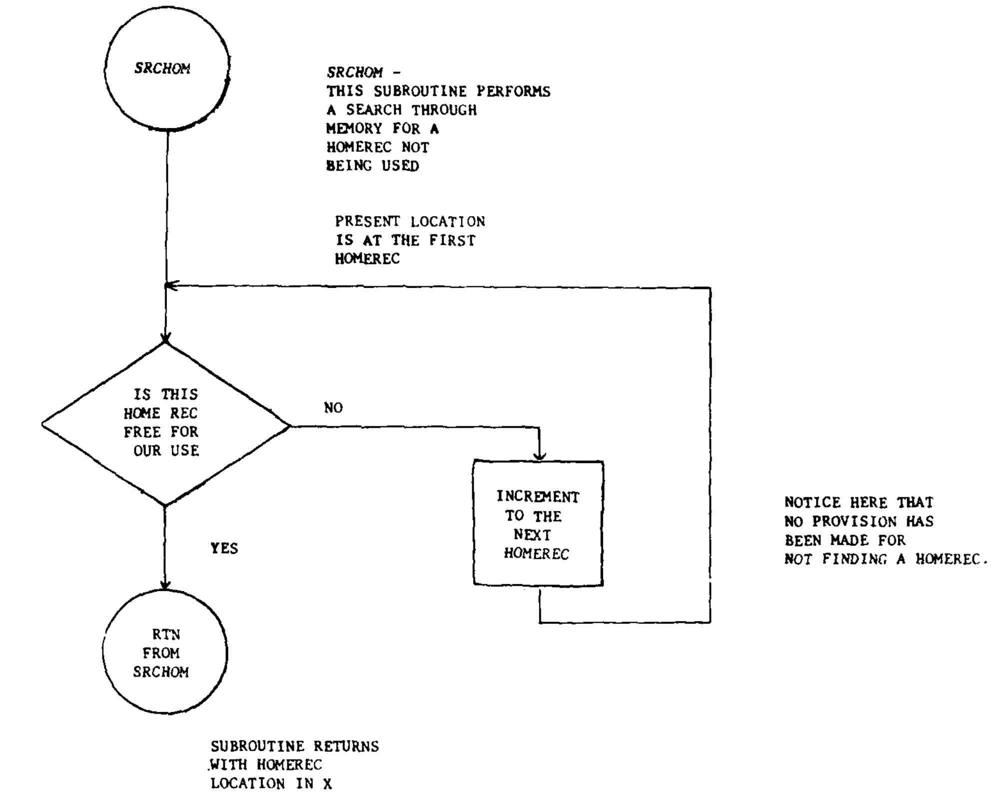
THE PURPOSE OF CHANGE IS TO RECALCULATE THE CRC PLACED WITH THE MESSAGE TO REFLECT THE NEW LOCAL SEQUENCE NUMBER PLACED WITH THE MESSAGE.

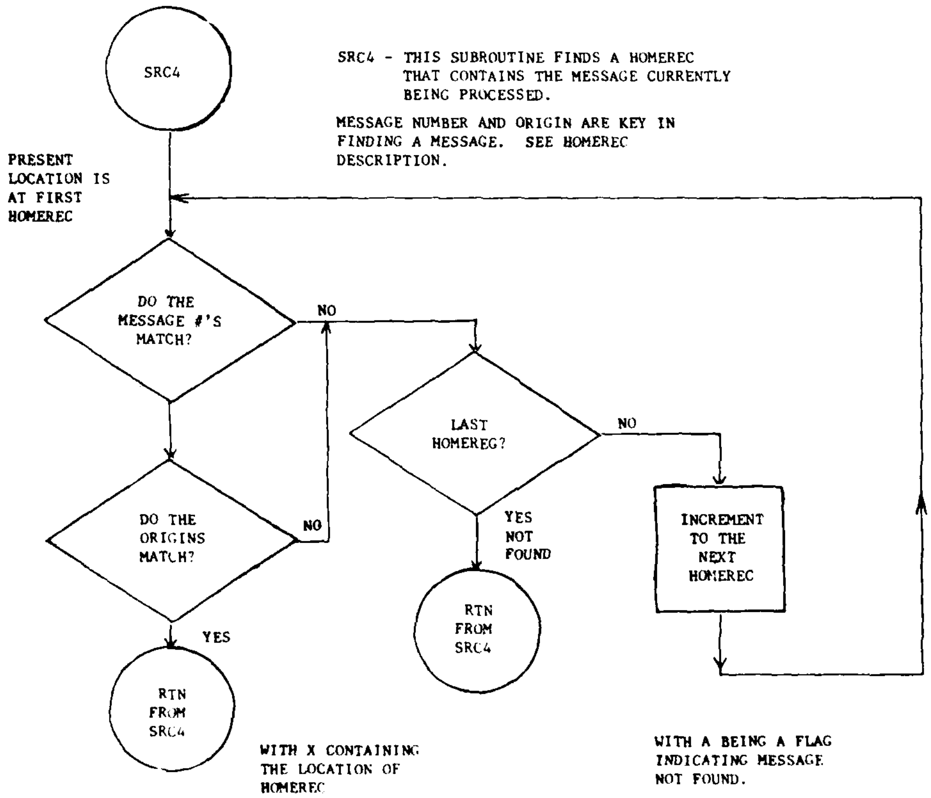
THIS CHANGE TAKES PLACE IN THE ASAU VARIABLES.

LOOK - THIS SUBROUTINE PERFORMS A SEARCH OF BEGSTR'S TO FIND MESSAGES THAT NEED TO BE PROCESSED. THE PROCESSING STATUS PROVIDES THE KEY. WHEN IT RETURNS, THE ADDRESS OF BEGSTR IS IN THE X REGISTER.

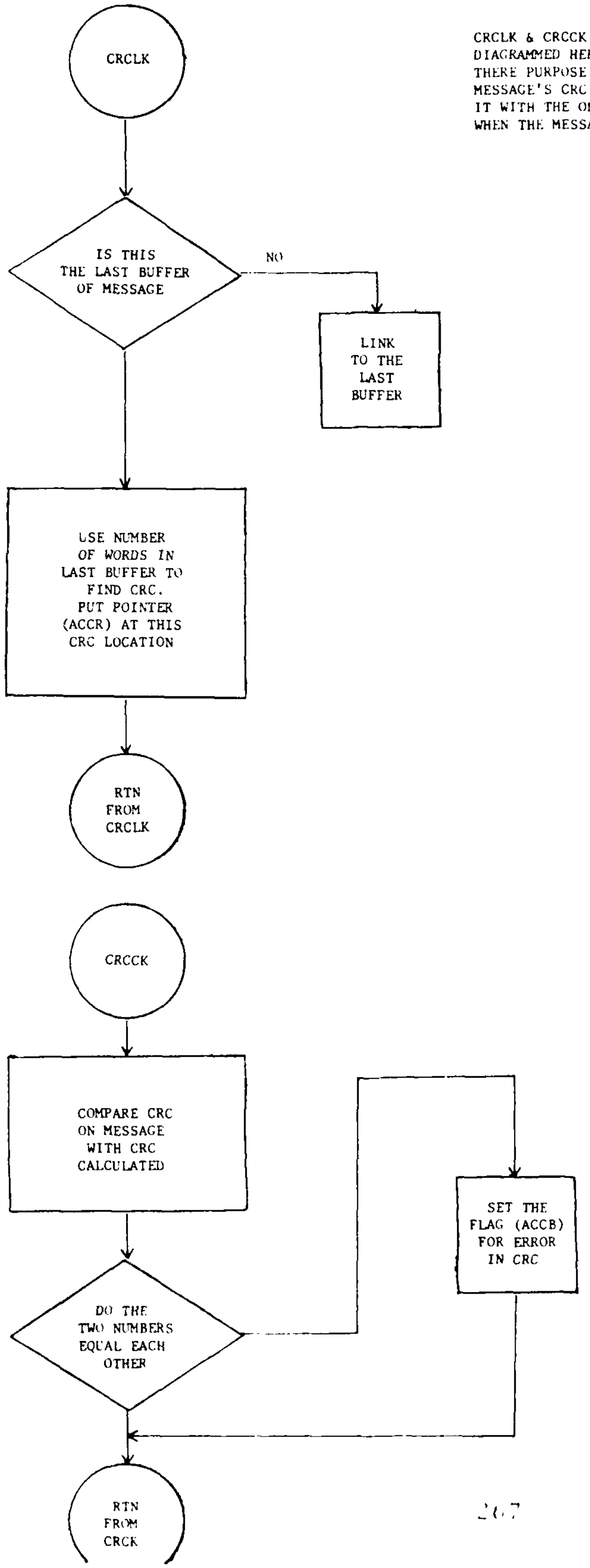



TAKE 5
TAKE5: TAKES A BEGSTR FROM RAM AND STORES IT IN THE ASAU VARIABLES.
(1) LOCATION OF FIRST BUFFER FAKE YX
(2) FIRST INPUT BUFFER UPPER ASA V0
(3) NUMBER OF BUFFERS ASA V1
(4) # OF WORDS IN LAST BUFFER ASAV2
(5) PROCESSING STATUS ASA V3
(6) HIGH ORDER CRC ASA V4
(7) LOW ORDER CRC ASA V5
RETURN W/ BEGSTR LOCATION IN X
RTN FROM TAKE 5
H1
H1 - THIS FINDS A HOME BOUND MESSAGE BY SEARCHING THROUGH BEGSTRS
PRESENT LOCATION IS TOP OF BEGSTR
IS THIS A HOME BOUND MESSAGE
YES
STORE THIS LOCATION OF BEGSTR IN XZ
NO
IS THIS THE LAST BEGSTR
NO
INCREMENT TO THE NEXT BEGSTR
YES
RTN FROM H1
PUTOUT
PUTOUT - WILL OUTPUT A HOME MESSAGE OVER THE HOME ACIA IF IT IS NOT BUSY.
IS HOME ACIA BUSS?
YES
RTN FROM PUTOUT
NO
FROM BEGSTR SET REAC FOR OUTPUT OVER THE HOME ACIA.
WE WANT TO OUT EVERYTHING IN THE MESSAGE BUT THE ACIA
OUTPUT THE FIRST WORD TO THE HOME ACIA, LET THE REST OF THE MESSAGE BE OUTPUTTED BY INTERUPT
RTN FROM PUTOUT

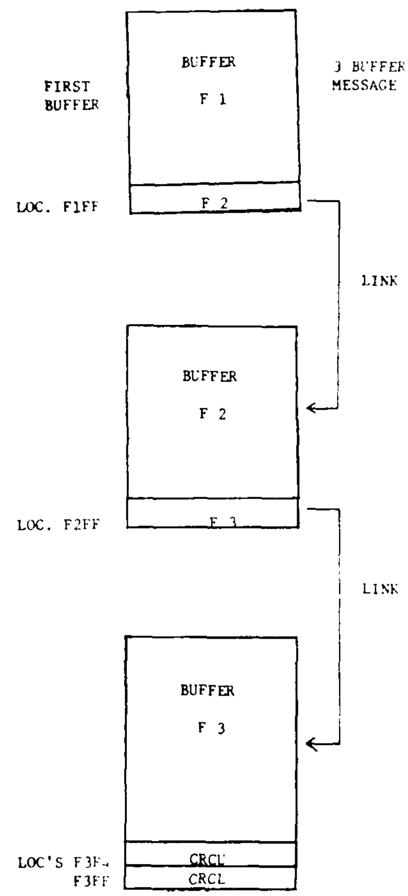

SRCHOM
SRCHOM -
THIS SUBROUTINE PERFORMS
A SEARCH THROUGH
MEMORY FOR A
HOMEREC NOT
BEING USED
PRESENT LOCATION
IS AT THE FIRST
HOMEREC
IS THIS
HOME REC
FREE FOR
OUR USE
NO
INCREMENT
TO THE
NEXT
HOMEREC
NOTICE HERE THAT
NO PROVISION HAS
BEEN MADE FOR
NOT FINDING A HOMEREC.
YES
RTN
FROM
SRCHOM
SUBROUTINE RETURNS
WITH HOMEREC
LOCATION IN X


SRC4
SRC4 - THIS SUBROUTINE FINDS A HOMEREC
THAT CONTAINS THE MESSAGE CURRENTLY
BEING PROCESSED.
MESSAGE NUMBER AND ORIGIN ARE KEY IN
FINDING A MESSAGE. SEE HOMEREC
DESCRIPTION.
PRESENT
LOCATION IS
AT FIRST
HOMEREC
DO THE
MESSAGE #'S
MATCH?
NO
LAST
HOMEREG?
NO
INCREMENT
TO THE
NEXT
HOMEREC
DO THE
ORIGINS
MATCH?
NO
YES
NOT
FOUND
RTN
FROM
SRC4
YES
RTN
FROM
SRC4
WITH X CONTAINING
THE LOCATION OF
HOMEREC
WITH A BEING A FLAG
INDICATING MESSAGE
NOT FOUND.

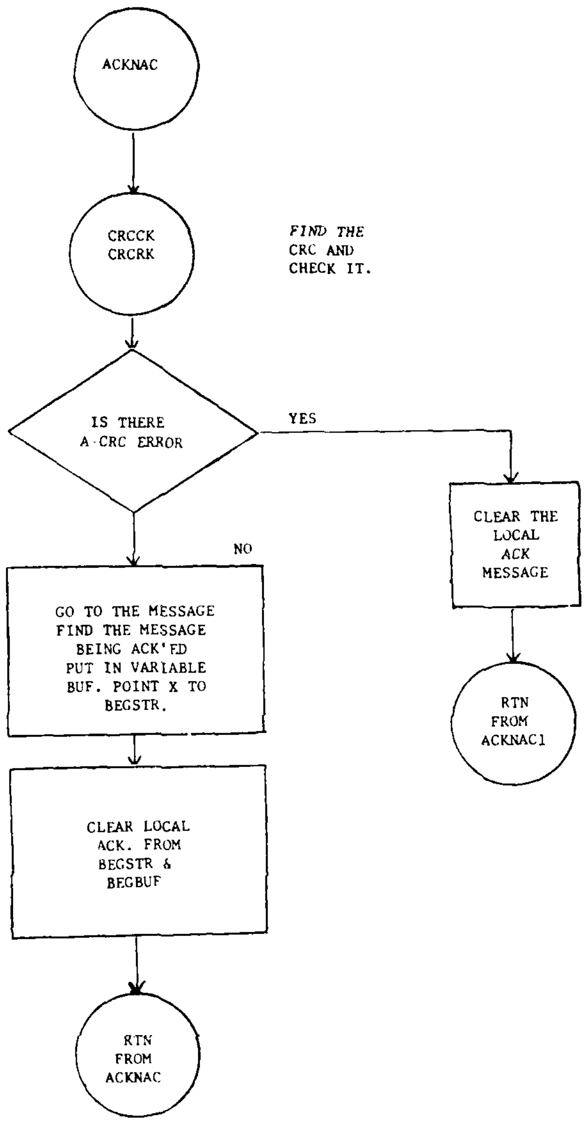
CRCLK & CRCCK BOTH WILL BE DIAGRAMMED HERE.
THERE PURPOSE IS TO FIND A MESSAGE'S CRC AND CHECK IT WITH THE ONE CALCULATED WHEN THE MESSAGE WAS INPUTTED

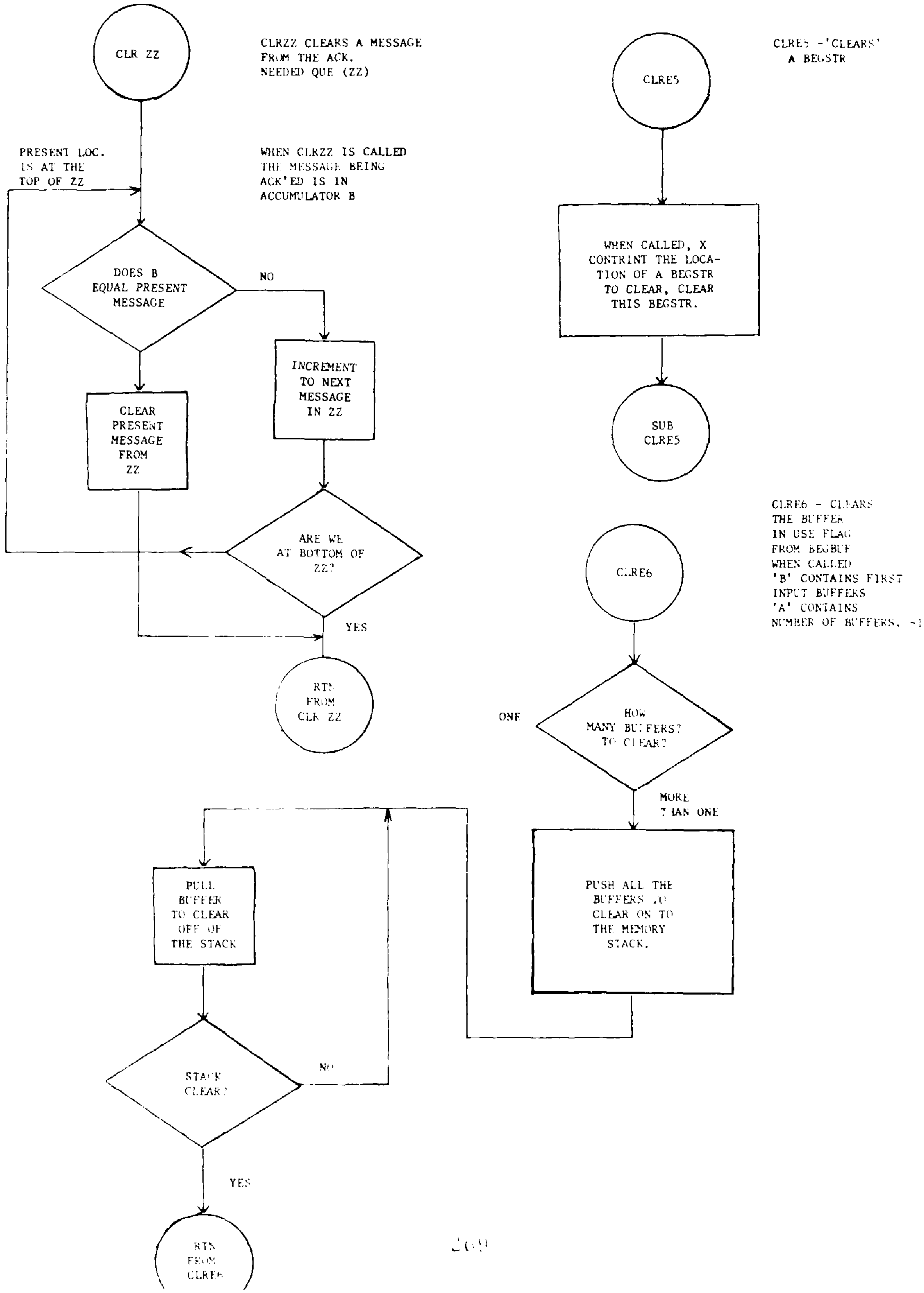


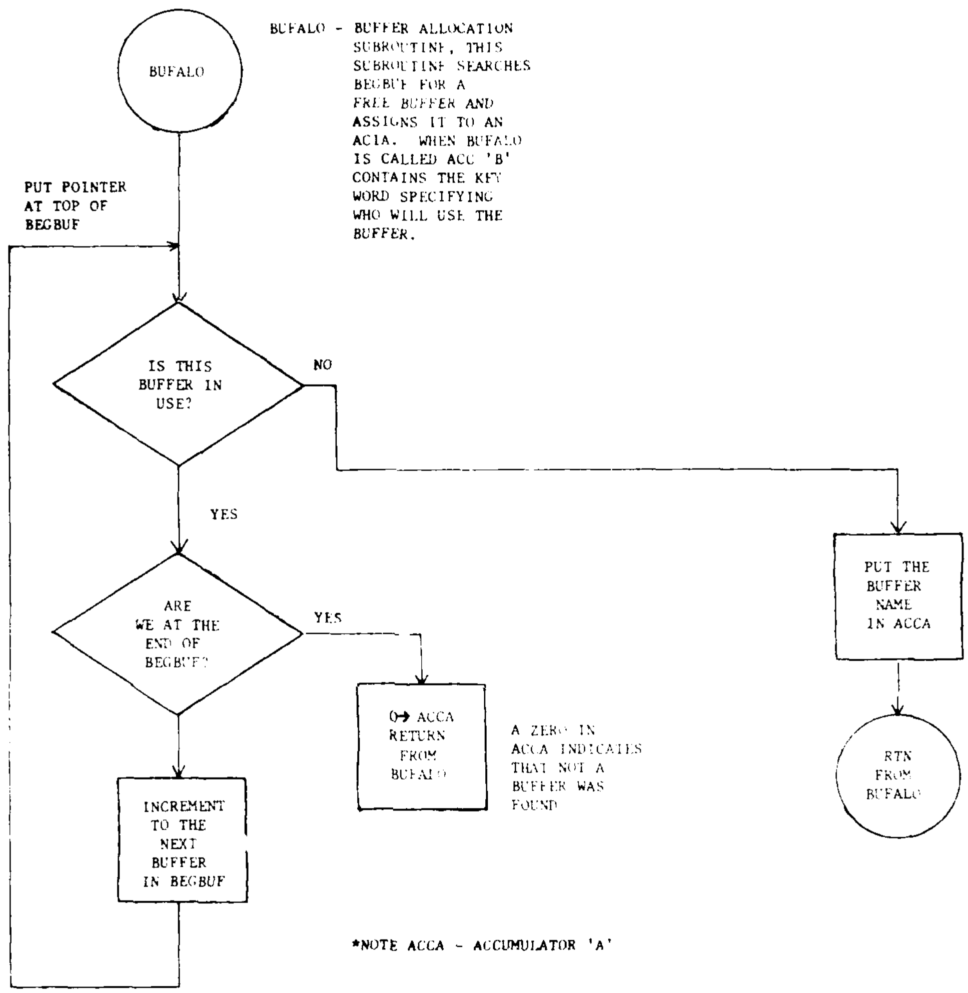
EXAMPLE OF BUFFER LINKAGE AND LOCATION OF CRC



ACKNAC CLEARS THE LOCAL ACKNOWLEDGEMENT MESSAGE FROM BEGSTR & BEGBUF.



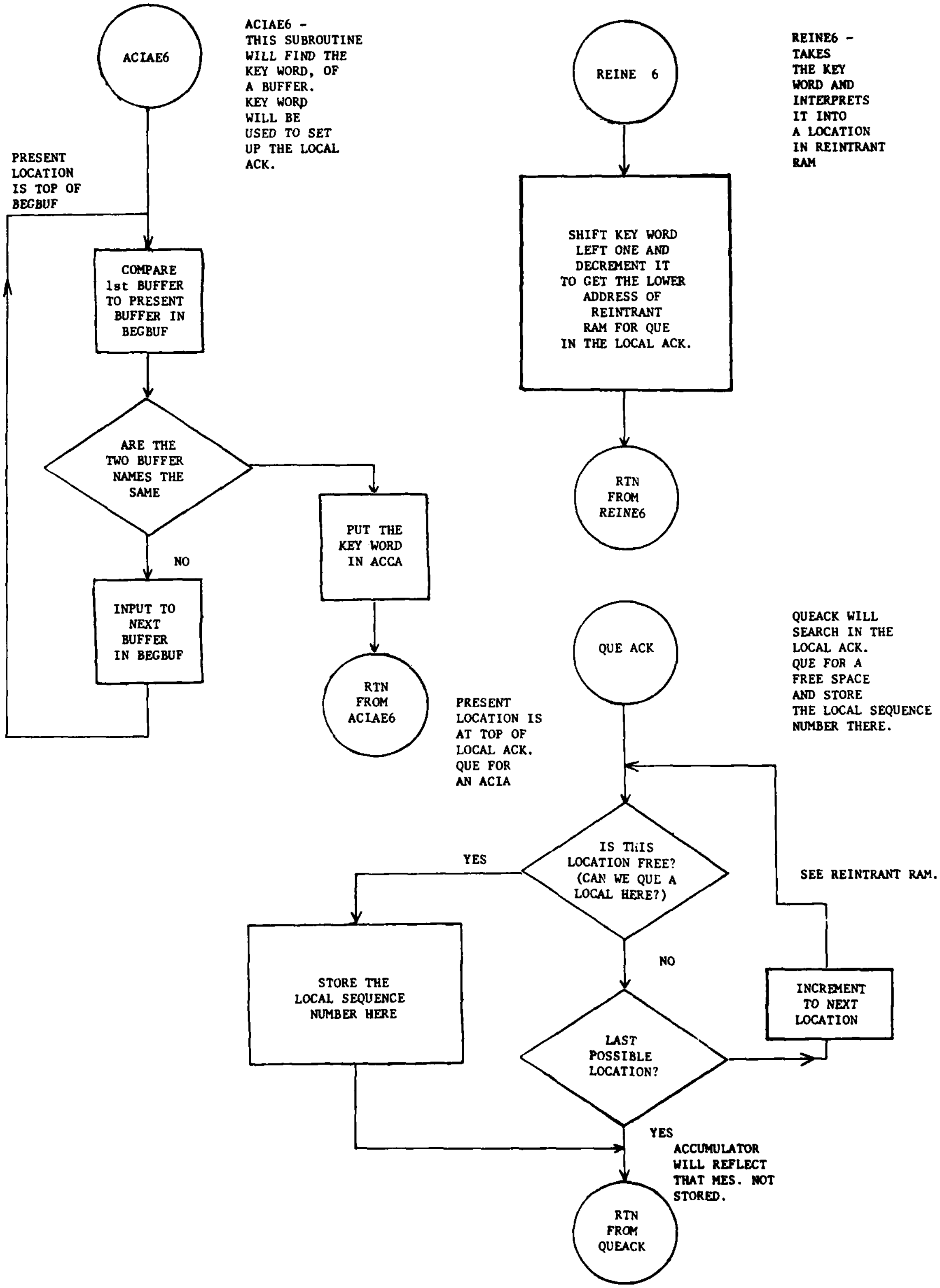

CLR ZZ
CLRZZ CLEARS A MESSAGE
FROM THE ACK.
NEEDED QUE (ZZ)
PRESENT LOC.
IS AT THE
TOP OF ZZ
WHEN CLRZZ IS CALLED
THE MESSAGE BEING
ACK'ED IS IN
ACCUMULATOR B
DOES B
EQUAL PRESENT
MESSAGE
NO
INCREMENT
TO NEXT
MESSAGE
IN ZZ
CLEAR
PRESENT
MESSAGE
FROM
ZZ
ARE WE
AT BOTTOM OF
ZZ?
YES
RTN
FROM
CLR ZZ
CLRE5
CLRE5 -'CLEARS'
A BEGSTR
WHEN CALLED, X
CONTRINT THE LOCA-
TION OF A BEGSTR
TO CLEAR, CLEAR
THIS BEGSTR.
SUB
CLRE5
CLRE6
CLRE6 - CLEARS
THE BUFFER
IN USE FLAG
FROM BEGBUF
WHEN CALLED
'B' CONTAINS FIRST
INPUT BUFFERS
'A' CONTAINS
NUMBER OF BUFFERS. -1
HOW
MANY BUFFERS?
TO CLEAR?
ONE
MORE
THAN ONE
PUSH ALL THE
BUFFERS TO
CLEAR ON TO
THE MEMORY
STACK.
PULL
BUFFER
TO CLEAR
OFF OF
THE STACK
STACK
CLEAR?
NO
YES
RTN
FROM
CLRE6

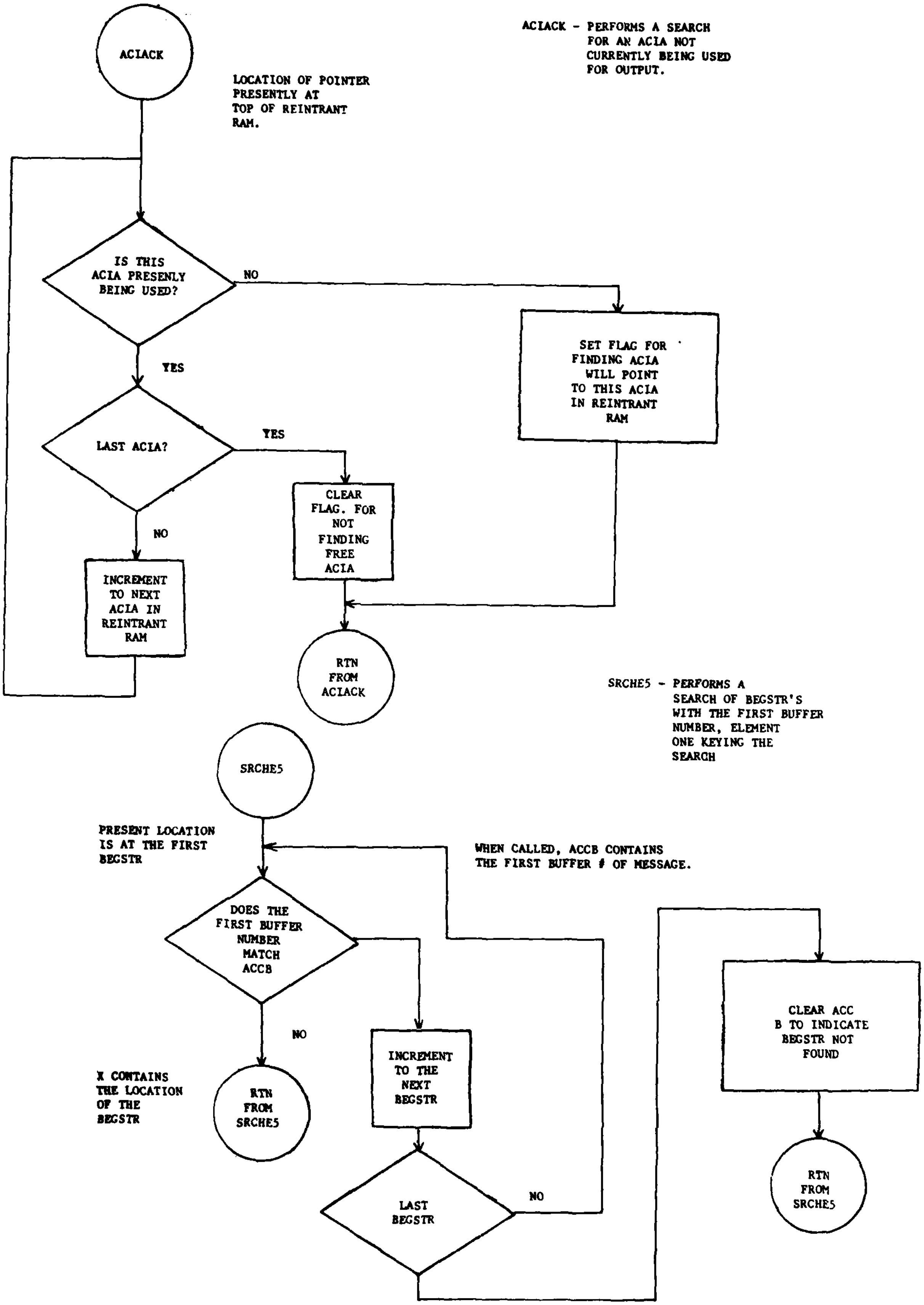

BUFALO
BUFALO - BUFFER ALLOCATION SUBROUTINE, THIS SUBROUTINE SEARCHES BEGBUF FOR A FREE BUFFER AND ASSIGNS IT TO AN ACIA. WHEN BUFALO IS CALLED ACC 'B' CONTAINS THE KEY WORD SPECIFYING WHO WILL USE THE BUFFER.
PUT POINTER AT TOP OF BEGBUF
IS THIS BUFFER IN USE?
NO
YES
PUT THE BUFFER NAME IN ACCA
RTN FROM BUFALO
ARE WE AT THE END OF BEGBUF?
YES
0→ ACCA RETURN FROM BUFALO
A ZERO IN ACCA INDICATES THAT NOT A BUFFER WAS FOUND
INCREMENT TO THE NEXT BUFFER IN BEGBUF
*NOTE ACCA - ACCUMULATOR 'A'

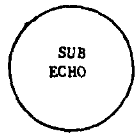
AD-A083 046   GEORGIA INST OF TECH ATLANTA SCHOOL OF ELECTRICAL EN--ETC F/G 9/2
THE FEASIBILITY OF IMPLEMENTING MULTICOMMAND SOFTWARE FUNCTIONS--ETC(U)
OCT 79 T P BARNWELL, J L HAMMOND, J H SCHLAG   DAAG29-78-G-0139
UNCLASSIFIED   ARO-15900.1-A-EL   NL

4 OF 4
AD A
083046

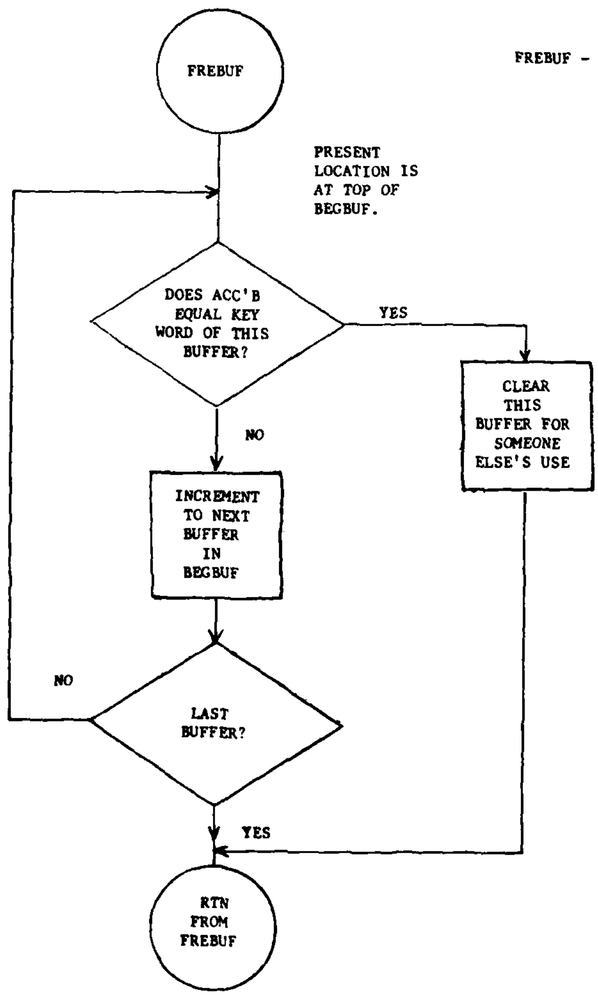
END
DATE
FILMED
5-80
DTIC


ACIAE6
ACIAE6 - THIS SUBROUTINE WILL FIND THE KEY WORD, OF A BUFFER. KEY WORD WILL BE USED TO SET UP THE LOCAL ACK.
PRESENT LOCATION IS TOP OF BEGBUF
COMPARE 1st BUFFER TO PRESENT BUFFER IN BEGBUF
ARE THE TWO BUFFER NAMES THE SAME
NO
INPUT TO NEXT BUFFER IN BEGBUF
PUT THE KEY WORD IN ACCA
RTN FROM ACIAE6
REINE 6
REINE6 - TAKES THE KEY WORD AND INTERPRETS IT INTO A LOCATION IN REINTRANT RAM
SHIFT KEY WORD LEFT ONE AND DECREMENT IT TO GET THE LOWER ADDRESS OF REINTRANT RAM FOR QUE IN THE LOCAL ACK.
RTN FROM REINE6
QUE ACK
QUEACK WILL SEARCH IN THE LOCAL ACK. QUE FOR A FREE SPACE AND STORE THE LOCAL SEQUENCE NUMBER THERE.
PRESENT LOCATION IS AT TOP OF LOCAL ACK. QUE FOR AN ACIA
IS THIS LOCATION FREE? (CAN WE QUE A LOCAL HERE?)
YES
SEE REINTRANT RAM.
STORE THE LOCAL SEQUENCE NUMBER HERE
NO
INCREMENT TO NEXT LOCATION
LAST POSSIBLE LOCATION?
YES
ACCUMULATOR WILL REFLECT THAT MES. NOT STORED.
RTN FROM QUEACK

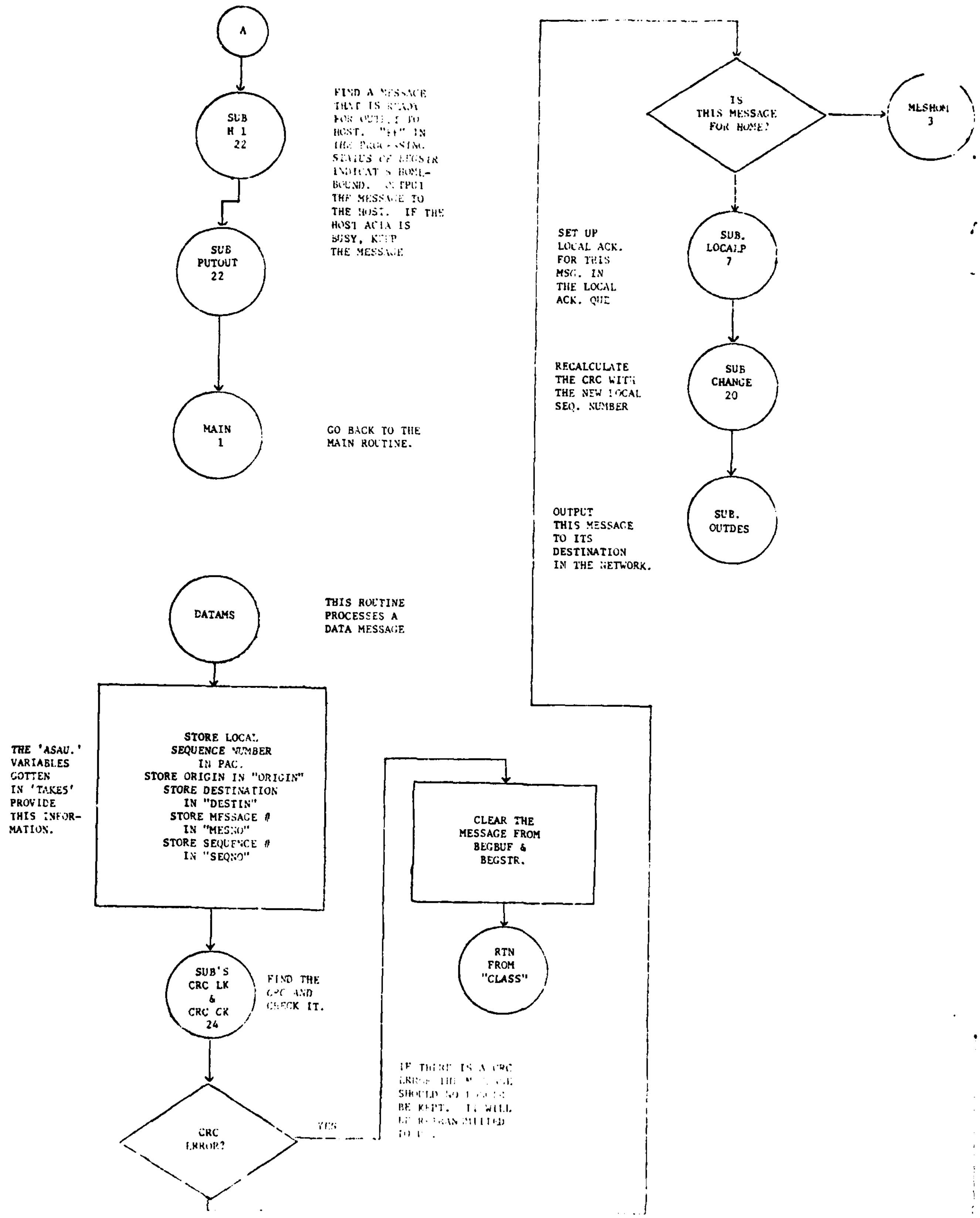

ACIACK - PERFORMS A SEARCH FOR AN ACIA NOT CURRENTLY BEING USED FOR OUTPUT.
ACIACK
LOCATION OF POINTER PRESENTLY AT TOP OF REINTRANT RAM.
IS THIS ACIA PRESENLY BEING USED?
NO
SET FLAG FOR FINDING ACIA WILL POINT TO THIS ACIA IN REINTRANT RAM
YES
LAST ACIA?
YES
CLEAR FLAG. FOR NOT FINDING FREE ACIA
NO
INCREMENT TO NEXT ACIA IN REINTRANT RAM
RTN FROM ACIACK
SRCHE5 - PERFORMS A SEARCH OF BEGSTR'S WITH THE FIRST BUFFER NUMBER, ELEMENT ONE KEYING THE SEARCH
SRCHE5
PRESENT LOCATION IS AT THE FIRST BEGSTR
WHEN CALLED, ACCB CONTAINS THE FIRST BUFFER # OF MESSAGE.
DOES THE FIRST BUFFER NUMBER MATCH ACCB
NO
X CONTAINS THE LOCATION OF THE BEGSTR
RTN FROM SRCHE5
INCREMENT TO THE NEXT BEGSTR
LAST BEGSTR
NO
CLEAR ACC B TO INDICATE BEGSTR NOT FOUND
RTN FROM SRCHE5



ECHO - THIS SUBROUTINE ECHOES THE INPUTTED CHARACTER BACK TO THE HOST. THE FLOWCHART OF THIS SUBROUTINE IS NOT COMPLETE.



FREBUF - THIS SUBROUTINE FREES A BUFFER FOR USE BY OTHER ACIA'S. WHEN CALLED THE KEY WORD OF THE BUFFER TO BE FREED IS IN THE 'B' ACCUMULATOR.


A
SUB
H 1
22
SUB
PUTOUT
22
MAIN
1
GO BACK TO THE
MAIN ROUTINE.
THE MESSAGE TO
THE HOST. IF THE
HOST ACIA IS
BUSY, KEEP
THE MESSAGE
IS
THIS MESSAGE
FOR HOME?
MESHOM
3
SET UP
LOCAL ACK.
FOR THIS
MSG. IN
THE LOCAL
ACK. QUE
SUB.
LOCALP
7
RECALCULATE
THE CRC WITH
THE NEW LOCAL
SEQ. NUMBER
SUB
CHANGE
20
OUTPUT
THIS MESSAGE
TO ITS
DESTINATION
IN THE NETWORK.
SUB.
OUTDES
DATAMS
THIS ROUTINE
PROCESSES A
DATA MESSAGE
THE 'ASAU.'
VARIABLES
GOTTEN
IN 'TAKES'
PROVIDE
THIS INFOR-
MATION.
STORE LOCAL
SEQUENCE NUMBER
IN PAC.
STORE ORIGIN IN "ORIGIN"
STORE DESTINATION
IN "DESTIN"
STORE MESSAGE #
IN "MESNO"
STORE SEQUENCE #
IN "SEQNO"
CLEAR THE
MESSAGE FROM
BEGBUF &
BEGSTR.
RTN
FROM
"CLASS"
SUB'S
CRC LK
&
CRC CK
24
FIND THE
CRC AND
CHECK IT.
BE KEPT.
CRC
ERROR?
YES

12. APPENDIX D--COBOL PROGRAM

```
MICROSOFT COBOL-80 V2.0... DEMO  COB    10/24/78  10:00:00     PAGE  1


         IDENTIFICATION DIVISION.
         PROGRAM-ID.
             INVENTORY-DEMO-PROG.
         DATE-WRITTEN. ORIGINALLY 7 FEB 1979
              COMPLETED APPROXIMATELY 20 JUL 1979.
        *
         ENVIRONMENT DIVISION.
         CONFIGURATION SECTION.
         SOURCE-COMPUTER.
               INTEL 8080.
         OBJECT-COMPUTER.
               INTEL 8080.
         INPUT-OUTPUT SECTION.
         FILE-CONTROL.
              SELECT DATA-FILE1 ASSIGN TO DISK
                ORGANIZATION IS INDEXED  ACCESS MODE IS DYNAMIC
                RECORD KEY IS PRTNO.
         DATA DIVISION.
         FILE SECTION.
         FD    DATA-FILE1
               LABEL RECORDS ARE STANDARD
               DATA RECORD IS DATA-BASE
               VALUE OF FILE-ID IS ":F0:DATA1.IND".
         01    DATA-BASE.
              05    PRTNO       PIC X(3).
              05    BLANK1       PIC X(5).
              05    PT-NM       PIC X(6).
              05    STOCK1        PIC 999.
              05    ON-ORDER1   PIC 999.
              05    THRESHOLD1  PIC 999.
              05    ORDER-SIZE1 PIC 999.
        *
        *FD    NETWORK-IN
        *      ABOVE FOR CS-20'S BENEFIT ONLY
        *FD    NETWORK-OUT
        *      ABOVE FOR CS-20 ONLY SO IGNORE
        *FD    AUDIT-FILE
        *      ABOVE FOR PDP-11 AUDIT TRAIL FILE
        *
         WORKING-STORAGE SECTION.
        *01  COMMUNICATIONS-STORAGE.
         01  MICRO-FLAG PIC 9.
         01  MESSAGE-SEND-LENGTH USAGE IS INDEX.
         01  MESSAGE-LENGTH USAGE IS INDEX.
         01  MESSAGE-BUFFER.
                 05  SOURCE-DESTINATION  PIC X.
                 05  MESSAGE-CONTROL.
                 10  MESSAGE-CLASS  PIC X.
                 10  LAST-OF-MESSAGE  PIC X.
                 10  FILLER        PIC X(5).
                 05  MESSAGE-DATA           PIC X(72).
             05  MESSAGE-DATA2 REDEFINES MESSAGE-DATA.
                 10  MESSAGE-DATA1 PIC X OCCURS 72 TIMES
```

```
                    INDEXED BY MES-INDEX.
                05  FILLER PIC X(8).
          01    MESSAGE-BUF REDEFINES MESSAGE-BUFFER.
                    05  MESSAGE-BUFF PIC X(70).
                    05  FILLER       PIC X(10).
          01    LOCAL-FLAG PIC X VALUE "Y".
          01    M6800-CODE PIC X VALUE "A".
          01    PDP11-CODE PIC X VALUE "B".
          01    INTEL-CODE PIC X VALUE "C".
          01    CS-20-CODE PIC X VALUE "D".
          01    RETURN-ADDRESS PIC X.
          01    REMOTE-COMMAND PIC X.
          01    REMOTE-ADDRESS PIC X.
          01    WAIT-FOR-ANSWER PIC X.
          01    COM-FUNCTION PIC X.
          01      DUM-DUM-TABLE.
                03     DUM-DUM PIC X OCCURS 2 TIMES.
          01    COMMAND-STRING.
                05  MAX-LENGTH       USAGE IS INDEX.
                05  PART-STRING1.
                    10  FIRST-CHARACTER  PIC X.
                    10  REST-COMMAND  PIC X OCCURS 29 TIMES.
                05  PART-STRING REDEFINES PART-STRING1.
                    10  PART-NUM         PIC X(13).
                    10  DIGITS REDEFINES PART-NUM.
                        15  DIGIT-3      PIC X(3).
                        15  FILLER       PIC X(10).
                    10  FILLER           PIC X(17).
                05  OUT-COMMAND REDEFINES PART-STRING.
                    10  COMMAND-OUT PIC X OCCURS 30 TIMES INDEXED BY I3.
                05  PART-NUMBER REDEFINES OUT-COMMAND PIC X OCCURS 30 TIMES
                                INDEXED BY I1, AACTUAL.
         *
          01    STRINGB.
                 05  MAXB USAGE IS INDEX.
                 05  STRING2 PIC X OCCURS 10 TIMES INDEXED BY I2.
          01    DEBUG     PIC X VALUE "N".
          01    STOP-FLAG    PIC X VALUE "Y".
          01    CURRENT-DATE   PIC X(8).
          01    LIST.
                05  MAX1  USAGE IS INDEX.
                05  FILLER PIC X(10) VALUE "LIST      ".
          01    INITIALIZE.
                05  MAX  USAGE IS INDEX.
                05  FILLER PIC X(10) VALUE "INITIALIZE".
          01    UPDATE.
                05  MAX2  USAGE IS INDEX.
                05  FILLER PIC X(10) VALUE "UPDATE    ".
          01    STOCK.
                05  MAX3  USAGE IS INDEX.
                05  FILLER PIC X(10) VALUE "STOCK     ".
          01    ON-ORDER.
                05  MAX4  USAGE IS INDEX.
                05  FILLER PIC X(10) VALUE "ONORDER   ".
          01    THRESHOLD.
```

```
              05  MAX5  USAGE IS INDEX.
              05  FILLER PIC X(10) VALUE "THRESHOLD ".
          01  ORDER-SIZE.
              05  MAX6 USAGE IS INDEX.
              05  FILLER PIC X(10) VALUE "ORDERSIZE ".
          01  COMMAND.
              05  ACTION    PIC X.
              05  EENTRY    PIC X.
              05  QUANTITY-SIGN PIC X VALUE "+".
              05  QUANTITY      PIC 999.
              05  PARTNO        PIC X(13) VALUE "             ".
              05  PART-NAME     PIC X(20) VALUE "                    ".
          01  FLAGS.
              05  ACTION-FLAG PIC X VALUE "N".
              05  REPEAT-FLAG  PIC X.
              05  ERROR-FLAG   PIC X.
          01  DUMMY-FLAG  PIC X.
          01  TRANS-FLAG  PIC X.
          01  SAME-FLAG  PIC X.
          01  NEW-BUFFER  PIC X.
          01  NO-INPUT-FLAG  PIC X VALUE "N".
          01  BUFFER-EMPTY PIC X VALUE "N".
          01  EMPTY-LINE PIC X.
          01  YES         PIC X VALUE "Y".
          01  NONO           PIC X VALUE "N".
          01  PARTNO-STORAGE.
              05  DUMMY-ARRAY PIC X OCCURS 100 TIMES
                      INDEXED BY I.
              05  TEM  PIC X.
         *
          01   DATA-BUFFER.
               05    PART-NUMB  PIC X(3).
               05    BLANK-BUF   PIC X(5).
               05    PRT-NME     PIC X(6).
               05    STCK        PIC 999.
               05    ON-ORDR     PIC 999.
               05    THRESHLD    PIC 999.
               05    ORDR-SIZE   PIC 999.
          01 HEADING-LINE.
              05  FILLER PIC X(11) VALUE "PART NUMBER".
              05  FILLER PIC XXX VALUE "   ".
              05  FILLER PIC X(9) VALUE "PART NAME".
              05  FILLER PIC X(4) VALUE "    ".
              05  FILLER PIC X(5) VALUE "STOCK".
              05  FILLER PIC X VALUE " ".
              05  FILLER PIC X(8) VALUE "ON ORDER".
              05  FILLER PIC X VALUE " ".
              05  FILLER PIC X(9) VALUE "THRESHOLD".
              05  FILLER PIC X VALUE " ".
              05  FILLER PIC X(10) VALUE "ORDER SIZE".
          01 PRINT-LINE.
              05  PART-NUM-OUT  PIC X(13).
              05  FILLER PIC X VALUE " ".
              05  PART-NAME-OUT PIC X(12).
              05  FILLER PIC XX VALUE "  ".
```

```
              05  STOCK-OUT PIC 999.
              05  FILLER PIC X(4) VALUE "    ".
              05  ON-ORDER-OUT PIC 999.
              05  FILLER PIC X(7) VALUE "       ".
              05  THRESHOLD-OUT PIC 999.
              05  FILLER PIC X(7) VALUE "       ".
              05  ORDER-SIZE-OUT PIC 999.
          01  ERROR-LINE.
              05  FILLER PIC X(12) VALUE "PART NUMBER ".
              05  ERROR-PRINT PIC X(13).
              05  FILLER PIC X(24) VALUE " IS NOT IN THE DATA BASE".
          01  DELETE-LINE.
              05  FILLER PIC X(12) VALUE "PART NUMBER ".
              05  DELETE-PRINT PIC X(13).
              05  FILLER PIC X(17) VALUE " HAS BEEN DELETED".
          01  ADD-LINE.
              05  FILLER PIC X(12) VALUE "PART NUMBER".
              05  ADD-PRINT PIC X(13).
              05  FILLER PIC X(15) VALUE "HAS BEEN ADDED".
          01  PRESENT-LINE.
              05  FILLER PIC X(12) VALUE "PART NUMBER".
              05  DATA-PRES-PRINT PIC X(13).
              05  FILLER PIC X(19) VALUE "IS ALREADY PRESENT".
              05  FILLER PIC X(16) VALUE "IN THE DATA BASE".
          01  INTERNAL-ERROR-ENTRY.
              05  FILLER PIC X(22) VALUE "INTERNAL ERROR. ENTRY=".
              05  ENTRY-ERROR PIC X.
          01  INTERNAL-ERROR-COMMAND.
              05  FILLER PIC X(24) VALUE "INTERNAL ERROR. COMMAND=".
              05  COMMAND-ERROR PIC X.
          01  VALUE-TOO-BIG PIC X(26)
                  VALUE "VALUE TOO LARGE - 999 USED".
          01  MISCELLANEOUS.
              05  ERROR1-FLAG  PIC X.
              05  TEMP  PIC 999.
          01  INDEX-CONSTANTS.
              05  ONE USAGE IS INDEX.
              05  FOUR  USAGE IS INDEX.
              05  EIGHTY USAGE IS INDEX.
          01  D-I-P-F  PIC X.
          01    M-D-R-F   PIC X.
          01  COMMAND-SPACE.
              05  COMMAND-LINE.
                  10  COMMAND-BUFFER  PIC X OCCURS 70 TIMES
                            INDEXED BY TEM-PTR, PTR, TPTR.
              05  FILLER PIC X(10).
          01  BUFFER-LENGTH USAGE IS INDEX.
          01   ADDIT.
               05 MAX7 USAGE IS INDEX.
               05 FILLER PIC X(10) VALUE "ADD       ".
          01  DELETE-IT.
              05  MAX8 USAGE IS INDEX.
              05  FILLER PIC X(10) VALUE "DELETE    ".
          01  STOP-IT.
              05  MAX9 USAGE IS INDEX.
```

```
               05  FILLER PIC X(10) VALUE "STOP    ".
           01  HELP.
               05  MAX10 USAGE IS INDEX.
               05  FILLER PIC X(10) VALUE "HELP    ".
           01  REMOTE-NAME.
               05 MAX11 USAGE IS INDEX.
               05 FILLER PIC X(10) VALUE "REMOTE    ".
           01  SEND-IT.
               05  MAX16 USAGE IS INDEX.
               05  FILLER PIC X(10) VALUE "SEND      ".
           01  M6800.
               05  MAX12 USAGE IS INDEX.
               05  FILLER PIC X(10) VALUE "M6800     ".
           01  PDP11.
               05  MAX13 USAGE IS INDEX.
               05  FILLER PIC X(10) VALUE "PDP11     ".
           01  INTEL.
               05  MAX14 USAGE IS INDEX.
               05  FILLER PIC X(10) VALUE "INTEL     ".
           01  CS20.
               05  MAX15 USAGE IS INDEX.
               05  FILLER PIC X(10) VALUE "CS-20     ".
           01    FIRST-TIME-THRU    PIC 9 VALUE 1.
          *  END OF WORKING STORAGE SECTION.
          *
           PROCEDURE DIVISION.
           MAIN-PROGRAM.
               PERFORM INITIALIZE-FOR-DAY.
                PERFORM COMMAND-PROCESS UNTIL STOP-FLAG = YES.
               PERFORM END-DAY.
               STOP RUN.
          *
           COMMUNICATE.
          *   THIS PARAGRAPH IS A SUBROUTINE THAT IS SPECIFIC TO EACH
          *    COMPUTER IT HANDLES THE INTERFACE WITH THE NETWORK.
          *       LOCAL-FLAG INDICATES WHETHER A TRANSACTION COMES FROM
          *                      THIS MACHINE.
          *       COM-FUNCTION TELLS WHETHER TO SEND RECEIVE OR INITIALIZE.
          *       MESSAGE-BUFFER CONTAINS THE DATA TO BE TRANSFERRED
          *       MESSAGE LENGTH IS THE NUMBER OF CHARACTERS TRANSFERRED.
          *
               IF DEBUG = YES
                   DISPLAY "NETWORK CALLED" COM-FUNCTION
               ELSE
                   PERFORM COM-DUMMY.
           COM-DUMMY.
               IF COM-FUNCTION = "I"
                   PERFORM NETWORK-INITIALIZE
               ELSE
               IF COM-FUNCTION = "S"
                   PERFORM NETWORK-SEND
               ELSE
               IF COM-FUNCTION = "R"
                   PERFORM NETWORK-RECEIVE
               ELSE
```

```
                DISPLAY "ILLEGAL COMMAND TO COMMUNICATE" COM-FUNCTION
               STOP RUN.
      *
       NETWORK-INITIALIZE.
           DISPLAY "INITIALIZE THE NETWORK".
             SET MESSAGE-LENGTH TO 4.
      *         USE MESSAGE-LENGTH TO PASS A REQUEST FOR 1200 BAUD.
             CALL "INITIALIZE" USING MESSAGE-LENGTH.
      *
       NETWORK-SEND.
             SET MESSAGE-SEND-LENGTH TO MESSAGE-LENGTH.
             CALL "SENDMESSAGE"
                       USING MESSAGE-BUFFER MESSAGE-SEND-LENGTH.
      *
       NETWORK-RECEIVE.
           MOVE SPACES TO MESSAGE-BUFFER.
           CALL "RECEIVEMESSAGE"
               USING MESSAGE-BUFFER MESSAGE-LENGTH.
      *
       INITIALIZE-FOR-DAY.
                   SET I TO 1.
           SET ONE TO I.
               SET I TO 3.
           SET MAX7 TO I.
               SET I TO 4.
           SET MAX1 TO I.
           SET MAX9 TO I.
           SET MAX10 TO I.
           SET MAX16 TO I.
           SET FOUR TO I.
               SET I TO 5.
           SET MAX3 TO I.
           SET MAX12 TO I.
           SET MAX13 TO I.
           SET MAX14 TO I.
           SET MAX15 TO I.
               SET I TO 6.
           SET MAX2 TO I.
           SET MAX8 TO I.
           SET MAX11 TO I.
               SET I TO 7.
           SET MAX4 TO I.
               SET I TO 9.
           SET MAX5 TO I.
           SET MAX6 TO I.
               SET I TO 10.
           SET MAX TO I.
               SET I TO 30.
           SET MAX-LENGTH TO I.
               SET I TO 80.
           SET EIGHTY TO I.
           MOVE SPACES TO BLANK-BUF.
           MOVE NONO TO REMOTE-COMMAND.
           MOVE NONO TO WAIT-FOR-ANSWER.
           DISPLAY "INVENTORY PROGRAM VERSION 1.0".
```

```
             DISPLAY "ENTER DATE (DD/MM/YY)".
             ACCEPT CURRENT-DATE.
             PERFORM OPEN-AUDIT-FILE.
             MOVE "I" TO COM-FUNCTION.
             PERFORM COMMUNICATE.
      *          ABOVE HAS INITIALIZED THE NETWORK PORT.
             MOVE "C" TO ACTION.
             MOVE "I" TO EENTRY.
             MOVE 0 TO QUANTITY.
             PERFORM TRANSACTION-PROCESSOR.
             MOVE NONO TO STOP-FLAG.
             DISPLAY "ENTER HELP FOR A LIST OF CURRENT CAPABILITIES".
      *
         OPEN-AUDIT-FILE.
             PERFORM DO-NOTHING.
      *      FOR MICRO THIS IS A DUMMY PARAGRAPH
      *
         END-DAY.
             MOVE "C" TO ACTION.
             MOVE "E" TO EENTRY.
             PERFORM TRANSACTION-PROCESSOR.
             DISPLAY "END OF DAY PLEASE REMOVE DISKETTE".
      *
         COMMAND-PROCESS.
             IF DEBUG = YES DISPLAY "COMMAND PROCESS ENTERED".
             SET MESSAGE-LENGTH TO FOUR.
             PERFORM PROCESS-MESSAGE UNTIL MESSAGE-LENGTH < ONE.
             IF WAIT-FOR-ANSWER = NONO
                 PERFORM ASK-FOR-INPUT.
      *
         ASK-FOR-INPUT.
                 DISPLAY "ENTER PART NUMBER OR COMMAND".
                 MOVE YES TO LOCAL-FLAG.
                 MOVE YES TO NEW-BUFFER.
                 PERFORM READ-INPUT.
                 IF FIRST-CHARACTER IS NOT ALPHABETIC
                 PERFORM PART-NUMBER-PROCESSOR
                 ELSE
                 PERFORM COMMAND-PROCESSOR.
      *
         PROCESS-MESSAGE.
             MOVE "?" TO COM-FUNCTION.
             PERFORM COMMUNICATE.
             IF MESSAGE-LENGTH NOT < ONE
                 IF MESSAGE-CLASS = "C"
                 PERFORM APPLY-COMMAND
                 ELSE
                 IF MESSAGE-CLASS = "D"
                 PERFORM DISPLAY-COMMAND
                 ELSE
                 IF MESSAGE-CLASS = "A"
                     PERFORM AUDIT-COMMAND
                 ELSE
                     DISPLAY SOURCE-DESTINATION MESSAGE-CONTROL
                DISPLAY MESSAGE-DATA.
```

```
      *
       DISPLAY-COMMAND.
           DISPLAY MESSAGE-DATA.
           IF LAST-OF-MESSAGE = YES
               MOVE NONO TO WAIT-FOR-ANSWER.
      *
       APPLY-COMMAND.
           MOVE NONO TO LOCAL-FLAG.
           MOVE SOURCE-DESTINATION TO RETURN-ADDRESS.
           MOVE MESSAGE-DATA TO COMMAND.
           PERFORM TRANSACTION-PROCESSOR.
           MOVE YES TO LOCAL-FLAG.
      *
       AUDIT-COMMAND.
           DISPLAY "AUDIT TRAIL MESSAGE --".
           DISPLAY SOURCE-DESTINATION MESSAGE-CONTROL.
           DISPLAY MESSAGE-DATA.
      *
       PART-NUMBER-PROCESSOR.
           IF DEBUG = YES DISPLAY "PART NUMBER PROCESSOR ENTERED".
           MOVE NONO TO ERROR-FLAG.
           MOVE "             " TO PARTNO.
           MOVE "L" TO ACTION.
           MOVE " " TO EENTRY.
           MOVE "+" TO QUANTITY-SIGN.
           MOVE 0 TO QUANTITY.
           MOVE "                         " TO PART-NAME.
           PERFORM PARTNO-CHECK.
           IF ERROR-FLAG = NONO
               PERFORM CHECK-OTHER-FIELDS.
      *
       CHECK-OTHER-FIELDS.
           MOVE YES TO REPEAT-FLAG.
           MOVE NONO TO ERROR-FLAG.
           PERFORM ACTION-CHECK UNTIL REPEAT-FLAG = NONO.
           IF ACTION-FLAG = YES
               MOVE YES TO REPEAT-FLAG
               MOVE NONO TO ERROR-FLAG
               PERFORM ENTRY-CHECK UNTIL REPEAT-FLAG = NONO
               MOVE YES TO REPEAT-FLAG
               MOVE NONO TO ERROR-FLAG
               PERFORM VALUE-CHECK UNTIL REPEAT-FLAG = NONO.
           PERFORM TRANSACTION-PROCESSOR.
      *
       PARTNO-CHECK.
           IF DEBUG = YES DISPLAY "PARTNO CHECK ENTERED".
           MOVE NONO TO ERROR-FLAG.
           PERFORM DIGIT-CHECK
              VARYING I1 FROM 1 BY 1
                     UNTIL (I1 > 13 OR ERROR-FLAG = YES).
           IF ERROR-FLAG = NONO
               MOVE PART-NUM TO PARTNO
           ELSE
              DISPLAY "PART-NUMBERS CONTAIN ONLY DIGITS.".
      *
```

```
       DIGIT-CHECK.
           MOVE PART-NUMBER(I1) TO TEM.
           IF TEM IS NUMERIC OR TEM = " "
               NEXT SENTENCE
           ELSE
               MOVE YES TO ERROR-FLAG.
      *
       READ-DATA.
           MOVE YES TO NEW-BUFFER.
           MOVE YES TO NO-INPUT-FLAG.
           PERFORM READ-INPUT.
      *
       ACTION-CHECK.
           IF DEBUG = YES DISPLAY "ACTION CHECK ENTERED".
           IF ERROR-FLAG = NONO
               MOVE NONO TO NEW-BUFFER
               PERFORM READ-INPUT.
           IF (NO-INPUT-FLAG = YES OR ERROR-FLAG = YES)
               DISPLAY "ENTER ACTION - LIST,UPDATE OR INITIALIZE"
               PERFORM READ-DATA.
           MOVE NONO TO REPEAT-FLAG.
           MOVE LIST TO STRINGB.
           PERFORM RECOGNIZE.
           IF SAME-FLAG = YES
               MOVE "L" TO ACTION
             MOVE NONO TO  ACTION-FLAG
           ELSE
           MOVE UPDATE TO STRINGB
           PERFORM RECOGNIZE
           IF SAME-FLAG = YES
               MOVE "U" TO ACTION
             MOVE YES TO ACTION-FLAG
           ELSE
               PERFORM ACTION-CHECK1.
       ACTION-CHECK1.
           MOVE INITIALIZE TO STRINGB
           PERFORM RECOGNIZE
           IF SAME-FLAG = YES
             MOVE "I" TO ACTION
               MOVE YES TO ACTION-FLAG
           ELSE
               DISPLAY "ILEGAL ACTION CODE  - RE-ENTER"
               MOVE YES TO REPEAT-FLAG
               MOVE YES TO ERROR-FLAG.
      *
       ENTRY-CHECK.
           IF DEBUG = YES DISPLAY "ENTRY CHECK ENTERED".
           IF ERROR-FLAG = NONO
               MOVE NONO TO NEW-BUFFER
               PERFORM READ-INPUT.
           IF (NO-INPUT-FLAG = YES OR ERROR-FLAG = YES)
               DISPLAY "ENTER CODE FOR ENTRY TO BE CHANGED"
               DISPLAY "     STOCK, ON ORDER, THRESHOLD OR ORDER SIZE"
               PERFORM READ-DATA.
           MOVE NONO TO REPEAT-FLAG.
```

```
                 MOVE STOCK TO STRINGB.
                 PERFORM RECOGNIZE.
                 IF SAME-FLAG = YES
                     MOVE "S" TO EENTRY
                 ELSE
                 MOVE ON-ORDER TO STRINGB
                 PERFORM RECOGNIZE
                 IF (SAME-FLAG = YES AND AACTUAL > 1)
                     MOVE "O" TO EENTRY
                 ELSE
                     PERFORM ENTRY-CHECK1.
            *
             ENTRY-CHECK1.
                  MOVE THRESHOLD TO STRINGB
                  PERFORM RECOGNIZE
                 IF SAME-FLAG = YES
                     MOVE "T" TO EENTRY
                 ELSE
                 MOVE ORDER-SIZE TO STRINGB
                 PERFORM RECOGNIZE
                 IF (SAME-FLAG = YES AND AACTUAL > 1)
                     MOVE "Z" TO EENTRY
                 ELSE
                     DISPLAY "ILLEGAL ENTRY CODE"
                     MOVE YES TO REPEAT-FLAG
                     MOVE YES TO ERROR-FLAG.
            *
             VALUE-CHECK.
                 IF DEBUG = YES DISPLAY "VALUE CHECK ENTERED".
                 IF ERROR-FLAG = NONO
                     MOVE NONO TO NEW-BUFFER
                     PERFORM READ-INPUT.
                 IF (NO-INPUT-FLAG = YES OR ERROR-FLAG = YES)
                     DISPLAY "ENTER THE NUMBER OF ITEMS"
                     PERFORM READ-DATA.
                 IF (PART-NUMBER(1) = "+" OR PART-NUMBER(1) = "-")
****  PUNCT?
            *            THIS CODE ASSUMES 3 DIGIT PART QUANTITIES
                     MOVE PART-NUMBER(1) TO QUANTITY-SIGN
                     MOVE PART-NUMBER(2) TO PART-NUMBER(1)
                     MOVE PART-NUMBER(3) TO PART-NUMBER(2)
                     MOVE PART-NUMBER(4) TO PART-NUMBER(3)
                     SET AACTUAL DOWN BY 1.
                 IF AACTUAL > 3 SET AACTUAL TO 3
                 IF AACTUAL < 1
                     SET AACTUAL TO 1
                     MOVE "A" TO PART-NUMBER(1).
                 MOVE NONO TO ERROR-FLAG.
                 PERFORM DIGIT-CHECK
                     VARYING I1 FROM 1 BY 1
                     UNTIL (I1 > AACTUAL OR ERROR-FLAG = YES).
                 IF ERROR-FLAG = NONO
                     PERFORM RIGHT-JUSTIFY-0-FILL
                     MOVE DIGIT-3 TO QUANTITY
                     MOVE NONO TO REPEAT-FLAG
```

```
               ELSE
                   DISPLAY "ENTER NUMBERS ONLY".
          *
          *
           RIGHT-JUSTIFY-0-FILL.
               MOVE PART-NUMBER(AACTUAL) TO PART-NUMBER(3).
               IF AACTUAL = 2
                   MOVE PART-NUMBER 1) TO PART-NUMBER(2)
                   MOVE "0" TO PART-NUMBER(1)
               ELSE
               IF AACTUAL = 1
                   MOVE "0" TO PART-NUMBER(2)
                  MOVE "0" TO PART-NUMBER(1).
          *
          *------------------------------------------------------------
           TRANSACTION-PROCESSOR.
               IF DEBUG = YES
                   DISPLAY "TRANSACTION PROCESSOR ENTERED"
                   DISPLAY COMMAND.
               MOVE "S" TO COM-FUNCTION.
               MOVE SPACES TO MESSAGE-BUFFER.
               SET MESSAGE-LENGTH TO EIGHTY.
               IF REMOTE-COMMAND = YES
                   PERFORM SEND-OUT-COMMAND
               ELSE
                   PERFORM TRANSACTION-PROCESSOR1.
          *
           SEND-OUT-COMMAND.
                   MOVE REMOTE-ADDRESS TO SOURCE-DESTINATION.
                   MOVE "C" TO MESSAGE-CLASS.
                   MOVE YES TO LAST-OF-MESSAGE.
                   MOVE COMMAND TO MESSAGE-DATA.
                   MOVE YES TO WAIT-FOR-ANSWER.
                   MOVE NONO TO REMOTE-COMMAND.
                   PERFORM COMMUNICATE.
          *  A MESSAGE ASKING IF WE SHOULD WAIT FOR RESULTS WOULD BE NICE.
                   DISPLAY "YOUR REQUEST HAS BEEN SENT. WAIT FOR RESULTS".
          *
           TRANSACTION-PROCESSOR1.
               MOVE PDP11-CODE TO SOURCE-DESTINATION.
               MOVE "A" TO MESSAGE-CLASS.
               MOVE YES TO LAST-OF-MESSAGE.
               MOVE COMMAND TO MESSAGE-DATA.
               MOVE NONO TO ERROR1-FLAG.
               MOVE YES TO TRANS-FLAG.
               IF ACTION = "C"
                   PERFORM DO-COMMAND
                ELSE
                   PERFORM FIND-PART
                   IF ERROR1-FLAG = NONO
                   PERFORM EXECUTE-TRANSACTION
                   ELSE
                   CLOSE DATA-FILE1
                   MOVE PARTNO TO ERROR-PRINT
                   IF LOCAL-FLAG = YES
```

```
                        DISPLAY ERROR-LINE
                        ELSE
                        PERFORM SETUP-TO-DISPLAY
                        MOVE ERROR-LINE TO MESSAGE-DATA
                        PERFORM COMMUNICATE.
       *
        SETUP-TO-DISPLAY.
            MOVE RETURN-ADDRESS TO SOURCE-DESTINATION.
            MOVE "D" TO MESSAGE-CLASS.
            MOVE YES TO LAST-OF-MESSAGE.
            MOVE SPACES TO MESSAGE-DATA.
       *
        EXECUTE-TRANSACTION.
            IF ACTION NOT = "L"
                PERFORM PROCESS-PART.
            IF (TRANS-FLAG = YES AND ACTION NOT = "L")
****  PUNCT?
       *        SEND OUT THE AUDIT TRAIL
                PERFORM COMMUNICATE.
            IF TRANS-FLAG = YES
                PERFORM PRINT-LINE-TO-DATA-BASE
                IF LOCAL-FLAG = YES
                DISPLAY HEADING-LINE
                DISPLAY PRINT-LINE
                ELSE
                PERFORM SETUP-TO-DISPLAY
                MOVE NONO TO LAST-OF-MESSAGE
                MOVE HEADING-LINE TO MESSAGE-DATA
                PERFORM COMMUNICATE
                PERFORM SETUP-TO-DISPLAY
                MOVE PRINT-LINE TO MESSAGE-DATA
                PERFORM COMMUNICATE
            ELSE
                MOVE ENTRY TO ENTRY-ERROR
                IF LOCAL-FLAG = YES
                DISPLAY INTERNAL-ERROR-ENTRY
                ELSE
                PERFORM SETUP-TO-DISPLAY
                MOVE INTERNAL-ERROR-ENTRY TO MESSAGE-DATA
                PERFORM COMMUNICATE.
            CLOSE DATA-FILE1.
       *
        FIND-PART.
            OPEN I-O DATA-FILE1.
            MOVE PARTNO TO PRTNO.
            START DATA-FILE1 KEY EQUAL PRTNO
                INVALID KEY MOVE YES TO ERROR1-FLAG.
            READ DATA-FILE1 INVALID KEY MOVE YES TO ERROR1-FLAG.
            PERFORM MOVE-TO-PRINT-LINE.
       *
        MOVE-TO-PRINT-LINE.
            MOVE PRTNO TO PART-NUM-OUT.
            MOVE PT-NM TO PART-NAME-OUT.
            MOVE STOCK1 TO STOCK-OUT.
            MOVE ON-ORDER1 TO ON-ORDER-OUT.
```

```
               MOVE THRESHOLD1 TO THRESHOLD-OUT.
               MOVE ORDER-SIZE1 TO ORDER-SIZE-OUT.
       *
        PROCESS-PART.
               IF DEBUG = YES DISPLAY "PROCESS PART ENTERED".
               IF EENTRY = "S"
                   MOVE STOCK-OUT TO TEMP
                   PERFORM PROCESS-TEMP
                   MOVE TEMP TO STOCK-OUT
               ELSE
               IF EENTRY = "T"
                   MOVE THRESHOLD-OUT TO TEMP
                   PERFORM PROCESS-TEMP
                   MOVE TEMP TO THRESHOLD-OUT
               ELSE
               IF EENTRY = "O"
                   MOVE ON-ORDER-OUT TO TEMP
                   PERFORM PROCESS-TEMP
                   MOVE TEMP TO ON-ORDER-OUT
               ELSE
               IF EENTRY = "Z"
                   MOVE ORDER-SIZE-OUT TO TEMP
                   PERFORM PROCESS-TEMP
                   MOVE TEMP TO ORDER-SIZE-OUT
               ELSE
                   MOVE NONO TO TRANS-FLAG.
       *
        PROCESS-TEMP.
               IF ACTION = "I"
                   MOVE 0 TO TEMP.
               IF QUANTITY-SIGN = "+"
                  ADD QUANTITY TO TEMP
                   ON SIZE ERROR
                       MOVE 999 TO TEMP
                       IF LOCAL-FLAG = YES
                       DISPLAY VALUE-TOO-BIG
                       ELSE
                       PERFORM SETUP-TO-DISPLAY
                       MOVE VALUE-TOO-BIG TO MESSAGE-DATA
                       MOVE NONO TO LAST-OF-MESSAGE
                       PERFORM COMMUNICATE
               ELSE
                   IF QUANTITY > TEMP
                       MOVE 0 TO TEMP
                   ELSE
                       SUBTRACT QUANTITY FROM TEMP.
       *
        PRINT-LINE-TO-DATA-BASE.
               MOVE STOCK-OUT TO STOCK1.
               MOVE THRESHOLD-OUT TO THRESHOLD1.
               MOVE ON-ORDER-OUT TO ON-ORDER1.
               MOVE ORDER-SIZE-OUT TO ORDER-SIZE1.
               REWRITE DATA-BASE;
                 INVALID KEY  MOVE YES TO DUMMY-FLAG.
       *
```

```
       DO-COMMAND.
           IF DEBUG = YES DISPLAY "DO COMMAND ENTERED".
           IF EENTRY = "L"
               PERFORM LIST-PARTS
           ELSE
           IF LOCAL-FLAG NOT = YES
               PERFORM SETUP-TO-DISPLAY
               MOVE "REMOTE COMMAND ERROR" TO MESSAGE-DATA
               PERFORM COMMUNICATE
           ELSE
               PERFORM LOCAL-COMMAND.
      *
       LOCAL-COMMAND.
           IF EENTRY = "D"
               PERFORM DELETE-PART
           ELSE
           IF EENTRY = "A"
               PERFORM ADD-PART
           ELSE
           IF EENTRY = "I"
               PERFORM INIT
           ELSE
           IF EENTRY = "E"
               PERFORM TERMINATE-RUN
           ELSE
               MOVE COMMAND TO COMMAND-ERROR
               DISPLAY INTERNAL-ERROR-COMMAND.
      *
       TERMINATE-RUN.
               PERFORM COMMUNICATE.
              DISPLAY "END OF DAY - SAVE THE DATA-BASE".
      *
       INIT.
           PERFORM COMMUNICATE.
           OPEN INPUT DATA-FILE1.
           READ DATA-FILE1 NEXT AT END MOVE NONO TO M-D-R-F.
           CLOSE DATA-FILE1.
           DISPLAY "INITIALIZE THE DATA BASE".
      *
       LIST-PARTS.
           IF LOCAL-FLAG = YES
               DISPLAY HEADING-LINE
           ELSE
               PERFORM SETUP-TO-DISPLAY
               MOVE NONO TO LAST-OF-MESSAGE
               MOVE HEADING-LINE TO MESSAGE-DATA
               PERFORM COMMUNICATE.
           OPEN INPUT DATA-FILE1.
           MOVE YES TO M-D-R-F.
           PERFORM LST-PRTS UNTIL M-D-R-F = NONO.
           CLOSE DATA-FILE1.
           IF LOCAL-FLAG NOT = YES
               PERFORM SETUP-TO-DISPLAY
               PERFORM COMMUNICATE.
      *
```

```
        LST-PRTS.
            READ DATA-FILE1 NEXT AT END MOVE NONO TO M-D-R-F.
            IF M-D-R-F NOT = NONO
            PERFORM PRINT-A-LINE1.
       *
        PRINT-A-LINE1.
            PERFORM MOVE-TO-PRINT-LINE.
            IF LOCAL-FLAG = YES
                DISPLAY PRINT-LINE
                ELSE
                PERFORM SETUP-TO-DISPLAY
                MOVE NONO TO LAST-OF-MESSAGE
                MOVE PRINT-LINE TO MESSAGE-DATA
                PERFORM COMMUNICATE.
       *
        PRINT-A-LINE.
            IF PRTNO NOT = "          "
               PERFORM MOVE-TO-PRINT-LINE
                DISPLAY PRINT-LINE.
       *
        DELETE-PART.
            PERFORM FIND-PART.
            IF ERROR1-FLAG = NONO
                 PERFORM DELETE-RECORD
            ELSE
                MOVE PARTNO TO ERROR-PRINT
                CLOSE DATA-FILE1
                DISPLAY ERROR-LINE.
       *
        DELETE-RECORD.
            DELETE DATA-FILE1;
                INVALID KEY  DISPLAY "INTERNAL ERROR DELETE"
                     CLOSE DATA-FILE1
                     STOP RUN.
            PERFORM COMMUNICATE.
            MOVE PARTNO TO DELETE-PRINT.
            CLOSE DATA-FILE1.
            DISPLAY DELETE-LINE.
       *
        ADD-PART.
       *  IN THIS SECTION
       *      D-I-P-F ABREVIATES DATA-IS ALREADY-PRESENT-FLAG.
            PERFORM COMMUNICATE.
            PERFORM SET-BUFF.
            OPEN I-O DATA-FILE1.
            MOVE NONO TO D-I-P-F.
            WRITE DATA-BASE FROM DATA-BUFFER;
                INVALID KEY MOVE YES TO D-I-P-F.
            IF DEBUG = YES DISPLAY "MID ADD PART " D-I-P-F.
            IF D-I-P-F = YES
                PERFORM VALUE-IS-PRESENT
            ELSE
                MOVE PARTNO TO ADD-PRINT
                DISPLAY ADD-LINE.
            CLOSE DATA-FILE1.
```

```
      *
       VALUE-IS-PRESENT.
            MOVE PARTNO TO DATA-PRES-PRINT.
            DISPLAY PRESENT-LINE.
      *
       SET-BUFF.
           MOVE PARTNO TO PART-NUMB.
           MOVE PART-NAME TO PRT-NME.
           MOVE "000" TO STCK.
           MOVE "000" TO ON-ORDR.
           MOVE "000" TO THRESHLD.
           MOVE "000" TO ORDR-SIZE.
      *
      *-----------------------------------------------------------------
       READ-INPUT.
           IF NEW-BUFFER = YES
               MOVE YES TO BUFFER-EMPTY
               PERFORM GET-NEW-BUFFER UNTIL BUFFER-EMPTY = NONO.
           IF DEBUG = YES
               DISPLAY COMMAND-LINE.
           MOVE BUFFER-EMPTY TO NO-INPUT-FLAG.
           IF BUFFER-EMPTY = NONO
               PERFORM DELETE-FIRST-FIELD.
      *
       GET-NEW-BUFFER.
           IF DEBUG = YES DISPLAY "TEST GET NEW BUFFER".
           MOVE SPACES TO COMMAND-LINE.
           ACCEPT COMMAND-LINE.
           INSPECT COMMAND-LINE REPLACING ALL "," BY "/".
           SET TPTR TO ONE.
           PERFORM CLEANUP-LINE.
           IF BUFFER-EMPTY = YES
               DISPLAY "RE-ENTER LAST LINE.".
      *
       DELETE-FIRST-FIELD.
           IF DEBUG = YES DISPLAY "DELETE FIRST FIELD ENTERED".
           MOVE SPACES TO OUT-COMMAND.
           PERFORM MOVE-FIRST-FIELD
               VARYING PTR FROM 1 BY 1
                   UNTIL (COMMAND-BUFFER(PTR) = "/" OR PTR = 30).
           IF COMMAND-BUFFER(PTR) = "/"
                   SET I3 TO PTR
               MOVE " " TO COMMAND-OUT(I3)
               SET AACTUAL TO PTR
               SET AACTUAL DOWN BY 1
           ELSE
               SET AACTUAL TO PTR
               PERFORM DO-NOTHING
               VARYING PTR FROM AACTUAL BY 1
                     UNTIL COMMAND-BUFFER(PTR) = "/".
           SET PTR UP BY 1.
           SET TPTR TO PTR.
           PERFORM CLEANUP-LINE.
      *
       CLEANUP-LINE.
```

```
            SET TEM-PTR TO ONE.
            PERFORM REMOVE-BLANKS-AND-PACK
                VARYING PTR FROM TPTR BY 1 UNTIL PTR > 70.
            PERFORM BLANK-REST-OF-LINE
                VARYING PTR FROM TEM-PTR BY 1 UNTIL PTR > 70.
            IF TEM-PTR = ONE
                MOVE YES TO BUFFER-EMPTY
            ELSE
                MOVE NONO TO BUFFER-EMPTY
                SET TEM-PTR DOWN BY 1
                SET BUFFER-LENGTH TO TEM-PTR
                IF COMMAND-BUFFER(TEM-PTR) NOT = "/"
                    SET TEM-PTR UP BY 1
                    SET BUFFER-LENGTH TO TEM-PTR
                    MOVE "/" TO COMMAND-BUFFER(TEM-PTR).
      *
       REMOVE-BLANKS-AND-PACK.
            IF COMMAND-BUFFER(PTR) NOT = " "
                MOVE COMMAND-BUFFER(PTR) TO COMMAND-BUFFER(TEM-PTR)
                SET TEM-PTR UP BY 1.
      *
       MOVE-FIRST-FIELD.
            SET I3 TO PTR.
            MOVE COMMAND-BUFFER(PTR) TO COMMAND-OUT(I3).
      *
       BLANK-REST-OF-LINE.
            MOVE " " TO COMMAND-BUFFER(PTR).
      *
       DO-NOTHING.
            SET PTR TO PTR.
      *
      *---------------------------------------------------------------
       RECOGNIZE.
            MOVE YES TO SAME-FLAG.
            PERFORM COMPARE
                VARYING I2 FROM 1 BY 1
                  UNTIL (SAME-FLAG = NONO OR I2 > AACTUAL OR I2 > MAXB).
      *
       COMPARE.
            SET I1 TO I2.
            IF PART-NUMBER(I1) NOT = STRING2(I2)
               MOVE NONO TO SAME-FLAG.
      *
      *---------------------------------------------------------------
       COMMAND-PROCESSOR.
            IF DEBUG = YES DISPLAY "COMMAND PROCESSOR ENTERED".
            MOVE "          " TO PARTNO.
            MOVE " " TO ACTION.
            MOVE " " TO ENTRY.
            MOVE "+" TO QUANTITY-SIGN.
            MOVE "000" TO QUANTITY.
            MOVE "                    " TO PART-NAME.
      *  THE FOLLOWING IS A CASE STATEMENT ON THE COMMAND NAMES.
            MOVE LIST TO STRINGB.
            PERFORM RECOGNIZE.
```

```
               IF SAME-FLAG = YES
                   PERFORM LIST-PROCESS
               ELSE
               MOVE STOP-IT TO STRINGB
               PERFORM RECOGNIZE
               IF SAME-FLAG = YES
                   MOVE YES TO STOP-FLAG
               ELSE
               MOVE ADDIT TO STRINGB
               PERFORM RECOGNIZE
               IF SAME-FLAG = YES
                   PERFORM ADD-PROCESS
               ELSE
               MOVE DELETE-IT TO STRINGB
               PERFORM RECOGNIZE
               IF SAME-FLAG = YES
                   PERFORM DELETE-PROCESS
               ELSE
                  PERFORM CHECK-OTHERS.
      *
       CHECK-OTHERS.
           MOVE HELP TO STRINGB.
           PERFORM RECOGNIZE.
           IF SAME-FLAG = YES
               PERFORM HELP-PROCESS
           ELSE
           MOVE REMOTE-NAME TO STRINGB
           PERFORM RECOGNIZE
           IF SAME-FLAG = YES
               PERFORM REMOTE-PROCESS
           ELSE
           MOVE SEND-IT TO STRINGB
           PERFORM RECOGNIZE
           IF SAME-FLAG = YES
               PERFORM SEND-PROCESS
           ELSE
               PERFORM ILLEGAL-COMMAND.
      *
       ILLEGAL-COMMAND.
           DISPLAY "ILLEGAL COMMAND - ENTER HELP FOR HELP".
      *
       LIST-PROCESS.
           MOVE "C" TO ACTION.
           MOVE "L" TO EENTRY.
           MOVE NONO TO NEW-BUFFER.
           PERFORM READ-INPUT.
           IF (NO-INPUT-FLAG = NONO AND PART-NUMBER(1) = "A")
**** PUNCT?
               MOVE "A" TO QUANTITY-SIGN.
           PERFORM TRANSACTION-PROCESSOR.
      *
       ADD-PROCESS.
           MOVE "C" TO ACTION.
           MOVE "A" TO EENTRY.
           MOVE YES TO REPEAT-FLAG.
```

```
              MOVE NONO TO ERROR-FLAG.
              PERFORM GET-PART-NUMBER UNTIL REPEAT-FLAG = NONO.
              MOVE YES TO REPEAT-FLAG.
              MOVE NONO TO ERROR-FLAG.
              PERFORM GET-PART-NAME UNTIL REPEAT-FLAG = NONO.
              PERFORM TRANSACTION-PROCESSOR.
              IF TRANS-FLAG = NONO
                  DISPLAY "DATA BASE FULL. PART NOT ADDED.".
      *
       GET-PART-NUMBER.
              IF ERROR-FLAG = NONO
                 MOVE NONO TO NEW-BUFFER.
                  PERFORM READ-INPUT.
              IF (NO-INPUT-FLAG = YES OR ERROR-FLAG = YES)
                  DISPLAY "ENTER PART NUMBER"
                  PERFORM READ-DATA.
              MOVE NONO TO ERROR-FLAG.
              PERFORM DIGIT2-CHECK
                  VARYING I1 FROM 1 BY 1
                  UNTIL (I1 > AACTUAL OR ERROR-FLAG = YES).
              IF ERROR-FLAG = NONO
                  MOVE PART-NUM TO PARTNO
                  MOVE NONO TO REPEAT-FLAG
              ELSE
                  MOVE YES TO REPEAT-FLAG
                  DISPLAY "PART NUMBERS CONTAIN ONLY DIGITS".
      *
       DIGIT2-CHECK.
              IF PART-NUMBER(I1) IS NOT NUMERIC
                 MOVE YES TO ERROR-FLAG.
      *
       GET-PART-NAME.
              IF ERROR-FLAG = NONO
                  MOVE NONO TO NEW-BUFFER
                  PERFORM READ-INPUT.
              IF (NO-INPUT-FLAG = YES OR ERROR-FLAG = YES)
                 DISPLAY "ENTER PART NAME"
                  PERFORM READ-DATA.
              MOVE PART-STRING TO PART-NAME.
              MOVE NONO TO REPEAT-FLAG.
      *
       DELETE-PROCESS.
              MOVE "C" TO ACTION.
              MOVE "D" TO EENTRY.
              MOVE YES TO REPEAT-FLAG.
              MOVE NONO TO ERROR-FLAG.
              PERFORM GET-PART-NUMBER UNTIL REPEAT-FLAG = NONO.
              PERFORM TRANSACTION-PROCESSOR.
      *
       HELP-PROCESS.
              DISPLAY "SEPERATORS ARE EITHER COMMAS OR SLASES (, OR /)".
              DISPLAY "---------".
              DISPLAY " THE FOLLOWING COMMANDS ARE IMPLIMENTED:".
              DISPLAY "    HELP - PRINTS THIS LISTING".
              DISPLAY "    LIST - DISPLAYS THE DATA BASE".
```

```
           DISPLAY "   STOP - TERMINATES THE PROGRAM".
           DISPLAY "   DELETE,PART NUMBER -".
           DISPLAY "           REMOVES AN ITEM FROM THE DATA BASE".
           DISPLAY "   ADD/PART NUMBER/PART NAME -".
           DISPLAY "           ADDS AN ITEM TO THE DATA BASE".
           DISPLAY "           ALL QUANTITIES ARE SET TO 0".
           DISPLAY "   REMOTE/DESTINATION/COMMAND - SEND COMMAND TO".
           DISPLAY "           DESTINATION MACHINE AND AWAITS RESPONSE".
           DISPLAY "   SEND/DESTINATION/MESSAGE - SEND MESSAGE TO ".
           DISPLAY "           DESTINATION MACHINE".
           DISPLAY "--------".
           DISPLAY "TO MODIFY THE QUANTITIES FOR ANY ITEM ENTER".
           DISPLAY "   PART NUMBER/ACTION/ENTRY/SIGNED QUANTITY".
           DISPLAY "WHERE".
           DISPLAY "   PART NUMBER IS A STRING OF DIGITS".
           DISPLAY "   ACTION IS LIST,UPDATE OR INITIALIZE THE ITEM".
           DISPLAY "   ENTRY IS STOCK,ON ORDER,THRESHOLD, ORDER SIZE".
      *
       REMOTE-PROCESS.
           MOVE YES TO REMOTE-COMMAND.
           MOVE YES TO REPEAT-FLAG.
           MOVE NONO TO ERROR-FLAG.
           PERFORM GET-DESTINATION UNTIL REPEAT-FLAG = NONO.
           MOVE NONO TO NEW-BUFFER.
           PERFORM READ-INPUT.
           IF NO-INPUT-FLAG = YES
               DISPLAY "ENTER PART NUMBER OR COMMAND FOR REMOTE COMMAND"
               MOVE YES TO NEW-BUFFER
               PERFORM READ-INPUT.
           IF FIRST-CHARACTER IS NOT ALPHABETIC
               PERFORM PART-NUMBER-PROCESSOR
           ELSE
               MOVE LIST TO STRINGB
               PERFORM RECOGNIZE
               IF SAME-FLAG = YES
               PERFORM LIST-PROCESS
               ELSE
               DISPLAY "ILLEGAL REMOTE COMMAND".
      *
       GET-DESTINATION.
           IF ERROR-FLAG = NONO
               MOVE NONO TO NEW-BUFFER
               PERFORM READ-INPUT.
           IF (NO-INPUT-FLAG = YES OR ERROR-FLAG = YES)
               DISPLAY "ENTER DESTINATION MACHINE CODE."
               PERFORM READ-DATA.
           MOVE NONO TO REPEAT-FLAG.
           MOVE M6800 TO STRINGB.
           PERFORM RECOGNIZE.
           IF SAME-FLAG = YES
               MOVE M6800-CODE TO REMOTE-ADDRESS
           ELSE
           MOVE PDP11 TO STRINGB
           PERFORM RECOGNIZE.
           IF SAME-FLAG = YES
```

```
                MOVE PDP11-CODE TO REMOTE-ADDRESS
            ELSE
            MOVE INTEL TO STRINGB
            PERFORM RECOGNIZE
            IF SAME-FLAG = YES
                MOVE INTEL-CODE TO REMOTE-ADDRESS
            ELSE
            MOVE CS20 TO STRINGB
            PERFORM RECOGNIZE
            IF SAME-FLAG = YES
                MOVE CS-20-CODE TO REMOTE-ADDRESS
            ELSE
                PERFORM BAD-DEST-CODE.
      *
        BAD-DEST-CODE.
                DISPLAY "ILLEGAL DESTINATION CODE."
                DISPLAY "     USE M6800, PDP11, INTEL, OR CS-20"
                MOVE YES TO REPEAT-FLAG
                MOVE YES TO ERROR-FLAG.
      *
        SEND-PROCESS.
            MOVE YES TO REPEAT-FLAG.
            MOVE NONO TO ERROR-FLAG.
            PERFORM GET-DESTINATION UNTIL REPEAT-FLAG = NONO.
            DISPLAY "ENTER TEXT - EMPTY LINE WILL TERMINATE.".
            MOVE YES TO REPEAT-FLAG.
            PERFORM SEND-TEXT UNTIL REPEAT-FLAG = NONO.
      *
        SEND-TEXT.
            MOVE SPACES TO MESSAGE-BUFFER.
            ACCEPT MESSAGE-DATA.
            MOVE YES TO EMPTY-LINE.
            PERFORM CHECK-EMPTY-LINE
                VARYING MES-INDEX FROM 1 BY 1 UNTIL MES-INDEX > 70.
            MOVE REMOTE-ADDRESS TO SOURCE-DESTINATION.
            MOVE "D" TO MESSAGE-CLASS.
            MOVE NONO TO LAST-OF-MESSAGE.
            IF EMPTY-LINE = YES
                MOVE YES TO LAST-OF-MESSAGE
                MOVE NONO TO REPEAT-FLAG.
            SET MESSAGE-LENGTH TO EIGHTY.
            MOVE "S" TO COM-FUNCTION.
            PERFORM COMMUNICATE.
      *
        CHECK-EMPTY-LINE.
            IF MESSAGE-DATA1(MES-INDEX) NOT = " "
                MOVE NONO TO EMPTY-LINE.
```

13. APPENDIX E--NETWORK COBOL RESERVED WORDS

PRECEDING PAGE BLANK-NOT FILMED

13. APPENDIX E

NETWORK COBOL RESERVED WORDS

ACCEPT
ACCESS
ACCESSABILITY
ACTUAL
ADD
ADDRESS
ADVANCING
AFTER
ALL
ALPHABETIC
ALSO
ALTER
ALTERNATE
AND
APPROXIMATE
ARE
AREA
AREAS
ASCENDING
ASCII
ASSIGN
AT
AUTHOR
AUTO
BACKWARD
BEEP
BEFORE
BEGINNING
BELL
BIT
BLANK
BLINK
BLOCK
BOTTOM
BREAK-KEY
BY
C-300
CALL
CAM
CANCEL
CCNL
CD
CDAC
CDIS
CF
CHANNEL2
CHARACTER
CHARACTERS
CINT
CIOC
CLOCK-UNITS
CLOSE
CMOD
COBOL
CODE
CODE-SET
COLLATING
COLUMN
COMMA
COMMUNICATION
COMP
COMP-1
COMP-2
COMP-3
COMPRESSION
COMPUTATIONAL

COMPUTATIONAL-1
COMPUTATIONAL-2
COMPUTATIONAL-3
COMPUTE
CONFIGURATION
CONSOLE
CONTAINS
CONTIGUOUS
CONTROL
CONTROLS
COPY
CORR
CORRESPONDING
COUNT
CR
CRCV
CREATE
CS-20
CS-40
CS-60
CSND
CURRENCY
DATA
DATA-SENSITIVE
DATE

DATE-COMPILED
DATE-WRITTEN
DAY
DE
DEBUG-CONTENTS
DEBUG-ITEM
DEBUG-LINE
DEBUG-NAME
DEBUG-SUB-1
DEBUG-SUB-2
DEBUG-SUB-3
DEBUG-SUB1
DEBUG-SUB2
DEBUG-SUB3
DEBUGGING
DECIMAL-POINT
DECLARATIVES
DEFINE
DELETE
DELIMITED
DELIMITER
DEPENDING
DESCENDING
DESTINATION
DETAIL

DISABLE
DISK
DISPLAY
DIVIDE
DIVISION
DOWN
DUPLICATES
DYNAMIC
EBCDIC
ECLIPSE
EGI ELSE
EMI
ENABLE
END
END-OF-PAGE
ENDING
ENTER
ENVIRONMENT
EQUAL
EQUALS
ERROR
ESI
EVEN
EVERY
EXCEPTION

EXCLUDE	GENERIC	INITIATE
EXCLUSIVE	GIVING	INPUT
EXHIBIT	GLOBAL	INPUT-OUTPUT
EXPIRATION	GO	INSPECT
EXPUNGE	GREATER	INSTALLATION
EXTEND	GROUP	INTO
FD	HEADER	INVALID
FEEDBACK	HEADING	INVERTED
FIELD	HIERARCHICAL	IS
FIELDS	HIGH	JUST
FILE	HIGH-VALUE	JUSTIFIED
FILE-CONTROL	HIGH-VALUES	KEY
FILE-ID	I-O	KEYBOARD
FILE-LIMIT	I-O-CONTROL	KEYS
FILE-LIMITS	ID	LABEL
FILLER	IDENTIFICATION	LABELS
FINAL	IF	LAST
FIRST	IMMEDIATE	LEADING
FIXED	IN	LEFT
FOOTING	INDEX	LENGTH
FOR	INDEXED	LESS
FORWARD	INDICATE	LEVELS
FROM	INFOS	LIBRARY
GENERATE	INITIAL	LIMIT
GENERATION	INITIALIZATION	LIMITS

LINAGE
LINAGE-COUNTER
LINE
LINE-COUNTER
LINES
LINK
LINKAGE
LOCAL
LOCK
LOGICAL
LOW-VALUE
LOW-VALUES
LRU
MANAGEMENT
MAXIMUM
MEMORY
MERGE
MERIT
MESSAGE
MODE
MODULES
MOVE
MULTIPLE
MULTIPLY
NAMED

NATIVE
NEGATIVE
NEXT
NO
NODE
NOT
NUMBER
NUMERIC
OBJECT-COUNTER
OCCURANCE
OCCURS
ODD
OF
OFF
OFFSET
OH
OMITTED
ON
ONLY
OPEN
OPTIONAL
OR
ORGANIZATION
OUTPUT
OV

OVERFLOW
OWNER
PAD
PAGE
PAGE-COUNTER
PARITY
PARTIAL
PERFORM
PF
PH
PHYSICAL
PIC
PICTURE
PLUS
POINTER
POSITION
POSITIVE
PRINTER
PRINTING
PROCEDURE
PROCEDURES
PROCEED
PROCESSING
PROGRAM
PROGRAM-ID

QUEUE
QUOTE
QUOTES
RANDOM
RD
READ
READY
RECEIVE
RECORD
RECORDING
RECORDS
REDEFINES
REEL
REFERENCES
RELATIVE
RELEASE
REMAINDER
REMARKS
REMOVAL
RENAMES
REPLACING
REPORT
REPORTING
REPORTS
RERUN

RESERVE
RESET
RETAIN
RETRIEVE
RETURN
REVERSED
REWIND
REWRITE
RF
RH
RIGHT
ROOT
ROUNDED
RUN
SAME
SAVE
SCREEN
SD
SEARCH
SECTION
SECURE
SECURITY
SEEK
SEGMENT-LIMIT
SELECT

SELECTED
SEND
SENTENCE
SEPARATE
SEQUENCE
SEQUENTIAL
SET
SIGN
SIZE
SORT
SORT-MERGE
SOURCE
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
STANDARD
STANDARD-1
STANDARD-2
STANDARD-3
START
STATIC
STATUS
STOP
STRING

SUB-INDEX
SUB-QUEUE-1
SUB-QUEUE-2
SUB-QUEUE-3
SUBTRACT
SUM
SUPPRESS
SWITCH
SYMBOLIC
SYNC
SYNCHRONIZED
TABLE
TALLY
TALLYING
TAPE
TEMPORARY
TEMINAGE
TERMINAL
TERMINATE
TEXT
THAN
THEN
THROUGH
THRU
TIME

TIMES
TO
TOP
TRACE
TRAILER
TRAILING
TRUNCATE
TYPE
UNDEFINED
UNDELETE
UNIT
UNLOCK
UNSTRING
UNTIL
UP
UPON
USAGE
USE
USER
USING
VALUE
VALUES
VARIABLE
VARYING
VERIFY

VOLUMN
WAIT
WHEN
WITH
WORDS
WORKING-STORAGE
WRITE
XECS
XMOD
XNMT
XPND
XTRN
ZERO
ZEROES
ZEROS