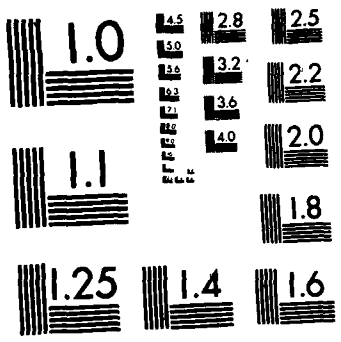
AD-A124 874 CONTINUED DEVELOPMENT AND IMPLEMENTATION OF THE PROTOCOLS FOR THE DIGITAL..(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. C H HAZELTON 1/3

UNCLASSIFIED DEC 82 AFIT/GE/EE/82D-37 F/G 9/2 NL


1.0
4.5
5.0
5.6
6.3
7.1
8.0
9.0
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

1

CONTINUED DEVELOPMENT AND IMPLEMENTATION OF THE PROTOCOLS FOR THE DIGITAL ENGINEERING LABORATORY NETWORK

THESIS

AFIT/GE/EE/82D-37

Craig H. Hazelton
Capt USAF

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY (ATC)
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GE/EE/82D-37

CONTINUED DEVELOPMENT AND IMPLEMENTATION OF THE PROTOCOLS FOR THE DIGITAL ENGINEERING LABORATORY NETWORK

THESIS

AFIT/GE/EE/82D-37

Craig H. Hazelton
Capt USAF

AFIT/GE/EE/82D-37

CONTINUED DEVELOPMENT AND IMPLEMENTATION OF THE PROTOCOLS FOR THE DIGITAL ENGINEERING LABORATORY NETWORK

THESIS

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

by
Craig H. Hazelton, B.S.
Capt USAF
Graduate Electrical Engineering

December 1982

Preface

This report presents further design and implementation of the Air Force Institute of Technology's Digital Engineering Laboratory Network (DELNET) operating system protocols. It is hoped that the protocol designs, of this thesis effort, will provide a firm base on which future and continued research can depend.

I would like to take this opportunity to express my sincere appreciation to Dr. Gary Lamont, my thesis advisor, for his guidance and patience during this effort. Dr. Lamont's philosophy of the thesis effort focusing on the aspects of learning and quality rather than mere quantity has made this thesis effort a truly rewarding experience. I would also like to thank Capt. Geno Cuomo whose concurrent thesis efforts provided me with assistance and insight into the hardware aspects of this project. I would like to thank the personnel of the DEL, especially Capt. Lee Baker and Mr. Dan Zambon, for their assistance and enthusiasm in all matters relating to this project. I would like to extend my appreciation and best wishes to my fellow students whose support and humor have made this project an enjoyable experience.

Foremost, I would like to dedicate this thesis to my wife, Karen. She has provided me with love and support for over fourteen years. For this, I will be eternally grateful.

Contents

	Page
Preface	ii
List of Figures	v
List of Tables	vi
Abstract	vii
I. Introduction	1- 1
Historical Perspective	1- 2
Background	1- 4
Problem and Scope	1- 6
Approach/Objectives	1- 6
Overview of the Thesis	1-11
II. Functional Requirements and Standards	2- 1
Introduction	2- 1
Global Requirements	2- 1
Flexibility	2- 1
Virtual Operation	2- 2
Performance Monitoring	2- 3
Global Standards	2- 3
Physical Layer	2- 6
Data Link Layer	2- 6
Network Layer	2- 6
Transport Layer	2- 6
Session Layer	2- 7
Presentation Layer	2- 7
Application Layer	2- 7
System Requirements	2- 7
Packet Switching Protocol	2- 8
Routing Techniques	2- 8
System Standards	2- 9
Detailed Requirements	2-10
Network Operating System	2-10
Detailed Standards	2-13
Physical Layer	2-13
Data Link Layer	2-15
Network Layer	2-15
Summary	2-18

Page

III. The Physical Layer 3- 1

Introduction 3- 1
Theory 3- 1
Twisted Wire Pair 3- 2
Coaxial-Cable 3- 3
Fiber Optics 3- 3
DELNET Implementation 3- 5
Summary 3- 6

IV. Data Link Layer 4- 1

Introduction 4- 1
Design Considerations 4- 1
DELNET Design and Implementation 4- 4
Summary 4-16

V. The Network Layer 5- 1

Introduction 5- 1
Design Considerations 5- 1
DELNET Design and Implementation 5- 3
Summary 5-13

VI Software Configuration and Validation 6- 1

Introduction 6- 1
Test Environment and Software Configuration . . 6- 1
DELNET Software Configuration 6- 3
Software Test and Validation 6- 9
Summary 6-15

VII. Conclusions and Recommendations 7- 1

Conclusions 7- 1
Recommendations 7- 4

Bibliography . . . . Bib- 1

Appendix A: Data Dictionary . . . . A- 1

Appendix B: Data Dictionary Cross Reference . . . . . . B- 1

Appendix C: Software Listings . . . . C- 1

Section I - Local Operating System . . . . . . C- 2
Section II - Network Operating System C-50
Section III - Shared Compnents C-87

Vita V-1

List of Figures

Figure		Page
1	Initial DELNET Configuration	1-10
2	ISO Protocol Model with UNID	2- 5
3	Protocol Hierarchy at the Systems Level	2-11
4	Link Access Procedure Frame Format	2-16
5	Hierarchy of DELNET Standards	2-17
6	DELNET Frame Format Scheme	4- 6
7	Main Driver for Network Operating System	4- 9
8	Route_In Procedure for Network Operting System . . .	4-10
9	Route_out Procedure for Network Operating System . .	4-11
10	Subordinate Procedures for Network Operating System.	4-12
11	Network Layer's View of Data Packet	5- 5
12	Local and Network Operating Systems' Table Buffers .	5- 6
13	Main Driver for Local Operating System	5- 8
14	Route_In Procedure for Local Operating System . . .	5- 9
15	Route_out Procedure for Local Operating System . . .	5-10
16	Subordinate Procedures for Local Operating System .	5-11
17	UNID Memory and Processor Configuration	6- 4

List of Tables

Table Page

2 Hierarchy of DELNET Requirements 2-14

1 Software Modules Implementing DELNET Operating System 6- 6

Abstract

Development of the Air Force Institute of Technology's Digital Engineering Laboratory Network (DELNET) was continued with the design and implementation of the first three layers of the DELNET's Operating System protocol structure. This effort centered on the actual software module development and their relationships to established standards for local area network protocol structures. The overall system organization and protocol structure followed the recommendations of the International Standards Organization's (ISO) Reference Model for Open Systems Interconnections. Within these guidelines, the Data Link Layer was developed utilizing a selected subset of the Link Access Procedure Protocol (LAP) adapted by the Consultive Committee for International Telephone and Telegraph (CCITT). The third protocol layer was developed utilizing an appropriate subset of the X.25 Packet Switching Protocol standards which were also adapted by of the CCITT. This study formulates the specific requirements, designs, and implementations of these protocol layers and presents recommendations for future research and development.

I. INTRODUCTION

The purpose of this thesis investigation is to continue the design and implementation of the software necessary to perform intercommunications between host-to-host computer devices and host-to-node computer devices for the Air Force Institute of Technology's (AFIT) Digital Engineering Laboratory Network (DELNET). AFIT's DELNET is a proposed local computer network (LCN) which will interconnect a series of independent stand alone minicomputers and microcomputers confined within a local area of the Digital Engineering Laboratory (DEL). The DELNET in turn will eventually connect to the AFIT Local Area Network (ALAN). Much of the research for this thesis effort is sponsored by the Rome Air Development Center (RADC) located at Griffis AFB, NY. under AFIT's post doctorial reseach program.

A tremendous concentration of research has been spawned by industry to place computers and their peripheral devices into LCNs (Ref 11). The driving force for this intense research is two fold. First is the tremendous power gained by combining several processors to accomplish a single task or multiplicity of tasks. And the second is the tremendous savings involved in sharing both software and hardware resources by reducing duplication of effort. The initial dominant influences for network design has evolved from private vendors whose designs were implemented with vendor unique hardware and software (Ref 19). Fortunately, much of the software design occured during the same time period as the general computer software community was transitioning to modularized design and top down analysis. For this reason, throughout industry there has been a general acceptance of structured design levels or rules called 'protocols'. It is through this philosophy of strictly defined levels of protocols that this investigation is based.

Historical Perspective

Since the advent of modern computers in the late 1940's, researchers have continually attempted to reduce the physical size and increase the speed of computers. As these goals are achieved, computers generally become more accessible and flexible in their applications. Several milestones in recent electronic history have made this goal a reality. The first was the development of the transistor and solid state electronics. The second was the development of integrated circuits and their large scale integration (Ref 6). Additional developments include the advancements in transmission line technology using broadband techniques rather than the more conventional baseband (Ref 12). Computers have become so diversified and relatively inexpensive that their availability is well within the reach of practically all institutions and many individuals.

But even with the greatly improved accessability and reduced size and cost of computers, many limitations still exist. For example, large main frame computers are required for a wide variety of applications. These computers remain large, expensive, and often out of reach of many potential users. Additionally, small minicomputers and microcomputers have limitations such as relatively small memories and slow computational speeds. Peripheral equipment such as secondary storage devices are costly and are only used a fraction of the time when they are connected to dedicated small computers. Large data bases are not only costly but must be shared by many users simultaneously in order to be effective and cost efficient. These are just a few reasons why networking has become a necessity. By interconnecting computers into a network, each computer's capability can be greatly increased and the costs of both hardware and software can be significantly reduced by sharing valuable resources.

Often when new technology is developed, additional forms of technology must be developed to function as a technological buffer or bridge before the new technology can be applied. The time span from first conception to actual application may be several years. Fortunately this is not the case for the development of LCNs. The phenominal advances in large scale integration which have made the advent of minicomputers and microcomputers a reality are also the vehicle which makes it possible to develop the interfaces for placing these computers into networks. These network interface units (NIU) are normally microprocessor controlled devices with inputs, outputs, and memory; the same components which compose the computers themselves. In fact, the NIUs may be thought of as special purpose computers which are architecturally designed for interfacing with other computers for the purpose of routing information into and out of the network.

Background

It became readily apparent to the military that the distributed processing and resource sharing of the LCNs were of great importance from both an economical and operational viewpoint. In 1977 a technical report was produced by the 1842 Electrical Engineering Group at Scott AFB, Ill (Ref 23). This report stated the necessity for computer/communications networks and presented assessments on the feasibility and economics concerning such a network. This report included a scenario for a typical military facility communications network which incorporates a multi-ring topology. The report specified five distinct types of NIUs to be used as the distributed communication concentrators. A 1980 AFIT thesis concluded that these five NIU types could be combined into a single universal NIU (Ref 3). Later that same year, another AFIT MS student designed a prototype Universal Network Interface Device (UNID) as his AFIT MS thesis (Ref 2). In late 1981, an upgraded prototype UNID was constructed and partially demonstrated as part of another AFIT MS thesis effort (Ref 17). Although the UNID was successfully demonstrated in part, it had several hardware design problems which required upgrading before its full capabilities could be demonstrated. Concurrent with this thesis effort, a continuation of the upgrade to the

UNID is being conducted (Ref 4).

As the UNID development progressed, the AFIT DEL became interested in using the UNID as the DELNET's interface medium (Ref 11). With its abundance of minicomputers and microcomputers, the DEL was an ideal environment to test both the UNID's suitability as a NIU and the feasibility of interconnecting the DEL's varied population into a LCN. There were three basic advantages of placing the DEL into a network. The first two of resource sharing and distributed processing were previously discussed. The third reason is that the DELNET would provide an ideal vehicle for state of the art research and study by AFIT students and faculty.

As a result of this interest, AFIT sponsored another thesis project to identify requirements and specify an initial design for the DELNET. That thesis included in the performance capabilities: virtual system transparency, software tool sharing, peripheral device sharing, file transfer, potential for additional network interface, and a distributed data base applications (Ref 11). The thesis included in its hardware specifications: loop network topology, a two UNID system connected by a fiber optic link for the network bus, and three host computers (Ref 11).

Another AFIT MS thesis effort continued the design of the DELNET from a software viewpoint (Ref 9). That investigation determined the overall structure of the DELNET protocols and partially implemented and successfully demonstrated several modules of software from both the local and network side of the UNID.

Problem and Scope

This study focused on the areas necessary that would make the UNID functional at its minimum level. A minimum of operations should consist of being able to have one host computer interject a message packet through its UNID and onto the network. A second UNID should be able to sieze the packet and properly route it to its appropriate host.

The purpose of this study was to continue the development of the host-to-host and host-to-node protocols required for DELNET implementation. It included continuation of development and implementation of the UNID operations, the implementation of a local and network operating systems for the UNID, and a generalized methodology for the host computer implementation.

Approach/Objectives

The intial approach in solving the problem of network design consisted of performing an extensive literature search to gather as much of the available information as possible. The enormous volumes of information was indicative of the attention focused on computer and communication networks in recent years. In addition to this information, all previous theses from AFIT that pertained to the DELNET and UNID were studied.

It is the objective of this investigation to build upon these previous thesis efforts and continue the protocol implementation and software development. As in the previous research, the main framework for the protocol structures is a modularized structure using a top down approach. It is also important to use established documented standards within this framework. This effort enables follow on development to continue with a minimum of rework; it enables modifications and revisions easily and efficiently; and it reduces the cost of additions.

The philosophy of top down design is sound and although it is seldom used, it is the accepted standard throughout industry (Ref 14). However, for actual implementation it has several drawbacks. For example, if the design begins at the highest level and sequences downward to smaller sub-modules, the proper perspective is maintained but the overall structure cannot be exercised until the bottom most module is complete. This is especially true if the top modules must use the lower modules to perform their tasks. The development of the DELNET's operating system combines the hierarchial structure of the protocol and structured programming which comprise the software for developing this protocol. This development is analogous to the construction of a high rise building. The design begins from the architectural view of the entire building and gradually becomes more specific and narrow in scope until finally the intricate details are designed. The actual construction of the building on the other hand, must begin on the ground level and slowly move toward the larger final product. The protocol hierarchy of the DELNET operating system is similar to that of a high rise building. Each protocol layer is like a level of the building of which the lower levels must be climbed before reaching the top. A similar analogy can be drawn between the hierarchial structure of the protocol levels and that of the top down design of structured programming. This comparison is not subtle, however. It is through the new Systems Engineering philosophy of structured analysis that both were derived.

While the overall framework and global design of the DELNET operting system is based on the top down approach, the actual implementation will be based on a bottom up approach. The lower levels of protocol are established, tested, and built upon. In this manner, the basic network functions can be used until the higher levels can be developed. Additionally, the lower levels can be used to help test and verify the upper levels during implementation. In using this approach two important points must be continually addressed. The first is that even though the levels are being implemented from the bottom, they must conform to the overall framework established in the top down design. And secondly, each protocol level in itself is treated as a complete entity and is designed in the top down fashion. Keeping these facts foremost, the design of the network began.

It was not feasable to attempt to incorporate all the minicomputers and microcomputers in the DEL for current thesis efforts. For this reason a small subset was chosen. The most logical choice was the Zilog MCZ 1/25 microcomputer which was used as the software development system for the DELNET. The most attractive aspect of this system is that it is dedicated to this thesis effort and was available at any time. The additional choices for implementation included the DEC LSI-11, DEC Vax 11/780, and Data General Eclipse S/250. These latter choices were made because each uses the Pascal programming language and support a variety of highly desirable peripheral equipment. The primary reasons for selecting Pascal as the programming language were due to its modularized structured design, and its availability to AFIT students both in hardware and detailed instruction. The proposed initial DELNET configuration is shown in Figure 1.

The initial plan was to incorporate the MCZ 1/25 and LSI-11 microcomputers into the network. The MCZ was chosen because of its availability and dedication to this project. The LSI-11 was chosen due to the familiarization to the personnel involved with this project and its availaility during the time period of this investigation. As the software for succesive levels of protocol was developed, it was to be tested and validated. When the system was able to properly transmit, route, and receive message packets from one host to the next, the additional computers were to be incorporated into the network.

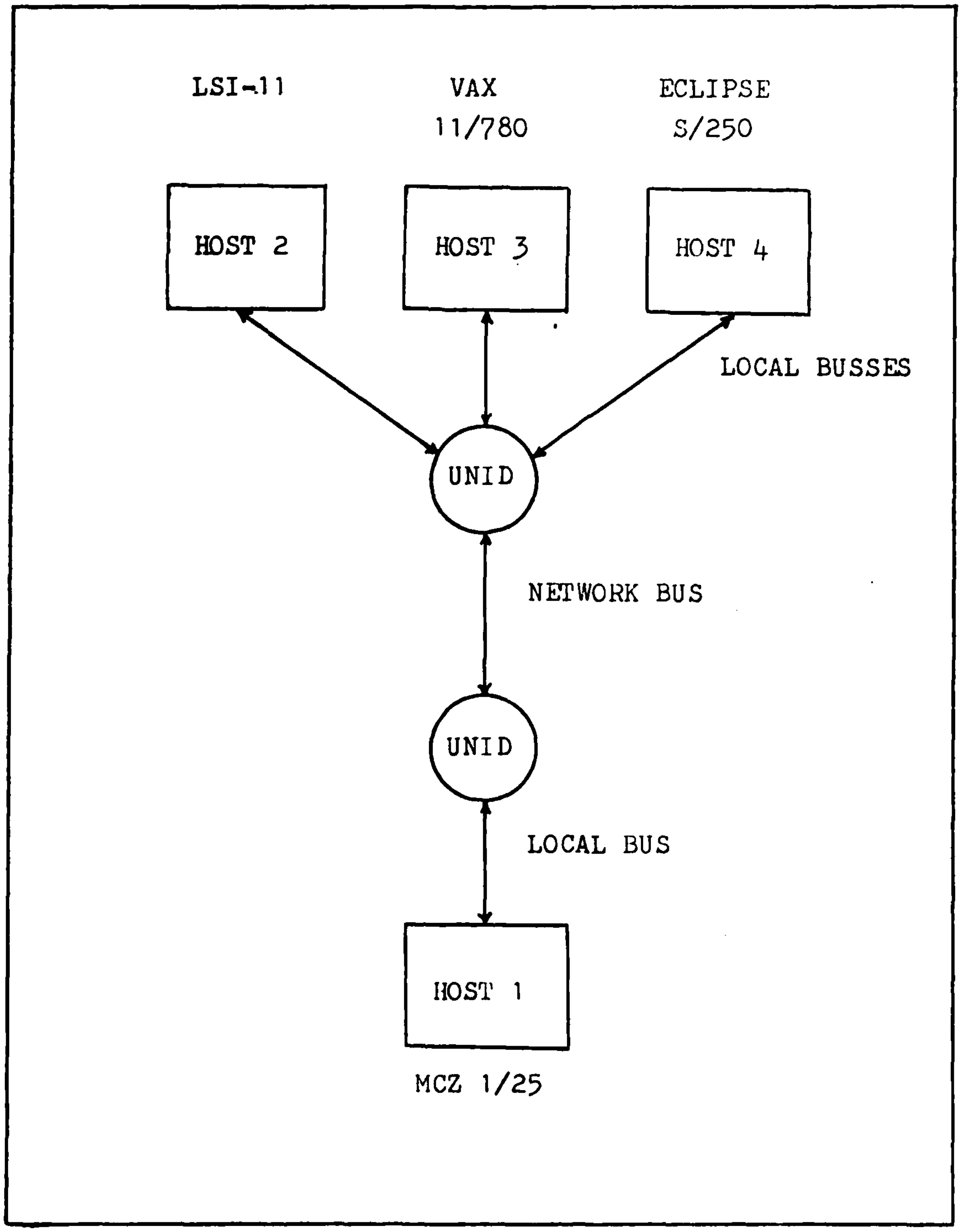


Figure 1. Initial Delnet Configuration

Overview of Thesis

The overall structure of this thesis parallels the developmental structure of the DELNET protocols. Chapter II presents the overall network structure which discusses the functional requirements and standards used in the protocol development. Subsequent chapters present a single level of protocol and its design and implementation. Chapter VI describes the software configuration, testing, and validation of the DELNET operating system. The final chapter summarizes the report and makes recommendations for future research and development. The appendices contain software and supporting documentation for the main portions of this thesis effort.

II. Functional Requirements and Standards

Introduction

This chapter introduces the functional requirements which are applicable to this project and the standards which govern it. These requirements and standards result from the initial design of the UNID and DELNET which were addressed in Chapter I. This chapter is separated into three sections: global requirements, system requirements, and detailed requirements.

The global level requirements are those which apply in general to all phases of the project. System level requirements are those which have multiple influences including the global requirements, technical aspects of the LCNs, and constraints of initial DELNET configuration. The detailed level requirements are those network functions which specifically apply to the operation and application of the DELNET. The standards which govern these requirements are presented in each section.

Global Requirements

Global requirements are basically abstractions and deal with the characteristics of an idealized system. They were initially defined by a 1980 AFIT Thesis (Ref 11) as the "Design-Oriented Functional Requirements". They include flexibility, virtual operation, and network performance monitoring.

Flexibility. Flexibility is a term which has become increasingly used in recent years whenever discussing new technology and designs. Because of growing costs, specifically in the area of new system design and development, it has become necessary to build in 'flexibility' so that a product can be used for a variety of tasks with little or no modifications. This philosophy is sound and has often been proven to save both time and money. Care must be taken, however, not to over design a new system or else it may become too universal in nature (Ref 8). In doing so, the system may not function optimally in any application and the cost may exceed the combined cost of individual nonflexible specific designs.

The design of the DELNET protocols must be such as to accomplish the specific objectives of the DELNET, but flexible enough to allow for expansion and reconfiguration. Just because the UNID is the device for interfacing the hosts into the DELNET, this does not imply that the protocols for the DELNET will necessarily function on a universal basis for all applications where the UNID is used. The flexibility, as it applies to the DELNET, specifically addresses the ease in which hardware, software, topologies, and network concepts can be modified. This is especially true in relation to the continual changes and upgrading of minicomputers and microcomputers which are presently being used in the DEL.

Virtual Operation. Virtual operation implies that within the DELNET, one host can communicate with another host on the same level of protocol. For example, in transferring files, the user only generates the proper command and the transfer takes place. The formatting of the file into packets, tagging the packet with the headers and trailers, and routing the the packet are all transparent to the user. These services take place at various levels throughout the protocol hierarchy. The user simply communicates at the user interface level. The transparency or virtual operation is an essential element of an efficient and effective network. Tradeoffs may be made to develop workable systems within the scope of this project.

Performance Monitoring. The ability to monitor the performance of the DELNET would be of great benefit for a variety of reasons. First, it would provide a means of testing and verifying proper system performance during its development. Secondly, it would provide a means of collecting data on the system for the purpose of real time evaluations and fault analysis, or more simply to insure proper operation once the DELNET becomes operational. Thirdly, future modifications can use the monitor to insure proper integration into the DELNET.

Global Standards

Many of the specific protocols for the DELNET are not directly transferable to other networks which use the UNID. This is because many of the services that the DELNET operating system provides pertain only to networks with similar characteristics such as topology, routing schemes, and flow control. The general framework for developing these protocols, however, can be used to develop or modify the specific protocols for other applications. There are many global schemes or standards used by industry for developing their protocol frameworks for LCNs. Most of these standards are derived from the Consultive Committee for International Telephone and Telegraph (CCITT), the International Standards Organization (ISO), the American National Standards Institute (ANSI), the Electronic Industries Association (EIA), or a proprietary standard design from a particular vendor (Ref 7).

At the present, most of these standards do not address the global aspect of protocol design, but rather concentrate on specific levels of protocol. The ISO has, however, developed one of the first complete global models for general LCN applications. It is called the Reference Model of Open Systems Interconnection (OSI). The ISO is a seven layer protocol model. Each layer in turn is governed by additional more specific standards (Ref 33). Figure 2 shows a pictorial representation of the ISO model.

In developing this model the ISO considered several important points. First, each abstraction of communication should be placed into its own protocol level. Second, this communication level should perform a specific function. Third, these functions should minimize flow across the protocol layer boundry. Fourth, the protocol layer boundries should be chosen to minimize data flow across interfaces. And lastly, each layer should be manageable and yet be able to support its function (Ref 33).

The following is a global description of the ISO seven layer protocol model (Ref 22).

The Physical Layer. This is the lowest and most basic link in the network. It deals with the physical realities of the network and is concerned with the transmitting of raw bits over a channel. It deals with connections, voltage levels, and transmission rates.

The Data Link Layer. This level creates, recognizes, and governs the flow of the logical bits created in the physical layer. This is generally accomplished by creating frames or packets of data. The effort placed into this layer will allow the next level (Network Layer) to accomplish its task in a more efficient manner.

Network Layer. This layer is concerned with the routing and management of the data packets. It largely determines the host-to-node interface and is subject to substantial design attention with concerns over the division of labor between the host and the node.

Transport Layer. This layer is concerned with establishing communication paths between hosts. It is sometimes referred to as the host-to-host layer. It manages buffer space and controls data flow. This is the highest layer concerned with the transport services and normally functions with communications taking place from a source host to a destination host with the NIU being transparent to

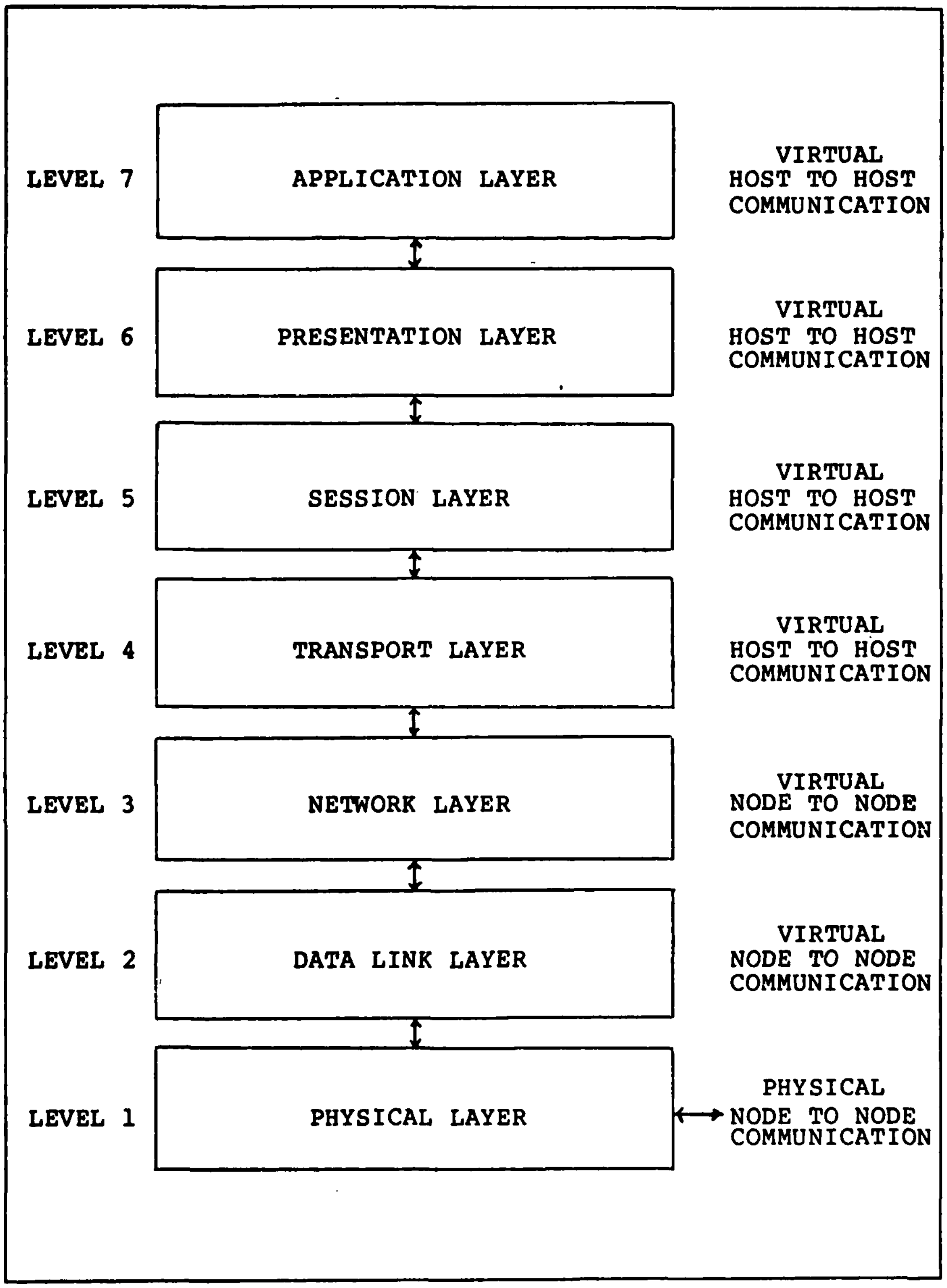

LEVEL 7
APPLICATION LAYER
VIRTUAL HOST TO HOST COMMUNICATION
LEVEL 6
PRESENTATION LAYER
VIRTUAL HOST TO HOST COMMUNICATION
LEVEL 5
SESSION LAYER
VIRTUAL HOST TO HOST COMMUNICATION
LEVEL 4
TRANSPORT LAYER
VIRTUAL HOST TO HOST COMMUNICATION
LEVEL 3
NETWORK LAYER
VIRTUAL NODE TO NODE COMMUNICATION
LEVEL 2
DATA LINK LAYER
VIRTUAL NODE TO NODE COMMUNICATION
LEVEL 1
PHYSICAL LAYER
PHYSICAL NODE TO NODE COMMUNICATION

Figure 2. ISO Protocol Model

the service.

Session Layer. This layer is the user's interface into the network by establishing a connection or session to manage the dialoque in an orderly fashion. An example may be time sharing or the transferring of a file from one host to another. The service includes setting up the connection, establishing agreement on the session options (called binding), managing the session, and disconnecting the session upon task completion.

Presentation Layer. This layer performs library functions for the network such as the transfer of files, format configuration, text compression, encryption, etc. The presentation layer attempts to alleviate inconsistencies in the network faced by different host users.

Application Layer. The content of this layer is determined by the users. They are normally application dependent but many services are common in nature such as file transfer and remote job execution.

Depending upon the size, complexity, and general application of the network, the three top levels of protocol may become blurred as to their specific tasks. In fact, in small special purpose systems, the three top levels may be grouped together into a single protocol layer called the "Applications Layer" (Ref 22).

System Requirements

The global requirements specified in the previous section dealt with the rather abstract qualites of the system such as virtual operation, flexibility, and performance monitoring. At the systems level these requirements become more specific. At the lower levels of the ISO seven layer model the system requirements are quite explicit. But as the levels increase so does their complexity. In fact, the complexity evolves into abstraction as the upper protocol levels are reached. This is due to the application dependence of the upper levels and inability to fix requirements and standards to systems that are application variant. For this reason, this section concentrates on the lower levels of protocol where the system requirements are well defined and yet remain under the requirements defined at the global level.

Packet Switching Protocol. The global requirement of flexibility establishes the necessity for a packet switching data transfer technique rather than that of dedicated physical connections. Additionally, the transparency requires all forms of data to be processed similarly. Efficiency requires the potential for parallel processing with time division multiplexing of message portions. Again, packet switching meets these requirements while meeting the transmission bandwidth (Ref 9). For these reasons the DELNET will use packet switching, store-and-foward protocol.

Routing Techniques. Routing algorithms can greatly influence the effectiveness of a network. This is especially true for multipath networks which use dynamic routing schemes (Ref 20). One advantage of implementing the DELNET with a loop or ring topology and store-and-foward packet switching is that the routing technique is relatively simple. It simply enables the UNID to interface a data packet into the normal traffic of the network. The more important aspect of this issue relates to the flexibility requirement of the network. The protocols developed for this investigation should be capable of absorbing additional nodes and host into the network.

System Standards

Using the ISO seven layer model as the global framework for the network standards, many specific standards exist for implementation of the bottom three levels of the protocol model (Ref 22). Few specific standards exist, however, for the top four levels since they are abstract in nature and tend to pertain to the specfic system and its applications. For this reason, this section will focus on the system requirements for the bottom three layers of protocol.

The (CCITT) has developed an international standard protocol for the bottom three layers of the ISO model. This standard is known as the X.25 standard (Ref 7). Investigation of other commercially available protocols found serious deficiencies including vendor dependent equipment, lack of technical sophistication, and the overdependence of specific hardware. The versatility of the X.25 standard and its endorsement by the CCITT led to its acceptance as the access protocol for the DELNET (Ref 11).

The X.25 recommendation defines the three layers of protocol through references to the X.21 standard for the Physical Layer, Link Access Procedure (LAP) for the Data Link Layer, and packet control at the Network Layer (Ref 22). Figure 3 shows the protocol structure at the systems level.

Detailed Requirements

At the detailed requirements level, the specific network functions to be encountered by this project are well defined. They include the operating system for the network and the application functions.

Operating System for the Network.

There are two basic approaches available for implementing an operating system within the DELNET. The first is to have one host which functions as a central node and control for the entire ring. Each host would route its message to this central host which in turn would perform any necessary conversions or network control functions and route the packet to the original destination. The central control node would thus have the majority of the network operating system contained within its memory. It would control all access commands and control the network functions. The second method that could be employed to implement the operating system for a network is to have each host within the ring function independently and on the same level. In this case the network operating system would be stored within the memory of each system host. Either method satisfies the requirement of a network

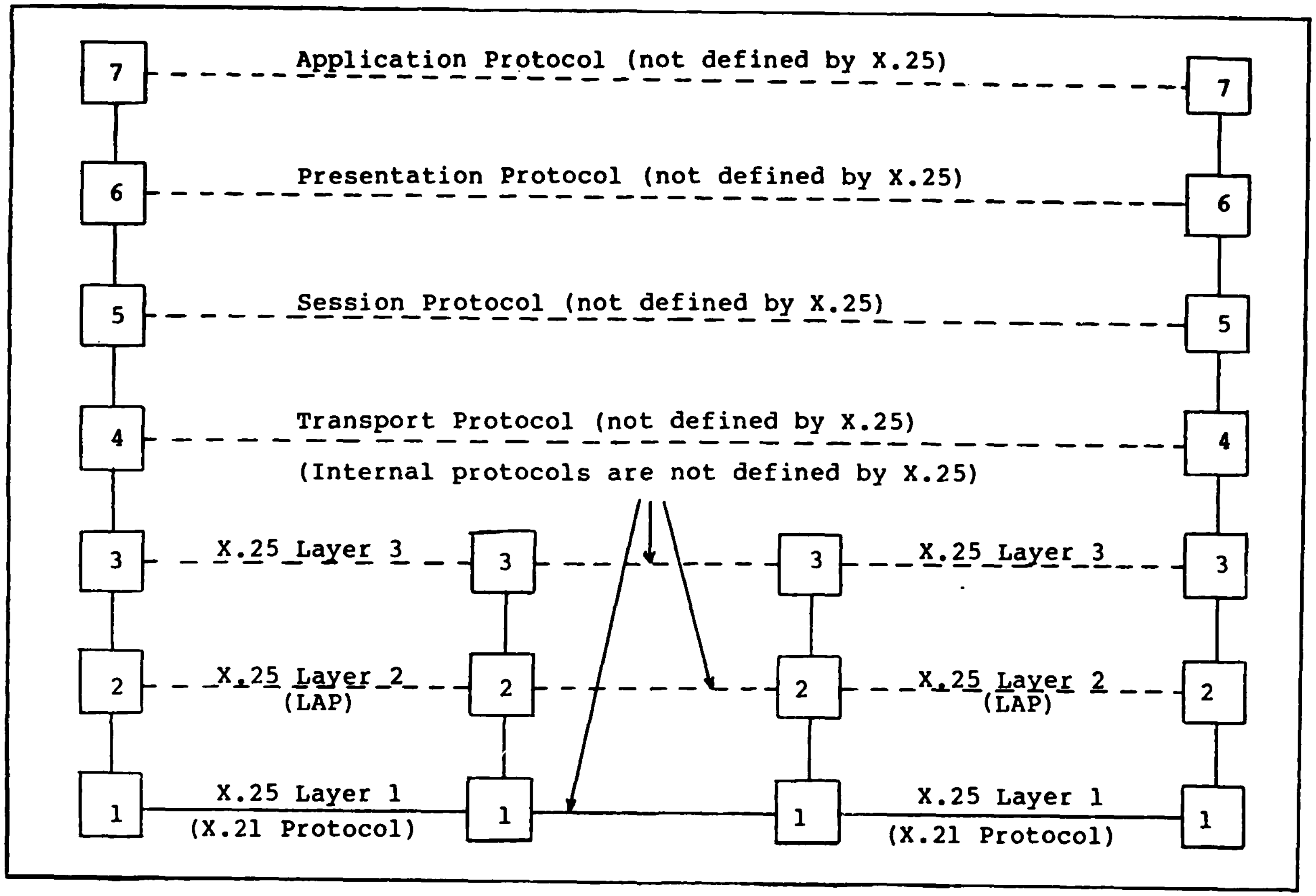

7
6
5
4
3
2
1
Application Protocol (not defined by X.25)
Presentation Protocol (not defined by X.25)
Session Protocol (not defined by X.25)
Transport Protocol (not defined by X.25)
(Internal protocols are not defined by X.25)
X.25 Layer 3
X.25 Layer 2
(LAP)
X.25 Layer 1
(X.21 Protocol)
X.25 Layer 3
X.25 Layer 2
(LAP)
X.25 Layer 1
(X.21 Protocol)

Figure 3. Protocol Hierarchy at the Systems Level

operating system that would provide certain user services. These services should include, but should not be limited to, commands to LOGIN, LOGOUT, and HELP. Following is a brief discussion of these commands as well as several special application functions which should be included as services.

When a user wishes to use the network, the LOGIN command will be used. This command will verify access authorization, identify the user to the network for data routing and status, and initializes the host for DELNET processing.

The LOGOUT command will perform the opposite process. The user is removed from the network configuration at the network operating system level, and the host dependent interface to the network is terminated.

The HELP command is required to provide any user with convenient information about the network. The information available must include network overview, current network status, network map for routing, and command syntax instructions (Ref 9).

The application functions are a minimum set of instructions required to perform network operations. They should include user messages for real time internetwork communications, file transfer, and remote job execution. The message transfer is to allow for direct communication from host to host (electronic mail). The file transfer requirement is included to enable data identified as a file on one host to be transfered to any other network host.

Finally, the remote job execution will allow command files on any host to be executed by any other network user (Ref 9).

Table 1 shows the relationships of the global, system, and detailed levels of requirements and how they apply to the DELNET.

Detailed Standards

As with the system level, specific standards for the upper levels of protocol do not exist due to their abstractions. In fact, future research on the DELNET protocols may elect to combine several of the higher levels of the ISO Model into a single Applicationns Layer (Ref 22).

Physical Layer. The standards for this layer at a detailed level are contained within the standards at the systems and global level peviously presented. The justification for these standards can be found in Reference 7. On the local side of the UNID the data link uses the RS-232C standard for twisted wire pairs and connectors. Only 9 out of the available 25 pins are used (Ref 17). The data transfer from host to UNID is in serial at a maximum rate of 19.2 kbps. On the network side, the data flow is again serial using a modified RS-449 standard and a fiber optic link at a maximum data rate of 2 mbps for the network bus. Future research may explore various data rates. If the maximums listed are exceeded, either the standards must be changed or modifications must be made to the established

REQUIREMENTS LEVEL	DEFINITION	REQUIREMENTS
I. Global	Overall general requirements pertaining to the DELNET operation	A. Flexibility B. Virtual Operation C. Performance Monitoring
II. Systems	The multiple influences of the DELNET including the Global requirements, technical aspects, and the limitations and constraints	A. Packet Switching B. Routing Techniques
III. Detailed	The specific requirements of the DELNET operations which are needed to accomplish the Global and Systems requirements	A. Network Operating System 1. Host to Node subordinate relationships 2. Commands 3. Applications

Table 1. Hierarchy of DELNET Requirements

standards. It must be emphasized that standard modifications should not be taken lightly. A subtle change at an early stage in development could substantially effect modifications or interfaces in the future.

<u>Data Link Layer</u>. As specified at the systems level, the data link access is governed specifically by the CCITT standard for Link Access Procedures (LAP). This LAP is very similar to the ISO standard for the High Level Data Link Control (HDLC). This standard specifies the packet frame format as shown in Figure 4.

<u>Network Layer</u>. The specific detail of the standards for this layer are basically the same as for the systems level. The routing for the DELNET is simple and does not require a great deal of attention. Frames simply travel unidirectional within the ring. The source and destination information is contained in the frame's header information. As the frame in injected into the ring, it proceeds from UNID to UNID until the address of the host is recognized by its connected UNID. At this point the packet is seized and routed to the proper host on the UNID's local side.

Figure 5 presents a hierarchy of all the DELNET standards. Figure 5, in conjunction with Table 1, should present the reader with a graphical representation of the functional requirements and standards, at each level, which govern the DELNET.

FLAG[1]	ADDRESS	CONTROL	INFORMATION (Packet)	FCS[2]	FLAG[1]
1 BYTE	1 BYTE	1 BYTE	VARIABLE[3]	2 BYTES	1 BYTE

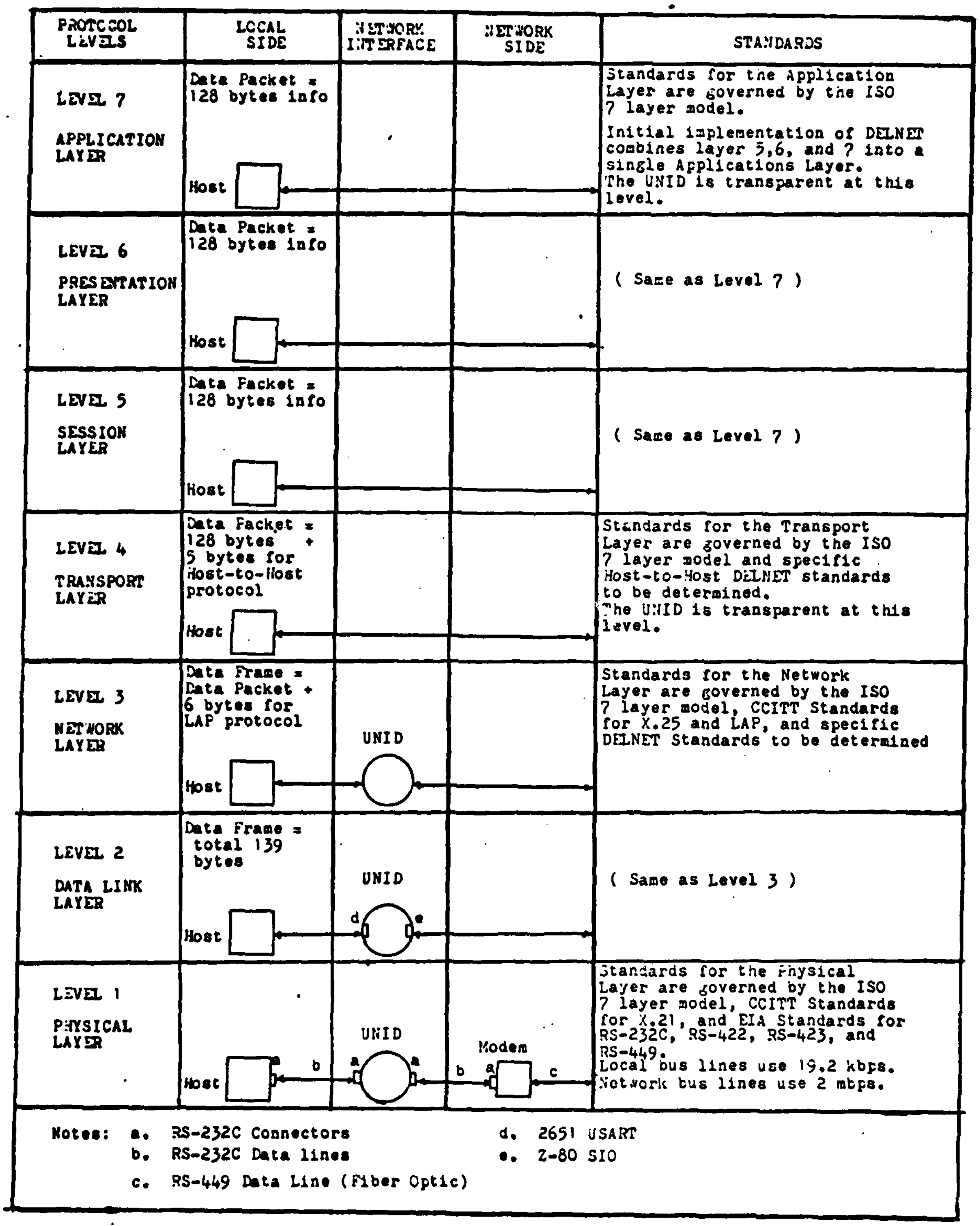
<u>Notes:</u>

1 - Flags are normally '01111110'

2 - Frame Checking Sequence used for error checking

3 - Information field is variable length - normally 128 bytes

Figure 4. Link Access Procedure Frame Format



PROTOCOL LEVELS	LOCAL SIDE	NETWORK INTERFACE	NETWORK SIDE	STANDARDS
LEVEL 7 APPLICATION LAYER	Data Packet = 128 bytes info; Host			Standards for the Application Layer are governed by the ISO 7 layer model. Initial implementation of DELNET combines layer 5,6, and 7 into a single Applications Layer. The UNID is transparent at this level.
LEVEL 6 PRESENTATION LAYER	Data Packet = 128 bytes info; Host			(Same as Level 7)
LEVEL 5 SESSION LAYER	Data Packet = 128 bytes info; Host			(Same as Level 7)
LEVEL 4 TRANSPORT LAYER	Data Packet = 128 bytes + 5 bytes for Host-to-Host protocol; Host			Standards for the Transport Layer are governed by the ISO 7 layer model and specific Host-to-Host DELNET standards to be determined. The UNID is transparent at this level.
LEVEL 3 NETWORK LAYER	Data Frame = Data Packet + 6 bytes for LAP protocol; Host	UNID		Standards for the Network Layer are governed by the ISO 7 layer model, CCITT Standards for X.25 and LAP, and specific DELNET Standards to be determined
LEVEL 2 DATA LINK LAYER	Data Frame = total 139 bytes; Host	UNID d e		(Same as Level 3)
LEVEL 1 PHYSICAL LAYER	Host a b	UNID a a	Modem b a c	Standards for the Physical Layer are governed by the ISO 7 layer model, CCITT Standards for X.21, and EIA Standards for RS-232C, RS-422, RS-423, and RS-449. Local bus lines use 19.2 kbps. Network bus lines use 2 mbps.

Notes: a. RS-232C Connectors
b. RS-232C Data lines
c. RS-449 Data Line (Fiber Optic)
d. 2651 USART
e. Z-80 SIO

Figure 5. Hierarchy of DELNET Standards

Summary

The purpose of this chapter is to identify the functional requirements and standards for the DELNET from a global, systems, and detailed viewpoint. The global level focused on the flexibility, virtual operation, and performance monitoring. The framework for the global standards is the ISO seven layer protocol model. At the systems level, the focus was on the techniques required for packet switching and routing using the store-and-foward method. The system level standards are governed by the X.25 standard developed by the CCITT. The scope of the investigation limits the detailed requirements to those which pertain to DELNET operations. These include the network operating system functions for access control and user help services, and application functions for message transfer, file transfer, and remote job execution. Lastly, the standards that govern the detailed requirements are specific for the bottom two layers of protocol. The Physical Layer uses RS-232C and RS-449 standards whereas the Data Link Layer uses LAP. The third and forth levels at the detailed level are the same as at the systems level. Also at the detailed level, the top three levels are combined to make a single applications layer. There have been several minor design considerations mentioned in this chapter so that the detailed requirements could be more specifically defined. The follc ing chapters define the actual designs of the first three protocol levels and the implementations to support them.

III. The Physical Layer

Introduction

This chapter presents the rationale used to determine the specific hardware necessary for designing and implementing the lowest level of the ISO seven layer model for protocol development. The chapter begins with a general theoretical discussion of various Physical Layer implementaion techniques. It then discusses the specific designs and implementations of the DELNET and how these designs relate to previous research for this project. All the actual implementations for the Physical Layer of the DELNET are governed by the standards set forth in Chapter II as shown.

Theory

The initial perception of the Physical Layer is one of fulfilling a rather simplistic requirement to insure a complete overview of all areas pertaining to data transfer within a network. In contrast, however, the actual theoretical considerations pertaining to this subject can become quite complex. In fact, a comprehensive analysis must consider the data bit stream as a periodic waveform and is therefore subject to the bandwidth limitations determined through complex Fourier Analysis. It is imperative that the bandwidth of the transfer medium be broad enough to support a sufficient number of harmonics of the basic frequency to successfully reproduce the square wave type bit stream.

Chapter 3 of Reference 22 presents a detailed description of this analysis. The results of this analysis contain several important points. For example, given a particular type of transfer medium or channel there exists a maximum data rate that can be transmitted on that type of medium due to bandwidth limitations.

The data rate is not the only factor used in determining the type of channel used for a LCN. Additional variables include length of channel, topology, troubleshooting and maintenance, availablity of channel interface equipment, susceptibility to electromagnetic interference (EMI), and cost. The types of transfer mediums most often used for LCNs are twisted-wire pair, coaxial cable, and fiber-optics. Each has definite limitations and advantages over the others as described in the following sections (Ref 13).

Twisted-wire Pair. Twisted-wire pairs are the most commonly used channel medium between conventional data communications equipment (DCE) and data termination equipment (DTE). The primary reasons are low cost and availability of the wire as well as its connectors. This type of channel is highly acceptable for normal communications between DCE and DTE especially for short runs of the cable (Ref 13). At the present time, 19.2 kbps and 9.6 kbps are the most widely implemented data rates due to the RS-232C standard. Twisted-wire pairs can even handle data rates up to 10 Mbps for short distances of less than

100 feet. One disadvantage of this type channel is its susceptibility to EMI and its inadvertant broadcasting of its own electromagnetic fields (Ref 8).

Within some networks one of the major disadvantages of twisted-wire pairs is its limitation to function as a baseband medium. Both fiber-optics and coaxial cable have the capability to be used as both baseband or broadband channel mediums. Baseband refers to the method of data transfer of placing the bit stream directly onto the channel; whereas, broadband refers to the method of modulating the data onto an RF carrier frequency. Using broadband channels, several customers have access to the same channel simultaneously by modulating their data streams at different rates (Ref 12 and 13).

Coaxial-cable. In LCNs, coaxial-cable is the most attractive medium for implementation due to its advantages over twisted-wire pairs and few real disadvantages. Not only can it support RF transmission for broadband capabilities, but it is relatively inexpensive and easily tapped, thus allowing easy additions to the network. It has a broad bandwidth and can support data rates of 10 Mbps for over 1000 feet or up to several miles for lower frequencies. The only real disadvantages are its small increase in cost over twisted-wire pairs, slight complexity, and nonavailability and nonconformaty of connectors for the DCE/DTE interface (Ref 13).

Fiber-optics. The use of fiber-optic channels is increasing proportionally as the technology of the subject increases. At the present time there are several large drawbacks to using fiber-optics within an LCN. The two predominant limitations are cost and complexity of installation. The fiber-optic modem which is used to interface the NIUs to the network bus, is considerally more expensive than the simple connectors used for twisted-wire or coaxial cable. Additionally, any breaks in the fiber-optic bus for the reasons of maintenance or additional hookups must be precise and often become very complicated. Once these obstacles are overcome, the fiber-optic channel is the 'most efficient' and versatile channel of the three. The primary benefits to using fiber-optics is that it is practically impervious to EMI (Ref 13) and it can transfer data at rates of over several hundred Mbps for distances ten times greater than coaxial cable (Ref 8).

Many of the larger and newer LCNs are using various combinations of the channel mediums mentioned. For example, an inter-office network might be connected by a baseband bus composed of either twisted-wire pairs or coaxial cable. These small LCNs might join into a larger network being supported by a broadband coaxial bus. This secondary network might interconnect offices over several city blocks from several different buildings. Finally, these secondary networks might be connected to other secondary networks on the other side of a large city via a fiber-optics channel. It is easy to visualize that the type channel implemented depends on a great many variables which are primarily determined by the use and size of the network (Ref 12).

DELNET Implementation

As stated in Chapter I, the role of the DELNET will have many purposes. It will greatly aid to increase the productivity of the DEL as well as providing a highly pedagogical platform for student and faculty research and study. For this reason, several decisions for the original design of the DELNET departed from the typical operational design considerations (Ref 9).

For example, in one of the original AFIT sponsored thesis projects, the design specified a fiber-optic link to be used as the network bus channel between the UNIDs (Ref 3). This design has carried foward and was incorporated as part of the DELNET. It was determined that the fiber-optic link would provide a vehicle for students to receive first hand experience with this system.

Although the DELNET's main network link is implemented by a fiber-optic bus, the UNID itself does not incorporate a fiber-optic modem nor connectors. For this reason, the UNID connections on the network side begin with an RS-232C connection and cable and lead into a fiber-optic modem (Fibronics Model TTK) for network bus interface. By connecting the network bus in this manner, it provides the pedagogical requirements or the original design yet minimized the cost and complexity of the UNID. Additionally, a very short run of twisted-wire pairs will not degrade the network link when working with data rates below 10 Mbps (Ref 13). For this and all preceeding DELNET research, the network bus has functioned as a baseband channel. With the fiber-optic cable functioning as its channel medium, the possibility exits for an upgrade to a broadband channel. The upgrade to a broadband channel would allow for a greater number of UNIDS to be connected to the DELNET without the adverse effects of increased traffic. Additionally, the DELNET could support analog types of data such as video and voice.

On the local side of the UNID, the four hosts are connected to the UNID on standard RS-232C serial links. The data rate between the hosts and the UNID is 19.2 Kbps. Reference 17 provides a complete schematic breakdown of these links, connectors, and pin assignments. There are no future plans to change the local side bus configuration.

Summary

Within the LCN community, the three basic types of channel mediums being used for the physical layer of protocol are the twisted-wire pairs, coaxial cable, and fiber-optics. Under the standards set forth in Chapter II, the DELNET incorporates a combination of twisted-wire pairs for the local side data link at 19.2 Kbps. The network side uses a fiber-optic link at 2 Mbps. All channels of the DELNET operate in the baseband configuration but could be expanded to broadband in the future.

IV. Data Link Layer

Introduction

This chapter presents the design considerations necessary for implementation of the second level of the ISO seven layer model for protocol development. The chapter begins with a discussion of various techniques that may be employed to design a network's Data Link Layer. It then discusses the specific design and implementation of the DELNET's data link protocol scheme and how this thesis effort integrated its findings into the design of the previous thesis efforts. All the actual implementations for the Data Link Layer of the DELNET are governed by the standards set forth in Chapter II as shown.

Design Considerations

The role of the Data Link Layer is to perform a variety of tasks which are totally transparent to the users. The tasks themselves vary widely in complexity in both concept as well as in actual implementation. The main task of this protocol level is to consider a raw transmission medium between NIUs and transform it into a sophisticated channel that appears free of errors to the next higher level of protocol. It normally accomplishes this task by placing the data packets into frames, transmitting the frames sequentially, and then processing the acknowledgement frames sent back from the receiving NIU (Ref 22).

The concept of the Data Link Layer is rather basic;

however, depending upon the number of protocol enhancements provided by a particular network, the design can become quite complex. These enhancements may include flow control, error detection, error correction, sequence management, and automatic reset and restart capabilities. The degree of effort spent in developing this layer of protocol is directly reflected in the higher protocol layers. That is, as many housekeeping tasks as possible should be implemented within the lower levels of the protocol structure thus freeing the higher levels to be used in a more efficient effective manner (Ref 22).

There are as many variations for developing the Data Link Layer as there are vendors and regulatory agencies which control standards. Even when two seperate networks are designed under the same identical standards, slight variations exist due to the changes in topology and utilization (Ref 18).

There are several major procedures used in industry for implementing the Data Link Layer of a LCN. Fortunately, most of these procedures are all very much alike, LAP (Ref 7), HDLC (Ref 7), SDLC (Ref 18). In fact, they are so common, many hardware devices incorporate modes of operation specifically designed to automatically accomplish many of the tasks of this protocol layer. One such device is the new Intel 8272 Programmable HDLC/SDLC Protocol Controller. Many other such devices are now in production (Ref 8). In fact, in the same time period as this thesis report was being prepared, the Digital Equipment Corporation (DEC) developed an NIU on a single LSI chip (Ref 8). In the past and until such hardware devices are commonly used, most of the tasks performed by the Data Link Layer will be accomplished through software realization.

The fundamental building block of the Data Link Layer is the data frame. As a packet of data is processed for transmission from node A to node B, a frame is built around the packet. Figure 3 of Chapter II shows a typical frame. The flag bits are normally set to 01111110 but may vary if protocols agree. These flag bits are appended onto the original data packet as are the address, control, and checksum fields. The address bits are for routing and flow control. The control bits provide information as to the type, purpose, sequence number, and acknowledgement of frames. The checksum bits are for error detection and in some cases for error correction. The schemes themselves may be as simple or complex as the networks which they control. The Data Link Layer is only concerned with the appending fields and normally has nothing to do with the data packet field. Reference 7 presents a detailed description of how these frame headers are formatted and used.

The overall throughput efficiency of a data frame from one node to another is proportional to the time spent on the analysis of the header information (Ref 1). For example, consider routing a frame from node A to node B through node C. The designer of the network must decide if there should be an acknowledgement between A and C and then C and B or just between A and B. Additionally, the designer must decide if error checking should be performed at every intermediate node or just at the destination node. Other decisions that the designer must make includes the numerical sequencing of the frames known as the 'modulo number' and the maximum number of frames that can be transmitted before an acknowledgement is required (Ref 22). Also, if an acknowledgement is not received, the designer must incorporate the time interval before retransmission occurs.

As with most design problems, there are many considerations that effect the performance of the network. These include the topology, the amount of traffic on the network, and perhaps the most important, the applications of the network (Ref 22). It is very possible to incorporate so many overhead enhancement features into the Data Link Layer that the throughput is actually reduced (Ref 1). The designer of the network protocol scheme must address these specific performance considerations.

DELNET Design and Implementation

Although there were many design decisions in regard to the DELNET protocol scheme, there was one consideration that would not normally affect a typical design environment. This was the fact that this project would be used in a continuing academic environment and would be passed from student to student over several thesis efforts. Additionally, in an acedemic environment a primary concern is flexibility which fosters continuing research investigations. Because of this, the overall design concepts used did not only incorporate the modularized approach as specified in Chapter I, but they have been kept as basic as possible while maintaining the ability to perform all the functions of the original design.

The network topology of a ring structured system creates an ideal environment for a very simple yet effective store-and-foward routing philosophy (Ref 22). While still under the standard of the LAP, this philosophy actually eliminates much of the data link control overhead and greatly reduces others. Additionally, the store-and-foward concept treats all UNIDs equally and independently and eliminates the master-slave relationships normally designed into an HDLC/SDLC type protocol scheme such as the LAP (Ref 22). Although the ring topology is not suited for all applications of networks, it is an ideal first step or starting point for the DELNET to build upon.

The data frame shown in Figure 4 was modified slightly for the DELNET scheme development as shown in Figure 6. There are two types of data frames used within the DELNET design. Both types have a fixed length of 139 bytes. The first is the information or I-frame and is used for transfering data between two UNIDs. The second is the supervisory or S-frame and is used for acknowledging the receipt of a good I-frame. As the formatting information of Figure 6 shows, the possibility of expanding the role of

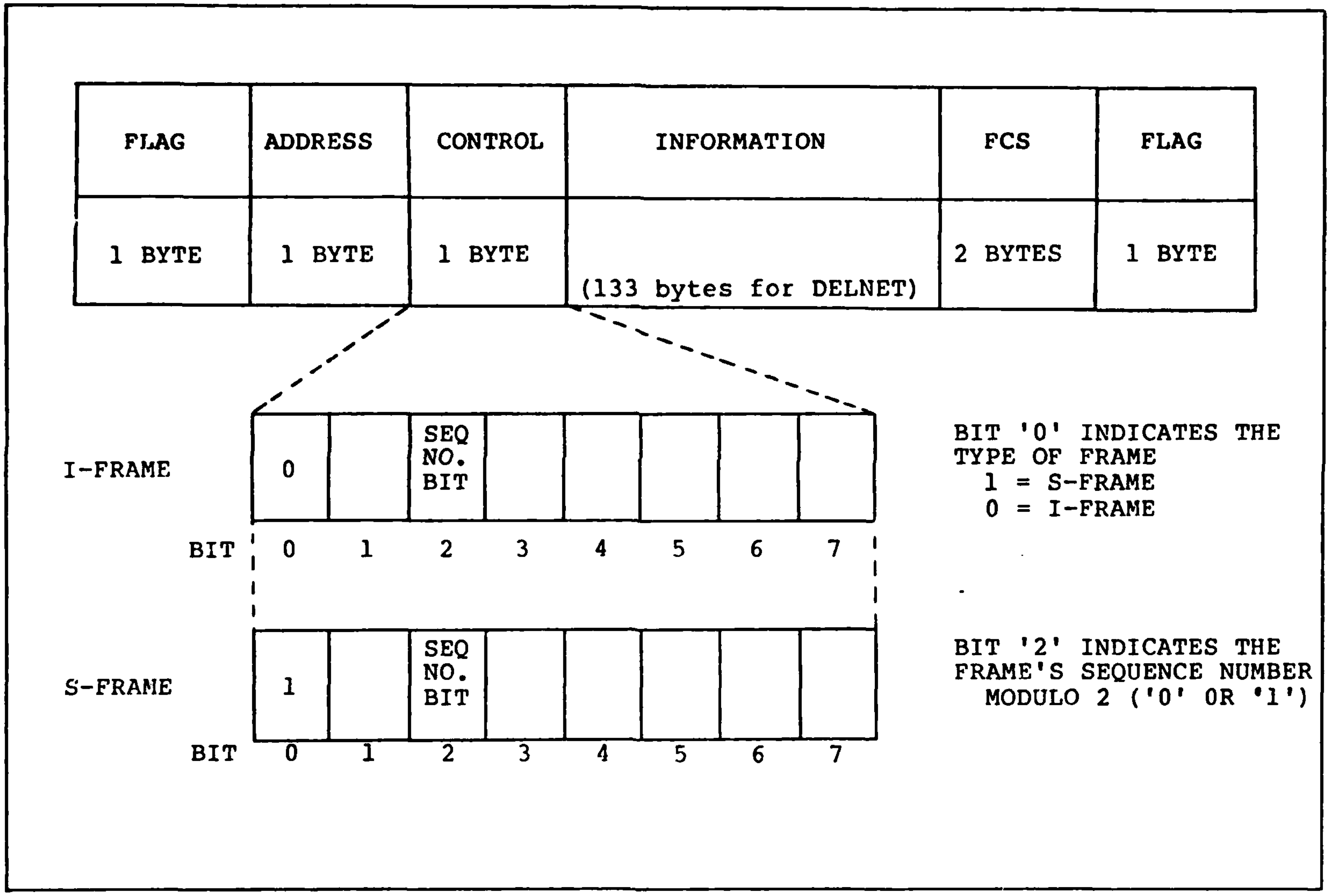


Figure 6. DELNET Frame Format Scheme

supervisory functions is open for future development. All the procedures and tasks pertaining to this level of protocol strictly function on the five appended fields of the frame. The information contained in the packet field is of no use at this level of protocol.

The software developed in the previous thesis effort (Ref 9) defined the basic framework of the procedures required to support the Data Link Layer under the basic guidelines of the LAP and X.25 standards. This software established the buffer tables and pointers to maintain the various routing of the frames, but made little or no analysis of the appended header information. The primary concern of this thesis effort, in relation to the Data Link Layer, was to incorporate this analysis into the framework previously designed.

The remainder of this section discusses the sequence of events for the Data Link Layer scheme. Reference 9 should be reviewed in order to understand the buffer table structures previously developed. The algorithms of this project were developed with the following philosophy of operation:

- Fixed length frames
- One directional communication on network bus
- Frame sequence numbering is modulo 2 (1 or 0)
- Frames received are physically moved to their appropriate tables in shared memory
- No attempt is made at error correction

- All frames are independent of each other
- The network does not approach saturation
- The percentage of network errors is low

The algorithms used to perform the services of the Data Link Layer, were developed using a three tier scheme which consisted of Data Flow Diagrams, Structure Charts, and Pseudo English. The Data Flow Diagrams provide a means of identifying the modularity of processing that is required to transform the input data into the final form of processing. The Structure Charts transform the Data Flow Diagrams into a physical structure of procedures which will accomplish the processing shown in the Data Flow Diagrams. Lastly, the Pseudo English is used to bridge the gap between the Structure Charts and actual code by combining understandable English statements and computer code. If performed correctly the transition between Pseudo English and actual code is straight foward.

The Data Flow Diagrams were developed during the initial phases of the DELNET design and are presented in Reference 9. The Structure Charts for the Data Link Layer are presented in Figure 7 through 10. The following paragraphs contain the Pseudo English description. In order to assist in the understanding of both the Structure Charts and Pseudo English constructs, Appendices A-C provide a comprehensive description of all descriptors and processing pertaining to this software.

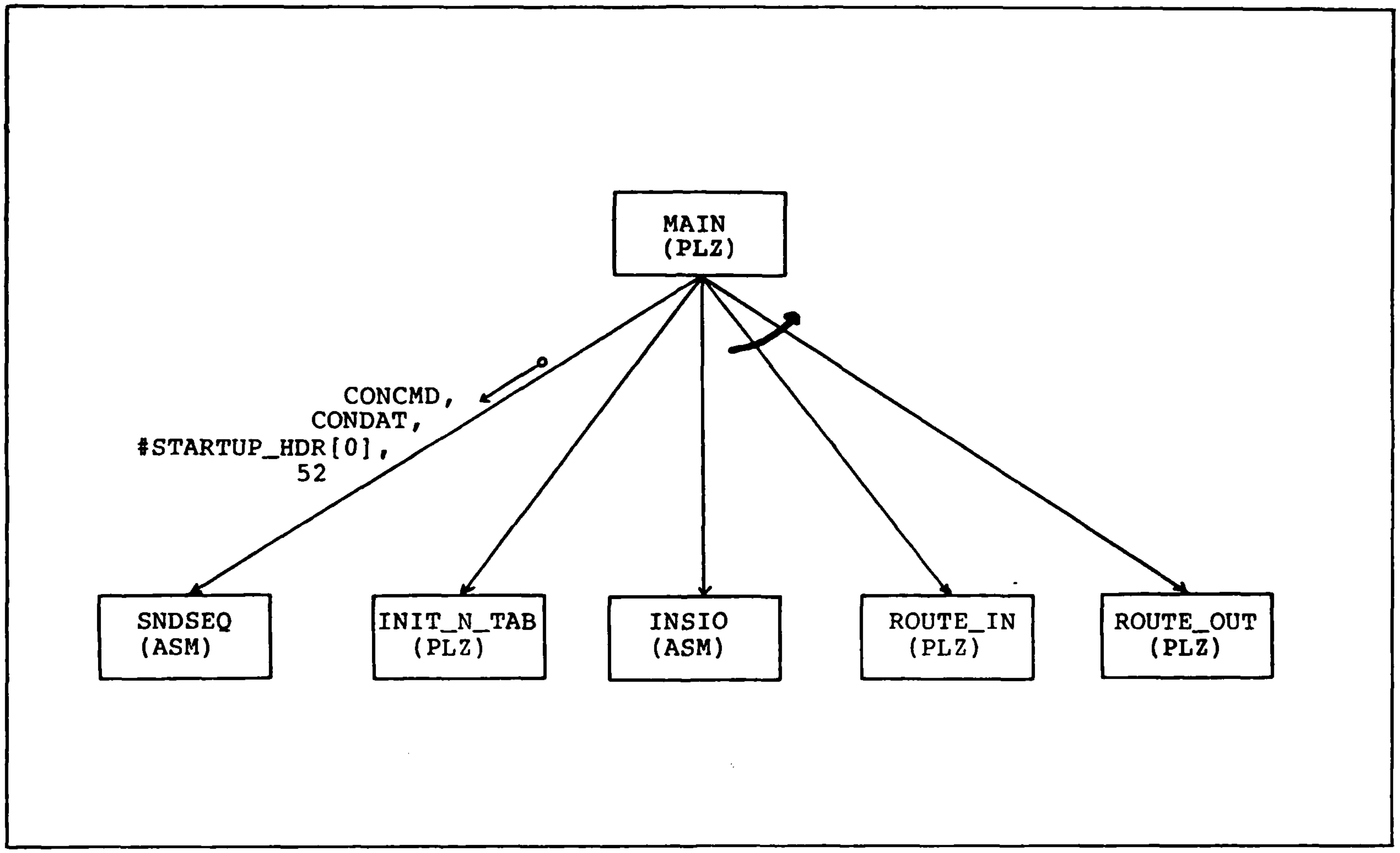


Figure 7. Main Driver For Network Operating System

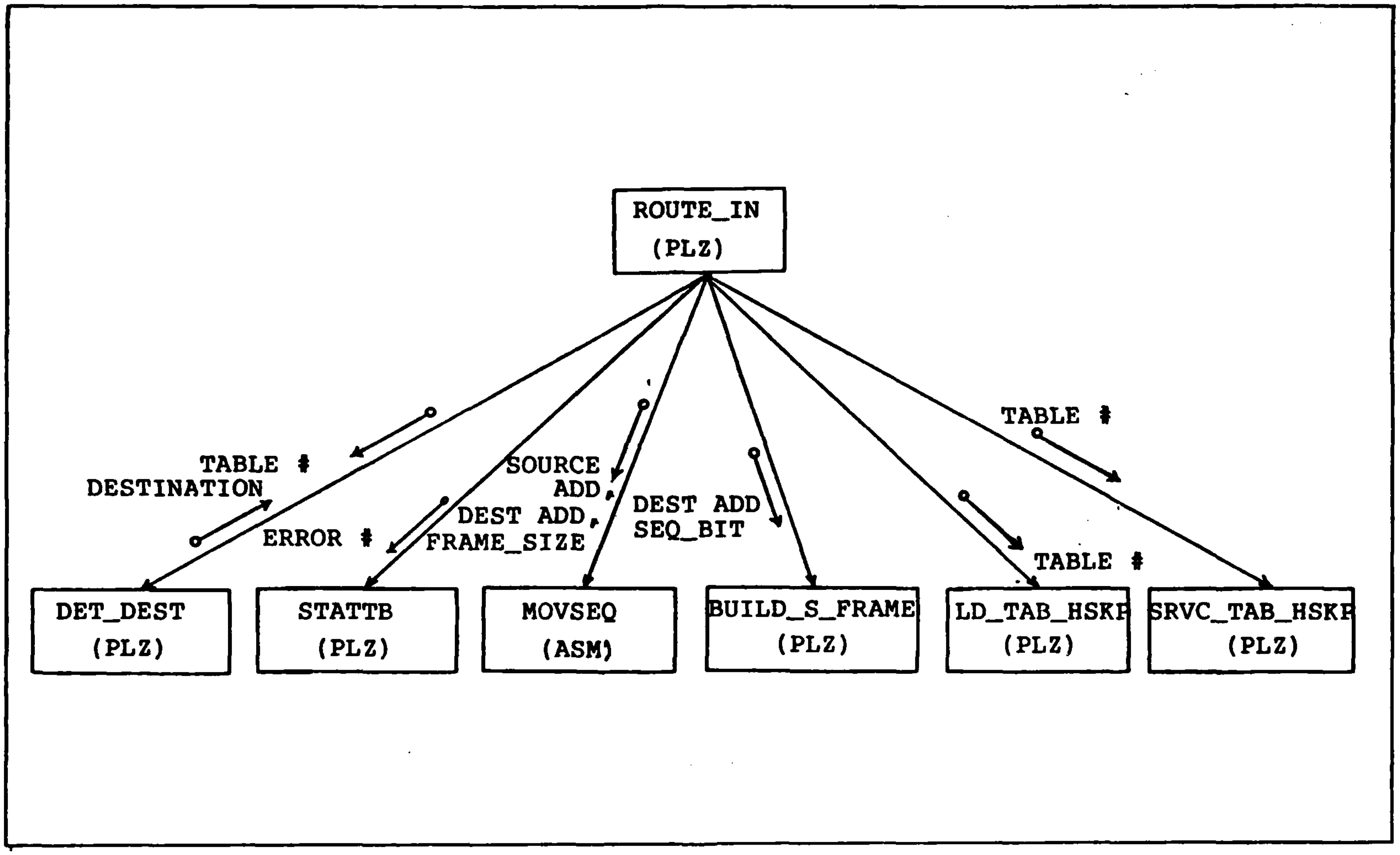


Figure 8. Route_in Procedure for Network Operating System

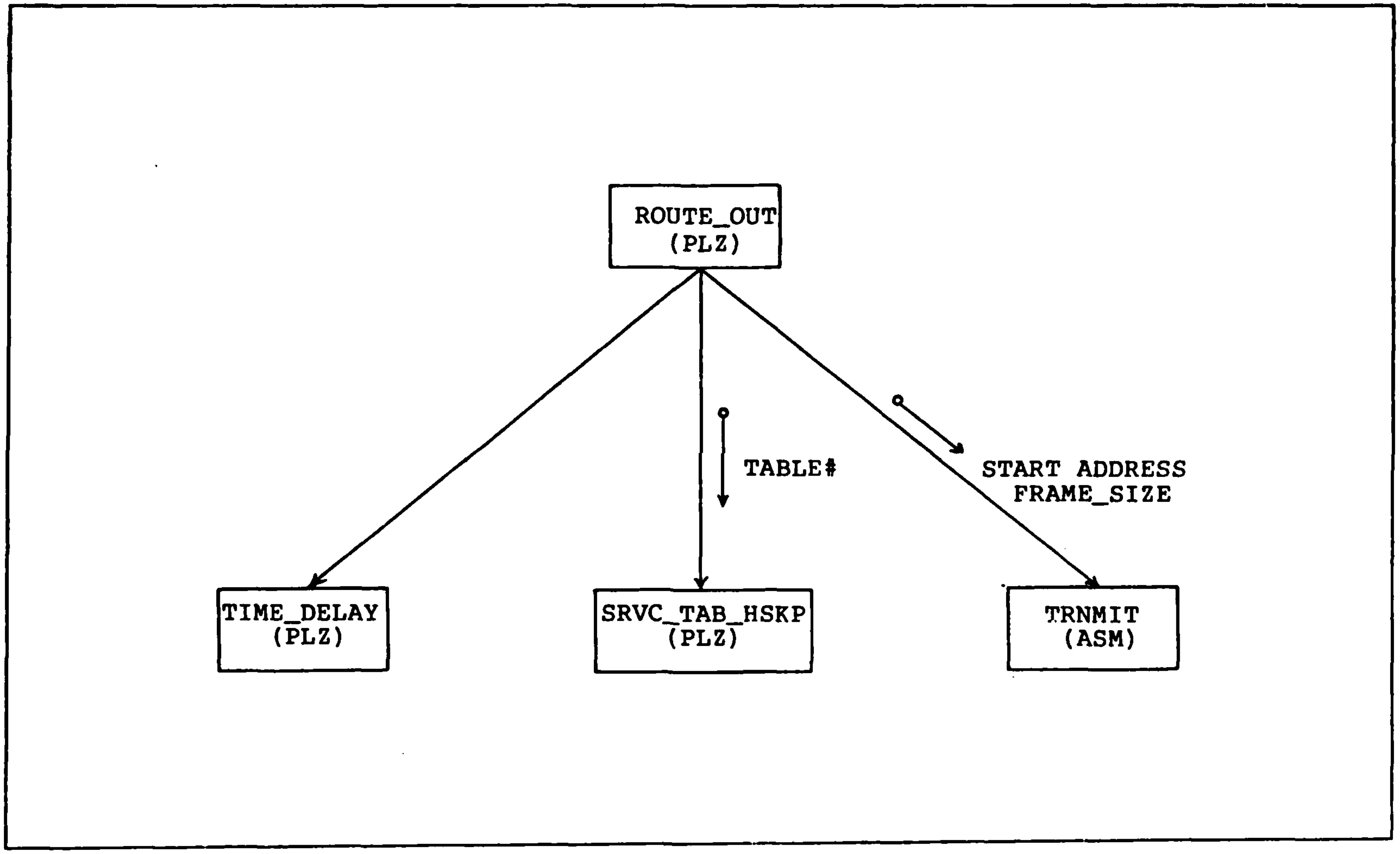


Figure 9. Route_Out Procedure for Network Operating System

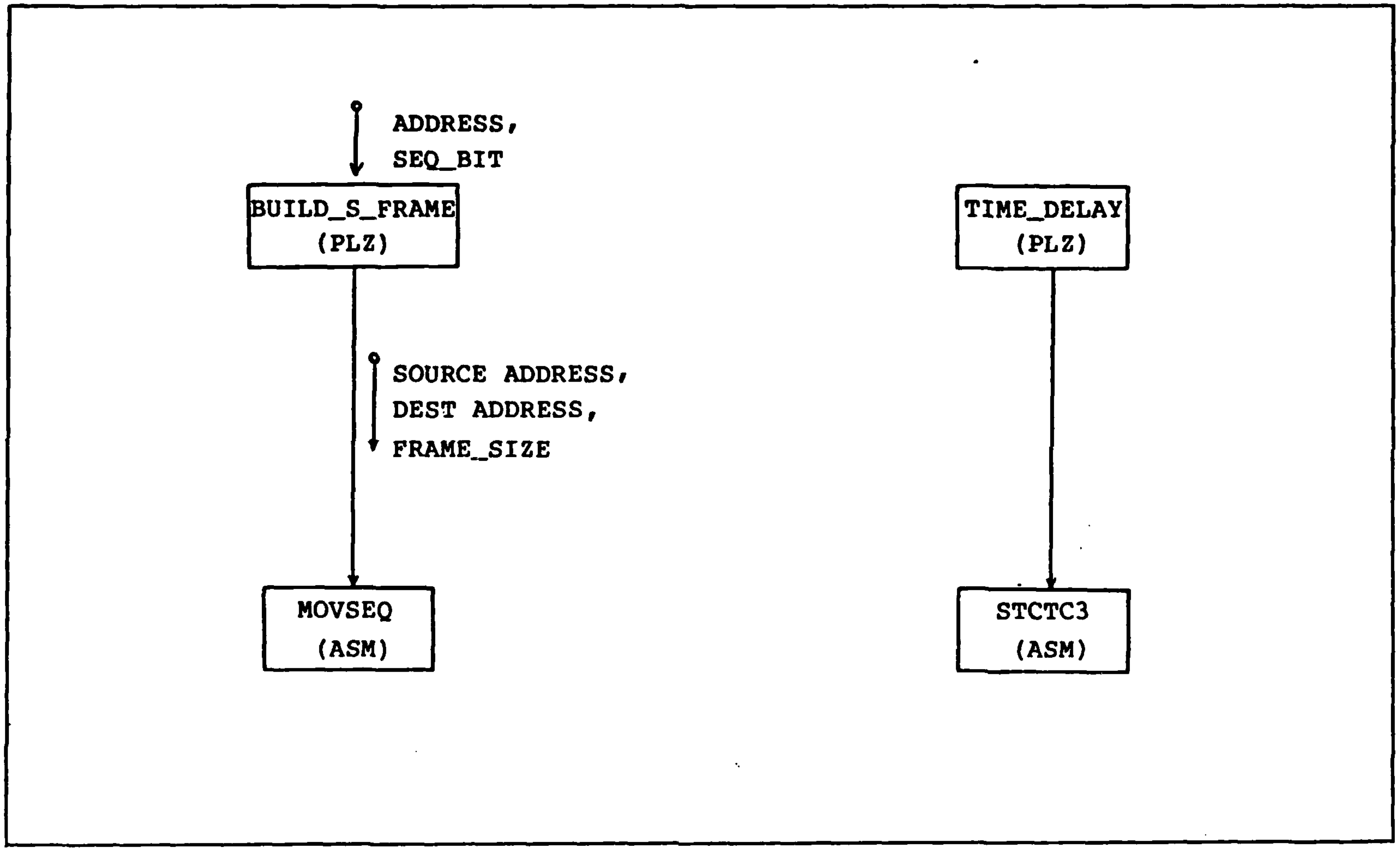


Figure 10. Subordinate Procedures for Network Operating System

After completing the initialization of the table buffers and their pointers, the processing enters an endless loop of calling procedure ROUTE_IN and ROUTE_OUT. Note that actual variable and procedure names are presented in all capital letters.

```
     Enter Procdure ROUTE_IN
If a frame is present in NT01TB then
   Determine its DESTINATION
   If DESTINATION = NTNTTB then
      MOVE frame to NTNTTB
      Update the NTNTTB pointers
   End If
   If DESTINATION = NTLCTB then
      If the frame is an S-frame then
         Determine if the S-frame is a positive
         ACKNOWLEDGEment of last transmitted I-frame
      Else  ( must be I-frame )
         Determine the INPUT_SEQ_BIT
         Call BUILD_S_FRAME to transmit ACKNOWLEDGEment
         MOVE frame to NTLCTB
         Update the NTLCTB pointers
       End If
   End If
   Update the NT01TB pointers
End If
End Procedure ROUTE_IN.

     Enter Procedure ROUTE_OUT
```

```
If a frame is present in NTNTTB then
   If the DESTINATION address < MAX_UNIDS then
      Call TRNMIT
      Update the NTNTTB pointers
   Else ( address out of limits )
      Increment status table, STATTB
      Update the NTNTTB pointers
   End If
If a frame is present in LCNTTB then
   If it is an S-frame then
      If the DESTINATION address < MAX_UNIDS then
         Call TRNMIT
         Update the LCNTTB pointers
      Else ( address out of limits )
         Increment status table, STATTB
         Update the LCNTTB pointers
      End If
   Else ( it was an I-frame )
      Place proper SEQ_BIT in control byte of frame
      If the DESTINATION address < MAX_UNIDS then
         If the TIME_DELAY is COMPLT then
            Call TIME_DELAY
         End If
         If ACKNOWLEDGE = FALSE 'AND' COMPLT = TRUE then
            Call TRNMIT
            Force ACKNOWLEDGE to FALSE until the reception
                of a good S-frame makes it TRUE
            Call TIME_DELAY to start timing sequence
         End If
         If ACKNOWLEDGE = TRUE then
            Update the LCNTTB pointers
            Compliment SEQ_BIT for next usage
         End If
      Else ( address out of limits )
         Increment status table, STATTB
         Update the LCNTTB pointers
      End If
   End If
End If
End Procedure ROUTE_OUT.
```

The method of positive acknowledgement and retransmission of I-frames and the discarding of any frame where an error may have occured has both advantages and disadvantages. It is relatively easy to implement, has little overhead, and gets the job done quite well and efficiently when few errors occur on the network. If the error rate is high, the throughput of the system will be reduced considerably (Ref 1). This general philosophy is often used in even large sophisticated networks although often network monitoring techniques are employed to guard against bottlenecks and breakdowns (Ref 18). Appendices A-C contain documentation of the software used to implement the Data Link Layer protocol.

The method employed to design and implement the actual

DELNET services provided by the Data Link Layer can actually be described as 'datagram' service. Each frame is transmitted independently of other frames. It will be the responsibility of the Transport Layer of protocol to place the data packets in proper order if required. Reference 22 has a good description of datagram service and how this relatively simple concept relates to the data link protocol layer using the store-and-foward concept.

Summary

The majority of the various protocol schemes present a framework from which very sophisticated and complex network services can be provided. Man_ of these services can be reduced or eliminated for simpler networks such as the initial DELNET implementation. In fact, the ring topology of the DELNET further relaxes the routing portions of the protocol scheme to an easily designed and implemented store-and-foward concept. This concept is consistent with the general philosophy of the DELNET to be a modularized and maintainable network. The data link protocol layer designed for the DELNET incorporates communications between UNIDs while maintaining the flow control, error detection, and virtual addressing. This datagram service, as it is sometimes called, is implemented through hardware using the Z-80 SIO and through software.

V. Network Layer

Introduction

This chapter presents the design considerations necessary for implementation of the third level of the ISO seven layer model for protocol development. The chapter begins with a discussion of the Network Layer philosophy and how this layer might be implemented. It then discusses the specific design and implementation of the DELNET's Network Layer protocol scheme and how this thesis effort was integrated into previous designs. All the actual implementations for the Network Layer of the DELNET are governed by the standards set forth in Chapter II as shown.

Design Considerations

The role of the Network Protocol Layer is to interface the Data Link Layer discussed in Chapter IV with the Transport Layer which has direct virtual communication with the Transport Layers of additional host computers (Ref 22). In simple terms, it transforms a frame of data from the network side of a NIU to a packet of data on the local side of a NIU. It is also up to the Network Layer to determine the actual routing of the data packet and pass this information to the Data Link Layer for transmission. Additionally, it is the task of the Network Layer to insure that messages are properly sequenced before delivery and upon arrival to the Data Link Layer (Ref 5).

The Network Layer is often called the packet or communications subnet layer (Ref 5). In either case, it refers to the Network Layer and its two subordinate layers in providing the actual communication transmissions between the NIUs and their host computers. This communication takes place by the routing of packets from the network layer to the subordinate layers and then over the physical channel to the destination NIU. From there it proceeds up the hierarchy to the Network Layer of the designated host.

As with the Data Link Layer, the Network Layer may be quite complex or rather simple depending on the complexity of the routing and control schemes of the network. In fact, the X.25 standard has five different packet types defined for full implementation if all services of this layer are required. For lesser services, a subset of these types may be used. This layer not only concerns itself with routing and sequencing as previously mentioned, but may control such areas as establishing virtual circuits, controlling collisions, preventing deadlocks, call confirmations and clears (Ref 22). This is what makes the X.25 standard universal in concept. It has services to control practically every situation that might arise.

If the Network Layer supports or does not support datagram services is perhaps one of the most critical questions the designer must ask. If no virtual circuit has to be established prior to transmission, the type of packet that sets up the virtual call may be disgarded. As in the Data Link Layer, using the loop topology, simplex communications, and store-and-foward routing philosophy, the complexity of the Network Layer is reduced considerably.

DELNET Design and Implementation

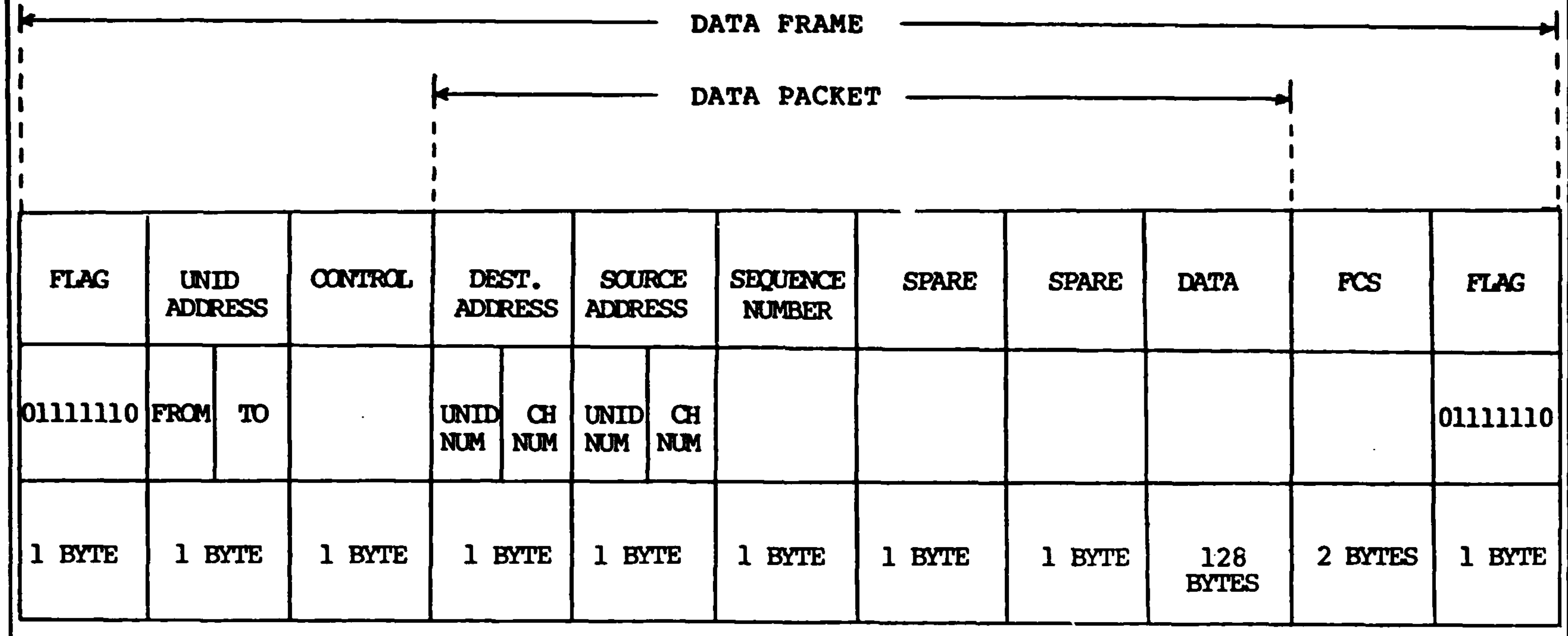
In developing the DELNET's layer scheme, the X.25 packet switching guidelines set forth by the CCITT were adhered to in philosophy, but varied slightly in actual format implementation. Since the design scheme incorporated the simple store-and-foward routing algorithm and the transmit-and-wait positive acknowledgement technique in the Data Link Layer, it was not necessry for the DELNET to establish a virtual circuit before transmission. For this reason, the 'Call Request' and 'Incoming Call' packet types were not needed. The source and destination DTE address fields were incorporated into a modified data type packet. This allowed the DELNET to utilize a single type of packet (called the data packet) and yet provide all the information necessary for virtual communications.

In addition to the actual data, this modified data packet contains five header fields of a single byte each. Bytes one and two are the destination and source host addresses respectively. Each contains both the UNID number and local channel number. Byte three contains the sequence number of the packet and will be used by a higher layer of protocol to sort the data. Bytes four and five are left blank for future development. Future efforts might use these bytes to incorporate an error checking scheme (CRC) at the network level or a word count for variable size packets. A wide variety of choices exit (Ref 22). The reader should keep in mind that since a hierarchial protocol scheme is being used, the data field from the Data Link Layer contains the five header bytes plus the data field of the Network Layer. Likewise, the data field of the network layer will contain some header fields from its higher Transport Layer. Figure 11 shows the complete data frame as viewed from the Network Layer.

The software development in the previous thesis effort (Ref 9) prepared the basic framework of most of the procedures required to support the Network Layer software for the UNID. This software established the buffer tables and pointers required to maintain the various UNID internal routings of the packets, but did not consider the appended header information. The primary task of this thesis effort was to implement the previously designed software by augmenting the header information scheme and processing the data flow accordingly.

The remainder of this section discusses the sequence of processing to control the Network Layer. As with the Data Link Layer, Reference 9, Chapter VII should be consulted to obtain an understanding of the buffer tables and pointers and how they are used to enhance the packet flow and processing. Figure 12 encapsultes these tables and the types of data contained in each.

The algorithms used to perform the services of the Network Layer were developed using the same three tier



FLAG	UNID ADDRESS		CONTROL	DEST. ADDRESS		SOURCE ADDRESS		SEQUENCE NUMBER	SPARE	SPARE	DATA	FCS	FLAG
01111110	FROM	TO		UNID NUM	CH NUM	UNID NUM	CH NUM						01111110
1 BYTE	1 BYTE		1 BYTE	1 BYTE		1 BYTE		1 BYTE	1 BYTE	1 BYTE	128 BYTES	2 BYTES	1 BYTE

NOTE: THE TWO FLAG BYTES AND THE FCS BYTES ARE APPENDED AND STRIPPED OFF AUTOMATICALLY BY THE Z-80 SIO.

Figure 11. Network Layer's View of Data Packet

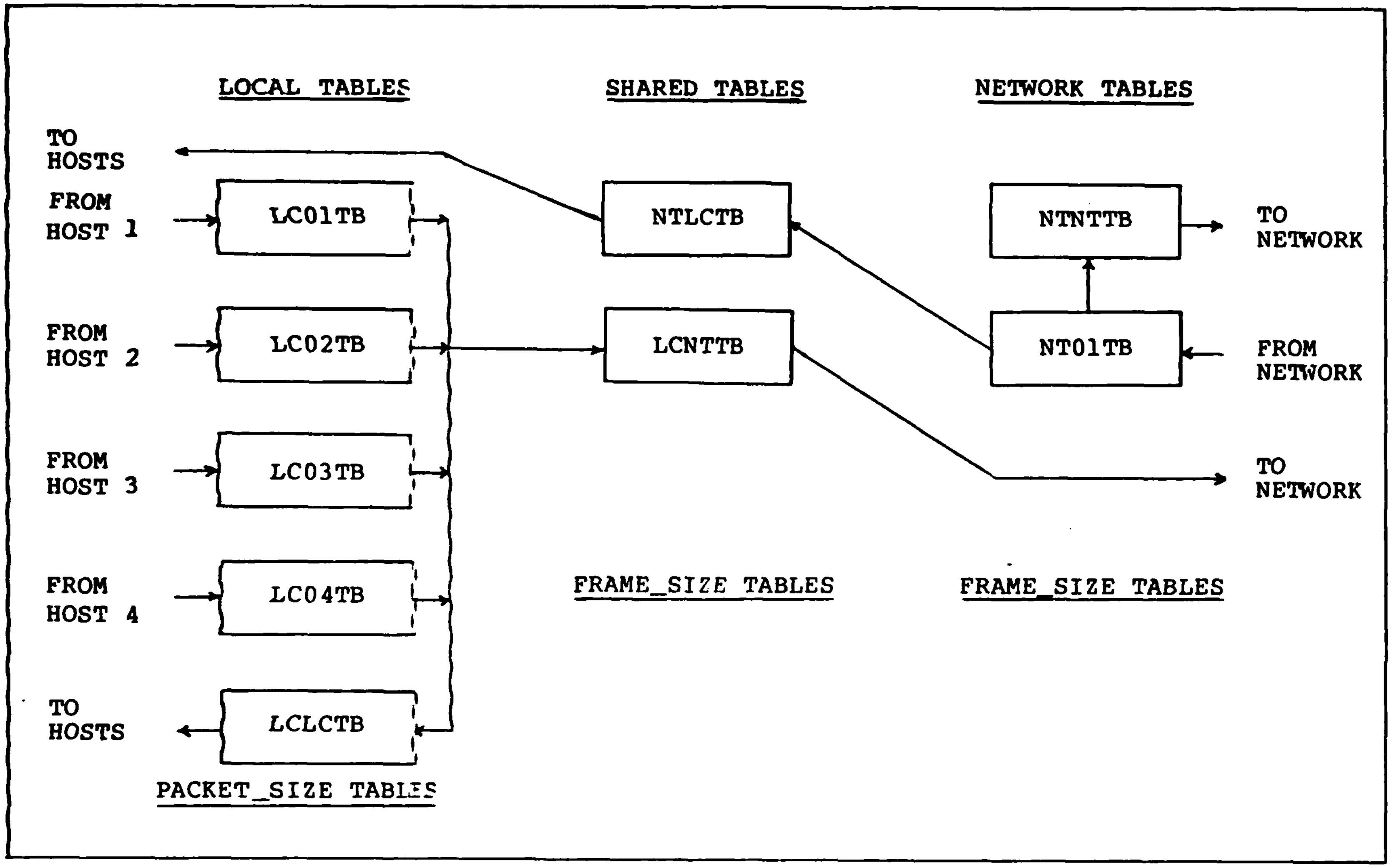


Figure 12. Local and Network Operating Systems' Table Buffers

approach used for the Data Link Layer. The Data Flow Diagrams are presented in Reference 9. Figures 13 through 16 present the Structure Charts and the following paragraphs descibe the processing using Pseudo English.

As in the Data Link Layer, the processing begins with the initialization of the table buffers and pointers and then enters an endless loop of calling procedures ROUTE_IN and ROUTE_OUT. Note that the actual variable and procedure names are presented in all capital letters.

```
     Enter Procedure ROUTE_IN
If a packet is present in LC01TB then
   Determine its DESTINATION
   If the DESTINATION is a Case of
      LCLCTB then
         MOVE packet to LCLCTB
         Update the LCLCTB pointers
      LCNTTB then
         Call BUILD_I_FRAME
         MOVE the new frame to LCNTTB
         Update the LCNTTN pointers
   Else  ( improper DESTINATION address )
         Increment status table,STATTB
   End If
   Update the LC01TB pointers
End If
Repeat this sequence for LC02TB through LC04TB
End Procedure ROUTE_IN
```

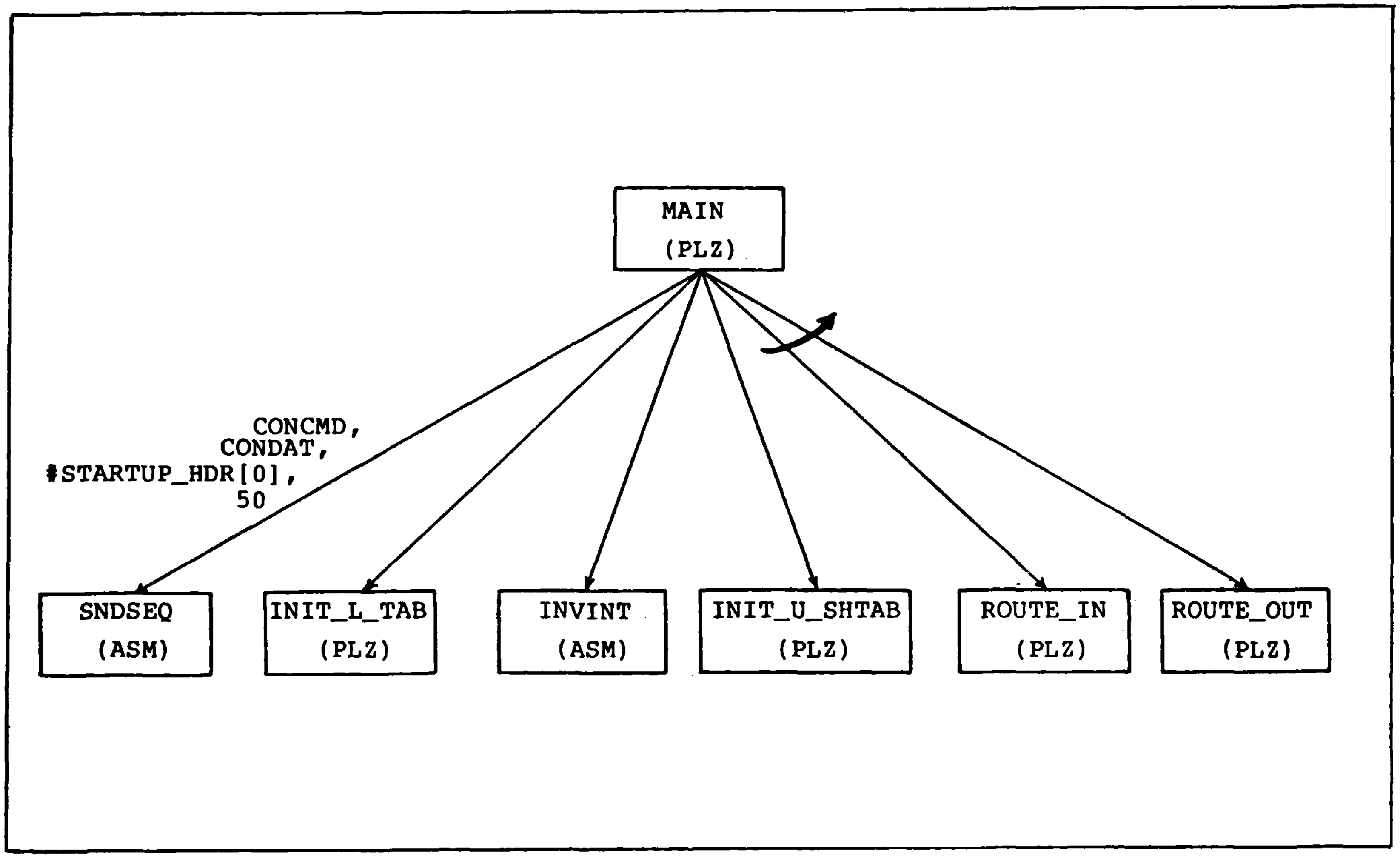


Figure 13. Main Driver for Local Operating System

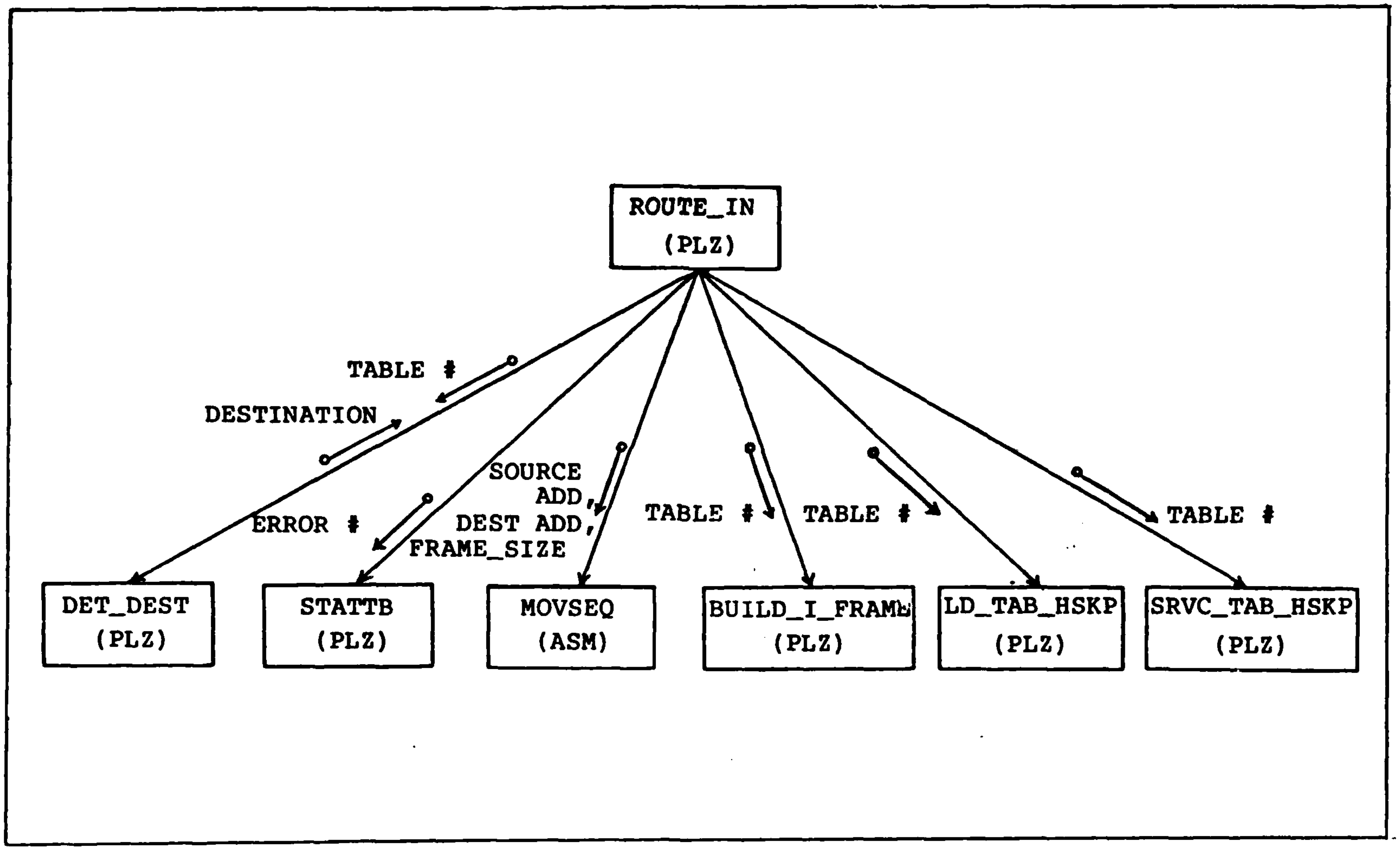


Figure 14. Route_In Procedure for Local Operating System

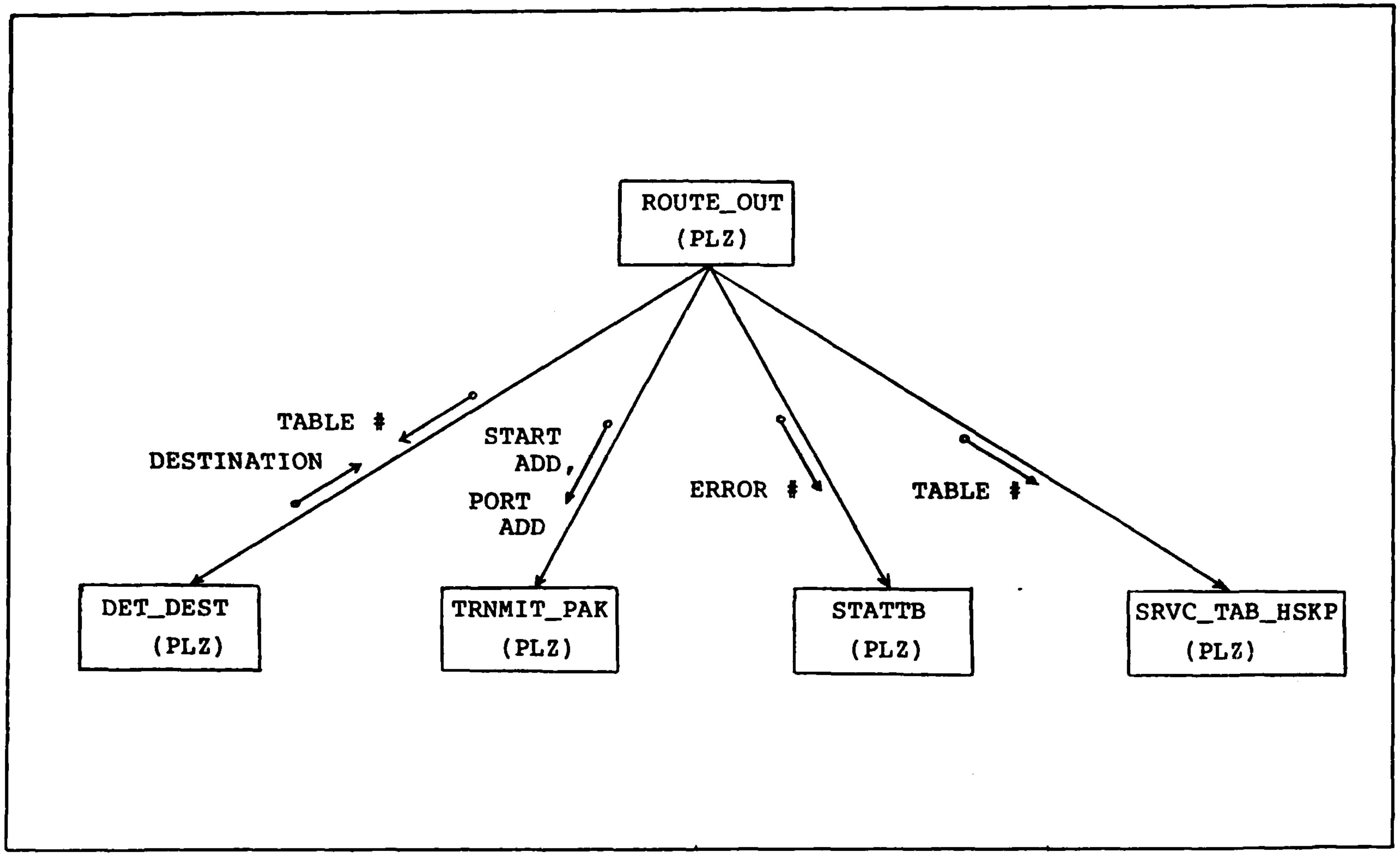


Figure 15. Route_Out Procedure for Local Operating System

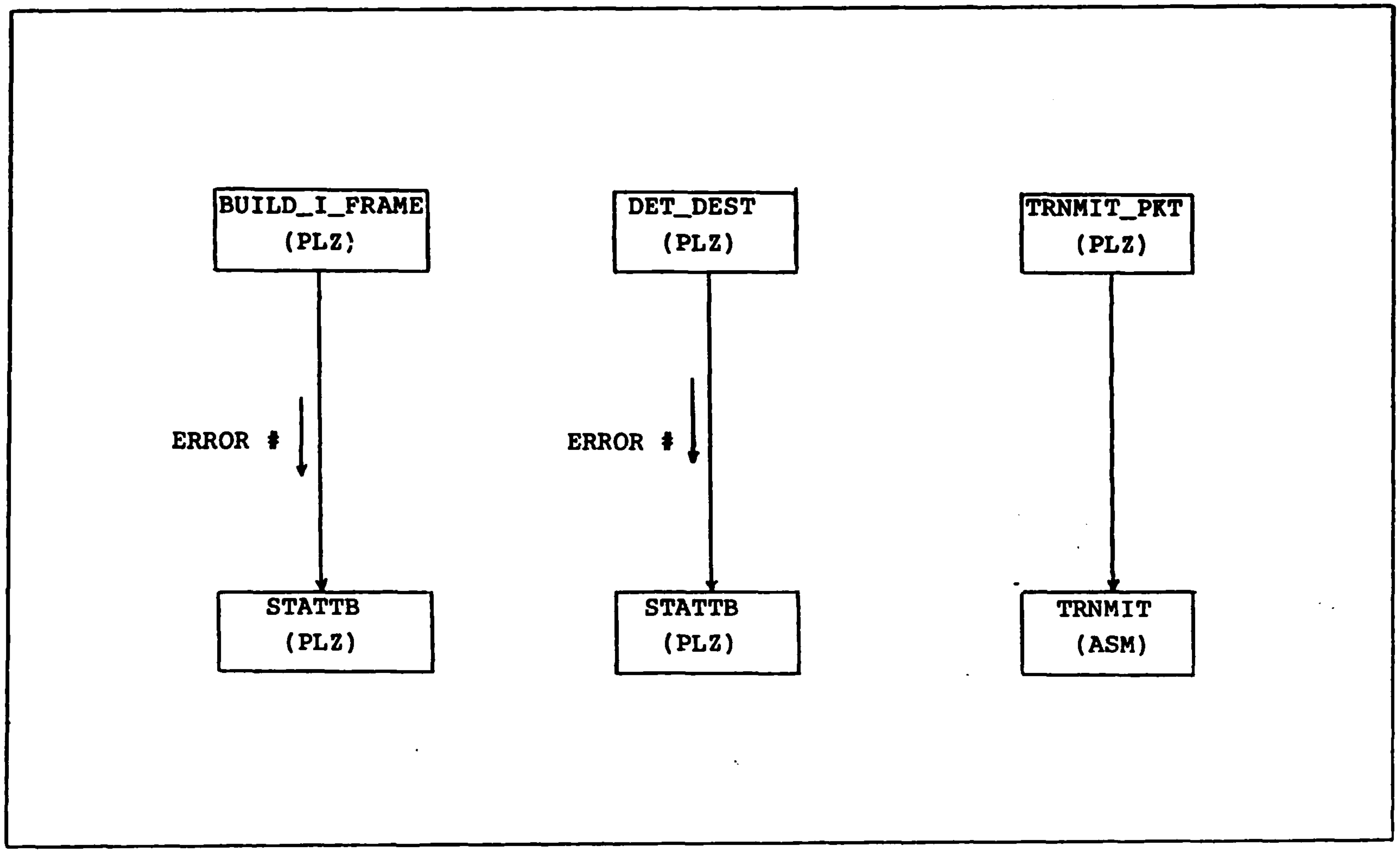


Figure 16. Subordinate Procedures for Local Operating System

```
     Enter Procedure ROUTE_OUT
If a packet is present in LCLCTB then
   Determine its DESTINATION
   If DESTINATION is a Case of
      Channel No. 1 then
         Call TRNMIT_PKT (To send to channel 1)
         Update the LCLCTB pointers
      Repeat this sequence for all Cases of
      Channel No. 2 through Channel No. 4
   Else ( Improper channel number )
      Increment status table,STATTB
      Update the LCLCTB pointers
   End If
End If


If a frame is present in the NTLCTB then
   Determine its DESTINATION
   If the DESTINATION is a Case of
      Channel No. 1 then
         Call TRNMIT_PKT (To send to channel 1)
         Update the NTLCTB pointers
      Repeat this sequence for all Cases of
      Channel No. 2 through Channel No. 4
   Else ( Improper channel number )
      Increment status table,STATTB
      Update the NTLCTB pointers
   End If
End If
End Procedure ROUTE-OUT
```

The processing for the Network Layer will transmit and receive packets of data between the UNID and its hosts. Additional services may be provided when the design of the Transport Layer is completed and the Network/Transport Layer interface is completed.

Summary

The Network Layer of protocol is the interface between the transport services layer and the basic Data Link Layer. For a rather complex network, it can use a wide variety of packet types to accomplish a multitude of tasks. But for a simple network such as the DELNET, these types can be combined into a single packet type. Within the scheme of the DELNET, the Network Layer simply transforms a data link frame, which is transmitted on the network bus, to a data packet, which is used for local processing. This transformation is accomplished by evaluating the packet headers and moving the packet to the proper location. With the completion of this layer of protocol, the UNIDs are capable to transmit and receive data frames from the network ports, route the frames internally, and receive and transmit the packets to the appropriate ports of their connected hosts.

VI. Software Configuration and Validation

Introduction

This chapter presents the procedures for configuring and testing the software developed during this thesis effort. The chapter begins with a discussion of the MCZ 1/25 software development system and the UNID environments and concludes with the actual test procedures conducted utilizing this environment to test the DELNET software. This chapter presents examples of the actual commands that were used to process the software under test. A working understanding of References 26, 29 and 30 would be helpful to fully comprehend the following text; however, a general knowledge of minicomputer operating systems will be sufficient.

Test Environment and Test Software Configuration

The Zilog MCZ 1/25 minicomputer, Zilog RIO operating system, and the PLZ programming language present an 'ideal' environment for the development and testing of an operating system such as the one for the DELNET. The PLZ language is primarily based on the concept of combining structured modules of software written in either the PLZ higher order language or Z-80 assembly level language. It is this diversification of combining and linking these modules together that create the flexibility desired for new operating system development. A second and perhaps equally important feature of this environment is that the modules are relocatable and may be placed into specific memory locations with simple linking commands (Ref 26).

For example, suppose we have two source code modules of software. The first module, Test_1.S, is written in PLZ and the other, Test_2.S, is written in Z-80 assembly language. Note that the RIO Operating System rules require the suffix '.S' for source code modules. The Test_1.S module must first be compiled using the 'PLZSYS' command.

```
    %PLZSYS TEST_1.S
```

The result of this compilation is a Test_1.L listing file and a Test_1.Z intermediate code file. The 'Z' code files are actually executable code which can be run by using the ZINTERP interpreter (Ref 26). The code is more efficient, however, if it is assembled by the PLZ Code Generator with the following command:

```
    %PLZCG TEST_1.Z
```

The result of this assembly is an object code file, Test_1.OBJ. It is recommended that for the DELNET operating systems, the latter method of generating an object code file be used. The execution time of the interpreted 'Z' code is much slower.

The Test_2.S assembly language module is simply assembled using the command:

```
    %ASM TEST_2.S
```

The result is an object code file, Test_2.OBJ. Once all the modules have been assembled and each has an object file attached, they are linked together and placed into memory using the 'PLINK' command:

```
%PLINK $=5000 TEST_1 $=8000 TEST_2
```

This command places the Test_1.OBJ code into memory beginning at location 5000 Hex and the Test_2.OBJ code beginning at location 8000 Hex. The linking information is placed directly following the last used memory location. If only a single address is specified, then each module is attached in sequential locations. For example:

```
%PLINK $=5000 TEST_1 TEST_2
```

After the linking is completed, the executable code is referenced by the name of the first module in the linking string. In the above example, Test_1 would be the programs name. To execute this program on the MCZ 1/25, it would be necessary to simply type 'TEST_1'. To obtain a memory map of the program, a '.MAP' suffix is attached following the linking process. To print a memory map just type:

```
%PRINT TEST_1.MAP
```

DELNET Software Configuration

The DELNET software for protocol layers two and three is located in the UNID's memories. Figure 17 shows the configuration of these memories and the Z-80 processors which controll them. Basically, the network operating system implements the Data Link Layer and the local operating system implements the Network Layer. The local processor has access to its 32 K (8000 Hex) of its local system memory and the 32 K of the shared memory. In the same manner, the network processor has access to its 32 K of

MEMORY (HEX)	UNDER CONTROL OF LOCAL Z-80 PROCESSOR	UNDER CONTROL OF NETWORK Z-80 PROCESSOR	MEMORY (HEX)
0000 0FFF	LOCAL SYSTEM ROM	NETWORK SYSTEM ROM	0000 0FFF
1000 1FFF	LOCAL SYSTEM RAM	NETWORK SYTEM RAM	1000 1FFF
2000 7FFF	LOCAL OPERATING SYSTEM AND TABLE BUFFERS	NETWORK OPERATING SYSTEM AND TABLE BUFFERS	2000 7FFF
8000 FFFF	SHARED MEMORY TABLE BUFFERS FOR ACCESS BY BOTH THE LOCAL AND NETWORK Z-80 PROCESSORS		8000 FFFF

Figure 17. UNID Memory and Processor Configuration

network system memory and the 32 K of the shared memory (Ref 4). In refering back to Figure 12, the reader can see why the NTNTTB and NT01TB are located in the network system memory, the LCLCTB and LC01TB-LC04TB are located in the local system memory, and the LCNTTB and NTLCTB are located in the shared memory.

There are a total of eight software modules presently being used to implement the DELNET operating system and which reside in the UNID memories. They were first presented in Chapters IV and V, but are condensed in Table II for continuity and clarity. After studying Table II, it should become clear to the reader why the relocatable options of PLZ are ideal for such software development.

In addition to the MCZ 1/25 environment, the UNID itself has several features which were previously developed to enhance its capabilities (Ref 2). The UNID incorporates 1 K of ROM and 1 K of RAM in each of the system memories which are used for the basic bootstrapping operations of the UNID and serve several monitoring functions as well. This monitoring program together with a video monitor enable the loading, filling, displaying, and moving of memory locations throughout the UNID. Reference 2 contains a complete desciption of the monitor options and procedures.

Each of the DELNET operating system software modules are compiled and/or assembled to produce the object codes for the modules. To place the modules in their correct locations in UNID memory, the following linking commands are

MODULE	LOCATION	PURPOSE OF MODULE
N.MAIN (PLZ)	NETWORK SYSTEM MEMORY	MAIN DRIVER FOR THE NETWORK OPERATING SYSTEM
N.TAB (PLZ)	NETWORK SYSTEM MEMORY	SETS UP AND INITIALIZES THE TABLE BUFFERS FOR THE NETWORK OPERATING SYSTEM
N.INSIO (ASM)	NETWORK SYSTEM MEMORY	SUPPORTS THE NETWORK OPERATING SYSTEM BY INITIALIZING THE I/O PROCESS AND TRANSMISSION OF NETWORK DATA
L.MAIN (PLZ)	LOCAL SYSTEM MEMORY	MAIN DRIVER FOR THE LOCAL OPERATING SYSTEM
L.TAB (PLZ)	LOCAL SYSTEM MEMORY	SETS UP AND INITIALIZES THE TABLE BUFFERS FOR THE LOCAL OPERATING SYSTEM
L.VINT (ASM)	LOCAL SYSTEM MEMORY	SUPPORTS THE LOCAL OPERATING SYSTEM BY INITIALIZING THE I/O PROCESS AND TRANSMISSION OF LOCAL DATA
U.LIB (ASM)	SHARED MEMORY	SUPPORTS BOTH THE LOCAL AND NETWORK OPERATING SYSTEMS WITH LIBRARY FUNCTIONS TO RECEIVE, TRANSMIT, AND MOVE BLOCKS OF DATA
U.SHTAB (PLZ)	SHARED MEMORY	SETS UP AND INITIALIZES THE TABLE BUFFERS FOR ACCESS BY BOTH THE LOCAL AND NETWORK OPERATING SYSTEMS

Table 2. Software Modules Implementing DELNET Operating System

envolked on the MCZ:

```
1).  %PLINK $=5000 L.VINT L.TAB L.MAIN $=7000
     ZINTERP.DATA $=8000 U.LIB U.SHTAB
2).  %PLINK $=5000 N.INSIO N.TAB N.MAIN $=7000
     ZINTERP.DATA $=8000 U.LIB U.SHTAB
```

First, note that ZINTERP.DATA is the linking information and is placed in each of the system memories. This prevents the local or network linking information from writing over each other. If it was left to the operating system, it would place the linking information at the end of the shared memory modules each time it loaded a different operating system, thus writing over each other. Both the local and network modules must each be linked with the shared memory modules since each must contain unique linking information.

Once all the modules of each operating system are properly linked together, they are ready to be placed into the UNID memories. First, the network operating system is loaded. Since the network monitor cannot interface the MCZ 1/25 directly (network side of UNID), the network operating system is initially loaded into the local side of the UNID. This is accomplished with the command on the UNID local monitor:

```
>L N.INSIO
```

At this point, the network operating system is loaded into the UNID's local and shared memory locations according to the previous linking information. The portion of the network operating system in local memory is then moved to shared memory by the following move command:

```
M A000 5000 2FFF
```

This command moves to location A000 Hex from location 5000 Hex a total of 2FFF Hex bytes. Now that the network operating system is located in shared memory, it is moved down into the network system memory with the following command envolked on the network monitor console:

```
M 5000 A000 2FFF
```

The local operating system is then simply placed into its system and shared memory by its load command. Upon the loading of the local operating system, the shared memory modules are written over in a one to one correspondence so both operating systems are linked to it independently. Once both these operating systems are loaded into their correct memory partitions, the processing can begin.

First, the memory maps of each operating system are checked to identify the starting addresses of each system. The starting address of the local operating system is set into the program counter (PC) and the stack pointer (SP) is set to a value away from used memory (3000 Hex). The local processing is begun with the 'go' command on the local monitor. This process is repeated for the network processing. Both the network and local processing enters enless loops of routing in and routing out of frames and packets of data respectively and uses the shared memory buffer tables to interchange the data.

Software Test and Validation

Perhaps the most challenging efforts of this thesis investigation was to develop and conduct a test plan that would adequately test and validate proper operation of the network and local operating systems. Since one of the constraints of the original software implementation was to design the software in a structured fashion, there were not a great many warnings and prohibited constraints incorporated into the original design. For this reason, the majority of the testing focused on those aspects of the DELNET operation that were suppose to occcur rather than on the endless permutations pertaining to Murphy's Law of 'What If' anomallies. All the safeguards that were built into the system were tested, however.

There was one significant obstacle in testing the DELNET's software. The UNIDs which were being upgraded during a concurrent thesis effort (Ref 4) were only completed during the final week of the testing period. For this reason, much of the software was tested using the MCZ 1/25 computer rather on the Z-80 processors inside the UNID. Since the MCZ utilizes the same Z-80 processor and the UNID processor scheme was designed under the framework of the MCZ, the differences were small. In fact, the RIO Operating System of the MCZ was much superior to the small monitor program of the UNID and it actually expedited development with its robust environment of memory manipulations and troubleshooting tools. The only real drawback was that the MCZ used only a single processor and the local and network operating systems had to be tested independenly without the capability of handshaking. Additionally, the MCZ could not actually transmit or receive frames of data on the network side or packets of data on the local side. The MCZ was used, however, to checkout the internal processing of each of the operating systems when certain conditions were intiated through manual manipulation of variables, table buffers, and pointers.

The global approach for testing the DELNET software was analogous to that of structured design. Each module or service was tested and validated to insure proper operation at the bottom most level. Next, the higher modules were tested which used the already validated sub-modules. Unfortunately, the complete end-to-end test of the DELNET at the global level could not be performed due to the previously mentioned problems encountered with the UNID development. Each internal module was tested; however, and provided confidence that the overall system would function properly if provided with a fully opertional set of UNIDs.

The following paragraphs describe the tests conducted using the MCZ 1/25 to simulate the UNID processing. Each test describes the setup and results. Appendix A may be refered to along with the many figures of preceeding chapters in order to follow the processing flow required and to obtain the results listed. In order to simplify the manipulations of the frames and packets for these tests, the packets contained a total of 30 bytes and the frames 32 bytes. In both cases the header fields were complete.

Each test performed validates specific servies provided by the DELNET operating system. Each service is identified along with the setup and results for the specific test.

<u>Test 1</u> - Reception of Local-to-Local Packet

<u>Setup</u>

a). Place 30 byte packet into LC01TB; all bytes = BB.

b). Set byte 1 to 03 (to UNID No.0, Channel No.3).

c). Set byte 2 to 01 (From UNID No.0, Channel No.1).

d). Set LC01NE to 1E Hex (30 bytes in table).

e). Jump around Init_L_Tab and Init_U_Shtab (this would reinitialize the LC01NE pointer to zero).

f). Set PC and SP

g). Go

<u>Results of Test 1</u>

The packet was properly routed to the LCLCTB and the LCLCNE pointer was updated to 1E Hex. All combinations of the addresses were then placed into the destination address byte 1. The destination address was then changed to an incorrect channel number. The error was properly noted and the STATTB was incremented accordingly. In all cases this software functioned correctly.

<u>Test 2</u> - Reception of Local-to-Network Packet

<u>Setup</u>

a). Place 30 byte packet into LC01TB; all bytes = BB.

b). Set byte 1 to 13 (to UNID No.1, Channel No.3).

c). Set byte 2 to 01 (From UNID No.0, Channel No.1).

d). Set LC01NE to 1E Hex (30 bytes in table).

e). Jump around Init_L_Tab and Init_U_Shtab (this would reinitialize the LC01NE pointer to zero).

f). Set PC and SP

g). Go

<u>Results of Test 2</u>

The packet was properly routed to the LCNTTB and the frame headers for the Data Link Layer were added correctly, thus transforming the packet (30 bytes) to a frame (32 bytes). The new frame header byte 1 was set to 10 and the byte 2 was set to 00. All pointers were properly changed after transfer; LCNTNE was changed to 20 Hex and LC03NS was changed to 1E Hex. All combinations were checked and the system performed in the correct manner. In all cases this software functioned correctly.

Test 3 - Reception of Network-to-Network Frame

<u>Setup</u>

a). Place 32 byte frame into NT01TB; all bytes = BB.

b). Set byte 1 to 10 (To UNID No.1 From UNID No.0)

c). Set byte 2 to 00 (I-Frame, Sequence No.0)

d). Set NT01NE to 20 Hex (32 bytes in table).

e). Jump around Init_L_Tab and Init_U_Shtab (this would reinitialize the NT01NE pointer to zero).

f). Set PC and SP

g). Go

Results of Test 3

The frame was routed to the NTNTTB properly. All pointers were properly updated; the NT01NS was set to 20 Hex and the NTNTNE was set to 20 Hex. The test was repeated with the destination address changed to UNID No.3. This value exceeded the variable Max_UNIDS and the SHATTB was properly incremented. In all cases the software functioned correctly.

Test 4 - Reception of Network-to-Local I-Frame

Setup

a). Place 32 byte frame into NT01TB; all bytes = BB.

b). Set byte 1 to 01 (To UNID No.0 From UNID No.1)

c). Set byte 2 to 00 (I-Frame, Sequence No.0)

d). Set NT01NE to 20 Hex (32 bytes in table).

e). Jump around Init_L_Tab and Init_U_Shtab (this would reinitialize the LC01NE pointer to zero).

f). Set PC and SP

g). Go

Results of Test 4

The frame was properly placed into the NTLCTB with the pointers being updated correctly; NTLCNE was set to 20 Hex and NT01NS was set to 20 Hex. An S-Frame was created and placed into the LCNTTB. The proper headers were placed on the S-Frame; byte 1 was set to 10 and byte 2 was set to 00. The LCNTNE was correctly set to 20 Hex with the addition of the S-Frame. In all cases the software functioned correctly.

<u>Test 5</u> - Reception of Network-to-Local S-Frame

<u>Setup</u>

a). Place 32 byte frame into NT01TB; all bytes = BB.

b). Set byte 1 to 10 (To UNID No.1 From UNID No.0)

c). Set byte 2 to A0 (S-Frame, Sequence No.1)

d). Set NT01NE to 20 Hex (32 bytes in table).

e). Jump around Init_L_Tab and Init_U_Shtab (this would reinitialize the LC01NE pointer to zero).

f). Set PC and SP

g). Go

<u>Results of Test 5</u>

The results were correct. The only action taken was that the pointer NT01NS was set to 20 Hex. The varialble 'acknowledge' should have been complimented during actual operation. But since the varialble is not global, its values could not be confirmed during this test. Both combinations of sequence numbers were tested. In all cases the software functioned correctly.

<u>Test 6</u> - Transmission to Network of I-Frame and S-Frame

<u>Setup</u>

a). Place 32 byte frame into LCNTTB; all bytes = BB.

b). Set byte 1 to 10 (To UNID No.1 From UNID No.0)

c). Set byte 2 to 00 (I-Frame, Sequence No.0)

d). Set LCNTNE to 20 Hex (32 bytes in table).

e). Jump around Init_L_Tab and Init_U_Shtab (this would reinitialize the LC01NE pointer to zero).

f). Set variable MAXNUM to 100 (allow 2.7 secs. between transmissions).

g). Set PC and SP

h). Go

Results of Test 6

The I-Frame was transmitted to the network once every 2.7 seconds. The pointer LCNTNS was never updated because the varialble 'acknowledge' could not be complimented by the arrival of an S-Frame. Between the successive transmissions of I-Frames, the normal processing of routing in and out data continued. Byte 2 of the frame in the LCNTTB was changed to A0 Hex to indicate an S-Frame. The S-Frame was transmitted once to the network and pointer LCNTNS was set to 20 Hex correctly. In all cases the software functioned correctly.

Summary

The Zilog MCZ 1/25 microcomputer and its supporting software were used to develop and support the testing of all software modules for this thesis effort. This system which supports the philosophy of structured software modules, relocatable code, and the combination of higher order and assembly language programming, provides a robust environment for the development of operating systems.

The DELNET operating system is composed of two basic components. The first is the network operating system which is controlled by a Z-80 processor within the network side of the UNID. It controls the operations pertaining to the Data Link protocol layer for frame traffic between UNIDs on the network bus. The second is the local operating system which is also controlled by a Z-80 processor but located on the local side of the UNID. It controls the local packet traffic of the Network protocol layer between the hosts and the UNID.

Both the local and network operating systems have access to their own memory partitions for unique system operations as well as to shared memory for intercommunications and packet/frame sharing. There are a total of eight software modules; three for the local side, three for the network side, and two for shared memory. Each software module was tested and validated to perform properly for all operations pertaining too inter-UNID processing and communication.

VII. Conclusions And Recommendations

The purpose of this investigation was to continue the initial design of the DELNET operating system and to implement it in such a manner as to make the UNID minimally functional. The majority of the specific objectives were accomplished. The continuing problems encountered with the UNID hardware, however, greatly hampered the testing and overall developmental efforts of this project. This report provides a firm foundation for continued development for the DELNET and its operating system.

Although the software is minimal in scope, it is based on accepted standards and has been developed in such a manner as to allow for expansion within these standards.

Conclusions

The foremost conclusion of this thesis effort pertains to the validity of the DELNET operating system which was initially designed by the previous AFIT MS student (Ref 9). This study confirms that the software and data structures initially designed into the operating system are sufficient to perform the tasks required for DELNET operation. Although perhaps not optimal in structure, they provide for an easily understandable and maintainable software system which can be extended.

Within the limitations of the test equipment available, the local and network operating system modules functioned properly. It is regrettable that the hardware limitations of the UNID (Ref 6) did not allow a more complete demonstration of the DELNET with true inter-UNID communications. Due to the UNID anomalies, the actual routing of the frames into and out of the UNID was not accomplished. This was demonstrated, however, during the previous thesis effort (Ref 9).

The designs and techniques used to implement the Physical Layer of protocol (layer 1) were demonstrated and found valid in the previous thesis effort (Ref 9). the RS-232C and RS-449 standards provide for all the services necessary for complete communications on the channels for both local and network traffic within the DELNET.

The designs and techniques used to implement the Data Link Layer of protocol (layer 2) were much more complicated than the Physical Layer due to the large variety of options for available services. Since the standards which govern this layer are truely universal in scope, great care was exercised to select a methology which did not violate standards yet allowed a simplified appraoch. Using a ring topology and store-and-foward simplex (unidirectional) routing approach, a protocol subset was chosen. The subset chosen was complete in that it provided for a minimum of services that would insure proper data link operation. These included such services as flow control, error detection, and virtual addressing. It accomplishes this by using a modulo 2 sequencing scheme, two types of frames (information and supervisory), and a 'transmit- wait for reply - retransmit if no reply' acknowledgement philosop'y. Much of the difficult flow control and error detection services were performed by the Z-80 SIO (Ref 16).

The designs and techniques used to implement the Network Layer of protocol (layer 3) were perhaps the most ambiguous. Athough Reference 7 contains the standards for this layer, the actual implementation is quite complex. This is because the standards allow for a wide variety of services, many of which the DELNET does not require. For a rather uncomplicated network such as the DELNET, a subset of services was selected.

A single packet type was chosen to incorporate this protocol level in its present configuration. Once the Transport Layer protocol services are defined, the Network/Transport Layer interface will probably require additional services and therefore the addition of various other types of packets. The present single packet type contains specific header fields which are used to insure correct packet routing and packet sequencing. Additional fields were left blank so that additional services could be provided under the existing standards.

When these three layers are combined, they provide for a minimally operational system. If provided with a set of fuctional UNIDs they can receive/transmit packets between the UNID and host computers, transform the packets into frames, and receive/tran:mit frames between UNIDs. Thus, the virtual information transfer for host to host has been achieved.

Recommendations

Because this thesis effort is a continuation of previous research, the overall recommendation is to continue with the DELNET development project. The specific objectives for future efforts fall into three major areas. The first is to improve upon the initial three levels of protocol thus far developed. The second is to continue to develop the successive higher levels of protocol. And the third is to develop a network monitoring system that can provide real time network monitoring and evaluation. In either case, the guidelines of standards set forth in Chapter II must be carefully followed. In particular, the future projects must pay close attention to the seven layer model of the ISO.

There are enough additional services or enhancements that could be incorporated into either the Data Link Layer or Network Layer that would require several dedicated thesis projects. The following recommendations, however, pertain only to those additional services and tasks that would be of benefit to the DELNET as it is viewed for use in the foreseeable future.

The first recommendation pertains to the mode of the communications. Complete LAP and HDLC protocols utilize full duplex communications rather than the DELNET's present simplex method. The DELNET should be upgraded to at least a half duplex if not full duplex capability. The second recommendation is to increase the sequence numbering scheme from the present modulo 2 to either modulo 8 or modulo 128. This would increase the I-frame traffic and greatly reduce the overhead of the S-frame traffic. Thirdly, the data link frames should be made of variable length. Under the present scheme of fixed buffers and pointers, this would require substantial rework of the data strctures. Next, the acknowledgement of I-frames should be made point to point (UNID to UNID) around the network instead of just from the destination UNID to the source UNID. Lastly, a new 'Request To Send' S-frame should be incorporated between adjacent UNIDs to reduce errors and increase flow control. The incorporation of each of these recommendations would widen the subset of services that the LAP protocol is suppose to provide for the Data Link Layer and bring the DELNET closer to full standard compliance.

The recommendations for the Network Layer of protocol basically parallels those recommendations for the Data Link Layer. The X.25 standard incorporates numerous packet types that provide a wide variety of services. The Network Layer for the DELNET should incorporate a packet numbering scheme that is similar to that of frames of the Data Link Layer. Additionally, the size of the packets should be made of variable length. Thirdly, the X.25 acknowlegement scheme should be developed between the UNID and its hosts. Next, a variety of Network Layer supervisory tasks should be incorporated according to the X.25 standard to allow for error detection and flow control.

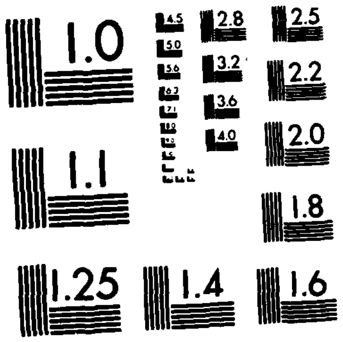
While improvements to the present DELNET protocol levels are important, continued development of the higher levels of protocol is essential in order to obtain a working DELNET in the foreseeable future. In doing so, the DELNET will sacrifice some quality but will realize an actual operational network. If follow-on research focuses on the continued development of higher protocol levels, it is essential that the ISO model be maintained as the overall framework for future development. It is imperative that if a subset of a higher protocol level is selected for DELNET operation, it must be implemented in such a manner as to allow for future enhancements to the full set of allocated serivces of the particular protocol.

The third major area of recommended future research deals with the development of a network monitor. The ability to monitor network operations would provide a means of testing and validating proper system performance. It would also provide the DEL with a pedagogical tool for network research.

Whichever choices are selected for future research, the central theme of the DELNET operating system development is 'quality'. The underlying theme is modularity and structured programming. This software engineering approach together with the established standards described in Chapter II, provide a firm foundation on which the DELNET can be developed. A fully operational network which implements full HDLC and X.25 standards is a tremendously complex system of enormous proportions. Full in-house development which attempts to include all the possible services is perhaps a unrealistic objective. But a fully operational network for the DEL is obtainable using carefully selected subsets of the services from those available.

AD-A124 874 CONTINUED DEVELOPMENT AND IMPLEMENTATION OF THE PROTOCOLS FOR THE DIGITAL..(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. C H HAZELTON 2/3

UNCLASSIFIED DEC 82 AFIT/GE/EE/82D-37 F/G 9/2 NL


1.0
4.5
5.0
5.6
6.3
7.1
8.0
9.0
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

Bibliography

1. Abrams,Marshal Dr. and Blanc, Bobert P. and Cotton, Ira W., "Computer Networks: A Tutorial (JH 3100-5C)," Proceedings of the IEEE, New York, 1978.

2. Baker, Capt Lee R., "Prototype and Software Development for Universal Network Interface Device," Unpublished MS Thesis. School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, 1980.

3. Brown, Capt Eric F. "Prototype Universal Network Interface Device," Unpublished MS Thesis. School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, 1980.

4. Cuomo, Capt Gennaro. "Continuation of UNID Hardware Design," Unpublished MS Thesis. School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, 1982.

5. Davies, D.W., and Barber, D.L.A., and Price, W.L., and Solomondies, C.M. Computer Networks and Their Protocols. Chichester, Great Britain: John Wiley and Sons, 1979.

6. de Sousa, Paulo J.T. and Ingle, Ashok D. and Sharma, Roshan Lal. Network Systems. New York, New York: Von Nostrand Reinhold Co., 1982.

7. Folts, Harold C. and Harry R. Carp (Editors). Data Communications Standards. New York: McGraw-Hill Publications Co.,1978.

8. Freeman, Harvey A. "Local Computer Networks," 26th Annual Symposium of the Central Ohio Chapter of the Association for Computing Machines. Columbus, Ohio, 12 May 1982.

9. Geist,Capt John W."Development of the Digital Engineering Laboratory Computer Network: Host-to-Node/ Host-to-Host Protocols," Unpublished MS Thesis. School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, 1981.

10. Graube, Maris. "Local Area Nets: A pair of Standards," Spectrum, Vol. 19 No.5: 60-64, New York, 1982.

11. Hobart, Capt William C. Jr. "Design of a Local Computer Network for the Air Force Institute of Technology Digital Engineering Laboratory," Unpublished MS Thesis. School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, 1981.

12. Hoskins, Gregory T. and Meisner, Norman B. "Choosing Between Broadband and Baseband Local Networks," Mini-Micro Systems, Vol XV : 265-274, (June 1982).

13. Karp, Peggy M. and Socher, Ivan D. "Designing Local-Area Networks," Mini-Micro Systems, Vol XV : 219-231, (April 1982).

14. Koffman, Elliot B. Problem Solving and Structured Programming in Pascal. Philippines: Addison-Wesley Publishing Co., 1981.

15. Leventhal, Lance A. Z80 Assembly Language Programming. Berkeley, Ca.: Adam Osborne & Associates Inc., 1979.

16. Osborne, Adam et al. An Introduction to Microcomputers (Vol 1-3), Berkeley Ca.: Osborne & Associates, Inc., 1979.

17. Papp, Charles E. "Prototype DELNET Using the Universal Network Interface Device," Unpublished MS Thesis. School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, 1981.

18. Randesi, Stephen J. "Interfacing Minis and Micros to IBM networks," Mini-Micro Systems, Vol XV, No 3: 159-168, (Mar 1982).

19. Roberts, Michael. "Multiprocessing Networks Versus Main Frames," Mini-Micro Systems, Vol XIII No. 10: 121-128, (Oct 1980).

20. Schwartz, Mischa. Computer-Communication Network Design and Analysis. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1977.

21. Sluzevich, Capt Sam C. "Preliminary Design of a Universal Network Interface Device," Unpublished MS Thesis. School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, 1978.

22. Tanenbaum, Andrew S. Computer Networks. Englewood Cliffs, N.J.: Prentice-Hall Inc., 1981.

23. Weaver, Thomas H. "An Engineering Assessment Toward Economic, Feasible, and Responsive Base-Level Telecommunications Through the 1980's", Technical Report 1842EEG/EEIC TR 78-5, 1842 Electonic Engineering Group, Richards-Gebaur AFB, Mo, 31 Oct 77.

24. Yourdan, Edward and Larry Constantine. Structured Design. New York: Yourdan Press, 1978.

25. Zilog, Inc. MCZ-1/20,25 Hardware User's Manual,

Manufacturer's data. Cupertino, Ca.: Zilog, Inc., 1977.

26. Zilog, Inc. PLZ User Guide (03-3096-01), Manufacturer's data. Cupertino, Ca.: Zilog, Inc., July 1979.

27. Zilog, Inc. Z80-MCB Hardware User's Manual, Manufacturer's data. Cupertino, Ca.: Zilog, Inc., 1977.

28. Zilog, Inc. Z80-MCB Software User's Manual (03-3004-01), Manufacturer's data. Cupertino, Ca.: Zilog, Inc., May 1978.

29. Zilog, Inc. Z80-RIO Operating System User's Manual (03-0072-01), Manufacturer's data. Cupertino, Ca.: Zilog, Inc., Sept 1978.

30. Zilog, Inc. Z80-RIO Relocating Assembler and Linker User's Manual, Manufacturer's data. Cupertino, Ca.: Zilog, Inc., 1978.

31. Zilog, Inc. Z80-SIB User's Manual (03-0051-00), Manufacturer's data. Cupertino, Ca.: Zilog, Inc., July 1978.

32. Zilog, Inc. Z80-SIO Technical Manual (03-3033-01), Manufacturer's data. Cupertino, Ca.: Zilog, Inc., Aug 1978.

33. Zimmermann, Hubert. "OSI Reference Model - The ISO Model of Architecture Open Systems Interconnection," IEEE Transactions on Communication, Vol COM 28(4): 425-432, (Apr 1980).

Appendix A

Data Dictionary

This appendix contains the data dictionary for the eight modules which compose the DELNET operating system in its present configuration. This data dictionary is organized by modules which are presented in alphabetical order. Each module contains a section for constants, variables, and procedures which are in turn listed in alphabetical order. Appendix B contains a cross reference list for all constants, variables, and procedures and the modules where they are located.

Table of Contents

Module	Page
L.MAIN	A- 2
L.TAB	A- 4
L.VINT	A- 6
N.MAIN	A- 7
N.TAB	A-10
N.INSIO	A-11
U.SHTAB	A-12
U.LIB	A-13

MODULE L.MAIN

The purpose of this module is to provide the local operating system with the main line of processing. The local operating system is required to input/output data from the four local channels or hand off and receive data from the network operating system.

Constants

CONCMD - Command port address for the USART on the local monitor console.

CONDAT - Data port address for the USART on the local monitor console.

F_TABLE_SIZE - Number of bytes in a frame table buffer.

FRAME_SIZE - Number of bytes in a frame.

L_RI_DEST_ERR - Local route in destination error.

L_RO_DEST_ERR - Local route out destination error.

P_TABLE_SIZE - Number of bytes in a packet table buffer.

PACKET_SIZE - Number of bytes in a packet.

PACKETS_IN_TABLE - Number of packets in a packet table buffer.

STAT_NBR - Number of the status entries to be included in the status table buffer.

********** NOTE **********

The next constant, UNID_NBR, must be unique for each copy of the module L.Main placed within each UNID or incorrect processing will result.

********** NOTE **********

UNID_NBR - Unique UNID number for the UNID performing the evaluation. See above note!

U01DAT - Local channel 1 USART data port address.

U02DAT - Local channel 2 USART data port address.

U03DAT - Local channel 3 USART data port address.

U04DAT - Local channel 4 USART data port address.

Variables

TDAADD - Global, type Pbyte - Starting address of data for to be transmitted out the USARTS.

TPRADD - Global, type byte - Data port address for the USARTS.

DESTINATION - Internal, type word - The destination address of a data packet or frame.

STARTUP_HDR - Internal, type array - A message to the console indicating proper operating system operation.

Procedures

BUILD_I_FRAME - A procedure which transforms a packet into a frame.

DET_DEST - 'Determine Destination' - Determines the destination of a packet or frame by evaluating its headers.

LD_TAB_HSKP - 'Load Table Housekeep' - Housekeeps a specified table after a new packet or frame has been loaded.

MAIN - This is the main procedure which drives other procedures through their proper sequencing.

ROUTE_IN - Routes in packets from their correct input table buffers and places them into their correct output table buffers for transmit.

ROUTE_OUT - Routes out packets from their correct output table buffers to their correct output channels.

SRVC_TAB_HSKP - 'Service Table Housekeep' - Housekeeps a specified table buffer whenever a packet or frame is removed.

TRNMIT_PKT - 'Transmit a Packet' - Transmits a packet out of one of the two output table buffers to one of the local channels.

MODULE L.TAB

The purpose of this module is to provide the local operating system with the table buffers necessary for storing the packets of data after reception and before transmission to the local hosts.

Constants

F_TABLE_SIZE - Number of bytes in a frame table buffer.

FRAME_SIZE - Number of bytes in a frame.

P_TABLE_SIZE - Number of bytes in a packet table buffer.

PACKET_SIZE - Number of bytes in a packet.

PACKETS_IN_TABLE - Number of packets in a packet table buffer.

Variables

LC01TB - Global, type array - Local input table buffer from that interfaces with channel number 1.

LC01NE - Global, type integer - Pointer for the next available position within LC01TB.

LC01NS - Global, type integer - Pointer for the next byte to be serviced within LC01TB.

LC01SZ - Global, type integer - Size of the LC01TB table buffer.

LC02TB - Global, type array - Local input table buffer from that interfaces with channel number 2.

LC02NE - Global, type integer - Pointer for the next available position within LC02TB.

LC02NS - Global, type integer - Pointer for the next byte to be serviced within LC02TB.

LC02SZ - Global, type integer - Size of the LC02TB table buffer.

LC03TB - Global, type array - Local input table buffer from that interfaces with channel number 3.

LC03NE - Global, type integer - Pointer for the next available position within LC03TB.

LC03NS - Global, type integer - Pointer for the next byte to be serviced within LC03TB.

LC03SZ - Global, type integer - Size of the LC03TB table buffer.

LC04TB - Global, type array - Local input table buffer from that interfaces with channel number 4.

LC04NE - Global, type integer - Pointer for the next available position within LC04TB.

LC04NS - Global, type integer - Pointer for the next byte to be serviced within LC04TB.

LC04SZ - Global, type integer - Size of the LC04TB table buffer.

LCLCTB - Global, type array - Local-to-local table buffer that receives packets from local hosts that are destined for other local hosts.

LCLCNE - Global, type integer - Pointer for the next available position within LCLCTB.

LCLCNS - Global, type integer - Pointer for the next byte to be serviced within LCLCTB.

LCLCSZ - Global, type integer - Size of the LCLCTB table buffer.

<u>Procedures</u>

INIT_L_TAB - 'Initialize Local Table Buffers' - Sets up the local table buffers and initializes the pointers to zero.

MODULE L.VINT

The purpose of this module is to support the local operating system and its processing. L.VINT is an assembly language module and does not have any declared constants or varialbes.

Procedures

INVINT - 'Intialize Vector Interrupt Mode' - The purpose of this procedure is to initialize the vector interrupt process through the use of the Priority Interrupt Controller (PIC).

INIURT - 'Initialize Local Card USARTS' - Initializes the 2651 USARTS on the UNID local board.

TRNMIT - 'Transmit' - The purpose of this procedure is to enable a transmit interrupt from a PLZ module.

URTR01 - 'I/O Receive Interrupt Controller' - The purpose of this procedure is to service local channel 01 interrupts.

URTR02 - 'I/O Receive Interrupt Controller' - The purpose of this procedure is to service local channel 02 interrupts.

URTR03 - 'I/O Receive Interrupt Controller' - The purpose of this procedure is to service local channel 03 interrupts.

URTR04 - 'I/O Receive Interrupt Controller' - The purpose of this procedure is to service local channel 04 interrupts.

URTTRN - 'I/O Transmit Interrupt' - The purpose of this procedure is to service the local channel transmit interrupts.

MODULE N.MAIN

The purpose of this module is to provide the network operating system with the main line of processing. The network operating system is required to input/output data from the network channel or hand off and receive data from the local operating system.

Constants

CONCMD - Command port address for the USART on the local monitor console.

CONDAT - Data port address for the USART on the local monitor console.

F_TABLE_SIZE - Number of bytes in a frame table buffer.

FALSE - Boolean word use for high order control.

FRAME_SIZE - Number of bytes in a frame.

FRAMES_IN_TABLE - Number of frames in a frame table buffer.

HDR00 - Frame header byte 00, address word.

HDR01 - Frame header byte 01, control word.

NET_RI_DEST_ERR - Network route in destination error.

NET_RO_DEST_ERR - Network route out destination error.

PACKET_SIZE - Number of bytes in a packet.

PACKETS_IN_TABLE - Number of packets in a packet table buffer.

STAT_NBR - Number of the status entries to be included in the status table buffer.

********** NOTE **********

The next constant, UNID_NBR, must be unique for each copy of the module L.Main placed within each UNID or incorrect processing will result.

********** NOTE **********

UNID_NBR - Unique UNID number for the UNID performing the evaluation. See above note!

Variables

ACKNOWLEDGE - Internal, type byte - Indicates either true or false if a good acknowledgement frame has been received.

COMPLT - Global, type byte - Indicates either true or false if the TIME_DELAY procedure is complete.

CTCCNT - Global, type byte - Counter for CTC. Incremented once each timeout of CTC.

DESTINATION - Internal, type word - Destination of data packet.

INPUT_SEQ_BIT - Internal, type byte - Sequence bit (modulo 2) to be entered into new frame.

********** NOTE **********

The next variable, MAX_UNIDS, must be set to the exact number of UNIDs in operation on the DELNET or improper processing will result.

********** NOTE **********

MAX_UNIDS - Internal, type byte - The maximum number of UNIDs connected to the DELNET. The number must be changed if UNIDs are added or removed or inproper processing will result. See note above.

MAXNUM - Global, type byte - The maximum number of times the CTC will cycle through its counting routine.

S_FRAMETB - Internal, type array - Supervisory frame table used to build up an S-frame.

SEQ_BIT - Internal, type byte - Sequence bit (modulo 2) of an active I-frame.

STARTUP_HDR - Internal, type array - A message to the console indicating proper operating system opertion.

THIS_SEQ_BIT - Internal, type byte - Sequence bit that is presently under examination.

Procedures

BUILD_S_FRAME - A procedure which builds an S-frame and places it into the proper location for network transmission.

DET_DEST - 'Determine Destination' - Determines the destination of a packet or frame by evaluating its headers.

LD_TAB_HSKP - 'Load Table Housekeep' - Housekeeps a specified table after a new packet or frame has been loaded.

MAIN - This is the main procedure which drives other procedures through their proper sequencing.

ROUTE_IN - Routes in frames from the network bus and places them into their correct table buffers for evaluation.

ROUTE_OUT - Routes out frames from their correct output table buffers to the network bus..

SRVC_TAB_HSKP - 'Service Table Housekeep' - Housekeeps a specified table buffer whenever a packet or frame is removed.

TIME_DELAY - Creates a time delay between succesive transmissions if I-frames.

MODULE N.TAB

The purpose of this module is to provide the network operating system with the table buffers necessary for storing the frames of data after reception and before transmission to the network bus.

Constants

F_TABLE_SIZE - Number of bytes in a frame table buffer.

FRAME_SIZE - Number of bytes in a frame.

FRAMES_IN_TABLE - Number of Frames in a frame table buffer.

PACKET_SIZE - Number of bytes in a packet.

PACKETS_IN_TABLE - Number of packets in a packet table buffer.

Variables

NT01TB - Global, type array - Network input table buffer from network bus.

NT01NE - Global, type integer - Pointer for the next available position within NT01TB.

NT01NS - Global, type integer - Pointer for the next byte to be serviced within NT01TB.

NT01SZ - Global, type integer - Size of the NT01TB table buffer.

NTNTTB - Global, type array - Network output table buffer for the network bus.

NTNTNE - Global, type integer - Pointer for the next available position within NTNTTB.

NTNTNS - Global, type integer - Pointer for the next byte to be serviced within NTNTTB.

NTNTSZ - Global, type integer - Size of the NTNTTB table buffer.

Procedures

INIT_N_TAB - 'Initialize the network table buffers' - Sets up the network table buffers and initializes the pointers to zero.

MODULE N.INSIO

The purpose of this module is to support the network operating system and its processing. N.INSIO is an assembly language module and does not have any declared constants or variables.

Procedures

INSIO - 'Initialize SIO' - The purpose of this procedure is to initialize the I/O process for frames transmitted and received on the network bus.

SIOREC - 'SIO Receive Interrupt Cntroller' - This procedure services the receive interrupt requests for frames coming into the UNID for the network bus.

STCTC3 - 'Start CTC Channel 3' - This procedure sets up the CTC 3 for proper operation.

TRNMIT - 'Transmit' - This procedure transmits a frame out on the network bus.

MODULE U.SHTAB

The purpose of this module is to provide both the local and network operating system with a shared interface for which they can exchange information. In the present form this interface is a pair of table buffers which will be located in the shared memory partition of the UNID memory.

Constants

F_TABLE_SIZE - Number of bytes in a frame table buffer.

FRAME_SIZE - Number of bytes in a frame.

P_TABLE_SIZE - Number of bytes in a packet table buffer.

PACKET_SIZE - Number of bytes in a packet.

PACKETS_IN_TABLE - Number of packets in a packet table buffer.

STAT_NBR - Number of the status entries to be included in the status table buffer.

Variables

LCNTTB - Global, type array - Table buffer for transferring packets from local side to the network bus.

LCNTNE - Global, type integer - Pointer for the next available position within LCNTTB.

LCNTNS - Global, type integer - Pointer for the next byte to be serviced within LCNTTB.

LCNTSZ - Global, type integer - Size of the LCNTTB table buffer.

NTLCTB - Global, type array - Table buffer used for storing frames received from network side and going to local hosts.

NTLCNE - Global, type integer - Pointer for the next available position within NTLCTB.

NTLCNS - Global, type integer - Pointer for the next byte to be serviced within NTLCTB.

NTLCSZ - Global, type integer - Size of the NTLCTB table buffer.

MODULE U.LIB

The purpose of this module is to support both the local and network operating system with a series of assembly laguage library routines. Because it is an assembly language routine, it does not have declared constants or variables.

Procedures

MOVSEQ - 'Move Sequence' - This is a procedure to move a block of data from one area of memory to the next.

RECSEQ - 'Receive Sequence' - This is a procedure to receive a squence of bytes from an identified port.

SNDSEQ - 'Send Sequence' - This is a procedure to send a sequence of data out of an identified port.

Appendix B

Data Dictionary Cross Reference

This appendix contains all the constants, variables, and procedures used in all eight modules of the software which compose the DELNET operating system. The identifiers are arranged in alphabetical order and list the specific modules from Appendix A where a full description may be found.

Identifier	Modules
ACKNOWLEDGE	N.MAIN
BUILD_I_FRAME	L.MAIN
BUILD_S_FRAME	N.MAIN
COMPLT	N.MAIN
CONCMD	L.MAIN, N.MAIN
CONDAT	L.MAIN, N.MAIN
CTCCNT	N.MAIN
DESTINATION	L.MAIN
DET_DEST	L.MAIN, N.MAIN
F_TABLE_SIZE	L.MAIN, L.TAB, N.MAIN, N.TAB, U.SHTAB
FALSE	N.MAIN
FRAME_SIZE	L.MAIN, L.TAB, N.MAIN, N.TAB, U.SHTAB
FRAMES_IN_TABLE	L.MAIN, L.TAB, N.MAIN, N.TAB
HDR00	N.MAIN
HDR01	N.MAIN
INIT_L_TAB	L.TAB
INIT_N_TAB	N.TAB
INPUT_SEQ_BIT	N.MAIN
INSIO	N.INSIO
INVINT	L.VINT
LCLCTB	L.TAB
LCLCNE	L.TAB
LCLCNS	L.TAB
LCLCSZ	L.TAB
LCNTTB	U.SHTAB
LCNTNE	U.SHTAB
LCNTNS	U.SHTAB
LCNTSZ	U.SHTAB
LC01TB	L.TAB
LC01NE	L.TAB
LC01NS	L.TAB
LC01SZ	L.TAB
LC02TB	L.TAB
LC02NE	L.TAB
LC02NS	L.TAB
LC02SZ	L.TAB
LC03TB	L.TAB
LC03NE	L.TAB
LC03NS	L.TAB
LC03SZ	L.TAB
LC04TB	L.TAB
LC04NE	L.TAB
LC04NS	L.TAB
LC04SZ	L.TAB
L_RI_DEST_ERR	L.MAIN
L_RO_DEST_ERR	L.MAIN
LD_TAB_HSKP	L.MAIN, N.MAIN
MAIN	L.MAIN, N.MAIN
MAX_UNIDS	N.MAIN
MAXNUM	N.MAIN
MOVSEQ	U.LIB
NET_RI_DEST_ERR	N.MAIN

NET_RO_DEST_ERR	N.MAIN
NTLCTB	U.SHTAB
NTLCNE	U.SHTAB
NTLCNS	U.SHTAB
NTLCSZ	U.SHTAB
NTNTTB	N.TAB
NTNTNE	N.TAB
NTNTNS	N.TAB
NTNTSZ	N.TAB
NT01TB	N.TAB
NT01NE	N.TAB
NT01NS	N.TAB
NT01SZ	N.TAB
P_TABLE_SIZE	L.MAIN, L.TAB, N.MAIN, N.TAB, U.SHTAB
PACKET_SIZE	L.MAIN, L.TAB, N.MAIN, N.TAB, U.SHTAB
PACKETS_IN_TABLE	L.MAIN, L.TAB, N.MAIN, N.TAB, U.SHTAB
RECSEQ	U.LIB
ROUTE_IN	L.MAIN, N.MAIN
ROUTE_OUT	L.MAIN, N.MAIN
S_FRAMETB	N.MAIN
SEQ_BIT	N.MAIN
SIOREC	N.INSIO
SNDSEQ	U.LIB
SRVC_TAB_HSKP	L.MAIN, N.MAIN
STARTUP_HDR	L.MAIN, N.MAIN
STAT_NBR	L.MAIN, L.TAB, N.MAIN, N.TAB, U.SHTAB
STCTC3	N.INSIO
TDAADD	L.MAIN
THIS_SEQ_BIT	N.MAIN
TIME_DELAY	N.MAIN
TPRADD	L.MAIN
TRNMIT	L.VINT, N.INSIO
TRNMIT_PKT	L.MAIN
TRUE	N.MAIN
UNID_NBR	L.MAIN, N.MAIN
URTR01	L.VINT
URTR02	L.VINT
URTR03	L.VINT
URTR04	L.VINT
URTTRN	L.VINT
U01DAT	L.MAIN
U02DAT	L.MAIN
U03DAT	L.MAIN
U04DAT	L.MAIN

Appendix C

DELNET Operating System Software Components

This appendix contains the actual software listings which compose the DELNET operating system. The modules are broken up into three major sections. Section I contains those modules which comprise the local operating system. Section II contains those modules which comprise the network operting system. Section III contains the modules which comprise the shared components of the DELNET operating system.

Table of Contents

Section		Page
Section I		C- 2
	L.MAIN	C- 3
	L.TAB	C-27
	L.VINT	C-30
Section II		C-49
	N.MAIN	C-50
	N.TAB	C-71
	N.INSIO	C-74
Section III		C-87
	U.SHTAB	C-88
	U.LIB	C-91

Appendix C Section I

This section of Appendix C contains the software listings which comprise the local operating sysstem.

```
!******************************************************************************
!
!******************************************************************************
!
!PROLOGUE -     MODULE L.MAIN       DATE 26 OCT 82

        THE PURPOSE OF THIS MODULE IS TO PROVIDE THE
    UNID LOCAL OPERATING SYSTEM (L.OS) WITH THE MAIN LINE
    OF PROCESSING.  THE L.OS IS REQUIRED TO INPUT/OUTPUT
    DATA FROM THE FOUR LOCAL CHANNELS OR THE NETWORK.
    THIS MODULE CONSISTS OF THE MAIN LINE PROCEDURE 'MAIN',
    AND SUBORDINATE PROCEDURES BUILD_I_FRAME, DET_DEST,
    LD_TAB_HSKP, SRVC_TAB_HSKP, TRNMIT_PKT, ROUTE_IN,
    AND ROUTE_OUT.
!
!******************************************************************************
!
!******************************************************************************
!
MAIN MODULE

    TYPE
        PBYTE ^BYTE

    CONSTANT
        CONCTC := %D5                   ! CONSOLE CTC PORT ADDRESS !
        CONCMD := %DF                   ! CONSOLE USART COMMAND PORT ADDRESS !
        CONDAT := %DE                   ! CONSOLE USART DATA PORT ADDRESS !
        L_RI_DEST_ERR := 00             ! LOC ROUTE_IN DEST ERROR ENTRY !
        L_RO_DEST_ERR := 01             ! LOC ROUTE_OUT DEST ERROR ENTRY !
        PACKET_SIZE := 30
        FRAME_SIZE := PACKET_SIZE + 2   ! TWO BYTES FOR FRAME HEADERS   !
        PACKETS_IN_TABLE := 10
        STAT_NBR := 20                  ! NBR OF STATUS ENTRIES IN STATTB !
        P_TABLE_SIZE := PACKET_SIZE * PACKETS_IN_TABLE
        F_TABLE_SIZE := FRAME_SIZE * PACKETS_IN_TABLE
        U01DAT := 00                    ! LOCAL CH 1 USART DATA PORT ADD !
        U02DAT := 04                    ! LOCAL CH 2 USART DATA PORT ADD !
        U03DAT := 08                    ! LOCAL CH 3 USART DATA PORT ADD !
    U04DAT := %0C                     ! LOCAL CH 4 USART DATA PORT ADD !
    UNID_NBR := 0                     ! UNIQUE ADDRESS OF THIS UNID    !

EXTERNAL                              ! IN L.VINT !
    INVINT PROCEDURE
    TRNMIT PROCEDURE

EXTERNAL                              ! IN U.LIB !
    MOVSEQ PROCEDURE (SRCADD PBYTE, DTDADD PBYTE, NUMBYT BYTE)
    SNDSEQ PROCEDURE (CMDPRT BYTE, DATPRT BYTE, BYTADD PBYTE, NUMBYT BYTE)
    RECSEQ PROCEDURE (CMDPRT BYTE, DATPRT BYTE, BYTADD PBYTE, NUMBYT BYTE)

EXTERNAL                              ! IN L.TAB !
    INIT_L_TAB PROCEDURE

    LC01TB ARRAY [P_TABLE_SIZE BYTE]
    LC01NS INTEGER
    LC01NE INTEGER
    LC01SZ INTEGER

    LC02TB ARRAY [P_TABLE_SIZE BYTE]
    LC02NS INTEGER
    LC02NE INTEGER
    LC02SZ INTEGER

    LC03TB ARRAY [P_TABLE_SIZE BYTE]
    LC03NS INTEGER
    LC03NE INTEGER
    LC03SZ INTEGER

    LC04TB ARRAY [P_TABLE_SIZE BYTE]
    LC04NS INTEGER
    LC04NE INTEGER
    LC04SZ INTEGER

    LCLCTB ARRAY [F_TABLE_SIZE BYTE]
    LCLCNS INTEGER
    LCLCNE INTEGER
    LCLCSZ INTEGER
```

```
EXTERNAL                               ! IN U.SHTAB !
    INIT_U_SHTAB PROCEDURE

    LCNTTB ARRAY [F_TABLE_SIZE BYTE]
    LCNTNS INTEGER
    LCNTNE INTEGER
    LCNTSZ INTEGER

    NTLCTB ARRAY [F_TABLE_SIZE BYTE]
    NTLCNS INTEGER
    NTLCNE INTEGER
    NTLCSZ INTEGER

    STATTB ARRAY [STAT_NBR BYTE]


GLOBAL                                 ! GLOBAL VARIABLES !
    TDAADD PBYTE                       ! LOC CHNL TRANSMIT DATA ADDRESS !
    TPRADD BYTE                        ! LOC CHNL TRANSMIT PORT ADDRESS !

INTERNAL                               ! INTERNAL VARIABLES USED !
                                       ! THROUGHOUT MODULE !
    DESTINATION WORD                   ! DESTINATION OF PACKET !

    STARTUP_HDR ARRAY [* BYTE] := '%R%R%L%L'
                                  'UNID LOCAL OS%R%L'
                                  'VERSION 27 AUG 82%R%L'
                                  'EXECUTING%R%L}'

INTERNAL

!***************************************************************************
 ***************************************************************************
```

```
PROCEDURE  DET_DEST    DETERMINE DESTINATION OF PACKET

        THE PURPOSE OF THIS PROC IS TO DETERMINE THE
   DESTINATION OF A SPECIFIED PACKET.

INPUT -    THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING
   THE TABLE LOCATION OF THE PACKET TO BE EVALUATED.

PROCESSING -  A CASE STATEMENT IS USED TO DETERMINE WHICH TABLE
   FROM WHICH THE EVALUATION IS TO BE MADE.  IF INPUT FROM THE
   LOCAL SIDE THEN THE PACKET IS ROUTED TO EITHER THE NETWORK
   OR BACK TO ONE OF THE LOCAL BUFFERS.  IF INPUT FROM A LOCAL
   HOST THEN IT IS ROUTED TO THE NETWORK BUFFER OR BACK TO ONE
   OF THE LOCAL BUFFERS. AN ERROR IS NOTED UPON NO MATCH.

OUTPUT -   THE PROC OUTPUTS A TWO CHARACTER ASCII VALUE
   INDICATING THE TABLE OR CHANNEL DESTINATION OF THE PACKET.

INTERFACE -    THIS PROC IS CALLED BY PROC ROUTE_IN FOR INPUT
   PACKETS, AND BY ROUTE_OUT FOR NETWORK PACKETS.

NOTES -  NONE.
*******************************************************************************
!
   INTERNAL

       DET_DEST PROCEDURE(TABLE WORD)
           RETURNS(DESTINATION WORD)
       LOCAL
           BYTE_01 BYTE
           BYTE_02 BYTE
       ENTRY

           IF TABLE                                     ! OBTAIN THE TABLE      !
               CASE '01'                                ! AND DETERMINE IF THE  !
                   THEN                                 ! DESTINATION FIELD OF  !
                   IF ((LC01TB [LC01NS] AND %F0)/ %10 ) <> UNID_NBR
                       THEN                             ! ADDRESS BYTE IS EQUAL !
                       BYTE_01 := 'N'                   ! TO THAT OF THE UNID.  !
```

```
    ELSE                                        ! IF SO THEN BYTE 1 = N !
        BYTE_01 := LC01TB [LC01NS] AND %03   ! FOR BACK TO THE !
    FI                                          ! NETWORK; ELSE THE CH   !
                                                ! OBTAIN THE TABLE       !
CASE '02'                                       ! AND DETERMINE IF THE   !
    THEN                                        ! DESTINATION FIELD OF   !
    IF ((LC02TB [LC02NS] AND %F0)/ %10 ) <> UNID_NBR
        THEN                                    ! ADDRESS BYTE IS EQUAL !
        BYTE_01 := 'N'                          ! TO THAT OF THE UNID.  !
    ELSE                                        ! IF SO THEN BYTE 1 = N !
        BYTE_01 := LC02TB [LC02NS] AND %03   ! FOR BACK TO THE !
    FI                                          ! NETWORK; ELSE THE CH   !
                                                ! OBTAIN THE TABLE       !
CASE '03'                                       ! AND DETERMINE IF THE   !
    THEN                                        ! DESTINATION FIELD OF   !
    IF ((LC03TB [LC03NS] AND %F0)/ %10 ) <> UNID_NBR
        THEN                                    ! ADDRESS BYTE IS EQUAL !
        BYTE_01 := 'N'                          ! TO THAT OF THE UNID.  !
    ELSE                                        ! IF SO THEN BYTE 1 = N !
        BYTE_01 := LC03TB [LC03NS] AND %03   ! FOR BACK TO THE !
    FI                                          ! NETWORK; ELSE THE CH   !
                                                ! OBTAIN THE TABLE       !
CASE '04'                                       ! AND DETERMINE IF THE   !
    THEN                                        ! DESTINATION FIELD OF   !
    IF ((LC04TB [LC04NS] AND %F0)/ %10 ) <> UNID_NBR
        THEN                                    ! ADDRESS BYTE IS EQUAL !
        BYTE_01 := 'N'                          ! TO THAT OF THE UNID.  !
    ELSE                                        ! IF SO THEN BYTE 1 = N !
        BYTE_01 := LC04TB [LC04NS] AND %03   ! FOR BACK TO THE !
    FI                                          ! NETWORK; ELSE THE CH   !

CASE 'LL'                                       ! IF LOCAL-TO-LOCAL TBL !
    THEN                                        ! THEN BYTE 1 = 'T' AND !
    BYTE_01 := 'T'                              ! BYTE 2 WILL EQUAL THE !
    BYTE_02 := LCLCTB [LCLCNS] AND %03   ! CHANNEL NUMBER        !
CASE 'NL'                                       ! IF NET-TO LOCAL TABLE !
    THEN                                        ! THEN BYTE 1 = 'T' AND !
    BYTE_01 := 'T'                              ! BYTE 2 WILL EQUAL THE !
    BYTE_02 := (NTLCTB [NTLCNS+2] AND %03)   ! CHANNEL NUMBER!
```

```
        ELSE                                        ! IF INCORRECT TABLE      !
            BYTE_01 := 'E'                          ! BYTE 1 EQUALS 'E'       !
    FI

  ! NOW MAKE DECISIONS BASED UPON THE BYTE VALUES !
  ! EITHER PLACE IN THE LOCAL-TO-LOCAL TABLE ('LL')   !
  ! OR THE LOCAL-TO-NETWORK TABLE ('LN').              !

    IF BYTE_01
        CASE 1
            THEN
            DESTINATION := 'LL'
        CASE 2
            THEN
            DESTINATION := 'LL'
        CASE 3
            THEN
            DESTINATION := 'LL'
        CASE 4
            THEN
            DESTINATION := 'LL'
        CASE 'N'
            THEN
            DESTINATION := 'LN'
        CASE 'T'
            THEN
            IF BYTE_02
                CASE 1
                    THEN
                    DESTINATION := '01'
                CASE 2
                    THEN
                    DESTINATION := '02'
                CASE 3
                    THEN
                    DESTINATION := '03'
                CASE 4
                    THEN
                    DESTINATION := '04'
```

```
                        ELSE
                            DESTINATION := 'ER'
                            STATTB [00] += 1        ! INCREMENT ERROR TABLE !
                    FI
                ELSE
                    DESTINATION := 'ER'
                    STATTB [01] += 1                ! INCREMENT ERROR TABLE !
            FI

        END DET_DEST

!******************************************************************************

!******************************************************************************
 ******************************************************************************
```

```
PROCEDURE  LD_TAB_HSKP     LOAD TABLE HOUSEKEEP

       THE PURPOSE OF THIS PROC IS TO HOUSEKEEP A SPECIFIED
   BUFFER TABLE AFTER THE LOADING OF A PACKET.

INPUT -    THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING
   THE TABLE REQUIRING HOUSEKEEPING.

PROCESSING -   THE PROC DETERMINES THE TABLE TO BE PROCESSED,
   ADVANCES THE NEXT-EMPTY-BYTE POINTER BY A PACKET_SIZE,
   AND ADJUSTS FOR BUFFER WRAP IF NECESSARY.

OUTPUT -   THE SPECIFIED TABLE HAS ITS NEXT-EMPTY-BYTE
   POINTER ADVANCED BY THE LENGTH OF A PACKET.

INTERFACE -    THIS PROC IS CALLED BY PROC ROUTE_IN AND ROUTE_OUT.

NOTES -    NONE.
*******************************************************************************
!
   INTERNAL

       LD_TAB_HSKP PROCEDURE(TABLE WORD)
       ENTRY

           IF TABLE
               CASE '01'                ! IF CALLED TO HSKP LOC CH 1 TAB !
                   THEN
                   LC01NE := LC01NE + PACKET_SIZE  ! ADV NEXT EMPTY PNTR !
                   IF LC01NE >= LC01SZ             ! IF TABLE WRAP !
                       THEN                        ! THEN SET PNTR TO 0 !
                       LC01NE := 0
                   FI

               CASE '02'                ! IF CALLED TO HSKP LOC CH 2 TAB !
                   THEN
                   LC02NE := LC02NE + PACKET_SIZE  ! ADV NEXT EMPTY PNTR !
                   IF LC02NE >= LC02SZ             ! IF TABLE WRAP !
                       THEN                        ! THEN SET PNTR TO 0 !
```

```
            LC02NE := 0
        FI

    CASE '03'                  ! IF CALLED TO HSKP LOC CH 3 TAB !
        THEN
        LC03NE := LC03NE + PACKET_SIZE  ! ADV NEXT EMPTY PNTR !
        IF LC03NE >= LC03SZ             ! IF TABLE WRAP !
            THEN                        ! THEN SET PNTR TO 0 !
            LC03NE := 0
        FI

    CASE '04'                  ! IF CALLED TO HSKP LOC CH 4 TAB !
        THEN
        LC04NE := LC04NE + PACKET_SIZE  ! ADV NEXT EMPTY PNTR !
        IF LC04NE >= LC04SZ             ! IF TABLE WRAP !
            THEN                        ! THEN SET PNTR TO 0 !
            LC04NE := 0
        FI

    CASE 'LL'                  ! IF CALLED TO HSKP LOC TO LOC TAB !
        THEN
        LCLCNE := LCLCNE + PACKET_SIZE  ! ADV NEXT EMPTY PNTR !
        IF LCLCNE >= LCLCSZ             ! IF TABLE WRAP !
            THEN                        ! THEN SET PNTR TO 0 !
            LCLCNE := 0
        FI

    CASE 'LN'                  ! IF CALLED TO HSKP LOCAL TO NET TAB !
        THEN
        LCNTNE := LCNTNE + PACKET_SIZE  ! ADV NEXT EMPTY PNTR !
        IF LCNTNE >= LCNTSZ             ! IF TABLE WRAP !
            THEN                        ! THEN SET PNTR TO 0 !
            LCNTNE := 0
        FI

FI

END LD_TAB_HSKP
```

```
PROCEDURE  SRVC_TAB_HSKP   SERVICE TABLE HOUSEKEEP

       THE PURPOSE OF THIS PROC IS TO HOUSEKEEP A SPECIFIED
   BUFFER TABLE AFTER SERVICING (REMOVING A PACKET).

INPUT -    THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING
   THE TABLE REQUIRING HOUSEKEEPING.

PROCESSING -   THE PROC DETERMINES THE TABLE TO BE PROCESSED,
   ADVANCES THE NEXT-BYTE-TO-BE-SERVICED POINTER BY A PACKET_SIZE,
   AND ADJUSTS FOR BUFFER WRAP IF NECESSARY.

OUTPUT -   THE SPECIFIED TABLE HAS ITS NEXT-BYTE-TO-BE-SERVICED
   POINTER ADVANCED BY THE LENGTH OF A PACKET.

INTERFACE -    THIS PROC IS CALLED BY PROC ROUTE_IN AND ROUTE_OUT.

NOTES -    NONE.
*****************************************************************************
!
   INTERNAL

       SRVC_TAB_HSKP PROCEDURE(TABLE WORD)
       ENTRY

           IF TABLE
               CASE '01'                ! IF CALLED TO HSKP LOC CH 1 TAB !
                   THEN
                   LC01NS := LC01NS + PACKET_SIZE  ! ADV NEXT SERVICE PNTR !
                   IF LC01NS >= LC01SZ             ! IF TABLE WRAP !
                       THEN                        ! THEN SET PNTR TO 0 !
                       LC01NS := 0
                   FI

               CASE '02'                ! IF CALLED TO HSKP LOC CH 2 TAB !
                   THEN
                   LC02NS := LC02NS + PACKET_SIZE  ! ADV NEXT SERVICE PNTR !
                   IF LC02NS >= LC02SZ             ! IF TABLE WRAP !
                       THEN                        ! THEN SET PNTR TO 0 !
```

```
            LC02NS := 0
        FI

    CASE '03'                   ! IF CALLED TO HSKP LOC CH 3 TAB !
        THEN
        LC03NS := LC03NS + PACKET_SIZE  ! ADV NEXT SERVICE PNTR !
        IF LC03NS >= LC03SZ             ! IF TABLE WRAP !
            THEN                        ! THEN SET PNTR TO 0 !
            LC03NS := 0
        FI

    CASE '04'                   ! IF CALLED TO HSKP LOC CH 4 TAB !
        THEN
        LC04NS := LC04NS + PACKET_SIZE  ! ADV NEXT SERVICE PNTR !
        IF LC04NS >= LC04SZ             ! IF TABLE WRAP !
            THEN                        ! THEN SET PNTR TO 0 !
            LC04NS := 0
        FI

    CASE 'LL'                   ! IF CALLED TO HSKP LOC TO LOC TAB !
        THEN
        LCLCNS := LCLCNS + PACKET_SIZE  ! ADV NEXT SERVICE PNTR !
        IF LCLCNS >= LCLCSZ             ! IF TABLE WRAP !
            THEN                        ! THEN SET PNTR TO 0 !
            LCLCNS := 0
        FI

    CASE 'NL'                   ! IF CALLED TO HSKP NET TO LOCAL TAB !
        THEN
        NTLCNS := NTLCNS + FRAME_SIZE   ! ADV NEXT EMPTY PNTR !
        IF NTLCNS >= NTLCSZ             ! IF TABLE WRAP !
            THEN                        ! THEN SET PNTR TO 0 !
            NTLCNS := 0
        FI

  FI

END SRVC_TAB_HSKP
```

```
PROCEDURE  TRNMIT_PKT      TRANSMIT A PACKET

       THE PURPOSE OF THIS PROC IS TO SET UP THE DATA AND PORT
   ADDRESSES FOR PACKET TRANSMISSION, AND DRIVE BYTE TRANSMISSION
   UNTIL AN ENTIRE PACKET HAS BEEN SENT.

INPUT -    THE PACKET'S FIRST BYTE ADDRESS AND THE USART DATA PORT
   ADDRESS ARE INPUT TO TRNMIT_PKT.

PROCESSING -   TWO GLOBAL VALUES, TDAADD (DATA ADD) AND TPRADD
   (PORT ADD), ARE LOADED WITH THE CORRECT INITIAL VALUES.
   THE PROC THEN LOOPS WITH BYTE TRANSMISSION VIA TRNMIT.
   THIS LOOP CONTINUES TO OUTPUT AND ADVANCE THE DATA ADDRESS
   UNTIL A FULL PACKET HAS BEEN TRANSMITTED.

OUTPUT -   A PACKET_SIZE NUMBER OF BYTES ARE TRANSMITTED ON THE
   CHANNEL SPECIFIED BY PRTADD.

INTERFACE -    THIS PROC IS CALLED BY ROUTE_OUT. IT CALLS PROC
   TRNMIT IN MODULE L.VINT FOR BYTE OUTPUT.

NOTES -    NONE.
*******************************************************************************
!
   INTERNAL

       TRNMIT_PKT PROCEDURE(SRCADD PBYTE, PRTADD BYTE)
       LOCAL
           IX INTEGER                     ! INDEX !
       ENTRY

           IX := 0                        ! SET LOOP IX TO START !
           TDAADD := SRCADD               ! LD DATA ADDRESS FOR TRNMIT !
           TPRADD := PRTADD               ! LD PORT ADDRESS FOR TRNMIT !
           DO                             ! LOOP AND TRNMIT A PACKET !
               TRNMIT
               IX += 1
               TDAADD := INC TDAADD
               IF IX = PACKET_SIZE
```

```
                    THEN
                    EXIT
                FI
            OD

        END TRNMIT_PKT

!*******************************************************************************
!
!*******************************************************************************
********************************************************************************
```

```
PROCEDURE BUILD_I_FRAME     PROC FOR TRANSFORMING A PACKET TO A FRAME

        THE PURPOSE OF THIS PROCEDURE IS TO TRANSFORM A PACKET OF
    DATA DELIVERED TO ONE OF THE LOCAL INPUT BUFFERS INTO A FRAME
    TO BE PLACED IN THE LOCAL-TO-NETWORK BUFFER.  THIS PROCEDURE
    ADDS ON THE TWO HEADER BYTES: ADDRESS BYTE AND CONTROL BYTE.

INPUT -  THIS PROC RECEIVES A TWO CHARACTER ASCII VALUE INDICATING THE
    LOCAL INPUT BUFFER TABLE WHERE THE PACKET IS LOCATED.

PROCESSING -  THE PROCESSING BEGINS WITH THE PASSING OF THE TABLE WHERE
    THE PACKET IS LOCATED.  THE DESTNATION ADDRESS IS READ FROM THE
    PACKET HEADER FIELD AND PLACED INTO THE  FRAME HEADER FIELD.
    THE SOURCE ADDRESS IS ADDED BY USING THE UNID_NBR VARIABLE.
    THE CONTROL FIELD BYTE IS INTIALIZED TO ZERO AND WILL BE CHANGED BASED
    UPON LATER ACTION BY THE ROUTING PROCEDURES.

OUTPUT -  THIS PROC PLACES THE FIRST TWO BYTES INTO THE LOCAL-TO-NETWORK
    BUFFER TABLE BEFORE THE PACKET IS TRANSFERRED OVER.

INTERFACE -  THE PROC IS CALLED FROM THE ROUTE_IN PROCEDURE FOR THOSE
    DATA PACKETS DESTINED FOR THE NETWORK ONLY.

NOTES -  THE HEADER INFORMATION IS BASED ON THE FIELDS DEFINED BY THIS
    THESIS ONLY.  EVEN THOUGH THEY CORRESPOND TO THE HDLC PROTOCOL,
    CAUTION SHOULD BE TAKEN BEFORE ADDING OF DELETING ADDITIONAL HEADERS.
******************************************************************************
!
INTERNAL

    BUILD_I_FRAME PROCEDURE (TABLE WORD)
        LOCAL ADDRESS_BYTE BYTE
              CONTROL_BYTE BYTE

    ENTRY
    IF TABLE                                          ! OBTAIN WHICH TABLE THE     !
        CASE '01'                                     ! PACKET IS CONTAINED.  THEN !
            THEN                                      ! DETERMINE THE DESTINATION  !
            ADDRESS_BYTE := (LC01TB [LC01NS] AND %F0)  ! ADDRESS OF THE PACKET!
```

```
                            OR (UNID_NBR)       ! AND ADD THE SOURCE ADDRESS.!
            CONTROL_BYTE := 0                   ! INITIALIZE CONTOL BYTE TO  !
            LCNTTB [LCNTNE] := ADDRESS_BYTE     ! ZERO.  THEN PLACE ADDRESS  !
            LCNTTB [LCNTNE+1] := CONTROL_BYTE   ! BYTE IN FIRST POSITION AND !
                                                ! CONTROL BYTE IN SECOND.    !
        CASE '02'
            THEN                                ! DETERMINE THE DESTINATION  !
            ADDRESS_BYTE := (LC02TB [LC02NS] AND %F0)  ! ADDRESS OF THE PACKET!
                            OR (UNID_NBR)       ! AND ADD THE SOURCE ADDRESS.!
            CONTROL_BYTE := 0                   ! INITIALIZE CONTOL BYTE TO  !
            LCNTTB [LCNTNE] := ADDRESS_BYTE     ! ZERO.  THEN PLACE ADDRESS  !
            LCNTTB [LCNTNE+1] := CONTROL_BYTE   ! BYTE IN FIRST POSITION AND !
                                                ! CONTROL BYTE IN SECOND.    !
        CASE ('03')
            THEN                                ! DETERMINE THE DESTINATION  !
            ADDRESS_BYTE := (LC03TB [LC03NS] AND %F0)  ! ADDRESS OF THE PACKET!
                            OR (UNID_NBR)       ! AND ADD THE SOURCE ADDRESS.!
            CONTROL_BYTE := 0                   ! INITIALIZE CONTOL BYTE TO  !
            LCNTTB [LCNTNE] := ADDRESS_BYTE     ! ZERO.  THEN PLACE ADDRESS  !
            LCNTTB [LCNTNE+1] := CONTROL_BYTE   ! BYTE IN FIRST POSITION AND !
                                                ! CONTROL BYTE IN SECOND.    !
        CASE ('04')
            THEN                                ! DETERMINE THE DESTINATION  !
            ADDRESS_BYTE := (LC04TB [LC04NS] AND %F0)  ! ADDRESS OF THE PACKET!
                            OR (UNID_NBR)       ! AND ADD THE SOURCE ADDRESS.!
            CONTROL_BYTE := 0                   ! INITIALIZE CONTOL BYTE TO  !
            LCNTTB [LCNTNE] := ADDRESS_BYTE     ! ZERO.  THEN PLACE ADDRESS  !
            LCNTTB [LCNTNE+1] := CONTROL_BYTE   ! BYTE IN FIRST POSITION AND !
                                                ! CONTROL BYTE IN SECOND.    !
        ELSE
            STATTB [00] += 1                    ! IF IMPROPER TABLE INCREMENT!
        FI                                      ! ERROR TABLE  !

    END BUILD_I_FRAME
!**************************************************************************
!
!***************************************************************************
 ***************************************************************************
```

```
PROCEDURE  ROUTE_IN        ROUTE PACKETS IN

        THE PURPOSE OF THIS PROC IS TO ROUTE PACKETS FROM
    THE FOUR INPUT BUFFERS TO THEIR CORRECT OUTPUT BUFFER.

INPUT -     DATA PACKETS ARE ROUTED VIA EVALUATION OF LCXXTB POINTERS
    AND INTERNAL PACKET ROUTING INFORMATION.

PROCESSING -    THE PROC CHECKS EACH INPUT BUFFER'S POINTERS FOR
    PACKET ARRIVAL.  IF A PACKET IS READY, THE DESTINATION IS
    DETERMINED VIA PROC DET_DEST, AND ROUTED VIA PROC MOVSEQ.
    BOTH THE BUFFER TABLE THAT IS LOADED AND THE TABLE THAT
    IS SERVICED HAVE THEIR POINTERS HOUSEKEPT BY LD_TAB_HSKP
    AND SRVC_TAB_HSKP.
    IF THE PACKETS ARE BEING PLACED INTO THE LOCAL-TO-NETWORK BUFFER
    TABLE, THEY ARE TRANSFORMED INTO FRAME SIZE BY ADDING 2 BYTES
    FOR THE HEADER SPACE.

OUTPUT -    A PACKET OF DATA IS MOVED TO A DESTINATION BUFFER.

INTERFACE -     THIS PROC IS CALLED IN AN ENDLESS LOOP BY PROC MAIN.

NOTES -     NONE.
******************************************************************************
!
    INTERNAL

        ROUTE_IN PROCEDURE
        ENTRY


            IF ((LC01NE-LC01NS) >= PACKET_SIZE)      !IF CH 1 PACKET IS RDY !
            ORIF (LC01NS > LC01NE)
                THEN                                 ! THEN !
                DESTINATION := DET_DEST('01')        ! DETERMINE DESTINATION !
                                                     ! MOVE DATA AND HSKP TBLS !
                IF DESTINATION
                    CASE 'LL'                        ! LOCAL DESTINATION !
                        THEN
            MOVSEQ( LC01TB[LC01NS],
                    LCLCTB[LCLCNE],
                    PACKET_SIZE)
            LD_TAB_HSKP('LL')

        CASE 'LN'                              ! NETWORK DESTINATION !
            THEN                               ! PLACE PACKET IN FRAME !
            BUILD_I_FRAME ('01')               ! AND MOVE TO DESTINATION !
            MOVSEQ( LC01TB[LC01NS],
                    LCNTTB[LCNTNE+2],
                    PACKET_SIZE)
            LD_TAB_HSKP('LN')

        ELSE
            STATTB[L_RI_DEST_ERR] += 1         ! COUNT ERROR !
    FI

    SRVC_TAB_HSKP('01')

FI

IF ((LC02NE-LC02NS) >= PACKET_SIZE)            !IF CH 2 PACKET IS RDY !
ORIF (LC02NS > LC02NE)
    THEN                                       ! THEN !
    DESTINATION := DET_DEST('02')              ! DETERMINE DESTINATION !
                                               ! MOVE DATA AND HSKP TBLS !
     ? DESTINATION
        CASE 'LL'                              ! LOCAL DESTINATION !
            THEN
            MOVSEQ( LC02TB[LC02NS],
                    LCLCTB[LCLCNE],
                    PACKET_SIZE)
            LD_TAB_HSKP('LL')

        CASE 'LN'                              ! NETWORK DESTINATION !
            THEN                               ! PLACE PACKET IN FRAME !
            BUILD_I_FRAME ('02')               ! PLACE PACKET IN FRAME !
            MOVSEQ( LC02TB[LC02NS],
                    LCNTTB[LCNTNE+2],
                    PACKET_SIZE)
                LD_TAB_HSKP('LN')

            ELSE
                STATTB[L_RI_DEST_ERR] += 1   ! COUNT ERROR !
        FI

        SRVC_TAB_HSKP('02')

    FI

    IF ((LC03NE-LC03NS) >= PACKET_SIZE)      !IF CH 3 PACKET IS RDY !
    ORIF (LC03NS > LC03NE)
        THEN                                 ! THEN !
        DESTINATION := DET_DEST('03')        ! DETERMINE DESTINATION !
                                             ! MOVE DATA AND HSKP TBLS !
        IF DESTINATION
            CASE 'LL'                        ! LOCAL DESTINATION !
                THEN
                MOVSEQ( LC03TB[LC03NS],
                        LCLCTB[LCLCNE],
                        PACKET_SIZE)
                LD_TAB_HSKP('LL')

            CASE 'LN'                        ! NETWORK DESTINATION !
                THEN                         ! PLACE PACKET IN FRAME !
                BUILD_I_FRAME ('03')         ! AND MOVE TO DESTINATION !
                MOVSEQ( LC03TB[LC03NS],
                        LCNTTB[LCNTNE+2],
                        PACKET_SIZE)
                LD_TAB_HSKP('LN')

            ELSE
                STATTB[L_RI_DEST_ERR] += 1   ! COUNT ERROR !
        FI

        SRVC_TAB_HSKP('03')

    FI
    IF ((LC04NE-LC04NS) >= PACKET_SIZE)     !IF CH 4 PACKET IS RDY !
    ORIF (LC04NS > LC04NE)
        THEN                                ! THEN !
        DESTINATION := DET_DEST('04')       ! DETERMINE DESTINATION !
                                            ! MOVE DATA AND HSKP TBLS !
        IF DESTINATION
            CASE 'LL'                       ! LOCAL DESTINATION !
                THEN
                MOVSEQ( LC04TB[LC04NS],
                        LCLCTB[LCLCNE],
                        PACKET_SIZE)
                LD_TAB_HSKP('LL')

            CASE 'LN'                       ! NETWORK DESTINATION !
                THEN                        ! PLACE PACKET IN FRAME !
                BUILD_I_FRAME ('04')        ! AND MOVE TO DESTINATION !
                MOVSEQ( LC04TB[LC04NS],
                        LCNTTB[LCNTNE+2],
                        PACKET_SIZE)
                LD_TAB_HSKP('LN')

            ELSE
                STATTB[L_RI_DEST_ERR] += 1  ! COUNT ERROR !
        FI

        SRVC_TAB_HSKP('04')

    FI

END ROUTE_˜N
!******************************************************************************
 ******************************************************************************
```

```
PROCEDURE  ROUTE_OUT       ROUTE PACKETS OUT

       THE PURPOSE OF THIS PROC IS TO ROUTE PACKETS FROM
   THE LOCAL-TO-LOCAL AND NETWORK-TO-LOCAL TABLES TO THE
   CORRECT OUTPUT CHANNEL.

INPUT -    DATA PACKETS ARE ROUTED VIA EVALUATION OF LCLCTB AND NTLCTB
   POINTERS AND INTERNAL PACKET ROUTING INFORMATION.

PROCESSING -   THE PROC CHECKS EACH INPUT BUFFER'S POINTERS FOR
   PACKET ARRIVAL.  IF A PACKET IS READY, THE DESTINATION IS
   DETERMINED VIA PROC DET_DEST, AND TRANSMITTED VIA PROC TRNMIT_PKT.
   THE TABLE THAT WAS SERVICED (PACKET REMOVED) HAS ITS POINTERS
   HOUSEKEPT BY SRVC_TAB_HSKP.
   WHEN THE FRAME IS TRANSMITTED TO THE HOSTS FROM THE NET-TO-LOCAL
   BUFFER TABLE, THE FIRST TWO BYTES OF HEADER INFORMATION IS STRIPPED
   OFF BEFORE TRANSMISSION.

OUTPUT -   A PACKET OF DATA IS TRANSMITTED TO A LOCAL CHANNEL.

INTERFACE -    THIS PROC IS CALLED IN AN ENDLESS LOOP BY PROC MAIN.

NOTES -    NONE.
*****************************************************************************
!
   INTERNAL

       ROUTE_OUT PROCEDURE
       ENTRY

           IF ((LCLCNE-LCLCNS) >= PACKET_SIZE)     ! IF LOCAL PACKET IS RDY !
           ORIF (LCLCNS > LCLCNE)
               THEN                                ! THEN !
               DESTINATION := DET_DEST('LL')       ! DETERMINE DESTINATION !
                                                   ! TRANSMIT DATA AND !
                                                   ! HSKP TABLES !

               IF DESTINATION
                   CASE '01'                       ! CH 1 DESTINATION !
                       THEN
```

```
            TRNMIT_PKT( LCLCTB[LCLCNS], ! TRANSMIT PKT !
                       U01DAT)
            SRVC_TAB_HSKP('LL')         ! HSKP LOC TO LOC TABLE !
        CASE '02'                       ! CH 2 DESTINATION !
            THEN
            TRNMIT_PKT( LCLCTB[LCLCNS], ! TRANSMIT PKT !
                       U02DAT)
            SRVC_TAB_HSKP('LL')         ! HSKP LOC TO LOC TABLE !
        CASE '03'                       ! CH 3 DESTINATION !
            THEN
            TRNMIT_PKT( LCLCTB[LCLCNS], ! TRANSMIT PKT !
                       U03DAT)
            SRVC_TAB_HSKP('LL')         ! HSKP LOC TO LOC TABLE !
        CASE '04'                       ! CH 4 DESTINATION !
            THEN
            TRNMIT_PKT( LCLCTB[LCLCNS], ! TRANSMIT PKT !
                       U04DAT)
            SRVC_TAB_HSKP('LL')         ! HSKP LOC TO LOC TABLE !
        ELSE                            ! ELSE !
            STATTB[L_RO_DEST_ERR] += 1  ! COUNT ERROR AND !
            SRVC_TAB_HSKP('LL')         ! HSKP LOC TO LOC TABLE !

    FI

FI

IF ((NTLCNE-NTLCNS) >= FRAME_SIZE)      ! IF NET FRAME IS RDY !
ORIF (NTLCNS > NTLCNE)
    THEN                                ! THEN !
    DESTINATION := DET_DEST('NL')       ! DETERMINE DESTINATION !
                                        ! TRANSMIT DATA AND !
                                        ! HSKP TABLES !
    IF DESTINATION
        CASE '01'                       ! CH 1 DESTINATION !
            THEN
            TRNMIT_PKT( NTLCTB[NTLCNS+2],   ! TRANSMIT PKT !
                       U01DAT)
            SRVC_TAB_HSKP('NL')         ! HSKP NET TO LOC TABLE !
        CASE '02'                       ! CH 2 DESTINATION !
```

```
                       THEN
                       TRNMIT_PKT( NTLCTB[NTLCNS+2],    ! TRANSMIT PKT !
                                  U02DAT)
                       SRVC_TAB_HSKP('NL')          ! HSKP NET TO LOC TABLE !
                   CASE '03'                        ! CH 3 DESTINATION !
                       THEN
                       TRNMIT_PKT( NTLCTB[NTLCNS+2],    ! TRANSMIT PKT !
                                  U03DAT)
                       SRVC_TAB_HSKP('NL')          ! HSKP NET TO LOC TABLE !
                   CASE '04'                        ! CH 4 DESTINATION !
                       THEN
                       TRNMIT_PKT( NTLCTB[NTLCNS+2],    ! TRANSMIT PKT !
                                  U04DAT)
                       SRVC_TAB_HSKP('NL')          ! HSKP NET TO LOC TABLE !
                   ELSE                             ! ELSE !
                       STATTB[L_RO_DEST_ERR] += 1   ! COUNT ERROR AND !
                       SRVC_TAB_HSKP('LL')          ! HSKP LOC TO LOC TABLE !

               FI

           FI

       END ROUTE_OUT

!*****************************************************************************
!



   GLOBAL

!*****************************************************************************
 *****************************************************************************
```

```
PROCEDURE  MAIN        PROC FOR MAIN LINE DRIVER OF LOCAL OS

       THE PURPOSE OF THIS PROC IS TO PROVIDE THE MAIN LINE
   OF PROCESSING FOR L.OS.

INPUT -    NONE.

PROCESSING -   THIS PROC SENDS A HEADER TO THE LOCAL MONITOR
   CONSOLE, INITIALIZES THE LOCAL BUFFERS AND STATUS BUFFER
   VIA INIT_L_TAB, INITIALIZES THE SHARED BUFFERS VIA INIT_U_SHTAB,
   USES PROC INVINT TO INITIALIZE I/O VECTOR INTERRUPTS,
   AND LOOPS ENDLESSLY ROUTING PACKETS IN AND OUT VIA PROCS
   ROUTE_IN AND ROUTE_OUT.

OUTPUT -   A START UP MSG IS SENT TO THE LOCAL MONITOR UPON
   START UP.

INTERFACE -    THIS PROC IS THE INITIAL ENTRY POINT FOR L.OS.
   IT OPERATES THROUGH SINGLE CALLS TO SNDSEQ, INIT_L_TAB, INIT_U_SHTAB,
   AND INVINT; AND REPETITIVE CALLS TO ROUTE_IN AND ROUTE_OUT.

NOTES -    NONE.
*****************************************************************************
!
       MAIN PROCEDURE
       ENTRY

                                         ! SEND START UP HEADER TO CONSOLE !
           SNDSEQ(CONCMD,
                  CONDAT,
                   STARTUP_HDR[0],
                  50)


           INIT_L_TAB                    ! INITIALIZE LOCAL BUFFER TABLES !


           INIT_U_SHTAB                  ! INITIALIZE UNID SHARED BUFFER TBLS !
```

```
            INVINT                       ! INITIALIZE I/O VECTOR INTERRUPTS !


            DO
                ROUTE_IN

                ROUTE_OUT
            OD


        END MAIN
END MAIN

!*******************************************************************************
!
!*******************************************************************************
!
```

```
!*****************************************************************************
*****************************************************************************
MODULE      L.TAB        LOCAL OS TABLES     26 OCT 82

        THE PURPOSE OF THIS MODULE IS TO PROVIDE L.OS WITH THE
    TABLES REQUIRED FOR DATA PROCESSING. THIS MODULE CONSISTS
    PRIMARILY OF TABLE DEFINITIONS WITH PROCESSING LIMITED TO
    THE INITIALIZATION OF THE DEFINED TABLES VIA INIT_L_TAB.

*****************************************************************************
!
L_TAB MODULE

    CONSTANT
        PACKET_SIZE := 30
        FRAME_SIZE := PACKET_SIZE + 2                ! TWO HEADER BYTES !
        PACKETS_IN_TABLE := 10
        P_TABLE_SIZE := PACKET_SIZE * PACKETS_IN_TABLE
        F_TABLE_SIZE := FRAME_SIZE * PACKETS_IN_TABLE

    GLOBAL
        LC01TB ARRAY [P_TABLE_SIZE BYTE]
        LC01NS INTEGER
        LC01NE INTEGER
        LC01SZ INTEGER

        LC02TB ARRAY [P_TABLE_SIZE BYTE]
        LC02NS INTEGER
        LC02NE INTEGER
        LC02SZ INTEGER

        LC03TB ARRAY [P_TABLE_SIZE BYTE]
        LC03NS INTEGER
        LC03NE INTEGER
        LC03SZ INTEGER

        LC04TB ARRAY [P_TABLE_SIZE BYTE]
        LC04NS INTEGER
        LC04NE INTEGER
        LC04SZ INTEGER

        LCLCTB ARRAY [P_TABLE_SIZE BYTE]
        LCLCNS INTEGER
        LCLCNE INTEGER
        LCLCSZ INTEGER

    GLOBAL

!*******************************************************************************
 *******************************************************************************
 PROCEDURE  INIT_L_TAB      PROC TO INITIALIZE LOCAL DATA BUFFERS

        THE PURPOSE OF THIS PROC IS TO INITIALIZE THE
    DATA BUFFER TABLES USED BY L.OS.

 INPUT -    NONE.

 PROCESSING -   THE PROCESS INITIALIZES THE FOUR LOCAL CHANNEL
    INPUT AND LOCAL-TO-LOCAL TABLES BY SETTING
    THE NEXT-BYTE-TO-BE-SERVICED AND THE NEXT-EMPTY-
    BYTE POINTERS TO ZERO, AND BY SETTING THE TABLE SIZE TO
    A MULTIPLE OF PACKET_SIZE.

 OUTPUT -   THE TABLE POINTERS AS NOTED UNDER PROCESSING  ARE
    MODIFIED.

 INTERFACE -    THIS PROC IS CALLED BY PROC MAIN.

 NOTES -    NONE.
 *******************************************************************************
!
        INIT_L_TAB PROCEDURE
        ENTRY

                                                ! INITIALIZE MEMORY BUFFER TABLE INFO
                                                  XXXXNS - NEXT BYTE TO BE SERVICED
                                                  XXXXNE - NEXT EMPTY BYTE
                                                  XXXXSZ - SIZE OF TABLE
```

```
                                       !
                                       ! INIT LOCAL CHANNEL INPUT TABLES !
        LC01NS := 0
        LC01NE := 0
        LC01SZ := P_TABLE_SIZE

        LC02NS := 0
        LC02NE := 0
        LC02SZ := P_TABLE_SIZE

        LC03NS := 0
        LC03NE := 0
        LC03SZ := P_TABLE_SIZE

        LC04NS := 0
        LC04NE := 0
        LC04SZ := P_TABLE_SIZE

                                       ! INIT 'LOCAL TO LOCAL' TABLE !
        LCLCNS := 0
        LCLCNE := 0
        LCLCSZ := F_TABLE_SIZE

    END INIT_L_TAB

!*****************************************************************************
!



END L_TAB

!*****************************************************************************
!
!*****************************************************************************
!
```

```
;
; NOTE:  TO PRINT OUT THIS MODULE SET TABSIZE TO '8'
;***************************************************************************
;***************************************************************************
;PROLOGUE -     MODULE L.VINT             DATE: 26 OCT 82
;
;               THIS MODULE IS AN ASSEMBLY PACKAGE BUILT TO SUPPORT
;       THE UNID LOCAL PROCESSOR OPERATING SYSTEM.  THE MODULE
;       CURRENTLY CONSISTS OF PROCEDURES INVINT WHICH INITIALIZES
;       THE VECTOR INTERRUPT PROCESS, INIURT WHICH SUPPORTS INVINT
;       WITH USART INITIALIZATION, TRNMIT WHICH SETS UP A TRANSMIT
;       INTERRUPT FOR PLZ SOFTWARE, FOUR RECEIVE I/O CONTROLLERS
;       FOR LOCAL CHANNEL INPUT, AND A GENERALIZED TRANSMIT I/O
;       CONTROLLER FOR LOCAL CHANNEL OUTPUT.
;
;
                GLOBAL INVINT TRNMIT


;***************************************************************************
*EJECT
;***************************************************************************
;***************************************************************************
;PROCEDURE      INVINT            INITIALIZE VECTOR INTERRUPT MODE
;
;               THE PURPOSE OF THIS PROC IS TO INITIALIZE THE
;       I/O VECTOR INTERRUPT PROCESS.  THIS MODULE IS ORG'D AT
;       TWO POINTS.  THE I/O VECTOR INTERRUPT TABLE (IOVCTB)
;       MUST BE LOCATED ON AN EVEN MEMORY BOUNDRY TO ENABLE
;       A CORRECT OFFSET POSITION DEVELOPMENT FOR I/O
;       CONTROLLER CALLS.  AN INTERRUPT GENERATES AN OFFSET
;       ADDRESS THROUGH THE 8212 PRIORITY INTERRUPT CONTROLLER
;       (PIC).  THIS OFFSET FROM THE START OF IOVCTB IDENTIFIES
;       THE CORRECT I/O HANDLER BY WHAT IS CONTAINED IN THAT
;       LOCATION.  THE CONTROLLER VALUES ARE SET IN IOVCTB
;       VIA DEFW COMMANDS IMMEDIATELY FOLLOWING THE ORG.
;               THE SECOND LOCATION TO BE ORG'D IS THE START
;       OF THE PROCEDURES THAT FOLLOW THE TABLE.  THESE PROCS
;       MUST BE LOCATED AT A POINT BEYOND THE END OF IOVCTB.
```

```
;
;INPUT -        THE ADDRESS FOR RETURNING TO THE CALLING
;       PROCEDURE IS LOCATED ON THE TOP OF THE STACK.
;
;PROCESSING -   THIS PROC BEGINS WITH A SAVE OF THE IX REG FOR
;       NORMALIZATION AT THE RETURN.  EACH LOCAL CARD USART IS
;       THEN INITIALIZED.  THIS INITIALIZATION IS ACCOMPLISHED
;       BY PASSING A DATA LIST ADDRESS THROUGH REG SET HL FOR
;       USE BY PROC INIURT.  PROC INIURT WILL THEN OUTPUT THE
;       COMMANDS ON THE DATA LIST TO THE ADDRESSES ON THE DATA
;       LIST.
;               INTERRUPT REGISTER (I) INITIALIZATION IS NEXT
;       WITH THE I REG BEING LOADED WITH THE ADDRESS OF THE I/O
;       VECTOR TABLE (IOVCTB).  THIS TABLE MUST BE LOCATED ON AN
;       EVEN 100 HEX MEMORY BOUNDRY (1600, 1700, ETC.).  WHEN AN
;       INTERRUPT IS IDENTIFIED BY THE PIC, THE I REG SUPPLIES
;       THE HIGH 8 BITS AND THE PIC SUPPLIES THE LOW 8 BITS OF THE
;       ADDRESS TO THE I/O CONTROLLER THAT WILL SERVICE THE INTERRUPT.
;       WITH THE IOVCTB TABLE ON AN EVEN MEMORY BOUNDRY, ONLY THE
;       8 HIGH BITS ARE REQUIRED TO BE LOADED AS THE LOWER BITS
;       ARE ALL ZEROS.
;               PIC INITIALIZATION FOLLOWS WITH INTERRUPT MODE
;       SET TO 2 AND INTERRUPTS ENABLED.  AT THIS POINT, THE UNID
;       IS CONFIGURED FOR INTERRUPT DRIVEN COMMUNICATIONS ON THE
;       LOCAL SIDE.
;               FINALLY, THE IX IS RESTORED, THE RETURN ADDRESS
;       RECOVERED, AND A RETURN EXECUTED.
;
;OUTPUT -       PROC INIURT IS PASSED DATA LIST ADDRESSES THROUGH
;       REG SET HL.  ADDITIONALLY, THE I REG IS SET WITH THE HIGH
;       8 BITS OF IOVCTB ADDRESS, THE PIC PRIORITY COMMAND IS
;       OUTPUT ON THE PIC ADDRESS PORT, AND INTERRUPTS ENABLED IN
;       MODE 2.
;
;INTERFACE -    THIS PROC IS CALLED FROM L.MAIN, THE MAIN LINE
;       DRIVER OF THE UNID LOCAL OPERATING SYSTEM.
;               THIS PROC ALSO CALLS PROC INIURT AND PASSES A
;       DATA LIST ADDRESS TO INIURT VAI THE HL REG SET.
;               THE INPUT TO THIS PROCEDURE IS OBTAINED VIA STACK
;       COMMUNICATION WITH THE CALLING PLZ MODULE.  THE INPUT
;       PARAMETERS ARE LOADED INTO THE STACK WITH A PUSH COMMAND
;       AND ARE RETREIVED WITH THE USE OF A BASE ADDRESS PLUS AN
;       OFFSET.  REFERENCE ZILOG PRODUCT DOCUMENT 03-3096-01,
;       PLZ USER GUIDE, SECTION 7 FOR DETAILS.
;
;NOTES -        NONE.
;
;******************************************************************************
                                ; **** NOTE ****
                ORG 0000H
                                ; THIS MODULE MUST BE ORG'D IN TWO AREAS:
                                ; THEREFORE WHEN PLINKING BE SURE THAT
                                ; THIS PORTION BEGINS AT AN EVEN BOUNDRY
                                ; LAYER SUCH AS:
                                ;       PLINK $=5000
                                ; IMPORTANT INFO ABOUT THIS ORG IS
                                ; IN THE PROC DOCUMENTATION ABOVE.
                                ; **** NOTE ****
IOVCTB                          ; IO VECTOR ADDRESS TABLE
URT04T  DEFW    URTTRN          ; USART 4 TRANSMIT CALLS URTT04
URT03T  DEFW    URTTRN          ; USART 3 TRANSMIT CALLS URTT03
URT02T  DEFW    URTTRN          ; USART 2 TRANSMIT CALLS URTT02
URT01T  DEFW    URTTRN          ; USART 1 TRANSMIT CALLS URTT01
URT04R  DEFW    URTR04          ; USART 4 RECEIVE CALLS URTR04
URT03R  DEFW    URTR03          ; USART 3 RECEIVE CALLS URTR03
URT02R  DEFW    URTR02          ; USART 2 RECEIVE CALLS URTR02
URT01R  DEFW    URTR01          ; USART 1 RECEIVE CALLS URTR01

                                ; **** NOTE ****
                ORG 0020H
                                ; THIS MODULE IS ORG'D IN TWO AREAS:
                                ; THE CODE, AND THE TABLE IOVCTB.
                                ; IMPORTANT INFO ABOUT THIS ORG IS
                                ; IN THE PROC DOCUMENTATION ABOVE.
                                ; **** NOTE ****

INVINT:                         ; PROC TO INITIALIZE VECTOR INTERRUPT MODE
                PUSH IX         ; STORE IX FOR RETURN
                LD HL,URT01L     ; LD LOC OF USART 1 DATA LIST
                CALL INIURT      ; INITIALIZE USART 1
                LD HL,URT02L     ; LD LOC OF USART 2 DATA LIST
                CALL INIURT      ; INITIALIZE USART 2
                LD HL,URT03L     ; LD LOC OF USART 3 DATA LIST
                CALL INIURT      ; INITIALIZE USART 3
                LD HL,URT04L     ; LD LOC OF USART 4 DATA LIST
                CALL INIURT      ; INITIALIZE USART 4

                LD HL,IOVCTB     ; LD ADD OF VECTOR ADDRESS TABLE
                LD A,H           ; LD HIGH 8 BITS OF ADD INTO I REG FOR BASE
                LD I,A

                LD A,(PICCMD)    ; LD PIC COMMAND
                OUT (PICADD),A   ; SEND COMMAND

                IM 2             ; SET INTERRUPT MODE TO VECTOR ADD MODE
                EI               ; ENABLE INTERRUPTS

                POP IX           ; RESTORE IX
                POP HL           ; RECOVER RETURN ADDRESS

                JP (HL)          ; RETURN

                ;EQUATES

PICADD          EQU     018H             ; PIC PORT ADDRESS
PICCMD          EQU     00001000B        ; ALLOW ANY INTERRUPT, NO PRIORITY

;*****************************************************************************
*EJECT
```

```
;****************************************************************************
;****************************************************************************
;PROCEDURE      INIURT          INITIALIZE LOCAL CARD USARTS
;
;               THE PURPOSE OF THIS PROC IS TO INITIALIZE THE 2651
;       USARTS ON THE UNID LOCAL BOARD.
;
;INPUT -        A DATA LIST ADDRESS IS PASSED TO INIURT VIA REG SET
;       HL.  THIS ADDRESS WILL IDENTIFY THE DATA TO BE USED BY INIURT.
;       THE DATA LISTS ARE DEFINED IMMEDIATELY FOLLOWING THIS PROC.
;
;PROCESSING -   A SEQUENCE OF COMMANDS TO THE USART AND ITS ASSOCIATED
;       COUNTER TIMER CIRCUIT (CTC) ARE REQUIRED FOR INITIALIZATION.
;       PROC INIURT SENDS A USART COMMAND WORD FOLLOWED BY TWO MODE
;       WORDS.  THE CTC IS THEN ADDRESSED AND A MODE WORD FOLLOWED BY
;       A BAUD RATE PRESCALER CONSTANT IS OUTPUT.  THESE ADDRESSES AND
;       THE DATA TO BE OUTPUT ARE PRESET IN A TABLE THAT IS IDENTIFIED
;       BY THE INPUT PARAMETER.
;
;OUTPUT -       THE USART RECEIVES ONE COMMAND WORD FOLLOWED BY TWO
;       MODE COMMANDS.  THE CTC ASSOCIATED WITH THE USART RECEIVES
;       A COMMAND WORD FOLLOWED BY A BAUD RATE CONSTANT.
;
;INTERFACE -    THIS PROC IS CALLED BY PROC INVINT.  A DATA LIST
;       ADDRESS IS PASSED VIA THE HL REG SET.  THIS ADDRESS
;       IDENTIFIES THE STARTING LOCATION OF THE DATA AND PORT
;       ADDRESSES TO BE USED.
;
;NOTES -        1.  INFORMATION CONCERNING THE 2651 USART AND THE CTC
;       CAN BE OBTAINED IN:
;               KANE, JERRY AND ADAM OSBORNE.
;               AN INTRODUCTION TO MICROCOMPUTERS
;               OSBORNE/MCGRAW-HILL, 1978.
;
;****************************************************************************
INIURT                          ; SUBROUTINE TO INITIALIZE A USART
                LD A,3          ; LD COMMAND PORT ADDRESS INTO C
                ADD A,(HL)
                LD C,A
                INC HL              ; SET HL AT COMMAND LOC IN LIST
                INC HL
                INC HL
                INC HL

                OUTI                ; SEND COMMAND

                DEC C               ; SET C TO MODE ADDRESS

                OUTI                ; SEND MODE COMMANDS
                OUTI

                LD C,(HL)           ; SET C TO CTC CH ADDRESS
                INC HL              ; SET HL TO CTC MODE COMMAND

                OUTI                ; SEND MODE COMMAND AND PRESCALER
                OUTI

                RET                 ; RETURN

                ;DEFINES

URT01L                              ; USART 1 INITIALIZATION DATA LIST
URT010  DEFB    00H                 ; DATA ADDRESS
URT011  DEFB    01H                 ; STATUS ADDRESS
URT012  DEFB    02H                 ; MODE ADDRESS
URT013  DEFB    03H                 ; COMMAND ADDRESS
URT014  DEFB    00000100B           ; USART COMMAND - ENAB TRAN, REC, DTR
URT015  DEFB    01001110B           ; USART MODE  1 - 1 STOP BIT, NO PARITY
                                    ;                 8 BIT, ASYNC, 16X
URT016  DEFB    00000000B           ; USART MODE  2 - CLK BY NOT TXC, RXC, 0 BD RT
URT017  DEFB    11H                 ; ASSOC CTC - CTC1, CH1 ADDRESS
URT018  DEFB    01000101B           ; CTC MODE COMMAND - CTR MODE, LD TIME CNST
URT019  DEFB    04H                 ; CTC TIME CNST PRESCALER FOR 19.2 KBAUD

URT02L                              ; USART 2 INITIALIZATION DATA LIST
URT020  DEFB    04H                 ; DATA ADDRESS
URT021  DEFB    05H                 ; STATUS ADDRESS
```

```
URT022  DEFB    06H             ; MODE ADDRESS
URT023  DEFB    07H             ; COMMAND ADDRESS
URT024  DEFB    00000100B       ; USART COMMAND - ENAB TRAN, REC, DTR
URT025  DEFB    01001110B       ; USART MODE  1 - 1 STOP BIT, NO PARITY
                                ;                 8 BIT, ASYNC, 16X
URT026  DEFB    00000000B       ; USART MODE  2 - CLK BY NOT TXC, RXC, 0 BD RT
URT027  DEFB    12H             ; ASSOC CTC - CTC1, CH2 ADDRESS
URT028  DEFB    01000101B       ; CTC MODE COMMAND - CTR MODE, LD TIME CNST
URT029  DEFB    04H             ; CTC TIME CNST PRESCALER FOR 19.2 KBAUD

URT03L                          ; USART 3 INITIALIZATION DATA LIST
URT030  DEFB    08H             ; DATA ADDRESS
URT031  DEFB    09H             ; STATUS ADDRESS
URT032  DEFB    0AH             ; MODE ADDRESS
URT033  DEFB    0BH             ; COMMAND ADDRESS
URT034  DEFB    00000100B       ; USART COMMAND - ENAB TRAN, REC, DTR
URT035  DEFB    01001110B       ; USART MODE  1 - 1 STOP BIT, NO PARITY
                                ;                 8 BIT, ASYNC, 16X
URT036  DEFB    00000000B       ; USART MODE  2 - CLK BY NOT TXC, RXC, 0 BD RT
URT037  DEFB    15H             ; ASSOC CTC - CTC2, CH1 ADDRESS
URT038  DEFB    01000101B       ; CTC MODE COMMAND - CTR MODE, LD TIME CNST
URT039  DEFB    08H             ; CTC TIME CNST PRESCALER FOR 9600 BAUD

URT04L                          ; USART 4 INITIALIZATION DATA LIST
URT040  DEFB    0CH             ; DATA ADDRESS
URT041  DEFB    0DH             ; STATUS ADDRESS
URT042  DEFB    0EH             ; MODE ADDRESS
URT043  DEFB    0FH             ; COMMAND ADDRESS
URT044  DEFB    00000100B       ; USART COMMAND - ENAB TRAN, REC, DTR
URT045  DEFB    01001110B       ; USART MODE  1 - 1 STOP BIT, NO PARITY
                                ;                 8 BIT, ASYNC, 16X
URT046  DEFB    00000000B       ; USART MODE  2 - CLK BY NOT TXC, RXC, 0 BD RT
URT047  DEFB    16H             ; ASSOC CTC - CTC2, CH2 ADDRESS
URT048  DEFB    01000101B       ; CTC MODE COMMAND - CTR MODE, LD TIME CNST
URT049  DEFB    08H             ; CTC TIME CNST PRESCALER FOR 9600 BAUD

;*****************************************************************************
*EJECT
```

```
;********************************************************************************
;********************************************************************************
;PROCEDURE      TRNMIT          TRANSMIT PROCEDURE
;
;                THE PURPOSE OF THIS PROCEDURE IS TO ENABLE A
;        TRANSMIT INTERRUPT FROM A PLZ MODULE.
;
;INPUT -         THIS PROC EXPECTS ONE INPUT PARAMETER: THE
;        DATA PORT ADDRESS OF THE USART SUPPORTING THE CHANNEL
;        OVER WHICH THE DATA IS TO BE OUTPUT.  THIS ADDRESS IS
;        TO BE IN GLOBAL PARAMETER TPRADD.
;
;PROCESSING -    THIS PROC BEGINS WITH A SAVE OF THE IX REG FOR
;        NORMALIZATION AT THE RETURN.
;                THE NEXT SECTION CONVERTS THE DATA PORT ADDRESS
;        TO THE COMMAND PORT ADDRESS AND LOADS IT INTO THE C REG.
;        THE TRANSMIT ENABLE BIT IS SETIN THE RETRIEVED COMMAND
;        WORD, AND SENT BACK OUT TO THE USART.  THIS ACTION
;        CAUSES THE ACTUAL INTERRUPT.
;                FINALLY, THE IX IS RESTORED, THE RETURN ADDRESS
;        RECOVERED, AND A RETURN EXECUTED.
;
;OUTPUT -        A COMMAND WORD IS OUTPUT TO THE APPROPRIATE
;        (AS IDENTIFIED BY THE INPUT PARAMETER) USART WHICH
;        CONTAINS A SET TRANSMIT ENABLE BIT.
;
;INTERFACE -     THE TRANSMIT INTERRUPT IS ENABLED THROUGH A
;        PRIORITY INTERRUPT CONTROLLER (PIC).  WHEN AN INTERRUPT
;        IS DESIRED, PROC TRNMIT IS CALLED WITH THE APPROPRIATE
;        DATA CHANNEL ADDRESS.  THE USART THEN HAS ITS TRANSMIT
;        ENABLE BIT SET.  THIS BIT-SETTING TRIGGERS THE PIC WHICH
;        IN TURN DEVELOPS AN ADDRESS OFFSET IN THE I/O VECTOR
;        TABLE (IOVCTB).  THIS OFFSET POSITION CONTAINS THE ADDRESS
;        OF THE CORRECT I/O INTERRUPT CONTROLLER WHICH ACTUALLY
;        OUTPUTS THE DATA.  THE TRANSMIT ENABLE BIT AND THE PIC
;        ARE RESET BY THE CONTROLLER WHEN INTERRUPTS CAN BE ALLOWED
;        AGAIN.
;
;                THE INPUT TO THIS PROC IS VIA A GLOBALLY DEFINED
;       PARAMETER.  THE REASON FOR THIS INTERFACE IS MULTIPLE USE
;       OF THIS VALUE BY THE TRANSMIT I/O CONTROLLER (URTTRN).
;       PARAMETER PASSING TO AN I/O CONTROLLER IS EASIER VIA A
;       GLOBAL DEFINITION SO PROC TRNMIT SIMPLY MAKES USE OF A
;       VALUE ALREADY REQUIRED.
;               ITEM TPRADD IS DEFINED IN MODULE L.MAIN.
;
;NOTES -        NONE.
;
;**************************************************************************
TRNMIT:                         ; PROC TO TRANSMIT A PACKET OF DATA
                EXTERNAL TPRADD ; IN MODULE L.MAIN

                PUSH IX         ; STORE IX FOR RETURN


                LD A,(TPRADD)   ; LD DATA PORT ADD
                LD C,3          ; CONVERT TO COMMAND PORT
                ADD A,C
                LD C,A

                IN A,(C)        ; ENABLE TRANSMIT
                SET 0,A
                OUT (C),A

                POP IX          ; RESTORE IX
                POP HL          ; RECOVER RETURN ADDRESS

                JP (HL)         ; RETURN

;**************************************************************************
*EJECT
```

```
;**************************************************************************
;**************************************************************************
;PROCEDURE      URTR01          I/O RECEIVE INTERRUPT CONTROLLER
;
;                   THE PURPOSE OF THIS PROCEDURE IS TO SERVICE LOCAL
;       CHANNEL 1 RECEIVE INTERRUPTS.
;
;INPUT -            THIS PROC USES THE THREE EXTERNALLY DEFINED VALUES
;       TO DETERMINE WHERE TO LOAD THE RECEIVED BYTE.  LC01NS IS
;       THE NEXT SERVICE POSITION; LC01NE IS THE NEXT EMPTY POSITION;
;       AND LC01SZ IS THE TOTAL NUMBER OF BYTE POSITIONS IN TABLE
;       LC01TB.
;
;PROCESSING -   THE PROC BEGINS WITH A SAVE OF THE INTERRUPTED
;       PROGRAM'S REGISTERS.  THE BYTE IS THEN INPUT WITH THE
;       PARITY BIT RESET (NOT USED).  THE BYTE IS LOADED INTO
;       THE BUFFER AND THE BUFFER LOCATION POSITIONS MODIFIED
;       FOR WRAPAROUND IF NECESSARY.  FINALLY, THE INTERRUPTED
;       PROGRAM'S REGISTERS ARE RESTORED, THE PIC RE-INITIALIZED,
;       INTERRUPTS ENABLED, AND A RETURN FROM INTERRUPT EXECUTED.
;
;OUTPUT -           THE BYTE RECEIVED IS LOADED INTO THE LC01TB AND
;       THE LC01NE POSITION IS UPDATED TO REFLECT THE BYTE
;       INSERTION.
;
;INTERFACE -    THIS PROC IS CALLED VIA INTERRUPT ACTION PROCESSED
;       BY THE PIC.  AS AN INTERRUPT IS IDENTIFIED, THE PIC
;       DEVELOPS AN ADDRESS OFFSET INTO THE I/O VECTOR TABLE
;       (IOVCTB).  THIS OFFSET POSITION CONTAINS THE ADDRESS OF
;       THE CORRECT I/O INTERRUPT CONTROLLER.
;
;NOTES -            NONE.
;
;**************************************************************************
URTR01:                             ; PROC TO HANDLE RECEIVE I/O INTERRUPTS

                EXTERNAL LC01TB LC01NE LC01SZ

                EX AF,AF'       ; SAVE REGS OF INTERRUPTED PROGRAM
                EXX
                PUSH IX
                PUSH IY

                IN A,(0)          ; INPUT THE BYTE AND RESET THE PARITY BIT
                RES 7,A

                LD DE,LC01TB      ; SET HL TO NEXT EMPTY BUFF LOCATION
                LD HL,(LC01NE)
                ADD HL,DE

                LD (HL),A         ; LD BYTE INTO EMPTY BUFF LOCATION

                LD HL,(LC01NE)    ; LD EMPTY LOC POINTER AND
                INC HL            ; INC EMPTY LOC POINTER
                LD (LC01NE),HL
                LD DE,(LC01SZ)    ; LD BUFF SIZE FOR CHECK
                SBC HL,DE         ; IF AT AND OF BUFF, RESET TO LOC ZERO
                JR NZ,UR1J10
                LD HL,0
                LD (LC01NE),HL
UR1J10
                LD A,(PICCMD)     ; LD PIC COMMAND
                OUT (PICADD),A    ; SEND COMMAND

                EX AF,AF'         ; RESTORE CALLING PROG'S REGS
                EXX
                POP IY
                POP IX

                EI                ; ENABLE INTERRUPTS

                RETI              ; RETURN

;*******************************************************************************
*EJECT
```

```
;*****************************************************************************
;*****************************************************************************
;PROCEDURE      URTR02          I/O RECEIVE INTERRUPT CONTROLLER
;
;               THE PURPOSE OF THIS PROCEDURE IS TO SERVICE LOCAL
;       CHANNEL 2 RECEIVE INTERRUPTS.
;
;INPUT -        THIS PROC USES THE THREE EXTERNALLY DEFINED VALUES
;       TO DETERMINE WHERE TO LOAD THE RECEIVED BYTE.  LC02NS IS
;       THE NEXT SERVICE POSITION; LC02NE IS THE NEXT EMPTY POSITION;
;       AND LC02SZ IS THE TOTAL NUMBER OF BYTE POSITIONS IN TABLE
;       LC02TB.
;
;PROCESSING -   THE PROC BEGINS WITH A SAVE OF THE INTERRUPTED
;       PROGRAM'S REGISTERS.  THE BYTE IS THEN INPUT WITH THE
;       PARITY BIT RESET (NOT USED).  THE BYTE IS LOADED INTO
;       THE BUFFER AND THE BUFFER LOCATION POSITIONS MODIFIED
;       FOR WRAPAROUND IF NECESSARY.  FINALLY, THE INTERRUPTED
;       PROGRAM'S REGISTERS ARE RESTORED, THE PIC RE-INITIALIZED,
;       INTERRUPTS ENABLED, AND A RETURN FROM INTERRUPT EXECUTED.
;
;OUTPUT -       THE BYTE RECEIVED IS LOADED INTO THE LC02TB AND
;       THE LC02NE POSITION IS UPDATED TO REFLECT THE BYTE
;       INSERTION.
;
;INTERFACE -    THIS PROC IS CALLED VIA INTERRUPT ACTION PROCESSED
;       BY THE PIC.  AS AN INTERRUPT IS IDENTIFIED, THE PIC
;       DEVELOPS AN ADDRESS OFFSET INTO THE I/O VECTOR TABLE
;       (IOVCTB).  THIS OFFSET POSITION CONTAINS THE ADDRESS OF
;       THE CORRECT I/O INTERRUPT CONTROLLER.
;
;NOTES -        NONE.
;
;*****************************************************************************
URTR02:                         ; PROC TO HANDLE RECEIVE I/O INTERRUPTS

                EXTERNAL LC02TB LC02NE LC02SZ

                EX AF,AF'       ; SAVE REGS OF INTERRUPTED PROGRAM
                EXX
                PUSH IX
                PUSH IY

                IN A,(4)          ; INPUT THE BYTE AND RESET THE PARITY BIT
                RES 7,A

                LD DE,LC02TB      ; SET HL TO NEXT EMPTY BUFF LOCATION
                LD HL,(LC02NE)
                ADD HL,DE

                LD (HL),A         ; LD BYTE INTO EMPTY BUFF LOCATION

                LD HL,(LC02NE)    ; LD EMPTY LOC POINTER AND
                INC HL            ; INC EMPTY LOC POINTER
                LD (LC02NE),HL
                LD DE,(LC02SZ)    ; LD BUFF SIZE FOR CHECK
                SBC HL,DE         ; IF AT AND OF BUFF, RESET TO LOC ZERO
                JR NZ,UR2J10
                LD HL,0
                LD (LC02NE),HL
UR2J10
                LD A,(PICCMD)     ; LD PIC COMMAND
                OUT (PICADD),A    ; SEND COMMAND

                EX AF,AF'         ; RESTORE CALLING PROG'S REGS
                EXX
                POP IY
                POP IX

                EI                ; ENABLE INTERRUPTS

                RETI              ; RETURN

;*****************************************************************************
*EJECT
```

```
;*****************************************************************************
;*****************************************************************************
;PROCEDURE      URTR03          I/O RECEIVE INTERRUPT CONTROLLER
;
;               THE PURPOSE OF THIS PROCEDURE IS TO SERVICE LOCAL
;       CHANNEL 3 RECEIVE INTERRUPTS.
;
;INPUT -        THIS PROC USES THE THREE EXTERNALLY DEFINED VALUES
;       TO DETERMINE WHERE TO LOAD THE RECEIVED BYTE.  LC03NS IS
;       THE NEXT SERVICE POSITION; LC03NE IS THE NEXT EMPTY POSITION;
;       AND LC03SZ IS THE TOTAL NUMBER OF BYTE POSITIONS IN TABLE
;       LC03TB.
;
;PROCESSING -   THE PROC BEGINS WITH A SAVE OF THE INTERRUPTED
;       PROGRAM'S REGISTERS.  THE BYTE IS THEN INPUT WITH THE
;       PARITY BIT RESET (NOT USED).  THE BYTE IS LOADED INTO
;       THE BUFFER AND THE BUFFER LOCATION POSITIONS MODIFIED
;       FOR WRAPAROUND IF NECESSARY.  FINALLY, THE INTERRUPTED
;       PROGRAM'S REGISTERS ARE RESTORED, THE PIC RE-INITIALIZED,
;       INTERRUPTS ENABLED, AND A RETURN FROM INTERRUPT EXECUTED.
;
;OUTPUT -       THE BYTE RECEIVED IS LOADED INTO THE LC03TB AND
;       THE LC03NE POSITION IS UPDATED TO REFLECT THE BYTE
;       INSERTION.
;
;INTERFACE -    THIS PROC IS CALLED VIA INTERRUPT ACTION PROCESSED
;       BY THE PIC.  AS AN INTERRUPT IS IDENTIFIED, THE PIC
;       DEVELOPS AN ADDRESS OFFSET INTO THE I/O VECTOR TABLE
;       (IOVCTB).  THIS OFFSET POSITION CONTAINS THE ADDRESS OF
;       THE CORRECT I/O INTERRUPT CONTROLLER.
;
;NOTES -        NONE.
;
;*****************************************************************************
URTR03:                         ; PROC TO HANDLE RECEIVE I/O INTERRUPTS

                EXTERNAL LC03TB LC03NE LC03SZ

                EX AF,AF'       ; SAVE REGS OF INTERRUPTED PROGRAM
                EXX
                PUSH IX
                PUSH IY

                IN A,(8)          ; INPUT THE BYTE AND RESET THE PARITY BIT
                RES 7,A

                LD DE,LC03TB      ; SET HL TO NEXT EMPTY BUFF LOCATION
                LD HL,(LC03NE)
                ADD HL,DE

                LD (HL),A         ; LD BYTE INTO EMPTY BUFF LOCATION

                LD HL,(LC03NE)    ; LD EMPTY LOC POINTER AND
                INC HL            ; INC EMPTY LOC POINTER
                LD (LC03NE),HL
                LD DE,(LC03SZ)    ; LD BUFF SIZE FOR CHECK
                SBC HL,DE         ; IF AT AND OF BUFF, RESET TO LOC ZERO
                JR NZ,UR3J10
                LD HL,0
                LD (LC03NE),HL
UR3J10
                LD A,(PICCMD)     ; LD PIC COMMAND
                OUT (PICADD),A    ; SEND COMMAND

                EX AF,AF'         ; RESTORE CALLING PROG'S REGS
                EXX
                POP IY
                POP IX

                EI                ; ENABLE INTERRUPTS

                RETI              ; RETURN

;****************************************************************************
*EJECT
```

```
;*******************************************************************************
;*******************************************************************************
;PROCEDURE      URTR04          I/O RECEIVE INTERRUPT CONTROLLER
;
;               THE PURPOSE OF THIS PROCEDURE IS TO SERVICE LOCAL
;       CHANNEL 4 RECEIVE INTERRUPTS.
;
;INPUT -        THIS PROC USES THE THREE EXTERNALLY DEFINED VALUES
;       TO DETERMINE WHERE TO LOAD THE RECEIVED BYTE.  LC04NS IS
;       THE NEXT SERVICE POSITION; LC04NE IS THE NEXT EMPTY POSITION;
;       AND LC04SZ IS THE TOTAL NUMBER OF BYTE POSITIONS IN TABLE
;       LC04TB.
;
;PROCESSING -   THE PROC BEGINS WITH A SAVE OF THE INTERRUPTED
;       PROGRAM'S REGISTERS.  THE BYTE IS THEN INPUT WITH THE
;       PARITY BIT RESET (NOT USED).  THE BYTE IS LOADED INTO
;       THE BUFFER AND THE BUFFER LOCATION POSITIONS MODIFIED
;       FOR WRAPAROUND IF NECESSARY.  FINALLY, THE INTERRUPTED
;       PROGRAM'S REGISTERS ARE RESTORED, THE PIC RE-INITIALIZED,
;       INTERRUPTS ENABLED, AND A RETURN FROM INTERRUPT EXECUTED.
;
;OUTPUT -       THE BYTE RECEIVED IS LOADED INTO THE LC04TB AND
;       THE LC04NE POSITION IS UPDATED TO REFLECT THE BYTE
;       INSERTION.
;
;INTERFACE -    THIS PROC IS CALLED VIA INTERRUPT ACTION PROCESSED
;       BY THE PIC.  AS AN INTERRUPT IS IDENTIFIED, THE PIC
;       DEVELOPS AN ADDRESS OFFSET INTO THE I/O VECTOR TABLE
;       (IOVCTB).  THIS OFFSET POSITION CONTAINS THE ADDRESS OF
;       THE CORRECT I/O INTERRUPT CONTROLLER.
;
;NOTES -        NONE.
;
;*******************************************************************************
URTR04:                         ; PROC TO HANDLE RECEIVE I/O INTERRUPTS

                EXTERNAL LC04TB LC04NE LC04SZ

                EX AF,AF'       ; SAVE REGS OF INTERRUPTED PROGRAM
                EXX
                PUSH IX
                PUSH IY

                IN A,(0CH)        ; INPUT THE BYTE AND RESET THE PARITY BIT
                RES 7,A

                LD DE,LC04TB      ; SET HL TO NEXT EMPTY BUFF LOCATION
                LD HL,(LC04NE)
                ADD HL,DE

                LD (HL),A         ; LD BYTE INTO EMPTY BUFF LOCATION

                LD HL,(LC04NE)    ; LD EMPTY LOC POINTER AND
                INC HL            ; INC EMPTY LOC POINTER
                LD (LC04NE),HL
                LD DE,(LC04SZ)    ; LD BUFF SIZE FOR CHECK
                SBC HL,DE         ; IF AT AND OF BUFF, RESET TO LOC ZERO
                JR NZ,UR4J10
                LD HL,0
                LD (LC04NE),HL
UR4J10
                LD A,(PICCMD)     ; LD PIC COMMAND
                OUT (PICADD),A    ; SEND COMMAND

                EX AF,AF'         ; RESTORE CALLING PROG'S REGS
                EXX
                POP IY
                POP IX

                EI                ; ENABLE INTERRUPTS

                RETI              ; RETURN

;******************************************************************************
*EJECT
```

9

```
;*******************************************************************************
;*******************************************************************************
;PROCEDURE       URTTRN          PROC TO HANDLE TRANSMIT I/O INTERRUPTS
;
;               THE PURPOSE OF THIS PROC IS TO SERVICE LOCAL
;       CHANNEL TRANSMIT INTERRUPTS.
;
;INPUT -        THIS PROC USES THE TWO EXTERNALLY DEFINED VALUES
;       TO DETERMINE WHERE TO SEND THE BYTE.  TDAADD IS
;       THE DATA SOURCE ADDRESS, AND TPRADD IS THE I/O PORT ADDRESS.
;
;PROCESSING -   THIS PROC BEGINS WITH A SAVE OF THE INTERRUPTED
;       PROGRAM'S REGISTERS.  THE DATA PORT ADDRESS IS CONVERTED
;       TO THE STATUS PORT ADDRESS AND THE PROC WAITS FOR A
;       TRANSMIT READY INDICATION.  THE STATUS PORT IS CONVERTED
;       BACK TO THE DATA PORT AND THE BYTE IS OUTPUT.  THE PROC
;       WAITS AGAIN FOR A READY CONDITION AND THEN RESETS THE
;       TRANSMIT ENABLE CONDITION TO GET OUT OF INTERRUPT.  FINALLY,
;       THE PRIORITY INTERRUPT CONTROLLER (PIC) IS RE-INITIALIZED,
;       AND A RETURN FROM INTERRUPT IS EXECUTED.
;
;OUTPUT -       A BYTE OF DATA IS OUTPUT ON THE DATA LINE, AND THE
;       PIC IS RESET TO ENABLE INTERRUPTS.
;
;INTERFACE -    THIS PROC IS CALLED VIA INTERRUPT ACTION PROCESSED
;       BY THE PIC.  AS AN INTERRUPT IS IDENTIFIED, THE PIC
;       DEVELOPS AN ADDRESS OFFSET INTO THE I/O VECTOR TABLE
;       (IOVCTB).  THIS OFFSET POSITION CONTAINS THE ADDRESS OF
;       THE CORRECT I/O INTERRUPT CONTROLLER.
;
;NOTES -        NONE.
;
;*******************************************************************************
URTTRN:                         ; PROC TO HANDLE TRANSMIT I/O INTERRUPTS

                EXTERNAL TDAADD TPRADD

                LD A,(TPRADD)   ; LD PORT
                LD C,+1          ; CNVRT TO STATUS PORT ADD
                ADD A,C
                LD C,A

UTRNL1          IN A,(C)         ; WAIT UNTIL READY TO TRANSMIT
                BIT 0,A
                JR Z,UTRNL1

                LD A,-1          ; CNVRT TO DATA PORT ADD
                ADD A,C
                LD C,A
                LD DE,(TDAADD)   ; LD DATA
                LD A,(DE)

                OUT (C),A        ; OUTPUT A BYTE

                LD A,+1          ; CNVRT TO STATUS PORT ADD
                ADD A,C
                LD C,A
UTRNL2          IN A,(C)         ; WAIT UNTIL READY TO TRANSMIT
                BIT 0,A
                JR Z,UTRNL2

                LD A,+2          ; CNVRT TO COMMAND PORT ADD
                ADD A,C
                LD C,A
                IN A,(C)         ; RESET TRANSMIT ENABLE
                RES 0,A
                OUT (C),A

                LD A,(PICCMD)    ; RESET PIC
                OUT (PICADD),A

                EI               ; ENABLE INTERRUPTS

                RETI             ; RETURN

;*****************************************************************************
```

Appendix C Section II

This section of Appendix C contains the software listings which comprise the network operating system.

```
!*****************************************************************************
!
!*****************************************************************************
!
!PROLOGUE -     MODULE N.MAIN       DATE 9 NOV 82

        THE PURPOSE OF THIS MODULE IS TO PROVIDE THE
    UNID NETWORK OPERATING SYSTEM (N.OS) WITH THE MAIN LINE
    OF PROCESSING.  THE N.OS IS REQUIRED TO INPUT/OUTPUT
    DATA PASSED TO IT FROM FOUR LOCAL CHANNELS OR RECEIVED
    FROM THE NETWORK CHANNEL.
        THIS MODULE CONSISTS OF THE MAIN LINE PROCEDURE 'MAIN',
    AND SUBORDINATE PROCEDURES INIT_N_TAB, DET_DEST, LD_TAB_HSKP,
    SRVC_TAB_HSKP, BUILD_S_FRAME, ROUTE_IN, TIME_DELAY, AND ROUTE_OUT..
!
!*****************************************************************************
!
!***************************************************************************
                      S P E C I A L   N O T E
   Because the Network Operating System must have certain
variables unique for each UNID station, this software
module must in turn be unique for each UNID.  The variable
MAX_UNIDS must be changed for every copy of the Network
Operating System located within each UNID whenever a new
UNID is added to the DELNET.  The variable UNID_NBR must
be unique for each UNID and changed whenever this software
is loaded into any UNID other than UNID  0.  These varialbes
have been made 'Global' so a memory map will produce their
location in the UNID memory and therefore make it easy to
change these values.  See the Data Dictionary and the software
comments below for further explanation.

**************************************************************************
!
!*****************************************************************************
!
MAIN MODULE

    TYPE
```

```
        PBYTE ^BYTE

    CONSTANT                                     ! GENERAL CONSTANTS !
        FALSE := 0                               ! USE AS FLAGS TO TEST!
        TRUE  := NOT FALSE                       ! BITS FOR BRANCHING  !
        CONCTC := %D5                            ! CONSOLE CTC PORT ADDRESS !
        CONCMD := %DF                            ! CONSOLE USART COMMAND PORT ADDRESS !
        CONDAT := %DE                            ! CONSOLE USART DATA PORT ADDRESS !
        NET_RI_DEST_ERR := 10                    ! NET ROUTE_IN DEST ERROR ENTRY !
        NET_RO_DEST_ERR := 11                    ! NET ROUTE_OUT DEST ERROR ENTRY !
        PACKET_SIZE := 30
        FRAME_SIZE := PACKET_SIZE + 2            ! ADD 2 BYTES FOR FRAME HEADER  !
        PACKETS_IN_TABLE := 10
        FRAMES_IN_TABLE := PACKETS_IN_TABLE ! ONE PACKET PER FRAME      !
        F_TABLE_SIZE := FRAME_SIZE * PACKETS_IN_TABLE
        STAT_NBR := 20                           ! NBR OF ENTRIES IN STATUS TABLE !


CONSTANT                                         ! CONSTANTS USED BY BUILD_S_FRAME!
        HDR00 := 00                              ! FIRST FRAME BYTE 00 - ADDRESS  !
        HDR01 := 01                              ! SECOND FRAME BYTE 01 - CONTROL !

    GLOBAL                          ! VARIABLES USED IN N.MAIN AND N.INSIO !
        CTCCNT BYTE := 0            ! PROGRESSIVE NUMBER OF LOOP COUNTS    !
        MAXNUM BYTE := %64          ! MAXIMUM NUMBER OF COUNTING LOOPS      !
        COMPLT BYTE := TRUE        ! FLAG FOR THE COMPLETION OF WAIT LOOP !
        UNID_NBR := 0                            ! THIS IS THE UNID NUMBER FOR
                                                   THIS PARTICULAR UNID.  IT IS
                                                   UNIQUE FOR EACH UNID AND MUST
                                                   BE DIFFERENT FOR THE SOFTWARE
                                                   IN EACH UNID OPERATING SYSTEM  !

        MAX_UNIDS       BYTE := %02              ! THE MAXIMUM NUMBER OF UNIDS THAT
                                                   ARE ATTACHED TO THE NETWORK.
                                                   THIS IS USED TO PROHIBIT FRAMES
                                                   WITH INCORRECT ADDRESS TO REACH
                                                   THE NETWORK.!
    !*** NOTE ***  MAX_UNIDS MUST BE REDEFINED IF THE NUMBER OF UNIDS
                   CONNECTED TO THE DELNET CHANGE    *** NOTE *** !
```

```
EXTERNAL                                  ! IN N.INSIO !
    INSIO PROCEDURE
    TRNMIT PROCEDURE (SRCADD PBYTE, NUMBYT WORD)
    STCTC3 PROCEDURE

EXTERNAL                                  ! IN U.LIB !
    MOVSEQ PROCEDURE (SRCADD PBYTE, DTDADD PBYTE, NUMBYT BYTE)
    SNDSEQ PROCEDURE (CMDPRT BYTE, DATPRT BYTE, BYTADD PBYTE, NUMBYT BYTE)
    RECSEQ PROCEDURE (CMDPRT BYTE, DATPRT BYTE, BYTADD PBYTE, NUMBYT BYTE)

EXTERNAL                                  ! IN N.TAB !
    INIT_N_TAB PROCEDURE

    NT01TB ARRAY [F_TABLE_SIZE BYTE]
    NT01NS INTEGER
    NT01NE INTEGER
    NT01SZ INTEGER

    NTNTTB ARRAY [F_TABLE_SIZE BYTE]
    NTNTNS INTEGER
    NTNTNE INTEGER
    NTNTSZ INTEGER

EXTERNAL                                  ! IN U.SHTAB !
    LCNTTB ARRAY [F_TABLE_SIZE BYTE]
    LCNTNS INTEGER
    LCNTNE INTEGER
    LCNTSZ INTEGER

    NTLCTB ARRAY [F_TABLE_SIZE BYTE]
    NTLCNS INTEGER
    NTLCNE INTEGER
    NTLCSZ INTEGER

    STATTB ARRAY [STAT_NBR BYTE]
```

```
INTERNAL                                 ! INTERNAL VARIABLES USED !
                                         ! THROUGHOUT MODULE !
    DESTINATION WORD                     ! DESTINATION OF PACKET !
    SEQ_BIT BYTE := FALSE                ! MODULO 2 SEQUENCE BIT OF    !
                                          ! ACTIVE DATA FRAME !
    INPUT_SEQ_BIT BYTE := FALSE          ! SEQUENCE BIT TO INPUT TO NEW FRAME !
    THIS_SEQ_BIT   BYTE :=  FALSE        ! SEQUENCE BIT UNDER EXAMINATION  !
    ACKNOWLEDGE    BYTE := FALSE            ! TRUE OR FALSE BIT THAT IDENTIFIES
                                          IF A GOOD ACKNOWLEDGEMENT (S-FRAME)
                                          HAS BEEN RECEIVED  !


INTERNAL                                 ! INTERNAL TABLES USED !
                                         ! THROUGHOUT MODULE !
    STARTUP_HDR ARRAY [* BYTE] :=        '%R%R%L%L'
                                         'UNID NETWORK OS%R%L'
                                         'VERSION 10 AUG 82%R%L'
                                         'EXECUTING%R%L}'

    S_FRAMETB ARRAY [FRAME_SIZE BYTE]    ! S_FRAME TABLE !

INTERNAL

!******************************************************************************
 ******************************************************************************
```

```
PROCEDURE  DET_DEST    DETERMINE DESTINATION OF PACKET

        THE PURPOSE OF THIS PROC IS TO DETERMINE THE
   DESTINATION OF A SPECIFIED PACKET.

INPUT -    THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING
   THE TABLE LOCATION OF THE PACKET TO BE EVALUATED.

PROCESSING -  THIS PROCEDURE LOOKS AT THE FIRST BYTE OF THE
   NETWORK INPUT BUFFER TABLE (NT01TB) AND MAKES THE DETERMINATION
   IF THE DATA FRAME IS DESTINED FOR THIS PARTICULAR UNID OR
   SOME OTHER UNID.  IF IT GOES TO THIS UNID THEN 'NL', IF TO
   SOME OTHER UNID THEN 'NN'.

OUTPUT -   THE PROC OUTPUTS A TWO CHARACTER ASCII VALUE
   INDICATING THE TABLE OR CHANNEL DESTINATION OF THE PACKET.

INTERFACE -    THIS PROC IS CALLED BY PROC ROUTE_IN FOR INPUT
   PACKETS.

NOTES -  FOR THIS PROCEDURE, S-FRAMES DESTINED FOR THIS UNID ARE 'NL'
*****************************************************************************
!
   INTERNAL

       DET_DEST PROCEDURE(TABLE WORD)
           RETURNS(DESTINATION WORD)
       ENTRY

           IF ((NT01TB [NT01NS] AND %F0)/ %10)  = UNID_NBR.
               THEN                             ! MASK OFF THE DESTINATION!
               DESTINATION := 'NL'              ! ADDRESS BITS AND SHIFT  !
           ELSE                                 ! THEM TO THE RIGHT.  THEN!
               DESTINATION := 'NN'              ! SEE IF THEY MATCH THE   !
           FI                                   ! UNID_NBR. MATCH = 'NL'  !
                                                ! NO MATCH = 'NN'         !

       END DET_DEST
```

```
PROCEDURE  LD_TAB_HSKP     LOAD TABLE HOUSEKEEP

        THE PURPOSE OF THIS PROC IS TO HOUSEKEEP A SPECIFIED
   BUFFER TABLE AFTER THE LOADING OF A PACKET.

INPUT -     THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING
   THE TABLE REQUIRING HOUSEKEEPING.

PROCESSING -   THE PROC DETERMINES THE TABLE TO BE PROCESSED,
   ADVANCES THE NEXT-EMPTY-BYTE POINTER BY A PACKET OR FRAME
   SIZE, AND ADJUSTS FOR BUFFER WRAP IF NECESSARY.

OUTPUT -    THE SPECIFIED TABLE HAS ITS NEXT-EMPTY-BYTE
   POINTER ADVANCED BY THE LENGTH OF A PACKET OR A FRAME.

INTERFACE -    THIS PROC IS CALLED BY PROC ROUTE_IN.

NOTES -     NONE.
*******************************************************************************
!
   INTERNAL

       LD_TAB_HSKP PROCEDURE(TABLE WORD)
       ENTRY

           IF TABLE
               CASE '01'                 ! IF CALLED TO HSKP NET CH 1 TAB !
                   THEN
                   NT01NE := NT01NE + FRAME_SIZE   ! ADV NEXT EMPTY PNTR !
                   IF NT01NE >= NT01SZ             ! IF TABLE WRAP !
                       THEN                        ! THEN SET PNTR TO 0 !
                       NT01NE := 0
                   FI

               CASE 'NN'                 ! IF CALLED TO HSKP NET TO NET TAB !
                   THEN
                   NTNTNE := NTNTNE + FRAME_SIZE   ! ADV NEXT EMPTY PNTR !
                   IF NTNTNE >= NTNTSZ             ! IF TABLE WRAP !
                       THEN                        ! THEN SET PNTR TO 0 !
```

```
            NTNTNE := 0
          FI

        CASE 'NL'               ! IF CALLED TO HSKP NET TO LOCAL TAB !
          THEN
          NTLCNE := NTLCNE + FRAME_SIZE     ! ADV NEXT EMPTY PNTR !
          IF NTLCNE >= NTLCSZ               ! IF TABLE WRAP !
            THEN                            ! THEN SET PNTR TO 0 !
            NTLCNE := 0
          FI

        CASE 'LN'               ! IF CALLED TO HSKP LOCAL TO NET TAB !
          THEN
          LCNTNE := LCNTNE + FRAME_SIZE     ! ADV NEXT EMPTY PNTR !
          IF LCNTNE >= LCNTSZ               ! IF TABLE WRAP !
            THEN                            ! THEN SET PNTR TO 0 !
            LCNTNE := 0
          FI
      FI

    END LD_TAB_HSKP

!*******************************************************************************
!*******************************************************************************
 *******************************************************************************
```

```
PROCEDURE  SRVC_TAB_HSKP   SERVICE TABLE HOUSEKEEP

      THE PURPOSE OF THIS PROC IS TO HOUSEKEEP A SPECIFIED
   BUFFER TABLE AFTER SERVICING (REMOVING A PACKET).

INPUT -    THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING
   THE TABLE REQUIRING HOUSEKEEPING.

PROCESSING -   THE PROC DETERMINES THE TABLE TO BE PROCESSED,
   ADVANCES THE NEXT-BYTE-TO-BE-SERVICED POINTER BY A PACKET OR
   FRAME SIZE, AND ADJUSTS FOR BUFFER WRAP IF NECESSARY.

OUTPUT -   THE SPECIFIED TABLE HAS ITS NEXT-BYTE-TO-BE-SERVICED
   POINTER ADVANCED BY THE LENGTH OF A PACKET OR FRAME.

INTERFACE -    THIS PROC IS CALLED BY PROC ROUTE_IN AND ROUTE_OUT.

NOTES -    NONE.
******************************************************************************
!
   INTERNAL

       SRVC_TAB_HSKP PROCEDURE(TABLE WORD)
       ENTRY

           IF TABLE
               CASE '01'                  ! IF CALLED TO HSKP NET INPUT TAB !
                   THEN
                   NT01NS := NT01NS + FRAME_SIZE    ! ADV NEXT SERVICE PNTR !
                   IF NT01NS >= NT01SZ              ! IF TABLE WRAP !
                       THEN                         ! THEN SET PNTR TO 0 !
                       NT01NS := 0
                   FI

               CASE 'NN'                  ! IF CALLED TO HSKP NET TO NET TAB !
                   THEN
                   NTNTNS := NTNTNS + FRAME_SIZE    ! ADV NEXT SERVICE PNTR !
                   IF NTNTNS >= NTNTSZ              ! IF TABLE WRAP !
                       THEN                         ! THEN SET PNTR TO 0 !
```

```
                    NTNTNS := 0
                FI

            CASE 'LN'                   ! IF CALLED TO HSKP LOCAL TO NET TAB !
                THEN
                LCNTNS := LCNTNS + FRAME_SIZE   ! ADV NEXT EMPTY PNTR !
                IF LCNTNS >= LCNTSZ             ! IF TABLE WRAP !
                    THEN                        ! THEN SET PNTR TO 0 !
                    LCNTNS := 0
                FI

        FI

    END SRVC_TAB_HSKP

!******************************************************************************
!
 !****************************************************************************
  ****************************************************************************
```

```
PROCEDURE BUILD_S_FRAME   BUILD A SUPERVISORY FRAME

        THE PURPOSE OF THIS PROC IS TO BUILD A SUPERVISORY FRAME
    AND PLACE IT INTO THE APPROPRIATE TABLE FOR TRANSMISSION TO
    THE NETWORK.

INPUT -   THIS PROCEDURE IS PASSED THE CONTENTS OF THE FIRST
    BYTE OF THE INCOMING I-FRAME (ADDRESS BYTE) AND THE
    SEQUENCE BIT OF THE INCOMING FRAME (MODULO 2).

PROCESSING -   THE PROCEDURE BEGINS WITH A SERIES OF LOGICAL
    OPERATIONS AND DIVISIONS WHICH SWAPS THE HIGH 4 ORDER
    BYTES OF THE ADDRESS WORD WITH THE LOW 4 ORDER BYTES.
    THIS SIMPLY INTERCHANGES THE DESTINATION AND SOURCE ADDRESSES.
    IT THEN SETS THE CONTROL WORD ACCORDING TO THE SEQUENCE BIT
    OF THE INCOMING I-FRAME.  THESE BYTES ARE THEN PLACED IN
    THE FIRST AND SECOND HEADER POSITIONS OF THE S-FRAME TABLE.
    THE S-FRAME IS THEN MOVED TO THE LOCAL-TO-NETWORK TABLE FOR
    TRANSMISSION.  TABLE POINTERS ARE THEN UPDATED.

OUTPUT -   SEE PROCESSING

INTERFACE -  THIS PROC IS CALLED BY ROUTE-IN WHENEVER A NEW I-FRAME
    ARRIVES.

NOTES -  THE X.25 AND LAP PROTOCOLS ALLOW FOR ADDITIONAL TYPES OF
    S-FRAMES WHICH MAY BE INCORPORATED DURING FURTHER DEVELOPMENT.
**********************************************************************
!
    INTERNAL

        BUILD_S_FRAME PROCEDURE (INPUT_SEQ_BIT BYTE)
        LOCAL
            ADDRESS_WORD BYTE
            CONTROL_WORD BYTE
        ENTRY

            ADDRESS_WORD := (((NT01TB[NT01NS] AND %0F)   ! MASK OFF RIGHT  !
                            * %10))  OR                   ! 4 BITS AND SHIFT!
```

```
                    (((NT01TB [NT01NS] AND %F0)  ! RIGHT.MASK OFF  !
                    / %10))                      ! LEFT 4 BIT AND  !
                                                 ! SHIFT LEFT.THEN !
                                                 ! COMBINE (SWAP). !
        IF INPUT_SEQ_BIT = TRUE                  ! IF SEQ BIT = 1  !
            THEN                                 ! PLACE 1010000 IN!
            CONTROL_WORD := %A0                  ! CONTROL WORD.   !
        ELSE                                     ! IF SEQ BIT = 0  !
            CONTROL_WORD := %80                  ! PLACE 1000000 IN!
        FI                                       ! CONTROL WORD.   !

        S_FRAMETB [HDR00] := ADDRESS_WORD        ! FIRST BYTE OF   !
        S_FRAMETB [HDR01] := CONTROL_WORD        ! S-FRAME BE THE  !
                                                 ! ADDRESS WORD AND!
                                                 ! SECOND BE THE   !
                                                 ! CONTROL WORD.   !
        MOVSEQ ( S_FRAMETB [HDR00],              ! THEN MOVE THE   !
                 LCNTTB [LCNTNE]   ,             ! NEWLY BUILD S-  !
                FRAME_SIZE         )             ! FRAME TO THE    !
                                                 ! LOC-NET TABLE.  !
        LD_TAB_HSKP ('LN')                       ! HOUSEKEEP TABLE !

    END BUILD_S_FRAME

!*******************************************************************************
!
!****************************************************************************
*****************************************************************************
```

```
PROCEDURE TIME_DELAY        PRODUCES A DELAY OF TIME

    THE PURPOSE OF THIS PROCEDURE IS TO PRODUCE A TIME DELAY TO BE
USED BETWEEN SUCCESSIVE TRANSMISSIONS OF I-FRAMES WHEN A VALID
ACKNOWLEDGEMENT HAS NOT BEEN RECEIVED.

INPUT -  NONE

PROCESSING -  THE PROCEDURE BEGINS BY CALLING THE ASSEMBLY ROUTINE
    STCTC3 WHICH INITIALIZES AND STARTS THE CTC CHANNEL  3.  EACH
    TIME THE CTC RUNS OUT, ITS INTERRUPT ROUTINE (TIMOUT) INCREMENTS
    THE GLOBAL VARIALBLE 'CTCCNT'.  WHENEVER THE CTCCNT VALUE IS GREATER
    OR EQUAL TO THE GLOBAL VARIABLE 'MAXNUM', THE GLOBAL VARIABLE 'COMPLT'
    IS CHANGED TO TRUE.

OUTPUT -  THE OUTPUT IS THE VARIABLE 'COMPLT'.  WHEN TRUE THE TIME DELAY
    SEQUENCE IS COMPLETE.

INTERFACE -  THIS PROCEDURE IS CALLED BY THE PROCEDURE ROUTE_OUT LOCATED
    WITHIN THIS SAME MODULE.  THIS PROCEDURE CALLS PROCEDURE STCTC3 WHICH
    IS LOCATED IN THE ASSEMBLY MODULE N.INSIO.

NOTES -  SEE PROCEDURE STCTC3 FOR AN EXAMPLE OF CALCULATING THE APPROPRIATE
    TIME DEALY.

****************************************************************************
!
    INTERNAL

        TIME_DELAY PROCEDURE
        ENTRY

            IF COMPLT = TRUE THEN
                STCTC3                          ! IF THE TIMER IS NOT  !
            FI                                  ! RUNNING THEN START IT!
            IF CTCCNT >= MAXNUM                 !          IF THE CTC  !
                THEN                            ! COUNTER IS GREATER OR  !
                COMPLT := TRUE                  ! EQUAL TO MAXNUM THEN LET!
                CTCCNT := 0                     ! COMPLT = TRUE AND ZERO !
        FI                                         ! OUT THE COUNTER.        !
      END TIME_DELAY
!****************************************************************************
!
!******************************************************************************
 ******************************************************************************
```

```
PROCEDURE  ROUTE_IN         ROUTE PACKETS IN

        THE PURPOSE OF THIS PROC IS TO ROUTE PACKETS FROM
    THE NETWORK INPUT BUFFER TO THEIR CORRECT OUTPUT BUFFER.
    IT INITIATES THE BUILDING OF AN S-FRAME FOR ACKNOWLEDGEMENT
    OF A GOOD I-FRAME.  IT ALSO INSURES THE PROPER SEQUENCING
    OF FRAMES BY THE USE OF MODULO 2 NUMBERING SCHEME.

INPUT -    DATA PACKETS OR FRAMES ARE ROUTED VIA EVALUATION
    OF NT01TB POINTERS AND FRAME HEADER ROUTING INFORMATION.

PROCESSING -  THE PROC CHECKS THE INPUT BUFFER'S POINTERS FOR
    FRAME ARRIVAL.  IT THEN DETERMINES IF THE FRAME IS DESTINED
    FOR ITS UNID OR ANOTHER UNID.  IF FOR ANOTHER UNID, THE PROC
    SIMPLY ROUTES IT BACK TO THE NETWORK.  IF FOR ITS UNID THEN
    IT DETERMINES THE TYPE OF FRAME.  IF IT IS AN I-FRAME THEN
    IT CALLS THE BUILD_S_FRAME PROC TO PROVIDE A POSITIVE
    ACKNOWLEDGEMENT AND THEN MOVES THE FRAME TO THE NET-TO-LOCAL
    BUFFER TABLE.  IF IT IS AN S-FRAME THEN IT TESTS TO SEE IF
    IT IS AN ACKNOWLEDGEMENT FOR ITS LAST TRANSMITTED I-FRAME.
    THIS PROC USES THE DET_DEST PROC TO ESTABLISH DESTINATION
    THE MOVSEQ PROC TO MOVE THE FRAME BETWEEN TABLES.
OUTPUT -   A FRAME OF DATA IS MOVED TO A DESTINATION BUFFER.

INTERFACE -    THIS PROC IS CALLED IN AN ENDLESS LOOP BY PROC MAIN.

NOTES -    NONE.
******************************************************************************
!
    INTERNAL

        ROUTE_IN PROCEDURE
        ENTRY

            IF ((NT01NE - NT01NS) >= FRAME_SIZE ! IF A FRAME FROM THE NET !
            ORIF (NT01NS > NT01NE))               ! IS READY IN THE INPUT    !
                THEN                            ! BUFFER TABLE (NT01TB)   !
                DESTINATION := DET_DEST ('01')  ! THEN DETERMINE ITS DEST-!
                                                ! TINATION.               !
```

```
IF DESTINATION                              ! IF THE DESTINATION IS    !
    CASE 'NN' THEN                          ! FOR ANOTHER UNID THEN    !
        MOVSEQ ( NT01TB [NT01NS],           ! SIMPLY MOVE THE FRAME    !
                 NTNTTB [NTNTNE],           ! TO THE NET-TO-NET TABLE !
                FRAME_SIZE        )         ! FOR TRANSMISSION TO NET !
        LD_TAB_HSKP ('NN')                  ! AND UPDATE TABLE.        !

    CASE 'NL' THEN                          ! IF DESTINATION IS FOR    !
        IF (NT01TB [NT01NS+1] AND %80)  = %80
            THEN                            ! THIS UNID, IS IT AN S-   !
            IF (NT01TB [NT01NS+1] AND %20) = %20
                THEN                        ! FRAME?  IF SO STRIP OFF !
                THIS_SEQ_BIT := TRUE! THE SEQ NUMBER AND SEE   !
                IF THIS_SEQ_BIT = SEQ_BIT ! IF IT IS A '1'.IF SO !
                    THEN                    ! DO WE HAVE A GOOD!
                    ACKNOWLEDGE := TRUE     ! ACKNOWLEDGEMENT? !
                FI
            ELSE                            ! IF SEQ NUMBER WAS A '0' !
                THIS_SEQ_BIT := FALSE  ! DOES IT MATCH THE SEQ!
                IF THIS_SEQ_BIT = SEQ_BIT  ! NUMBER OF THE LAST !
                    THEN                    ! TRANSMITTED FRAME?  IF SO !
                    ACKNOWLEDGE := TRUE   ! THERE IS A     !
                FI                          ! GOOD ACKNOWLEDGEMENT.  !
            FI
        ELSE                                ! IF IT IS AN I-FRAME     !
            IF (NT01TB [NT01NS+1] AND %20) = %20
                THEN                        ! DETERMINE THE INCOMING !
                INPUT_SEQ_BIT := TRUE     ! SEQ BIT.  IF IT IS !
                                          ! A '1' THEN BUILD AN!
                BUILD_S_FRAME (INPUT_SEQ_BIT) ! S-FRAME    !
                                              ! WITH A '1'  !
            ELSE                            ! AS THE SEQ BIT          !
                INPUT_SEQ_BIT := FALSE   ! IF THE SEQ BIT     !
                BUILD_S_FRAME (INPUT_SEQ_BIT)    ! IS A '0' !
                                                 ! THEN BUILD  !
            FI                              ! AN S-FRAME WITH A '0'. !
                                            ! AFTER THE S-FRAME IS BUILT !
        MOVSEQ ( NT01TB [NT01NS],           ! THEN MOVE THE NEW       !
                 NTLCTB [NTLCNE],           ! I-FRAME TO THE NET-TO- !
```

```
                            FRAME_SIZE      )   ! LOCAL BUFFER TABLE      !
                    LD_TAB_HSKP ('NL')          ! AND HOUSEKEEP TABLE.    !
                FI
            FI
            SRVC_TAB_HSKP ('01')                ! HOUSEKEEP TABLE '01'    !
        FI

        END ROUTE_IN

!****************************************************************************
!
!*******************************************************************************
 *******************************************************************************
```

PROCEDURE ROUTE_OUT ROUTE PACKETS OUT

THE PURPOSE OF THIS PROC IS TO ROUTE FRAMES FROM THE NET-TO-NET AND THE LOCAL-TO-NET TABLES TO THE NETWORK OUTPUT CHANNEL.
IT ALSO MAINTAINS FLOW CONTROL AND ERROR RECOVERY BY WAITING FOR THE ROUTE_IN PROCEDURE TO RECEIVE POSITIVE ACKNOWLEDGEMENT FOR I-FRAMES TRNSMITTED BY THIS PROCEDURE.

INPUT - DATA FRAMES ARE ROUTED VIA EVALUATION OF NTNTTB AND LCNTTB POINTERS AND FRAME HEADER ADDRESS INFORMATION.

PROCESSING - THE PROC CHECKS EACH INPUT BUFFER'S POINTERS FOR FRAMES TO BE TRANSMITTED ONTO THE NETWORK. BEFORE ANY TRANSMISSION OCCURS, THE VARIABLE MAX_UNIDS IS COMPARED AGAINST THE DESTINATION ADDRESS TO INSURE A FRAME CANNOT GET ONTO THE NETWORK WHICH HAS AN ADDRESS GREATER THAT THE AVAILABLE NUMBER OF UNIDS. NET-TO-NET FRAMES ARE SIMPLY TRANSFERED TO THE NETWORK. ALL FRAMES WHICH ORIGINATE AT ITS UNID ARE CHECKED TO IDENTIFY THE TYPE. S-FRAMES ARE SIMPLY TRANSMITTED. I-FRAMES ARE TRANSMITTED AND ENTER A TIME DELAY TO WAIT FOR A GOOD ACKNOWLEDGEMENT. IF A GOOD ACKNOWLEDGEMENT IS NOT RECEIVED BY THE END OF THE WAIT PERIOD, IT IS TRANSMITTED AGAIN UPON THE NEXT CYCLE THROUGH THIS PROCEDURE.

OUTPUT - A FRAME OF DATA IS TRANSMITTED TO THE NETWORK CHANNEL.

INTERFACE - THIS PROC IS CALLED IN AN ENDLESS LOOP BY THE MAIN PROCEDURE.

NOTES - THERE ARE TWO VERY IMPORTANT PARAMETERS RELATING TO THIS PROCEDURE THAT MUST BE CONSIDERED PRIOR TO THE MODIFICATION OF THIS PROC. THE FIRST IS VARIABLE MAX_UNIDS. ALL ACTIVE UNIDS ON THE NET SHOULD HAVE THEIR ADDRESSES BEGINNING WITH '00' AND INCREMENT PER UNID. IN THIS WAY THE ADDRESS CAN BE COMPARED TO MAX_UNIDS TO BE SURE NON EXISTENT ADDRESS WILL NOT APPEAR IN TRANSMITTED FRAMES. THE SECOND PARAMETER IS MAXNUM. MAXNUM IS THE MAXIMUM NUMBER OF TIMES THE TIMEOUT CYCLE LOOPS THROUGH THE RETRANSMISSION PERIOD FOR AN I-FRAME. THIS PERIOD ALLOWS TIME FOR A VALID ACKNOWLEDGEMENT TO BE RECEIVED AFTER THE I-FRAME

```
    IS TRANSMITTED.   THE BASIC PERIOD IS SET THROUGH N.INSIO
    FOR 27 MSEC.  IF MAXMUN = 10 THEN THE WAIT LOOP IS 270 MSEC.
*******************************************************************
!
    INTERNAL

        ROUTE_OUT PROCEDURE
        ENTRY
            IF ((NTNTNE - NTNTNS) >= FRAME_SIZE)   ! IF A FRAME IS READY    !
            ORIF (NTNTNS > NTNTNE)                ! IN THE NET-TO-NET TABLE  !
                THEN                              ! STRIP OFF THE ADDRESS    !
                IF ((NTNTTB[NTNTNS] AND %F0) / %10   ! BITS AND SHIFT    !
                < MAX_UNIDS)                          ! RIGHT. COMPARE WITH MAX  !
                    THEN                          ! NBR OF UNIDS. IF LESS, THEN !
                    TRNMIT ( NTNTTB [NTNTNS],     ! TRANSMIT FRAME AND THEN  !
                            FRAME_SIZE       )    ! HOUSEKEEP THE TABLE      !
                    SRVC_TAB_HSKP ('NN')
                ELSE                              ! IF ADDRESS IS OUT OF     !
                    STATTB [11] += 1              ! LIMITS THEN INCREMENT    !
                    SRVC_TAB_HSKP ('NN')          ! THE STATUS TABLE (ERROR) !
                FI
            FI

            IF ((LCNTNE - LCNTNS) >= FRAME_SIZE) ! IF A FRAME IN THE LOCAL- !
            ORIF (LCNTNS > LCNTNE)                ! TO-NET TABLE IS READY THEN !
                THEN                              ! IS IT AN S-FRAME?  IF SO !
                IF LCNTTB [LCNTNS+1] AND %80 = %80  ! IS ITS ADDRESS WITHIN !
                    THEN                          ! LIMITS?                  !
                    IF ((LCNTTB [LCNTNS] AND %F0) / %10
                    < MAX_UNIDS)
                        THEN
                        TRNMIT( LCNTTB [LCNTNS],! THEN TRANSMIT THE FRAME  !
                                FRAME_SIZE       )
                        SRVC_TAB_HSKP ('LN')      ! AND HOUSEKEEP THE TABLE. !
                    ELSE                          ! IF THE ADDRESS IS OUT OF !
                        STATTB [11] += 1          ! LIMITS INCREMENT STATUS  !
                        SRVC_TAB_HSKP ('LN')      ! TABLE AND HOUSEKEEP TABLE.!
                    FI
                ELSE                              ! IF IT IS AN I-FRAME THEN !
```

```
          IF SEQ_BIT = TRUE             ! IF SEQ BIT = '1' THEN    !
              THEN                      ! PLACE IT IN THE PROPER   !
              LCNTTB [LCNTNS+1] := %20   ! LOCATION IN CONTROL     !
          ELSE                          ! WORD.  IF SEQ BIT = '0'  !
              LCNTTB [LCNTNS+1] := %00  ! PLACE IT IN THE CONTROL !
          FI                            ! WORD                     !
          IF ((LCNTTB [LCNTNS] AND %F0) / %10
          < MAX_UNIDS)
              THEN
              IF COMPLT = FALSE THEN        ! IF THE TIMER IS NOT  !
                  TIME_DELAY                ! COMPLETE THEN        !
              FI                            ! CHECK ITS PROGRESS.  !
              IF (ACKNOWLEDGE = FALSE)      ! IF ADDRESS IS IN LIMITS !
              ANDIF (COMPLT = TRUE)         ! AND NO ACK HAS BEEN RE- !
                  THEN                      ! CEIVED AND IF THE TIMER !
                  TRNMIT ( LCNTTB [LCNTNS],! IS RUN OUT THEN X'MIT  !
                          FRAME_SIZE)        ! THE I-FRAME AGAIN AND  !
                  ACKNOWLEDGE := FALSE       ! FORCE ACK TO FALSE UNTI!
                  TIME_DELAY                ! ROUTE_IN MAKES IT TRUE  !
              FI                            ! THEN START THE TIMER   !
              IF ACKNOWLEDGE = TRUE THEN
                  SRVC_TAB_HSKP ('LN')      ! SERVICE TABLE ONCE     !
                  SEQ_BIT := NOT SEQ_BIT    ! AND UPDATE THE SEQ BIT !
              FI
          ELSE                          ! IF THE ADDRESS IS OUT   !
              STATTB [11] += 1          ! OF LIMITS INCREMENT THE !
              SRVC_TAB_HSKP ('LN')      ! STATUS TABLE & HOUSEKEEP!
          FI
      FI
  FI
END ROUTE_OUT

!**************************************************************************
!

  GLOBAL

!**************************************************************************
 **************************************************************************
```

```
PROCEDURE  MAIN        PROC FOR MAIN LINE DRIVER OF NETWORK OS

        THE PURPOSE OF THIS PROC IS TO PROVIDE THE MAIN LINE
   OF PROCESSING FOR N.OS.

INPUT -    NONE.

PROCESSING -   THIS PROC SENDS A HEADER TO THE NETWORK MONITOR
   CONSOLE, INITIALIZES THE NETWORK BUFFERS VIA INIT_N_TAB,
   USES INSIO TO INITIALIZE THE SIO INTERRUPTS,
   AND LOOPS ENDLESSLY ROUTING PACKETS IN AND OUT VIA PROCS
   ROUTE_IN AND ROUTE_OUT.

OUTPUT -   A START UP MSG IS SENT TO THE NETWORK MONITOR UPON
   START UP.

INTERFACE -    THIS PROC IS THE INITIAL ENTRY POINT FOR N.OS.
   IT OPERATES THROUGH SINGLE CALLS TO SNDSEQ, INIT_N_TAB, AND
   INSIO; AND REPETITIVE CALLS TO ROUTE_IN AND ROUTE_OUT.

NOTES -    NONE.
*****************************************************************************
!
       MAIN PROCEDURE
       ENTRY

                                        ! SEND START UP HEADER TO CONSOLE !
           SNDSEQ(CONCMD,
                  CONDAT,
                   STARTUP_HDR[0],
                  52)

           INIT_N_TAB                   ! INITIALIZE NET MEMORY BUFFER AREA !

           INSIO                        ! INITIALIZE SIO INTERRUPTS !

           DO                           ! BEGIN PROCESS LOOP !

               ROUTE_IN
```

```
                ROUTE_OUT
            OD                          ! END OF PROCESS LOOP !
        END MAIN
END MAIN

!*****************************************************************************
!
!*****************************************************************************
!
```

```
!***************************************************************************
 ***************************************************************************
 MODULE     N.TAB       NETWORK OS TABLES          9 NOV 82

        THE PURPOSE OF THIS MODULE IS TO PROVIDE N.OS WITH THE
    TABLES REQUIRED FOR DATA PROCESSING. THIS MODULE CONSISTS
    PRIMARILY OF TABLE DEFINITIONS WITH PROCESSING LIMITED TO
    THE INITIALIZATION OF THE DEFINED TABLES VIA INIT_N_TAB.
        THE TABLE SIZE IS A FUNCTION OF FRAME SIZE RATHER
    THAN PACKET SIZE. THIS IS DUE TO THESE TABLES OPERATING AT
    THE LINK-LEVEL PROTOCOL LAYER WHERE PACKETS ARE ALREADY
    FRAMED FOR NETWORK TRANSMISSION. CURRENTLY, MINIMAL X.25
    FRAMING IS BEING ACCOMPLISHED WITH A THREE BYTE HEADER AND
    A TWO BYTE TRAILER (FIRST AND LAST BYTE IS THE HARDWARE
    APPENDED SLDC FLAG).

 ***************************************************************************
!
N_TAB MODULE

    CONSTANT
        PACKET_SIZE := 30
        PACKETS_IN_TABLE := 10
        FRAME_SIZE := PACKET_SIZE + 3
        FRAMES_IN_TABLE := 10
        F_TABLE_SIZE := FRAME_SIZE * FRAMES_IN_TABLE

    GLOBAL
        NT01TB ARRAY [F_TABLE_SIZE BYTE]
        NT01NS INTEGER
        NT01NE INTEGER
        NT01SZ INTEGER

        NTNTTB ARRAY [F_TABLE_SIZE BYTE]
        NTNTNS INTEGER
        NTNTNE INTEGER
        NTNTSZ INTEGER

!***************************************************************************
```

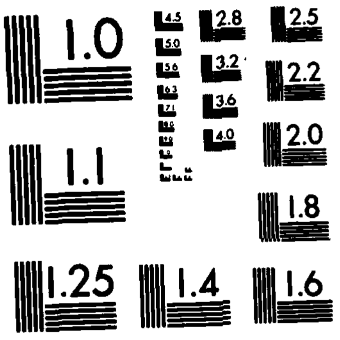
```
******************************************************************************
PROCEDURE  INIT_N_TAB      PROC TO INITIALIZE NET DATA BUFFERS

       THE PURPOSE OF THIS PROC IS TO INITIALIZE THE
   DATA BUFFER TABLES USED BY N.OS.

INPUT -    NONE.

PROCESSING -   THE PROCESS INITIALIZES THE NETWORK CHANNEL
   INPUT AND NETWORK-TO-NETWORK TABLES BY SETTING
   THE NEXT-BYTE-TO-BE-SERVICED AND THE NEXT-EMPTY-
   BYTE POINTERS TO ZERO, AND BY SETTING THE TABLE SIZE TO
   A MULTIPLE OF FRAME_SIZE.

OUTPUT -   THE TABLE POINTERS AND STATUS TABLE AS NOTED UNDER
   PROCESSING ARE MODIFIED.

INTERFACE -    THIS PROC IS CALLED BY PROC MAIN.

NOTES -    NONE.
******************************************************************************
!
       INIT_N_TAB PROCEDURE
       ENTRY
                                             ! INITIALIZE MEMORY BUFFER TABLE INFO
                                               XXXXNS - NEXT BYTE TO BE SERVICED
                                               XXXXNE - NEXT EMPTY BYTE
                                               XXXXSZ - SIZE OF TABLE
                                             !
                                             ! INIT NETWORK CHANNEL INPUT TABLE
                                             !
           NT01NS := 0
           NT01NE := 0
           NT01SZ := F_TABLE_SIZE

                                             ! INIT 'NETWORK TO NETWORK' TABLE !
           NTNTNS := 0
           NTNTNE := 0
```

```
            NTNTSZ := F_TABLE_SIZE

        END INIT_N_TAB
!*******************************************************************************
!



END N_TAB
!*******************************************************************************
!
!*******************************************************************************
!
```

```
;         TO PRINT OUT THIS PROCEDURE SE  TABSIZE=8
;*****************************************************************************
;*****************************************************************************
;PROLOGUE -     MODULE N.INSIO          DATE 9 NOV 82
;
;               THIS MODULE IS AN ASSEMBLY PACKAGE BUILT TO SUPPORT
;       THE UNID NETWORK PROCESSOR OPERATING SYSTEM.  THE MODULE
;       CURRENTLY CONSISTS OF PROCEDURES INSIO WHICH INITIALIZES
;       THE SERIAL INPUT/OUTPUT PROCESS, AND TRNMIT WHICH
;       TRANSMITS A FRAME OF DATA OUT THE NETWORK PORT.
;
;
                GLOBAL INSIO TRNMIT STCTC3

;*****************************************************************************
*EJECT
```

```
;*******************************************************************************
;*******************************************************************************
;PROCEDURE      INSIO           INITIALIZE SIO
;
;               THE PURPOSE OF THIS PROC IS TO INITIALIZE THE
;       SERIAL INPUT/OUTPUT PROCESS.  THIS MODULE IS ORG'D AT
;       TWO POINTS.  THE I/O VECTOR INTERRUPT TABLE (IOVCTB)
;       MUST BE LOCATED ON AN EVEN MEMORY BOUNDRY TO ENABLE
;       A CORRECT OFFSET POSITION DEVELOPMENT FOR I/O
;       CONTROLLER CALLS.  AN INTERRUPT GENERATES AN OFFSET
;       ADDRESS THROUGH THE SERIAL I/O COMPONENT (SIO).
;       THIS OFFSET FROM THE START OF IOVCTB IDENTIFIES
;       THE CORRECT I/O HANDLER BY WHAT IS CONTAINED IN THAT
;       LOCATION.  THE CONTROLLER VALUES ARE SET IN IOVCTB
;       VIA DEFW COMMANDS IMMEDIATELY FOLLOWING THE ORG.
;               THE SECOND LOCATION TO BE ORG'D IS THE START
;       OF THE PROCEDURES THAT FOLLOW THE TABLE.  THESE PROCS
;       MUST BE LOCATED AT A POINT BEYOND THE END OF IOVCTB.
;
;INPUT -        THE ADDRESS FOR RETURNING TO THE CALLING
;       PROCEDURE IS LOCATED ON THE TOP OF THE STACK.
;
;PROCESSING -   THIS PROC BEGINS WITH A SAVE OF THE IX REG FOR
;       NORMALIZATION AT THE RETURN.  THE SIO AND ASSOCIATED CTC ARE
;       THEN INITIALIZED.  THIS INITIALIZATION IS ACCOMPLISHED
;       BY USING TWO DATA LISTS CONTAINING THE INITIALIZATION
;       COMMANDS AND PORT ADDRESSES FOR THE SIO AND CTC.
;               INTERRUPT REGISTER (I) INITIALIZATION IS NEXT
;       WITH THE I REG BEING LOADED WITH THE ADDRESS OF THE I/O
;       VECTOR TABLE (IOVCTB).  THIS TABLE MUST BE LOCATED ON AN
;       EVEN 100 HEX MEMORY BOUNDRY (1600, 1700, ETC.).  WHEN AN
;       INTERRUPT IS IDENTIFIED BY THE SIO, THE I REG SUPPLIES
;       THE HIGH 8 BITS AND THE SIO SUPPLIES THE LOW 8 BITS OF THE
;       ADDRESS TO THE I/O CONTROLLER THAT WILL SERVICE THE INTERRUPT.
;       WITH THE IOVCTB TABLE ON AN EVEN MEMORY BOUNDRY, ONLY THE
;       8 HIGH BITS ARE REQUIRED TO BE LOADED AS THE LOWER BITS
;       ARE ALL ZEROS.
;               THE Z80 INTERRUPT MODE IS SET TO 2
;       AND INTERRUPTS ENABLED.  AT THIS POINT, THE UNID
;       IS CONFIGURED FOR INTERRUPT DRIVEN COMMUNICATIONS ON THE
;       NETWORK SIDE.
;               FINALLY, THE IX IS RESTORED, THE RETURN ADDRESS
;       RECOVERED, AND A RETURN EXECUTED.
;               TWO PROC STUBS, ONE FOR SPECIAL RECEIVE ACTION
;       AND ONE FOR CTC TIMEOUT, ARE INCLUDED FOR FUTURE
;       EXPANSION.
;
;OUTPUT -       THE SIO AND CTC ARE PASSED INITIALIZATION COMMANDS.
;       ADDITIONALLY, THE I REG IS SET WITH THE HIGH
;       8 BITS OF IOVCTB ADDRESS, AND INTERRUPTS ENABLED IN MODE 2.
;
;INTERFACE -    THIS PROC IS CALLED FROM N.MAIN, THE MAIN LINE
;       DRIVER OF THE UNID NETWORK OPERATING SYSTEM.
;               THE INPUT TO THIS PROCEDURE IS OBTAINED VIA STACK
;       COMMUNICATION WITH THE CALLING PLZ MODULE.  THE INPUT
;       PARAMETERS ARE LCADED INTO THE STACK WITH A PUSH COMMAND
;       AND ARE RETREIVED WITH THE USE OF A BASE ADDRESS PLUS AN
;       OFFSET.  REFERENCE ZILOG PRODUCT DOCUMENT 03-3096-01,
;       PLZ USER GUIDE, SECTION 7 FOR DETAILS.
;
;NOTES -        NONE.
;
;****************************************************************************
                ORG 0000H       ; **** NOTE ****
                                ; THIS MODULE IS ORG'D IN TWO AREAS:
                                ; THE CODE, AND THE TABLE IOVCTB.
                                ; IMPORTANT INFO ABOUT THIS ORG IS
                                ; IN THE PROC DOCUMENTATION ABOVE.
                                ; **** NOTE ****

IOVCTB                          ; IO VECTOR ADDRESS TABLE
SIO01B  DEFS    08              ; SIO B CHANNEL NOT USED
SIO01A  DEFS    02              ; SIO A CHANNEL
        DEFS    02
        DEFW    SIOREC          ; SIO A RECEIVE CALLS SIOREC
        DEFW    SPCREC          ; SIO A TRANSMIT CALLS SPCREC (STUB, NOT USED)
CTCI01  DEFS    06              ; CTC INTERRUPT
        DEFW    TIMOUT          ; CTC TIME OUT CALLS TIMOUT (STUB, NOT USED)
```

```
               ORG 0020H        ; **** NOTE ****
                                ; THIS MODULE IS ORG'D IN TWO AREAS:
                                ; THE CODE, AND THE TABLE IOVCTB.
                                ; IMPORTANT INFO ABOUT THIS ORG IS
                                ; IN THE PROC DOCUMENTATION ABOVE.
                                ; **** NOTE ****

               EXTERNAL  COMPLT, CTCCNT, MAXNUM; VARIABLES FROM N.MAIN
INSIO:                          ; PROC TI INITIALIZE SIO COMMUNICATIONS
               PUSH IX          ; STORE IX FOR RETURN

               LD HL,SIOLST     ; LD ADD OF SIO PARAMETER LIST
               LD C,(HL)        ; LD ADD OF SIO PORT A CMD/STATUS

               INC HL           ; INC TO NEXT BYTE IN LIST
               LD B,(HL)        ; LD NBR OF ENTRIES IN PORT A LIST

               INC HL           ; INC TO NEXT BYTE IN LIST
               OTIR             ; OUTPUT REMAINING BYTES TO SIO

               INC C            ; INC C TO SIO PORT B CMD/STATUS
               LD B,(HL)        ; LD NBR OF ENTRIES IN PORT B LIST

               INC HL           ; INC TO NEXT BYTE IN LIST
               OTIR             ; OUTPUT REMAINING BYTES TO SIO

                                ; INITIALIZE THE CTC
               LD HL,CTCLST     ; LD ADD OF CTC PARAMETER LIST
               LD C,(HL)        ; LD ADD OF CTC CHANNEL 0

               INC HL           ; INC TO NEXT BYTE IN LIST
               LD E,(HL)        ; LD LOW BYTE OF INTERRUPT ADD
               OUT (C),E        ; OUTPUT VECTOR ADD TO CTC CH 0

               INC HL           ; INC TWO BYTES IN LIST
               INC HL

               OUTI             ; SET OPERATING MODE
```


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
7.1

```
                OUTI            ; SET TIME CONSTANT

                INC C           ; INC C TO CTC CH 1 PORT ADD
                OUTI            ; SET OPERATING MODE
                OUTI            ; SET TIME CONSTANT

                INC C           ; INC C TO CTC CH 2 PORT ADD
                OUTI            ; SET OPERATING MODE
                OUTI            ; SET TIME CONSTANT

                INC C           ; INC C TO CTC CH 3 PORT ADD
                OUTI            ; SET OPERATING MODE
                OUTI            ; SET TIME CONSTANT

                LD HL,IOVCTB    ; LD ADD OF VECTOR ADDRESS TABLE
                LD A,H          ; LD HIGH 8 BITS OF ADD INTO I REG FOR BASE
                LD I,A

                IM 2            ; SET INTERRUPT MODE TO VECTOR ADD MODE
                EI              ; ENABLE INTERRUPTS

                POP IX          ; RESTORE IX
                POP HL          ; RECOVER RETURN ADDRESS

                JP (HL)         ; RETURN

SPCREC          NOP             ; SIO SPECIAL REC STUB. NOT CURRENTLY USED.
                RET             ; RETURN


TIMOUT:         PUSH IX         ; SAVE IX REG
                LD HL,CTCCNT    ; PLACE ADDRESS OF CTCCNT IN HL
                INC (HL)        ; INCREMENT CTC COUNTER BY '1'
                POP IX          ; RESTORE IX REG
                POP HL          ; GET RETURN ADDRESS IN HL
                JP (HL)         ; RETURN TO CALLING PROGRAM

A_DATA          EQU     00H             ; SIO PORT A DATA ADDRESS
B_DATA          EQU     01H             ; SIO PORT B DATA ADDRESS
```

```
A_STAT          EQU     02H             ; SIO PORT A STATUS/COMMAND ADDRESS
B_STAT          EQU     03H             ; SIO PORT B STATUS/COMMAND ADDRESS

CTC_0           EQU     04H             ; CTC CHANNEL 0 ADDRESS
CTC_1           EQU     05H             ; CTC CHANNEL 1 ADDRESS
CTC_2           EQU     06H             ; CTC CHANNEL 2 ADDRESS
CTC_3           EQU     07H             ; CTC CHANNEL 3 ADDRESS
TC3             EQU     00H             ; TIME CONSTANT = 256
FALSE           EQU     00H             ; FOR TRUE OR FALSE FLAG

                ;DEFINES

CTCLST  DEFB    CTC_0                   ; CTC CH 0 ADDRESS
        DEFW    CTCI01                  ; CTC INTERRUPT VECTOR ADDRESS
        DEFB    01000111B               ; CH 0 - RESET, INTERRUPTS OFF
        DEFB    00000001B
        DEFB    01000111B               ; CH 1 - CTR MODE, CNST NXT, RESET
        DEFB    10000000B               ;      - 9600 BAUD
        DEFB    01000111B               ; CH 2 - RESET, INTERRUPTS OFF
        DEFB    00000001B
        DEFB    01000111B               ; CH 3 - RESET, INTERRUPTS OFF
        DEFB    00000001B

                                        ; PORT A
SIOLST  DEFB    A_STAT                  ; SIO PORT A CMD/STAT ADD
        DEFB    12D                     ; NBR OF ENTRIES IN LIST
        DEFB    00000001B               ; SELECT WR1
        DEFB    00011000B               ; REG 1 - INT ALL RX CHAR
        DEFB    10010011B               ; SELECT WR3, RESET CRC GEN, EXT/STAT
        DEFB    11001001B               ; REG 3 - RX 8 BITS, CRC ENAB, ENAB
        DEFB    01010100B               ; SELECT WR4, RESET CRC CKR, EXT/STAT
        DEFB    00100000B               ; REG 4 - X1, SLDC MODE, SYNC ENAB
        DEFB    00000101B               ; SELECT WR5
        DEFB    11100010B               ; REG 5 - DTR, TX 8 BITS, SLDC, RTS
        DEFB    00000110B               ; SELECT WR6
        DEFB    00000000B               ; NONE
        DEFB    00000111B               ; SELECT WR7
        DEFB    01111110B               ; REG 7 - SLDC FLAG
                                        ; PORT B
        DEFB    04D                     ; NBR OF ENTRIES IN LIST
        DEFB    00000001B               ; SELECT WR1
        DEFB    00000100B               ; REG 1 - STATUS AFFECTS VECTOR
        DEFB    00000010B               ; SELECT WR2
        DEFB    00000000B               ; INTERRUPT VECTOR


;*******************************************************************************
*EJECT
```

```
;*******************************************************************************
;*******************************************************************************
;PROCEDURE      TRNMIT          TRANSMIT PROCEDURE
;
;               THE PURPOSE OF THIS PROCEDURE IS TO TRANSMIT A
;       FRAME OF DATA OUT THE NETWORK PORT.
;
;INPUT -        THIS PROC EXPECTS TWO INPUT PARAMETERS: THE
;       ADDRESS OF THE FIRST BYTE TO BE OUTPUT, AND THE NUMBER
;       OF BYTES TO BE SENT.
;
;PROCESSING -   THIS PROC BEGINS WITH A SAVE OF THE IX REG FOR
;       NORMALIZATION AT THE RETURN.  THE PUSH ESTABLISHES
;       THE BASE LOCATION FOR THE STACK.  ALL INPUT PARAMETERS
;       WILL BE OFFSET FROM THIS BASE LOCATION.
;               THE NEXT SECTION LOADS THE NUMBER OF BYTES TO BE
;       SENT AND THE ADDRESS OF THE FIRST BYTE FROM THE STACK.
;       TRANSMIT ENABLE CODE IS THEN OUTPUT TO THE SIO,
;       FOLLOWED BY A LOOP TO OUTPUT THE FRAME OF DATA.
;               FINALLY, THE IX IS RESTORED, THE RETURN ADDRESS
;       RECOVERED, THE STACK DEALLOCATED, AND A RETURN EXECUTED.
;
;OUTPUT -       A FRAME OF DATA IS OUTPUT ON THE NETWORK PORT.
;
;INTERFACE -    THE INPUT TO THIS PROCEDURE IS OBTAINED VIA STACK
;       COMMUNICATION WITH THE CALLING PLZ MODULE.  THE INPUT
;       PARAMETERS ARE LOADED INTO THE STACK WITH A PUSH COMMAND
;       AND ARE RETREIVED WITH THE USE OF A BASE ADDRESS PLUS AN
;       OFFSET.  REFERENCE ZILOG PRODUCT DOCUMENT 03-3096-01,
;       PLZ USER GUIDE, SECTION 7 FOR DETAILS.
;
;NOTES -        NONE.
;
;*******************************************************************************
TRNMIT:                         ; PROC TO TRANSMIT A FRAME OF DATA
                PUSH IX         ; STORE IX FOR RETURN

                LD IX,0         ; SET IX TO BASE OF STACK
                ADD IX,SP
                LD B,(IX+4)       ; LD   BYTES TO SEND

                LD L,(IX+6)       ; LD ADDRESS OF DATA TO SEND
                LD H,(IX+7)

                LD C,A_DATA       ; LD PORT A DATA PORT ADDRESS

                LD A,10000000B    ; RESET PORT A TRANSMIT CRC GENERATOR
                OUT (A_STAT),A

                LD A,00000101B    ; SET PORT A CMD REG TO 5
                OUT (A_STAT),A

                LD A,01101001B    ; ENABLE PORT A TRANSMITTER AND CRC GEN
                OUT (A_STAT),A

                OUTI              ; OUTPUT FIRST BYTE

                LD A,11000000B    ; RESET CRC/SYNC SENT/SENDING LATCH
                OUT (A_STAT),A    ; SET PORT A CMD REG TO 0

TRNJ10          IN A,(A_STAT)     ; READ PORT A STATUS REG
                BIT 2,A           ; AND WAIT UNTIL TX BUFFER IS EMPTY
                JP Z,TRNJ10

TRNJ20          OUTI              ; OUTPUT NEXT BYTE
                JP NZ,TRNJ10      ; IF NMBR BYTES > 0, SEND ANOTHER
                                  ; ELSE CONTINUE
                POP IX            ; RESTORE IX
                POP HL            ; RECOVER RETURN ADDRESS

                POP DE            ; DEALLOCATE STACK
                POP DE

                JP (HL)           ; RETURN
```

```
;******************************************************************************
;******************************************************************************
;PROCEDURE      SIOREC          SIO RECEIVE INTERRUPT CONTROLLER
;
;               THE PURPOSE OF THIS PROCEDURE IS TO SERVICE NETWORK
;       RECEIVE INTERRUPTS.
;
;INPUT -        THIS PROC USES THE THREE EXTERNALLY DEFINED VALUES
;       TO DETERMINE WHERE TO LOAD THE RECEIVED BYTE.  NT01NS IS
;       THE NEXT SERVICE POSITION; NT01NE IS THE NEXT EMPTY POSITION;
;       AND NT01SZ IS THE TOTAL NUMBER OF BYTE POSITIONS IN TABLE
;       NT01TB.
;
;PROCESSING -   THE PROC BEGINS WITH A SAVE OF THE INTERRUPTED
;       PROGRAM'S REGISTERS.  THE BYTE IS THEN INPUT FROM THE SIO.
;       THE BYTE IS LOADED THE NETWORK RECEIVE BUFFER AND THE BUFFER
;       DATA POINTERS MODIFIED FOR WRAPAROUND IF NECESSARY.  FINALLY
;       THE INTERRUPTED PROGRAM'S REGISTERS ARE RESTORED,
;       INTERRUPTS ENABLED, AND A RETURN FROM INTERRUPT EXECUTED.
;
;OUTPUT -       THE BYTE RECEIVED IS LOADED INTO THE NT01TB AND
;       THE NT01NE POSITION IS UPDATED TO REFLECT THE BYTE.
;       INSERTION.
;
;INTERFACE -    THIS PROC IS CALLED VIA INTERRUPT ACTION PROCESSED
;       BY THE SIO.  AS AN INTERRUPT IS IDENTIFIED, THE SIO
;       DEVELOPS AN ADDRESS OFFSET INTO THE I/O VECTOR TABLE
;       (IOVCTB).  THIS OFFSET POSITION CONTAINS THE ADDRESS OF
;       THE RECEIVE I/O INTERRUPT CONTROLLER.
;
;NOTES -        NONE.
;
;******************************************************************************
SIOREC:                         ; PROC TO HANDLE RECEIVE I/O INTERRUPTS

                EXTERNAL NT01TB NT01NE NT01SZ

                EX AF,AF'       ; SAVE REGS OF INTERRUPTED PROGRAM
                EXX
                IN A,(A_DATA)     ; INPUT THE BYTE

                LD DE,NT01TB      ; SET HL TO NEXT EMPTY BUFF LOCATION
                LD HL,(NT01NE)
                ADD HL,DE

                LD (HL),A         ; LD BYTE INTO EMPTY BUFF LOCATION

                LD HL,(NT01NE)    ; LD EMPTY LOC POINTER AND
                INC HL            ; INC EMPTY LOC POINTER
                LD (NT01NE),HL
                LD DE,(NT01SZ)    ; LD BUFF SIZE FOR CHECK
                SBC HL,DE         ; IF AT AND OF BUFF, RESET TO LOC ZERO
                JR NZ,SRIJ10
                LD HL,0
                LD (NT01NE),HL
SRIJ10
                EX AF,AF'         ; RESTORE CALLING PROG'S REGS
                EXX

                EI                ; ENABLE INTERRUPTS

                RETI              ; RETURN

;***************************************************************************
;
*EJECT
```

```
;*****************************************************************************
;PROCEDURE        STCTC3           START CTC CHANNEL 3
;                          THE PURPOSE OF THIS PROCEDURE IS TO START THE
;                 COUNTING OF CTC CHANNEL  3.  THIS PROCEDURE PROVIDES
;                 THE TIMING LOOP FOR TRANSMITTING AN I-FRAME OF DATA
;                 ONTO THE NETWORK DATA LINK.
;
;INPUT -  NONE
;
;PROCESSING -  THIS PROC BEGINS WITH THE SAVING OF THE IX AN A REGISTERS
;        FOR NORMALIZATION OF RETURN.  IT THEN SETS UP THE CTC CH NUMBER 3
;        FOR INTERRUPT MODE AND LOADS IN A TIME CONSTANT.  THIS TIME
;        CONSTANT IS 256 AT THE PRESENT TIME.  THE PROC CONTINUES WITH
;        THE INITIALIZATION OF THE GLOBAL VARIABLE 'COMPLT' TO FALSE.
;
;        AN EXAMPLE OF THE COMPLETE TIMING LOOP IS:
;                (Z-80 CLOCK)* TC * PRESCALAR * MAXNUM = TIME DELAY
;
;        ( 1 / (2.4X10^6) ) * 256 * 256 * 10 = 270 MILLISECONDS
;
;OUTPUT -  THE ONLY OUTPUT OF THIS PROCEDURE IS GLOBAL.  IT IS THE
;        VARIALBLE 'COMPLT' AND IS LOADED WITH THE VALUE TRUE UPON
;        THE COMPLETION OF THE TIMING CYCLE.
;
;INTERFACE -  THIS PROCEDURE IS CALLED BY THE PROCEDURE TIME_DELAY
;   IN THE MAIN MODULE.
;
;NOTES -  CAUTION SHOULD BE TAKEN IN INITIALIZING THE CONSTANTS FOR
;        THE CTC OPERATION.  BOTH THE TIME CONSTANT AND PRESCALAR
;        SHOULD BE SET SO THAT THE TIME DURATION OF THE CYCLE ALLOWS
;        SUFFICIENT TIME TO COMPLETE THE PROPER ROUTING OUT OF THE
;        DATA FRAMES.
;
;        SEE ZILOG DATA BOOK FOR THE Z8430 CTC OPERATION GUIDELINES.
;
;*****************************************************************************

STCTC3: PUSH IX                 ; SAVE THE IX REG
                PUSH AF         ; SAVE THE A REG
```

```
         LD A, 10100111B ; SET UP THE CTC  3 FOR INTERRUPT
         OUT (CTC_3),A   ; AND RESET.  THEN OUTPUT TO CTC
         LD A, TC3       ; THEN LOAD AND OUTPUT THE
         OUT (CTC_3),A   ; TIME CONSTANT

         LD A, FALSE     ; PLACE VALUE FALSE IN ACCUMULATOR
         LD HL, COMPLT   ; ADDRESS OF COMPLT IN HL
         LD (COMPLT), A  ; MAKE COMPLT = FALSE (INITIALIZATION)
         POP IX          ; RESTORE IX REGISTER
         POP AF          ; RESTORE AF REGISTER SET
         POP HL          ; OBTAIN THE RETURN ADDRESS
         JP (HL)         ; RETURN TO CALLING PROCEDURE
;  END PROCEDURE STCTC3
;*****************************************************************************
```

Appendix C Section III

This section of Appendix C contains the software listings which comprise the shared components of the DELNET operating system.

```
!****************************************************************************
 ****************************************************************************
 MODULE      U.SHTAB    UNID SHARED TABLE MODULE    DATE: 26 OCT 82

        THE PURPOSE OF THIS MODULE IS TO PROVIDE L.OS AND N.OS
    WITH THE TABLES SHARED FOR INTERFACE BETWEEN THE TWO PROCESSES.
    ADDITIONALLY, IT PROVIDES A STATUS TABLE FOR STATUS MONITORING.
    THIS PROC CONSISTS PRIMARILY OF TABLE DEFINITIONS WITH
    PROCESSING LIMITED TO THE INITIALIZATION OF THE DEFINED
    TABLES VIA INIT_U_SHTAB.


 NOTES -     1. STATTB ENTRIES ARE AS FOLLOWS:
                 00 - CUMULATIVE LOC ROUTE_IN DEST ERRORS
                 01 - CUMULATIVE LOC ROUTE_OUT DEST ERRORS
                 02 - NOT USED
                 03 - NOT USED
                 04 - NOT USED
                 05 - NOT USED
                 06 - NOT USED
                 07 - NOT USED
                 08 - NOT USED
                 09 - NOT USED
                 10 - CUMULATIVE NET ROUTE_IN DEST ERRORS
                 11 - CUMULATIVE NET ROUTE_OUT DEST ERRORS
                 12 - NOT USED
                 13 - NOT USED
                 14 - NOT USED
                 15 - NOT USED
                 16 - NOT USED
                 17 - NOT USED
                 18 - NOT USED
                 19 - NOT USED
 ****************************************************************************
!
U_SHTAB MODULE

    CONSTANT
        PACKET_SIZE := 30
        FRAME_SIZE  := PACKET_SIZE + 2          ! 2 BYTES OF HEADER !
        PACKETS_IN_TABLE := 10
        STAT_NBR := 20
        P_TABLE_SIZE := PACKET_SIZE * PACKETS_IN_TABLE
        F_TABLE_SIZE := FRAME_SIZE * PACKETS_IN_TABLE

    GLOBAL
        LCNTTB ARRAY [F_TABLE_SIZE BYTE]
        LCNTNS INTEGER
        LCNTNE INTEGER
        LCNTSZ INTEGER

        NTLCTB ARRAY [F_TABLE_SIZE BYTE]
        NTLCNS INTEGER
        NTLCNE INTEGER
        NTLCSZ INTEGER

        STATTB ARRAY [STAT_NBR BYTE]

!***************************************************************************
 ***************************************************************************
 PROCEDURE  INIT_U_SHTAB        PROC TO INITIALIZE DATA BUFFERS

        THE PURPOSE OF THIS PROC IS TO INITIALIZE THE
    DATA BUFFER TABLES SHARED BY L.OS AND N.OS.

 INPUT -    NONE.

 PROCESSING -   THE PROCESS INITIALIZES THE LOCAL-TO-NETWORK
    AND THE NETWORK-TO-LOCAL TABLES BY SETTING
    THE NEXT-BYTE-TO-BE-SERVICED AND THE NEXT-EMPTY-
    BYTE POINTERS TO ZERO, AND BY SETTING THE TABLE SIZE TO
    A MULTIPLE OF PACKET_SIZE.  THE PROC FINISHES BY CLEARING
    THE STATUS TABLE OF ALL STATUS INFORMATION.

 OUTPUT -   THE TABLE POINTERS AND STATUS TABLE AS NOTED UNDER
    PROCESSING ARE MODIFIED.

 INTERFACE -    THIS PROC IS CALLED BY PROC MAIN IN L.OS.
```

```
NOTES -    NONE.
**************************************************************************
!
       INIT_U_SHTAB PROCEDURE
       LOCAL IX WORD
       ENTRY
                                          ! INITIALIZE MEMORY BUFFER TABLE INFO
                                            XXXXNS - NEXT BYTE TO BE SERVICED
                                            XXXXNE - NEXT EMPTY BYTE
                                            XXXXSZ - SIZE OF TABLE
                                          !

                                          ! INIT 'LOCAL TO NETWORK' TABLE !
           LCNTNS := 0
           LCNTNE := 0
           LCNTSZ := F_TABLE_SIZE

                                          ! INIT 'NETWORK TO LOCAL' TABLE !
           NTLCNS := 0
           NTLCNE := 0
           NTLCSZ := F_TABLE_SIZE

           IX := 0                        ! INIT STATUS TABLE TO ZEROS !
           DO
               STATTB[IX] := 0
               IX += 1
               IF IX = STAT_NBR
                   THEN
                   EXIT
               FI
           OD

       END INIT_U_SHTAB

!**************************************************************************
!
```

```
;
; NOTE: TO PRINT MODULE SET TABSIZE TO '8'.
;*****************************************************************************
;*****************************************************************************
;PROLOGUE -     MODULE U.LIB                    DATE:26 OCT 82
;
;               THIS MODULE IS AN ASSEMBLY LIBRARY PACKAGE BUILT
;       TO SUPPORT PLZ/SYS SOFTWARE THAT WAS DEVELOPED TO OPERATE
;       OUTSIDE OF THE NORMAL ZILOG SUPPORTED ENVIRONMENT.  ITS
;       PURPOSE IS TO PROVIDE THE NECESSARY INTERFACE AND LIBRARY
;       FUNCTIONS NOT DIRECTLY AVAILABLE WITH PLZ/SYS.  THE MODULE
;       CURRENTLY CONSISTS OF PROCEDURES MOVSEQ, RECSEQ, AND SNDSEQ.
;
                GLOBAL MOVSEQ RECSEQ SNDSEQ

;*****************************************************************************
*EJECT
```

9

```
;*************************************************************************
;*************************************************************************
;PROCEDURE      MOVSEQ          MOVE A SEQUENCE OF BYTES IN MEMORY
;
;               THE PURPOSE OF THIS PROCEDURE IS TO MOVE A SEQUENCE
;       OF BYTES FROM ONE LOCATION IN MEMORY TO ANOTHER.
;
;INPUT -        THIS PROC EXPECTS THREE VALUES: THE SOURCE (FROM)
;       MEMORY ADDRESS, THE DESTINATION (TO) MEMORY ADDRESS, AND
;       THE NUMBER OF BYTES TO BE TRANSFERED.  ALL THREE PARAMETERS
;       ARE OF WORD LENGTH.
;
;PROCESSING -   THE PROCESS BEGINS WITH THE SAVE OF TH IX REG
;       FOR NORMALIZATION AT THE RETURN.  THE PUSH ESTABLISHES
;       THE BASE LOCATION FOR THE STACK.  ALL INPUT PARAMETERS
;       WILL BE OFFSET FROM THIS BASE LOCATION.
;               THE NEXT SECTION RETRIEVES THE THREE INPUT
;       PARAMETERS AND LOADS THEM INTO THE BC, DE, AND HL REG SETS.
;               THE LDIR COMMAND WILL MOVE THE CONTENTS OF THE
;       LOCATION INDICATED BY HL TO THE LOCATION INDICATED BY DE.
;       IT WILL INCREMENT HL AND DE, AND DECREMENT BC.  THIS
;       PROCESS WILL CONTINUE UNTIL BC IS EQUAL TO 0.
;               AFTER COMPLETION OF THE LDIR, IX IS RESTORED, THE
;       RETURN ADDRESS IS RECOVERED, THE INPUT PARAMETERS ON THE
;       STACK DEALLOCATED, AND A RETURN JUMP TO THE CALLING
;       MODULE IS EXECUTED.
;
;OUTPUT -       NONE.
;
;INTERFACE -    THE INPUT TO THIS PROCEDURE IS OBTAINED VIA STACK
;       COMMUNICATION WITH THE CALLING PLZ MODULE.  THE INPUT
;       PARAMETERS ARE LOADED INTO THE STACK WITH A PUSH COMMAND
;       AND ARE RETRIEVED WITH THE USE OF A BASE ADDRESS PLUS AN
;       OFFSET.  REFERENCE ZILOG PRODUCT DOCUMENT 03-3096-01,
;       PLZ USER GUIDE, SECTION 7 FOR DETAILS.
;
;
;NOTES -        1. THE NUMBER OF BYTES THAT CAN BE TRANSFERED
;       BY THIS PROC IS LIMITED BY THE REGISTER SET SIZE. ATTTEMPTING
;       TO TRANSFER A NUMBER OF BYTES GREATER THAN WHAT CAN BE STORED
;       IN 16 BITS WILL HAVE UNPREDICTABLE RESULTS.
;
;
;*******************************************************************************
MOVSEQ:                         ; PROC TO MOVE A SEQUENCE OF BYTES
                PUSH IX         ; STORE IX FOR RETURN

                LD IX,0         ; SET IX TO BASE OF STACK
                ADD IX,SP

                LD C,(IX+4)     ; LD   BYTES TO MOVE
                LD B,(IX+5)

                LD E,(IX+6)     ; LD DESTINATION ADDRESS
                LD D,(IX+7)

                LD L,(IX+8)     ; LD SOURCE ADDRESS
                LD H,(IX+9)

                LDIR            ; MOVE BYTES

                POP IX          ; RESTORE IX
                POP HL          ; RECOVER RETURN ADDRESS

                POP DE          ; DEALLOCATE STACK
                POP DE
                POP DE

                JP (HL)         ; RETURN

;*******************************************************************************
*EJECT
```

```
;***************************************************************************
;***************************************************************************
;PROCEDURE      RECSEQ          RECEIVE SEQUENCE OF BYTES
;
;                THE PURPOSE OF THIS PROCEDURE IS TO RECEIVE A
;       SEQUENCE OF BYTES FROM AN IDENTIFIED PORT.
;
;INPUT -         THIS PROC EXPECTS FOUR INPUT VALUES: THE
;       USART COMMAND PORT ADDRESS, THE USART DATA PORT ADDRESS,
;       THE STARTING MEMORY LOCATION OF WHERE TO SEND THE DATA,
;       AND THE NUMBER OF BYTES TO RECEIVE.  THE TWO USART PORT
;       ADDRESSES AND THE NUMBER OF BYTES ARE BYTE SIZE WHILE
;       THE MEMORY ADDRESS PARAMETER IS A FULL WORD IN LENGTH.
;
;PROCESSING -    THE PROCESS BEGINS WITH THE SAVE OF TH IX REG
;       FOR NORMALIZATION AT THE RETURN.  THE PUSH ESTABLISHES
;       THE BASE LOCATION FOR THE STACK.  ALL INPUT PARAMETERS
;       WILL BE OFFSET FROM THIS BASE LOCATION.
;                THE NEXT SECTION RETRIEVES THE FOUR INPUT
;       PARAMETERS AND LOADS THEM INTO THE B, C, D, AND HL REGS.
;                A SYNC LOOP IS NEXT FOR CHECKING READY STATUS. THIS
;       LOOP SYNCHRONIZES THE CODE WITH THE CHOSEN BAUD RATE. THE
;       ACTUAL INPUT CODE FOLLOWS WITH THE APPROPRIATE INCREMENT
;       AND DECREMENT OF POINTERS.  IF THERE ARE BYTES REMAINING
;       TO BE RECEIVED, A LOOP BACK TO THE SYNC LOOP CONTINUES THE
;       INPUT PROCESS.  OTHERWISE, THE IX REG IS RESTORED, THE
;       RETURN ADDRESS IS RECOVERED, THE INPUT PARAMETERS ON THE
;       STACK DEALLOCATED, AND A RETURN JUMP TO THE CALLING
;       MODULE IS EXECUTED.
;
;OUTPUT -        NONE.
;
;INTERFACE -     THE INPUT TO THIS PROCEDURE IS OBTAINED VIA STACK
;       COMMUNICATION WITH THE CALLING PLZ MODULE.  THE INPUT
;       PARAMETERS ARE LOADED INTO THE STACK WITH A PUSH COMMAND
;       AND ARE RETRIEVED WITH THE USE OF A BASE ADDRESS PLUS AN
;       OFFSET.  REFERENCE ZILOG PRODUCT DOCUMENT 03-3096-01,
;       PLZ USER GUIDE, SECTION 7 FOR DETAILS.
;
```

```
;
;NOTES -         1. THE NUMBER OF BYTES THAT CAN BE RECEIVED
;        BY THIS PROC IS LIMITED BY THE REGISTER SIZE. ATTTEMPTING
;        TO RECEIVE A NUMBER OF BYTES GREATER THAN WHAT CAN BE STORED
;        IN 8 BITS WILL HAVE UNPREDICTABLE RESULTS.
;
;*****************************************************************************
RECSEQ:                         ; PROC TO RECEIVE SEQUENCE OF BYTES
                PUSH IX         ; STORE IX FOR RETURN

                LD IX,0         ; SET IX TO BASE OF STACK
                ADD IX,SP

                LD B,(IX+4)     ; LD   BYTES TO RECEIVE

                LD L,(IX+6)     ; LD ADRESS FOR RECEIPT OF DATA
                LD H,(IX+7)

                LD D,(IX+8)     ; LD DATA PORT ADDRESS

                LD C,(IX+10)    ; LD COMAND PORT ADDRESS

RECLP1          IN A,(C)        ; WAIT UNTIL READY TO RECEIVE
                BIT 1,A
                JR Z,RECLP1

                LD C,D          ; LD DATA PORT ADDRESS
                LD A,(HL)       ; LD DATA ADDRESS
                IN A,(C)        ; RECEIVE BYTE

                LD (HL),A       ; STORE DATA
                INC HL          ; ADVANCE DATA ADDRESS POINTER
                LD C,(IX+10)    ; LD COMMAND PORT ADDRESS
                DJNZ RECLP1     ; IF NMBR BYTES LEFT > 0, RECEIVE ANOTHER
                                ; ELSE CONTINUE
                POP IX          ; RESTORE IX
                POP HL          ; RECOVER RETURN ADDRESS

                POP DE          ; DEALLOCATE STACK
                POP DE
                POP DE
                POP DE

                JP (HL)          ; RETURN

;
;**********************************************************************************
*EJECT
```

```
;******************************************************************************
;******************************************************************************
;PROCEDURE      SNDSEQ          SEND SEQUENCE OF BYTES
;
;               THE PURPOSE OF THIS PROCEDURE IS TO SEND A
;       SEQUENCE OF BYTES TO AN IDENTIFIED PORT.
;
;INPUT -        THIS PROC EXPECTS FOUR INPUT VALUES: THE
;       USART COMMAND PORT ADDRESS, THE USART DATA PORT ADDRESS,
;       THE STARTING MEMORY LOCATION OF THE DATA TO BE SENT,
;       AND THE NUMBER OF BYTES TO RECEIVE.  THE TWO USART PORT
;       ADDRESSES AND THE NUMBER OF BYTES ARE BYTE SIZE WHILE
;       THE MEMORY ADDRESS PARAMETER IS A FULL WORD IN LENGTH.
;
;PROCESSING -   THE PROCESS BEGINS WITH THE SAVE OF TH IX REG
;       FOR NORMALIZATION AT THE RETURN.  THE PUSH ESTABLISHES
;       THE BASE LOCATION FOR THE STACK.  ALL INPUT PARAMETERS
;       WILL BE OFFSET FROM THIS BASE LOCATION.
;               THE NEXT SECTION RETRIEVES THE FOUR INPUT
;       PARAMETERS AND LOADS THEM INTO THE B, C, D, AND HL REGS.
;               A SYNC LOOP IS NEXT FOR CHECKING READY STATUS. THIS
;       LOOP SYNCHRONIZES THE CODE WITH THE CHOSEN BAUD RATE. THE
;       ACTUAL OUTPUT CODE FOLLOWS WITH THE APPROPRIATE INCREMENT
;       AND DECREMENT OF POINTERS.  IF THERE ARE BYTES REMAINING
;       TO BE SENT, A LOOP BACK TO THE SYNC LOOP CONTINUES THE
;       OUTPUT PROCESS.  OTHERWISE, THE IX REG IS RESTORED, THE
;       RETURN ADDRESS IS RECOVERED, THE INPUT PARAMETERS ON THE
;       STACK DEALLOCATED, AND A RETURN JUMP TO THE CALLING
;       MODULE IS EXECUTED.
;
;OUTPUT -       NONE.
;
;INTERFACE -    THE INPUT TO THIS PROCEDURE IS OBTAINED VIA STACK
;       COMMUNICATION WITH THE CALLING PLZ MODULE.  THE INPUT
;       PARAMETERS ARE LOADED INTO THE STACK WITH A PUSH COMMAND
;       AND ARE RETRIEVED WITH THE USE OF A BASE ADDRESS PLUS AN
;       OFFSET.  REFERENCE ZILOG PRODUCT DOCUMENT 03-3096-01,
;       PLZ USER GUIDE, SECTION 7 FOR DETAILS.
;
```

```
;
;NOTES -         1. THE NUMBER OF BYTES THAT CAN BE SENT
;       BY THIS PROC IS LIMITED BY THE REGISTER SIZE. ATTTEMPTING
;       TO SEND A NUMBER OF BYTES GREATER THAN WHAT CAN BE STORED
;       IN 8 BITS WILL HAVE UNPREDICTABLE RESULTS.
;
;******************************************************************************
SNDSEQ:                         ; PROC TO SEND SEQUENCE OF BYTES
                PUSH IX         ; STORE IX FOR RETURN

                LD IX,0         ; SET IX TO BASE OF STACK
                ADD IX,SP

                LD B,(IX+4)     ; LD   BYTES TO SEND

                LD L,(IX+6)     ; LD ADDRESS OF DATA TO SEND
                LD H,(IX+7)

                LD D,(IX+8)     ; LD DATA PORT ADDRESS

                LD C,(IX+10)    ; LD COMAND PORT ADDRESS

SNDLP1          IN A,(C)        ; WAIT UNTIL READY TO TRANSMIT
                BIT 0,A
                JR Z,SNDLP1

                LD C,D          ; LD DATA PORT ADDRESS
                LD A,(HL)       ; LD DATA ADDRESS
                OUT (C),A       ; SEND BYTE

                INC HL          ; ADVANCE DATA ADDRESS POINTER
                LD C,(IX+10)    ; LD COMMAND PORT ADDRESS
                DJNZ SNDLP1     ; IF NMBR BYTES LEFT > 0, SEND ANOTHER
                                ; ELSE CONTINUE
                POP IX          ; RESTORE IX
                POP HL          ; RECOVER RETURN ADDRESS

                POP DE          ; DEALLOCATE STACK
                POP DE
```

```
                    POP DE
                    POP DE

                    JP (HL)          ; RETURN

;*******************************************************************************
;*******************************************************************************
```

Vita

Captain Craig H. Hazelton was born on 10 January, 1949 in New York City. In 1952 he moved to Atlanta, Georgia and later to Bradenton, Florida where he graduated from high school in 1967. In 1968 he enlisted in the United States Air Force and served as a missile electronics technician until 1978 when he entered the University of Central Florida under the Air Force's Airman Education and Commissioning Program. After graduating with a Bachelor of Science degree in Electrical Engineering, he received his commission and was assigned to Eglin AFB, Florida where he served as lead test engineer for the F-15 electronic warfare system. He entered the Air Force Institute of Technology in June 1981.

Permanent Address: 3316 32nd Street West
Bradenton, Florida 33505

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/GE/EE/82D-37	2. GOVT ACCESSION NO. AD-A124874	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) CONTINUED DEVELOPMENT AND IMPLEMENTATION OF THE PROTOCOLS FOR THE DIGITAL ENGINEERING LABORATORY NETWORK		5. TYPE OF REPORT & PERIOD COVERED MS Thesis
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Craig H. Hazelton, Capt, USAF		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Air Force Institute of Technology(AFIT/EN) Wright-Patterson AFB, OH 45433		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Institute of Technology(AFIT/EN) Wright-Patterson AFB, OH 45433		12. REPORT DATE December 1982
		13. NUMBER OF PAGES 213
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)

Approval for public release;,distribution unlimited

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

18. SUPPLEMENTARY NOTES

Approved for public release: IAW AFR 190-17.
[illegible]
LYNN E. WOLAVER
Dean for Research and Professional Development
Air Force Institute of Technology (AIC)
Wright-Patterson AFB OH 45433

4 JAN 198

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

Local Computer Networks
Local Area Networks
Computer Network Protocols
Computer Interfaces

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

Development of the Air Force Institute of Technology Digital Engineering Laboratory's local computer network (DELNET) operating system was continued. The DELNET operating system was developed under the standards of the International Standards Organization's 7 layer protocol model. This report contains the design and implementation of the protocol layers 1,2,and 3. This report also presents the tests and validations conducted to verify proper software development. Conclusions and recommendations are also presented.

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

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

END