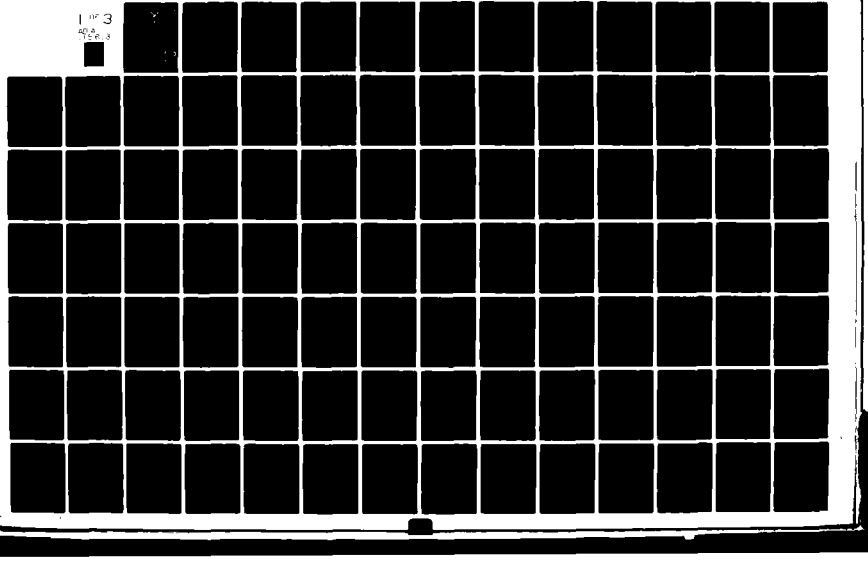
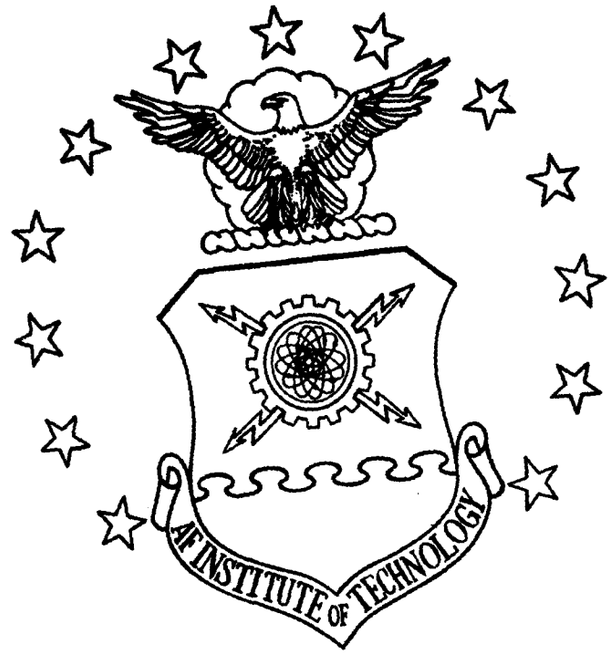
AD-A115 613 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/6 9/2
DEVELOPMENT OF THE DIGITAL ENGINEERING LABORATORY COMPUTER NETW--ETC(U)
DEC 81 J W GEIST

UNCLASSIFIED AFIT/GCS/EE/81D-8 NL





DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY (ATC)

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/EE/81D-8

DEVELOPMENT OF THE DIGITAL ENGINEERING LABORATORY COMPUTER NETWORK: HOST-TO-NODE/HOST-TO-HOST PROTOCOLS

THESIS

AFIT/GCS/EE/81D-8 John W. Geist
Capt USAF

Approved for public release; distribution unlimited.

AFIT/GCS/EE/81D-8

DEVELOPMENT OF THE DIGITAL ENGINEERING LABORATORY COMPUTER NETWORK: HOST-TO-NODE/HOST-TO-HOST PROTOCOLS

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
John W. Geist, B.S.
Capt USAF
Graduate Computer Systems

December 1981

Preface

This report presents further analysis and design of a host to host and host to node protocol structure. The work is in support of the continued development of the Air Force Institute of Technology Digital Engineering Laboratory Local Computer Network (DELNET). It is hoped that the capabilities developed, and the structure designed, will provide a sound basis for the DELNET's future development and operational implementation.

I wish to express my appreciation to Dr. Gary B. Lamont, my thesis advisor, for his valued support and guidance through the many paths this thesis has taken. Also, I thank my thesis readers, Lt Col James P. Rutledge and Maj Walter D. Seward for their cooperation and constructive comments. Additionally, I am indebted to Capt Charles E. Papp, whose concurrent thesis efforts were key to my progress.

Most of all, I would like to thank my wife, Fiona, and daughter Corinne, for their tolerance and support during this difficult period.

Contents

List of Figures

List of Tables

Abstract

Development of the Air Force Institute of Technology Digital Engineering Laboratory's local computer network (DELNET) was continued with requirements formulation, environment evaluation, organization and structure design, network node software implementation, and host software design. System requirements were evaluated from an implementor's viewpoint. System organization and protocol structure emphasized the International Standards Organization's Reference Model of Open Systems Interconnection protocol standard. A locally developed network node, the Universal Network Interface Device (UNID), was advanced and confirmed as operative. Methodology was established for targeting node operating system software, packages consisting of both high order language Zilog PLZ/SYS and assembly modules, to the UNID. An initial node software package was organized and implemented. The study continued with an evaluation of the host computer environment, applicable host requirements, and a map of the ISO protocol model on the DELNET topology. ISO model layers were defined for the DELNET, and design alternatives were suggested. The study formulates conclusions concerning DELNET development, and presents recommendations for follow-on research.

I. Introduction

The objective of this investigation was to continue the development of the host-to-node and host-to-host protocols required for the initial demonstration of the Air Force Institute of Technology (AFIT) Digital Engineering Laboratory Local Computer Network (DELNET). This network is the result of a number of influences including the recent explosion of computer network technology, USAF operational interest in base telecommunications, and AFIT's concern for both advancing research and optimizing resources. Interest in these areas spawned a sequence of projects that laid the necessary groundwork for a network configuration. This investigation built on that foundation and contributed to the initial DELNET implementation.

Historical Perspective

The use of communications technology to free the computer from its physical bounds propelled the computer industry into the network era. The ability to share computing resources over great distances offered users access to large main frames, specialized data bases, and unique hardware and software, at a fraction of the cost of discrete development. In the 1960's, distributed operations became a reality with large scale networks like GE Information Services, TYMNET, ARPANET et. al (Ref 18). These systems became the vehicles for technological

advancement as their operation became the subject of intense research and development.

Against this background, the computer industry experienced phenomenal advances in hardware technology. Large Scale Integration (LSI) development resulted in the emergence of small, inexpensive computers. Proliferation of these mini and micro computers enabled organizations to acquire multiple systems, each dedicated to a specific purpose. Unfortunately, the dramatic cost decreases experienced with processors was not accompanied by equivalent decreases in the cost of support elements - peripherals, software, mass storage etc. The desire to share those relatively expensive resources focused attention on networking the computers at the local level. In an effort to operate as efficiently as possible, network technology stemming from the large scale applications of the 1960's and 1970's, was applied to local environments. The result was the local computer network (LCN), a system owned by a single organization, and characterized by computers operating in close proximity, while communicating with each other at very high speeds.

Background

The potential for shared resources and distributed processing has focused attention on the LCN. In 1977 a technical report was produced by the 1842 Electrical Engineering Group at Scott AFB, Ill (Ref 23). This report addressed the assessment of an economic, feasible, and

responsive base level communication system for the 1980's. The last part of the report included a scenario for typical base communications, and introduced a multi-ring network as the prime candidate for topology concept. This multi-ring network used five different types of node devices to concentrate or distribute communications. An evaluation of the node requirements for this network concluded that the different types of nodes could be engineered as a single device (Ref 4). MS thesis efforts at AFIT resulted in the development of a prototype Universal Network Interface Device (UNID) (Ref 1).

Concurrent with the UNID development, the AFIT Digital Engineering Laboratory (DEL) became interested in the possibility of implementing its own local computer network. With its abundance of mini and micro systems, the laboratory was an excellent environment for a local network. By assembling the systems into a network, not only could the equipment be used more efficiently, but increased opportunities could be realized for research into the newly emerging areas of distributed processing and computer networking.

As a result of this interest, AFIT/ENG sponsored a thesis effort to identify requirements and specify an initial design for the DELNET (Ref 10). Besides an overriding demand for flexibility, the desired performance capabilities included: virtual system transparency, software tool sharing, peripheral device sharing, a file transfer

capability, a multiple network interface potential, and a capability for distributed data base applications. The derived hardware specifications included: a loop network topology, a two-UNID network concentrator system using a fiber optic link, and three host computers. The suggested initial configuration is represented in Figure 1.

Problem and Scope

This study focused on the problems that remained for the initial demonstration of the DELNET. There existed two prototype UNID's. The recommended initial configuration consisted of a network of the two UNID's with the VAX-11/780, Data General NOVA, and Intel Series II systems serving as host computers. The purpose of the study was to continue the development of the host-to-node and host-to-host protocols required for the initial DELNET configuration. It included the development and verification of UNID operations, the implementation of a local and network operating system for the UNID, and a generalized methodology for the host computer implementation.

Approach/Objectives

The first step in this study consisted of a search of the current information available on protocols and local computer networks. The explosion of network technology is reflected in the vast amounts of references in the current literature. Numerous articles, documents, studies, and texts, as well as four thesis reports (Ref 1, 4, 10, 19)

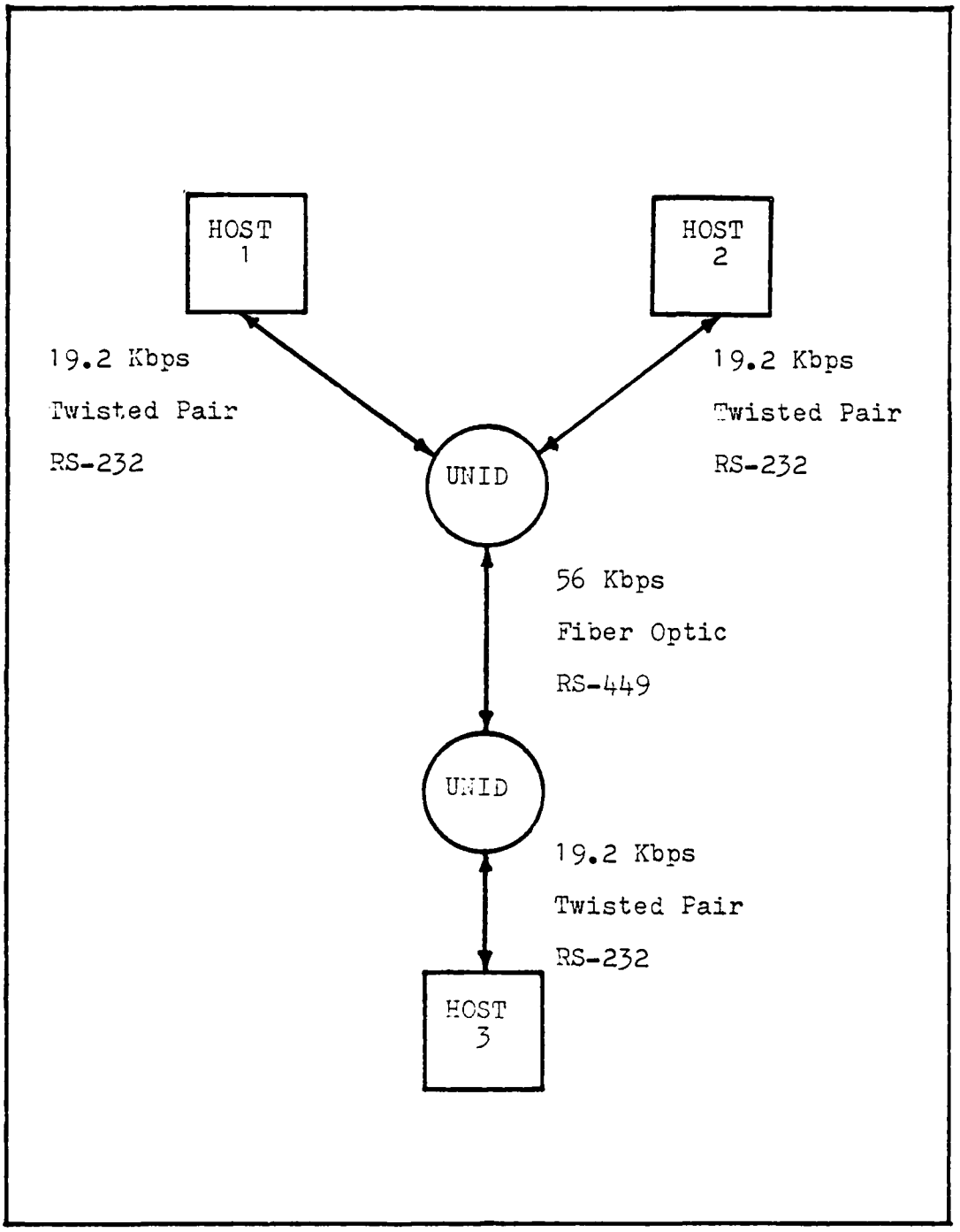


Figure 1. Initial DELNET Configuration

provided an abundance of information. Additionally, the variety of equipment, software, and operating system environments presented by the potential host computers and Z80 software development host was reviewed to enable a working knowledge of those systems.

The next objective was to develop an overall framework for the network protocols. This effort parallels a top-down, structured approach where the global perspective on structure, constraints, and standards applicable to the system were established. This framework proved to be an important asset as it provided a consistent development reference, minimized confusion, and kept design alternatives visible.

With the establishment of those standards, the functional specifications for the protocols were addressed. The low-level protocols specified by Hobart, along with their software transformations, were evaluated and integrated with appropriate upper-level protocol technology. This upper-level area was derived from both the standards developed earlier, as well as from current literature and proven commercial applications.

Once the functional specifications were complete, the environment available for development was evaluated to determine its potential for project implementation. The DELNET requirements were re-examined in light of the capabilities found, and a global project plan was established.

With the framework identified, the specifications prescribed, and the project approach determined, the development process was begun. Host-to-node protocol was the first task to be addressed for two reasons. First, since the UNID is the core device in the network, it was only natural that development begin with what is central to the network. Unique aspects discovered with the UNID created standards for the following host interface. Second, completion of the host-to-node area speeded parallel thesis work. With a dependable communication capability, UNID hardware development could rely on demonstration packages for diagnostic and reliability testing.

The concept of top down development was particularly important with the variety of untested equipment being used in the network. In general, a scheme of implementation was developed from specifications and implemented. Once confidence was established in the general technique, the scheme was modified appropriately for the different equipment and the validation repeated. In this manner, obstacles were addressed with minimized variables, and measurable milestones were identified throughout the development process.

After the successful verification of the UNID operations, design and development of a local and network operating system proceeded. With this development, the full capabilities of the UNID were exercised, providing for a critical evaluation of its operational potential and

reliability.

The final phase of the project analyzed the host environment and applied specific protocol model functions to the DELNET topology. This phase identified the remaining technical problems, and completed the structural plan required for DELNET implementation.

Overview of Thesis

The structure of this report parallels the sequence given under the approach. Chapter II presents the groundwork and structure for the network protocols. Chapter III develops the DELNET functional requirements as applicable to this thesis. Chapter IV presents the environmental elements available for this study. DELNET design and approach to testing is presented in Chapter V. Chapter VI discusses the problems faced with the UNID environment. Chapter VII describes UNID software design and implementation. Chapter VIII develops the host structure, while Chapter IX summarizes the report and gives recommendations for follow-on research.

II. Protocol

Introduction

The purpose of this chapter is to present the motivation for and a discussion of the protocol standard to be used for this project. Pouzin and Zimmermann (Ref 16) describe protocols as the common tools designed for controlling information transfer between computer systems. Definitional simplicity is desired since the concept of protocols spans both logical and physical bounds. Any agreed upon set of interface rules, from the physical description of RS-232, to a logic handshake between two processors, is a protocol.

It was the global scale implementations of the late 1960's that spawned the term protocol. Until then, processor communications was largely "face to face" within a single system and a single manufacturer. With the attempt to connect dissimilar computers over large distances, an entire new technology was born. LCN's face that same dissimilarity and require the control and service products of protocols. The key point is that with the LCN, the attributes of close proximity, high data transfer rates, and intermittent man-machine operations, significantly reduce the problems of overlaying a system with a protocol. The discussion of protocols can be better addressed in terms of concepts and structure.

Protocol Concept

The range of protocol concepts is limited only by imagination and the physical constraints of the network topology. Star topologies imply some type of polling process (Ref 18). Loop or bus topologies present a distributed environment where the mechanisms for control are made available at all nodes. Clark describes a number of concepts currently popular with bus or loop topologies including daisy chaining, control token, message slot, register insertion, and bus contention (Ref 5). The key point to these techniques and what enables their simplicity, is the ability of the local computer network to sacrifice transmission bandwidth. With inexpensive but high speed network transmission, LCN's can take advantage of their generally low utilization rate to enable distributed control and redundant transmission.

The recommended DELNET topology reflects the desire for flexibility, easy expansion, and optimized message throughput. Additionally, since the DEL does not support operational missions, availability was not a major concern and redundancy requirements were eliminated. A basic loop architecture was chosen to enable simple message routing and flexible modification. Additionally, multiple hosts are allowed to be connected to a single node (a star subnetwork). With this configuration, node costs are minimized and high inter-host traffic can be isolated at a single node to eliminate traffic from the network. This

configuration resulted in a store-and-forward type, packet-driven protocol (Ref 10).

Protocol Structure

It should be obvious that the job of a protocol could be quite large. The computer industry has been trying to formalize, at least in conceptual terms, the structure of protocol systems. The result has been a movement towards a hierarchial or layered structure. Schneider (Ref 17) provides three advantages to this approach. First, there can be a logical separation of functions. As in any complex communication system, dividing processes into smaller parcels facilitates their management. Second, responsibility for resource management can be segregated. With some divisions, each level of the hierarchy can be resonsible for a different class of resource. Third, and most important in the LCN environment, evolutionary change can be easily supported. With each level of protocol transparent to other levels, changes at one level will not affect the entire system. In fact, if the interface specifications between the levels do not change, no changes outside the affected level will be required. This is a particularly essential element which can enable graceful evolution in the volatile LCN environment.

While there is agreement on the requirement for layers, there exists little consensus on what the layers should contain. The International Standards Organization (ISO) has developed The Reference Model of Open Systems

Interconnection (OSI) in an effort to provide international standardization of various protocols. In developing this model's seven layers, the ISO applied several major principals (Ref 34). First, a different layer should be created for each different level of abstraction. Second, each layer should perform a well defined function. Third, the functions chosen should minimize data flow across the interfaces. Fourth, the layer boundaries should be chosen to minimize data flow across the interface. Fifth, the layers should be large enough to support distinct functions without becoming unwieldy.

Figure 2 and the following information stem from the excellent description of the ISO model by Tanenbaum and comments by Berglund, Schneider, and Walden (Ref 3, 17, 20, 22).

The Physical Layer. This is the lowest link between two nodes in a network. Its design deals with the mechanical, electrical, and procedural interfacing. It is basically concerned with transmitting raw bits of data over a communication channel and answering questions about connections, voltages, transmission capabilities etc. An example of a physical layer is the RS-232C standard.

The Data Link Layer. Since the physical layer accepts and transmits data bits without any regard to their meaning, the data link layer must create and recognize logical data entities and govern their flow. This is generally accomplished by creating frames of data. The creation and

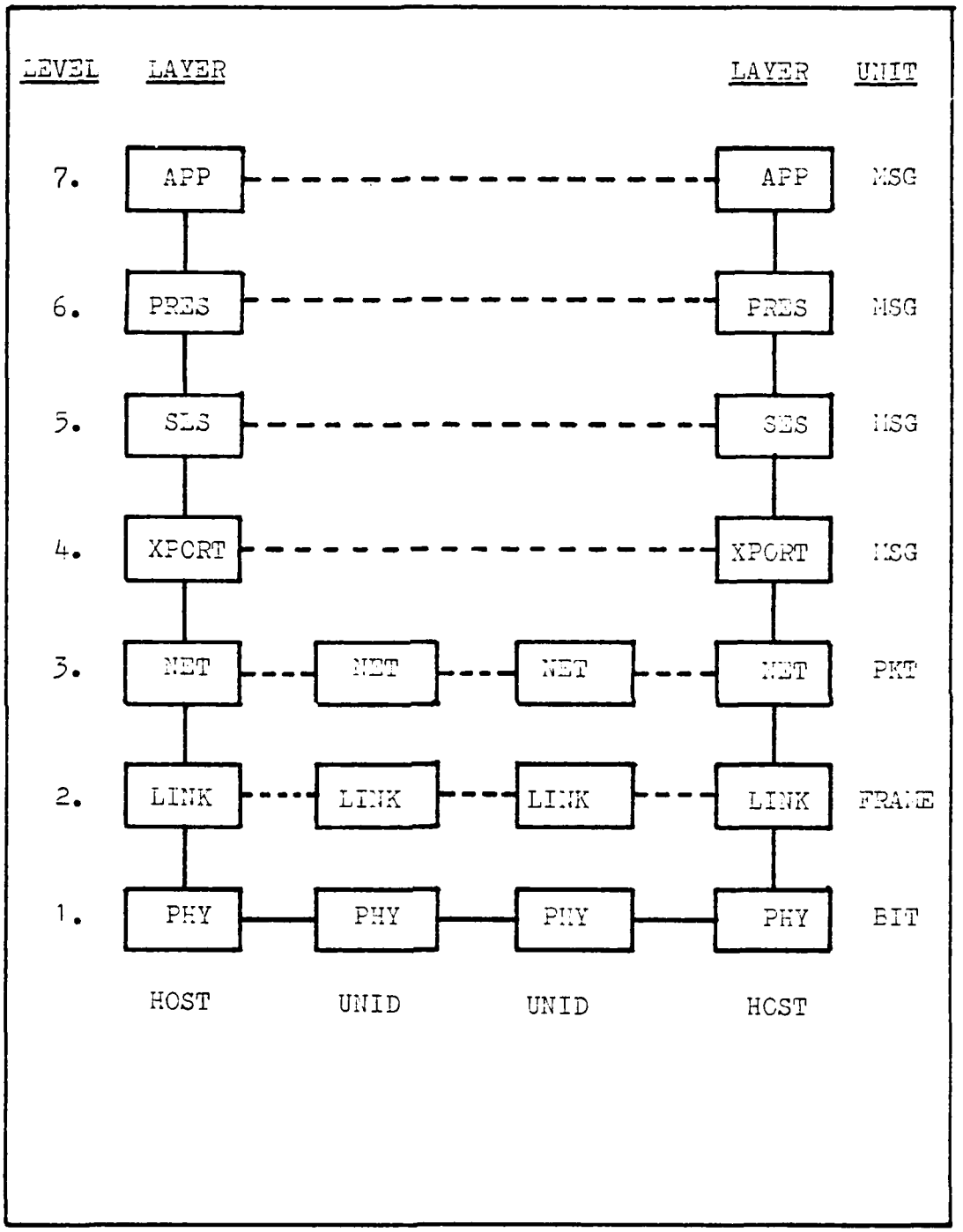


Figure 2. ISO Protocol Model With UNID

management of these frames will provide the network layer with an error free transmission of data. In this respect, the data link layer may provide the network layer with several different types of services, each of a different quality and price.

The Network Layer. This layer is concerned with the routing and management of packets. It largely determines the characteristics of 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.

The Transport Layer. Also known as the host-to-host layer, its job is to supply the methods for establishing communication paths between hosts, managing buffer space at both ends, controlling data flow, etc. This is an end-to-end layer where a source machine carries on a conversation with a destination machine. It is the highest layer associated with actual transport services, and as such, generally provides virtual services to the users of the transport service above, that is, the session, presentation, and application layers.

The Session Layer. This layer is the user's interface into the network. Its job is to establish a connection and manage the dialogue in an orderly fashion. This connection or session may be used, for example, to allow for time sharing or for transferring a file from a source host to a destination host. This process includes setting up the connection, establishing agreement on session options

(called binding), managing the session, and disconnecting the session.

The Presentation Layer. This layer is tasked to perform functions that can be offered as library routines. This includes format compatability conversion, text compression, encryption, terminal standardization, etc. The presentation layer attempts to alleviate inconsistencies in the network faced by different host users.

The Application Layer. The content of this layer is determined by the users. The set of requests and appropriate actions are heavily host computer dependent, but several generic processes are common. These include actions such as remote job entry or file transfers.

Summary

This chapter presented the protocol model to be used during this investigation. The model provides for seven layers of protocol from the specific physical transmission definition, to the more abstract application layer. It is the intent of this study to develop the host-to-node and host-to-host protocols in accordance with the philosophy and guidelines of the ISO model. While commercial experience has shown problems with one-to-one mapping of applications, using the model provides a coherent picture of the network in terms of protocol. Where mappings are difficult, sound logic and reasoning must prevail, and departure carefully examined as appropriate.

III. Functional Requirements

Introduction

This chapter introduces the functional requirements applicable to this investigation. These requirements stem from the global concepts developed from DELNET system requirements, the specific translation of those concepts into system specifications, and finally the scope of this investigation.

Global Requirements

The global requirements are those which apply in general to all phases of the project. With their abstraction, they amount to goals rather than specific measurable requirements. The source of these goals stemmed from Hobart's definition of "Design-oriented Functional Requirements" (Ref 10), and are reinforced by the concepts of the ISO protocol model. Specifically they include: flexibility, virtual operation, and performance monitoring.

Flexibility. The need for flexibility is derived primarily from concerns for reconfiguration. The variety of equipment, academic interests, and projected system expansion justify those concerns. The ease in which hardware, software, topologies, and network concepts can be modified will directly influence the DELNET's use and its future. As with other local computer networks, the flexibility issue is key, and must be considered in all

phases of development.

Virtual Operation. While not addressed directly in Hobart's user survey, the implied operation of the network was to be virtual. Giving the user the illusion of operating a single system is a complex problem. Issues of common network commands, user required knowledge of service and servers, and transparency of operations from host to host, are all problems where a trade-off of service costs versus system transparency takes place. Throughout this development, reasonable priority will be given to transparency, and tradeoffs will be evaluated against the goal of maintaining effective virtual operations.

Performance Monitoring. The requirement for a performance monitoring capability is derived from several directions. First, in order to validate the system's implementation, some mechanism for operations monitoring must be in place. Second, anticipated pedagogical interest in the network can be enhanced through a capability that can provide research data. Third, the growth and future modification of the network would benefit greatly with real-time feedback of the system's reaction to changes. Development of the network therefore must encompass a performance measurement concept.

System Requirements

These requirements are the results of multiple issue influences including the global requirements previously discussed, technical aspects of local computer networks, and

constraints of the initial DELNET configuration. They include the requisite for packet-switch protocol, X.25 protocol, and a simple message routing technique.

Packet-Switch Protocol. A number of issues led to the desire for a packet-switch protocol. Flexibility demanded logical rather than physical circuit establishment. Network response goals eliminated circuit monopolization by a single user. Transparency required all forms of data to be processed similarly. Efficiency suggested a parallel processing potential with effective time division multiplexing (TDM) of message portions. Packet-switching met these concerns while providing effective use of network transmission bandwidth and allowing multiple quasi-concurrent use of physical circuits. The DELNET will effect a packet-switched, store-and-forward type protocol.

X.25. This recommendation by the International Consultative Committee on Telephones and Telegraph (CCITT) is the proposed international standard network access protocol for ISO model layers one, two, and three (Ref 20). Investigation of other commercially available protocols found significant deficiencies including a lack of technical sophistication and the over-dependence on specific hardware. The versatility of X.25 and its endorsement by CCITT led to its specification as the access protocol for the DELNET (Ref 10).

The X.25 recommendation defines the three layers of protocol through references to X.21 at the physical layer,

Link Access Procedure (LAP) at the data link layer, and packet control at the network layer (Ref 20: 167-172). Figure 3 represents this X.25 application on the ISO protocol model. This study will structure X.25 at the network layer, and interface with the adjacent data link layer.

Routing Technique. Beyond a requirement for network capability information at the session level of protocol, the specification of a simple loop topology eliminates the need for a sophisticated routing concept. The store-and-forward technique will simply enable the UNID to extract from the network traffic the information destined for one of its local ports. Additionally, the UNID will inject data from one of is associated host computers into the network stream when appropriate. The more important aspect of this requirement relates to the flexibility issue. The protocol structure developed during this investigation must be capable of absorbing a more complex routing algorithm coupled with a more complex network topology.

Detail Requirements

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

Network Operating System. To eliminate the need for each host to manage its role in the network, and to provide

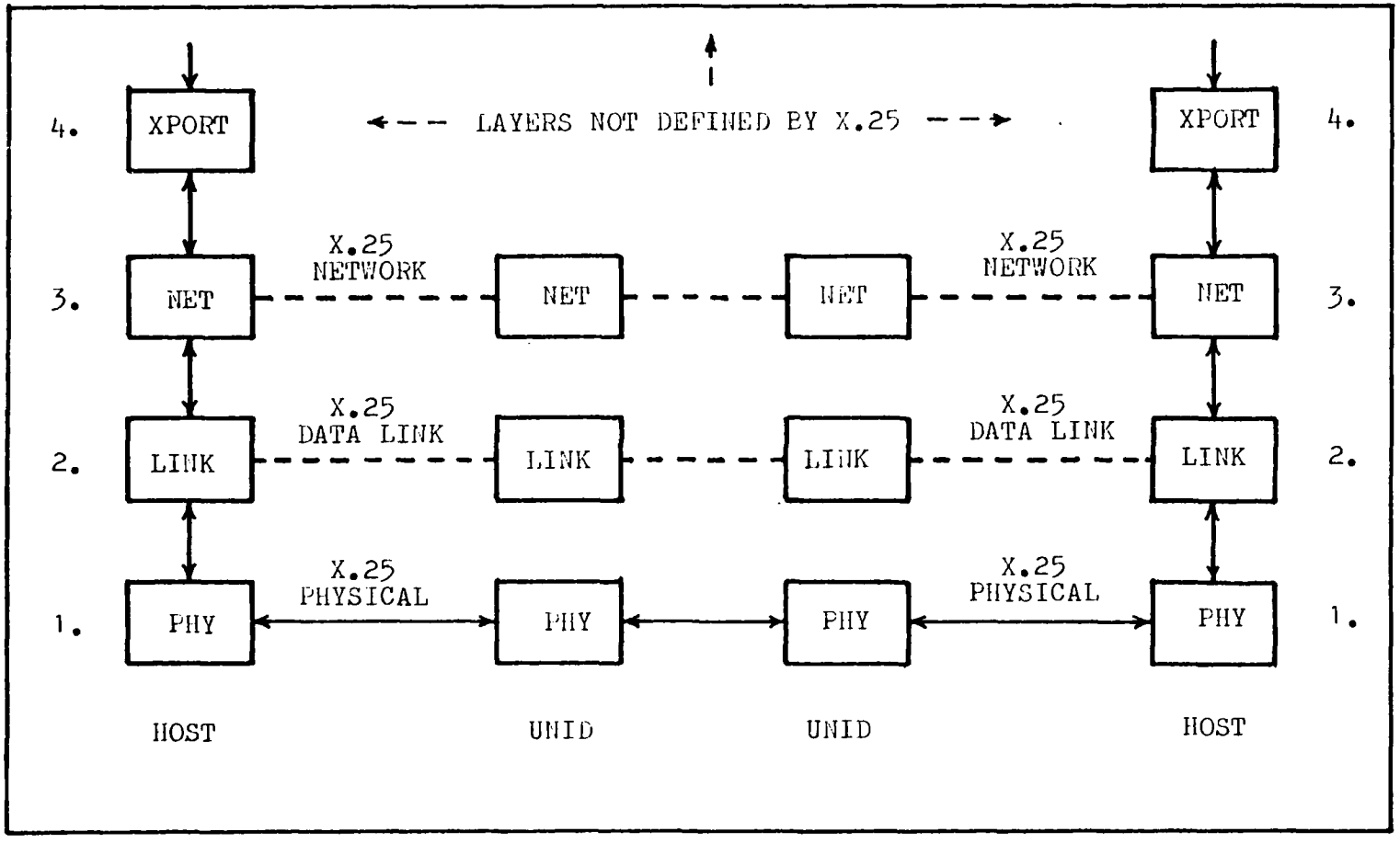


Figure 3. X.25 Application On ISO Model

uniform and consistent management of the network, several functions are required to process with a global scope. The functions specified for the initial DELNET operation include network access control and user help services.

To enable a user to operate in the network, a LOGIN command will be used. This command will verify access authorization, identify the user to the network for data routing and status, and initialize the host for DELNET processing.

With a LOGOUT command, the reverse process of LOGIN will be performed. The user is removed from the network configuration at the NOS level, and the host-dependent interface to the network is terminated.

HELP commands are required to provide any user with convenient information about the network. The information available must include a network overview, current network status, and command syntax instructions.

Application Functions. The application functions are those required to demonstrate viable DELNET operation. They include the capacity for user message transfer, file transfer, and remote job execution.

The function of the message transfer capability is to enable user console communications from host to host. The file transfer requirement is included to enable data identified as a file on one host to be transferred to any other network host. Finally, the remote job execution will allow command files on any host to be executed by any

network user.

Summary

The purpose of this chapter is to identify the functional requirements for this investigation. These requirements are derived from several views of the DELNET system. At the global level goals of flexibility, virtual operation, and performance monitoring have been established. For system requirements, packet-switching, X.25 protocol, and a simple store-and-forward routing technique have been defined. Finally, the scope of this investigation limits detailed requirements to those which will demonstrate the DELNET operation. They include the network operating access control and user help services; and application functions of message transfer, file transfer, and remote job execution.

IV. Development Environment

Introduction

The functional requirements of chapter three described the "what" of this investigation. This chapter will address the initial elements concerned with the project environment. Its purpose is to identify the significant aspects of the hardware and software environment applicable to this project. Specifically, it will present environmental information about the UNID, the Zilog MCZ-1/25 microcomputer system, and selected host computers.

UNID

As was stated earlier, the UNID is the product of USAF interest in the development of a universal interface device for use as a network concentrator. The concept was developed through Sluzevich's evaluation of operational requirements and available technology, and the application of that technology by Baker and Papp (Ref 1, 15, 19).

The significance of the UNID to this investigation rests in its selection as the DELNET concentrator. Basically, the UNID is a dual Z80-based (Ref 2) microprocessor system. One processor is configured as the "local" processor, and is tasked to support interface and communications to four host computers. The second Z80 processor is configured as the "network" processor, and communicates directly with the network. The two processors

communicate with each other through a shared memory technique. Host computers tied to the UNID can input data to the local processor which can route the data to another attached host or to the network processor for output on the network. Messages input to the UNID from the network will be sent through the network processor to the local processor for distribution to the proper host. Messages input to the UNID from the network, but not addressed to a host on that UNID, will simply be forwarded back out on the network.

Both the local and the network processor have the same memory configuration. Each processor is configured with 2K (1K = 1024 decimal) of erasable, programmable, read-only memory (EPROM); 2K of empty EPROM (socket only); 4K of individual system random access memory (RAM) physically located on the processor board; 24K of individual system RAM made available through the use of a "system" memory card; and 32K of "shared" memory on a special shared memory board.

The local and network processors differ substantially in terms of communications interface. Both processors include the standard Universal Synchronous/Asynchronous Receiver/Transmitter (USART) configuration on its processor board (Ref 2: 247-248). The local side contains an additional four USART's for host interface, while the network side processes network communication through a Z80 Serial Input/Output (SIO) component (Ref 33). The additional four USART's and the SIO component are housed on

specially constructed boards that extend the local and network boards respectively. Figure 4 presents a general view of the UNID structure.

Zilog MCZ-1/25

The MCZ-1/25 is a disk-based microprocessor system that utilizes the Z80 family of microcomputer cards (Ref 2: 247-258). It has been the subject of numerous projects in the DEL including both course and thesis development. The significance of this system rests in its role as a software development bed that can target software for the UNID.

The MCZ-1/25 contains a single Z80 microprocessor and a complement of support equipment. It includes a Cathode Ray Tube (CRT) terminal with keyboard for operator interface, a dual floppy disk drive, an NEC Spinwriter for printed output, and a special Zilog Analog Input/Output (AIO) processor board.

The memory configuration differs slightly with the UNID processors. The MCZ-1/25 memory has 3K of EPROM, 1K of empty EPROM (socket only), 16K of RAM physically located on the processor board, and 48K of RAM located on a special memory/disk controller board. It should be noted that while memory physically exists past location FFFF hexadecimal, the MCZ-1/25 is constrained by word size from addressing beyond FFFF or 65535 decimal.

The communications interface for the MCZ-1/25 includes the standard USART on the processor board, a Parallel Input/Output (PIO) module for two port, 8-bit parallel

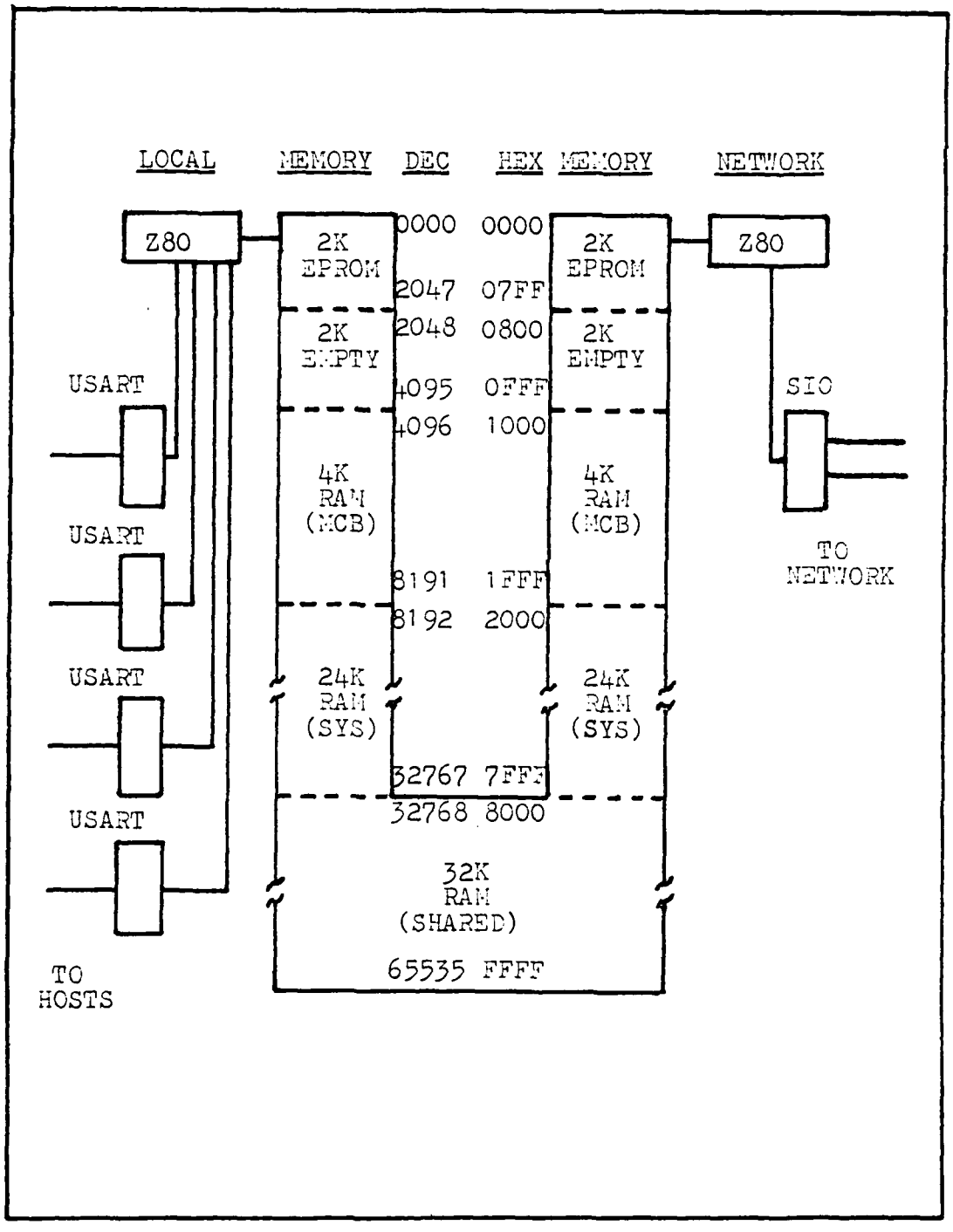


Figure 4. UMID Functional Diagram

transfers, and a Serial Interface Board (SIB) with four serial ports (Ref 32). The serial ports currently provide interface to the Spinwriter, and to the UNID for software transfer. Figure 5 presents a general view of the MCZ-1/25 structure. The interested reader can obtain additional information from the Zilog MCZ-1/25 product documentation (Ref 26, 28, 29).

Host Computers

The host computers provided the initial departure from the recommended implementation by Hobart. The information following is a brief discussion of the original three systems, and a suggested variation for implementation.

The VAX-11/780 is a full featured, state-of-the-art minicomputer, capable of a wide variety of multiprogramed tasks. The hardware is characterized by a comprehensive instruction set; a 32-bit, byte addressable word; stack orientation; and a page oriented memory management scheme augmented by two removable hard disks. I/O communications are currently interfaced via a UNIBUS subsystem providing for standard RS-232 connections at a variety of speeds up to 9600 baud (Ref 8).

General purpose in nature, the VAX is an exceptionally powerful device, able to support the most complex network role. Its multiprograming ability makes it an obvious choice for the initial network operating system host. It is currently supported by a rich software environment that includes: virtual memory management; event-driven priority

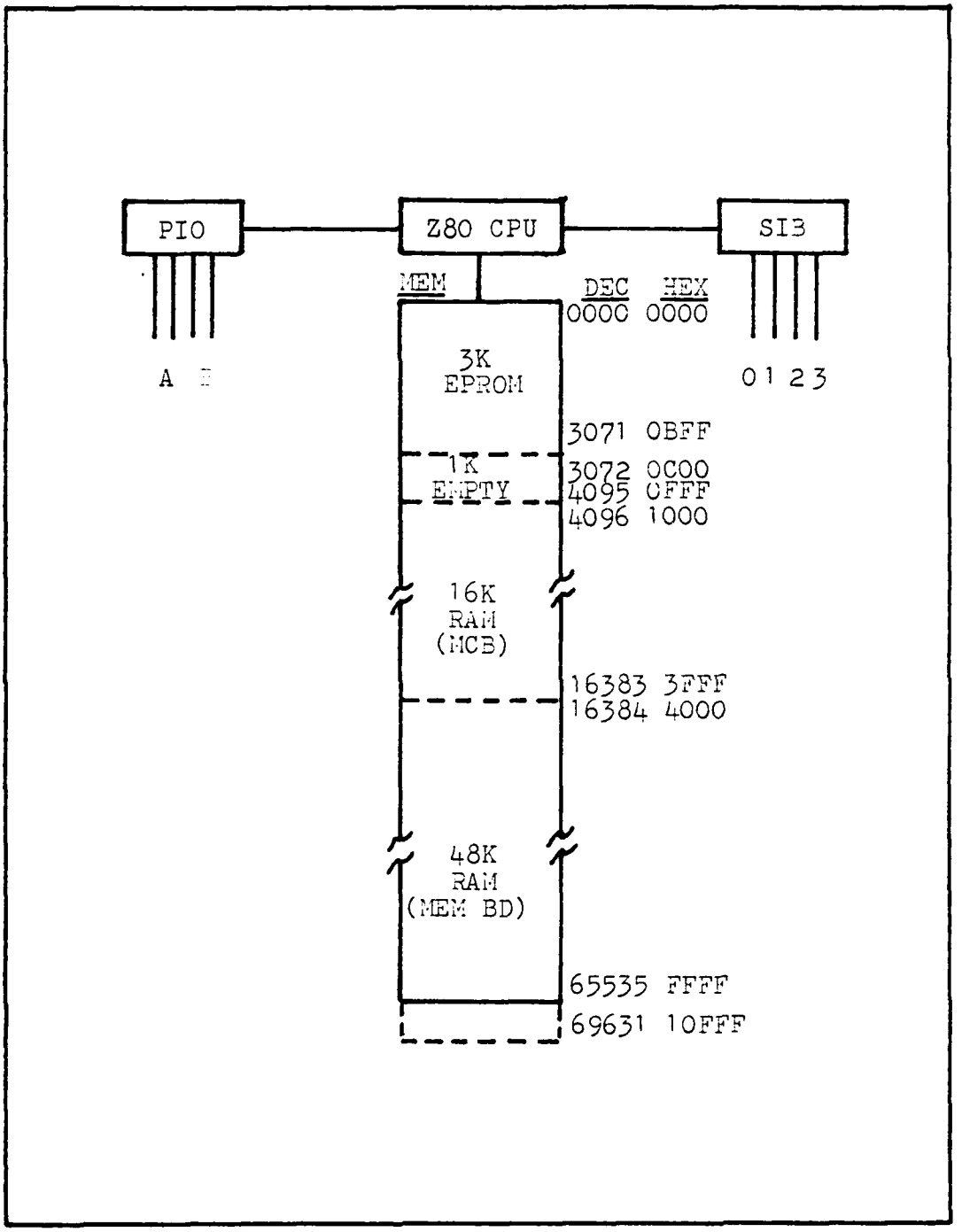


Figure 5. ZILOG MCZ-1/25 Functional Diagram

scheduling; shared memory, file, and interprocess communications; and a diverse set of system services for process and subprocess control (Ref 8). Common high order languages, including PASCAL, are available along with a wide variety of tools necessary for effective software development.

The Nova/Eclipse system is a mixed configuration designed to support signal and pattern recognition research at the DEL. The Eclipse supports foreground/background work, and communicates with the Nova through a shared disk system. The Nova is a single user, 16-bit processor, that interfaces with an additional Cromenco Z80 I/O processor for external communications. This system, in a variety of configurations, could support DELNET development.

The Intel Series II is a low cost, 8080 based, 8-bit, development system (Ref 11). As a single task microcomputer, it is equiped with RS-232 compatible channels capable of up to 9600 baud asynchronous operation. The system includes an eight level, vectored priority interrupt system for user programmed real-time response. Standard features such as a system monitor with diagnostics, a text editor, and an assembler, qualify the Series II as adequately supported for consideration as a viable DELNET option.

While the VAX was an obvious choice for DELNET inclusion, concern was felt over the selection of the Nova/Eclipse and Intel Series II for initial development.

The concern expressed was based not on a single, overwhelming problem, but rather on a series of small issues, and the availability of an excellent alternative. The Nova/Eclipse connection was originally targeted for the Nova/Z80 interface, and required Nova operating system modification for I/O tasking. Since the Z80 was not supported by an HOL, an assembly language interface would have to be developed. Additionally, the Nova is not currently supported with PASCAL, requiring either another language or assembly development of host processes. The Intel Series II presented problems of limited development availability with a significant course workload, low reliability, and the lack of PASCAL support. Also its physical position in the DEL required substantial cabling for its projected interface with the UNID's.

These technical concerns, while not overwhelming individually, presented a serious challenge to any initial implementation. In an effort to simplify the implementation burden, an alternative was sought. The DEL possessed three LSI-11 computer systems with six more projected for acquisition. These computers, while varied in their configurations, basically contained the 16-bit processor with operator console, dual floppy disk storage, and a capability for supporting four serial ports (Ref 7). Besides being plentiful, these systems were fully supported with software development tools and UCSD PASCAL. With a central location and ease of operations, they presented an

excellent option for initial inclusion into the DELNET.

Additionally, with the PASCAL support, a potential for singular host development exists. Software developed in PASCAL on the VAX could be transported on compatable floppy disks, via the VAX maintenance subsystem, to an LSI. With minor modifications, the one-time developed software could be made operational on both the VAX, and a large number of LSI's.

These capabilities, coupled with the concerns over the earlier systems, make the LSI-11 option very attractive. Using the VAX-11/780 as a development host and targeting software to the LSI's reduces the clutter of technical problems, and presents a simplified route to DELNET operations. With a configuration consisting of the VAX and two LSI's, the implementor can concentrate on network issues, rather than the myriad of technical implementation details that are sure to arise in multiple systems. Consistent communications interface, PASCAL availability, a host/target software potential, increased availability of hardware, and reduced cabling problems, support the selection of the LSI-11 as an initial DELNET participant.

Summary

In this chapter, the three major elements of the environment were discussed. The UNID provides a dual Z80-based microprocessor system with a complex memory of both system and shared portions. The UNID additionally communicates with four local ports via USART's and with a

network port via an SIO component. The MCZ-1/25 is a disk-based microprocessor which provides a software development bed for UNID targeted software. While differing slightly with the UNID, the MCZ-1/25 will enable both development and testing of targeted software. The VAX-11/780 provides a powerful tool for host and NOS functions. Replacement of the other two originally selected host computers is suggested. The LSI-11 computers offer substantial advantages, while eliminating numerous technical concerns from the initial implementation.

V. Design and Test Approach

Introduction

With both the requirements and environment identified, an organization can be established to support the goals of this study. The purpose of this chapter is to map the DELNET functional requirements onto the environment identified in the previous chapter. Additionally, this chapter will present both the development approach and test philosophy pursued during this investigation.

Requirements Map

The three major requirements divisions presented in chapter three were re-evaluated at this point. The purpose of this "second look" was to establish that each requirement was being addressed within the context of the working environment and had a reasonable expectation of successful implementation.

Global Requirements. The global requirements consisted of goals of flexibility, virtual operations, and performance monitoring. The goal of flexibility is nebulous, lacking a distinct, measurable result. In terms of this project, it was addressed with emphasis on the application of high order languages, modularized software development, and generalized programming techniques that were built to enhance system flexibility.

High order languages (HOL) were used as the software

base in the UNID as well as all host computers. HOL's such as the PLZ and PASCAL systems that were used, inherently support the structured precepts of modern programming practices.

Modular software development was supported by both the HOL's and the development techniques used to organize the software. In designing the processes addressed in this report, use was made of the data flow diagrams (DFD's). Basically, this technique maps data as it passes through transformations in a large process. This map is then converted via transform analysis into a structure chart that generally suggests program module scope and organization. While not an exact technique, DFD's can force a more thorough analysis phase eliminating process ommissions and generating a more complete plan. Additionally, they provide a strong, logical treatment for program module definition, substituting for the potentially erroneous intuitive approach. The interested reader can find a description of this technique in texts by Weinberg or Yourdon (Ref 24, 25).

This HOL and modularity concept was further refined with application along the philosophical boundaries of the ISO protocol model. With the functional division established through the experience reflected with this model, confidence could be gained with DELNET modular divisions along the same lines. While all ISO layers are not currently present within the DELNET structure, this

project considered the inclusion of all layers as an eventuality, and developed the system with that future goal in mind.

The question of virtual operations was addressed with host dependent software. Each host was required to have a network interface program (NIP) executing during DELNET operations. This NIP would provide several protocol layer functions in addition to the presentation/application layer area. Again, program modularity was used to provide a flexible, maintainable NIP since this appears to be the area most prone to change and subject to a large maintenance requirement.

The performance monitoring requirement calls for a capabililty for both program development and operational monitoring. This combination of throughput monitoring and trace evaluation will be enabled with performance monitor stations at each UNID and host. The UNID was designed to contain buffer monitoring software for throughput monitoring at network traffic concentration points. Each host computer was organized to buffer information concerning its operational situation, and the NOS host was structured to provide access to this information via monitoring requests.

System Requirements. The system requirements of packet switching, CCITT recommendation X.25, and store-and-forward protocol are all met with the implementation of X.25. This recommendation implements the first three ISO layers with a requirement of 128 byte packet transmission. Additionally

the store-and-forward protocol will be implemented in the UNID's network processor operating system.

Detail Requirements. The most specific requirements, those of the NOS and application functions, were implemented at various "locations" in the network. The NOS functions of LOGIN, LOGOUT, and HELP require a single source location in the NOS host. This host, besides being an active participant host in the DELNET, will provide additional services to the network in the form of these NOS functions. This single entity enables distributed control with only initially centralized authorization for network access.

The application examples of message transfer, file transfer and remote job execution require distribution of the capabilities to the hosts. These capabilities were absorbed into the NIP structure as modules to enable functional separation and easy modification.

The final product of this requirements map is represented in Figure 6. It is the intention of this map to provide both the underlying philosophy and the development thrust to the bulk of this project.

Development Plan

The scope of this investigation was centered on demonstrating a network capability with the DELNET. With that goal in mind, restrictions on full implementation were required. The requirements presented in chapter three and

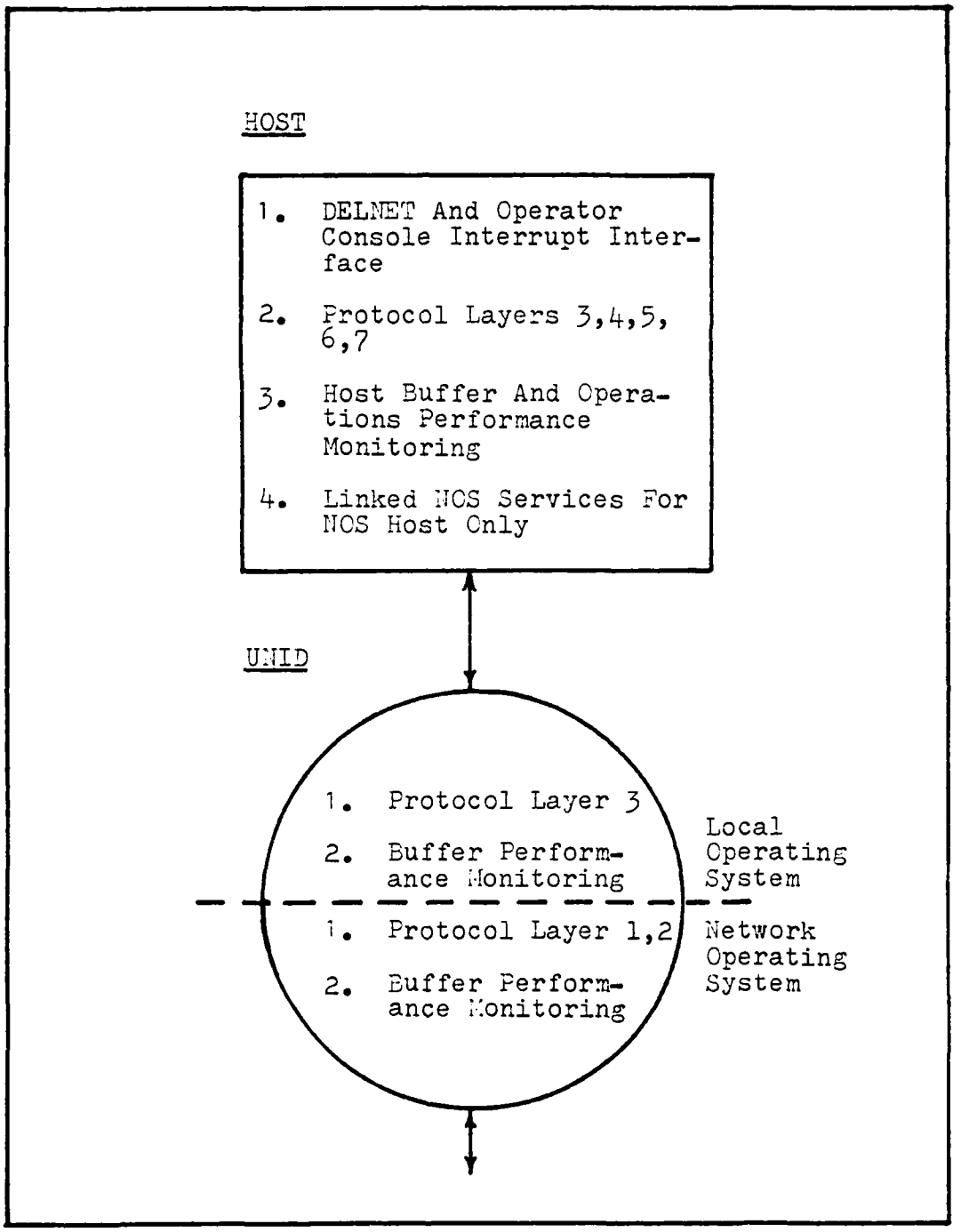


Figure 6. Requirements Map

refined in this chapter, set the stage for a global design viewpoint. It was not realistic, however, to implement the entire set of requirements within the timeframe allotted for this study. The thrust of this report focused on delivering the structure and organization necessary for a sound global network foundation, and the specific developmental elements required for a network demonstration.

The global structure was developed at three levels. First, an initial DELNET operating concept. Second, a general UNID structure and flow outline. Third, a universal host interface and organization. These three areas stem largely from the requirements definition as tempered by environmental constraints, and the ISO protocol model.

The specific elements developed included the plan for local and network operating systems for the UNID, a network operating system (NOS) structure, and a generalized network interface program (NIP) as applied to projected host environments. Throughout the specific development, concept and organizational structure took precedence over detail implementation. This was especially true in the case of X.25, where the protocol development is singularly complex. It is expected that once the general concepts are demonstrated, the details of the DELNET development will be addressed through course and thesis attention.

Test Plan

The initial phases of this study require little formal testing. Specifically, the UNID environment area and the

host communication areas are primarily investigatory in nature. Use of the UNID monitor debugger and demonstration software to evaluate operation was sufficient for verifying operations and establishing reliability. The monitor debugger is a modified version of the standard 1K MCB monitor offered by Zilog. The modifications by Baker kept the original commands such as Memory Display, Memory Fill, Register Display, Register Modify, and Program Step; and added commands for loading targeted software from the MCZ-1/25 (Ref 1). These capabilities, coupled with Zilog PLZ/SYS, assembly, and disassembly reference packages, created an adequate environment for program evaluation and testing.

Operational testing proceeded with a combination of system input response and memory evaluation exercises. Tests called for packet process combinations establishing operational completeness and reliability. These tests were additionally examined against PLZ provided memory map data for buffer counter and control data integrity.

Summary

This chapter reevaluated the major DELNET requirements with a view toward the environment available for implementation. Use of techniques available for system flexibility were stressed. HOL's, modular development, and ISO model application, were used to satisfy the global requirements. System requirements were met largely through the X.25 specification. Detail requirements were addressed

through functional alignment of network services either at the host or the UNID. The development plan focused on structure and organization with concept development, UNID operational implementation, and host structure. Finally, a test approach using standard monitor tools and operational packet response was established.

VI. UNID Environment

Introduction

Developing the UNID to where it could support the goals of this investigation was the initial objective. The device was never fully configured for its full memory or I/O porting capability. The software developed by Baker was largely of a test variety and not operational (Ref 1). The thrust of this investigation suggested a reasonably sophisticated software environment including a high order language base with assembly level module support interface. The purpose of this chapter is to describe the UNID environment, the initial problems encountered, the solutions chosen, and the results of applying those solutions.

UNID Hardware

The UNID configuration, as tested by Baker, was short of its full potential on several points. First, the shared memory configuration was limited to 8K. Second, the system memory cards, the cards extending each processor's system memory to a full 32K, were not complete. Third, the local card, the card which presented the USART interface to the hosts, was configured for only two host systems. The expansion of the prototype UNID to an operational configuration, as well as the development of a second UNID, was accomplished by Papp (Ref 15).

With these changes, however, the UNID presented itself

largely as an unknown quantity. This investigation faced the problem of establishing an operational software base in a changing and uncertain hardware environment. As a result, a conservative approach to UNID development was taken. With this approach, each new area of development was addressed in three general steps. First, the specific area of concern in the UNID was tested with the capabilities offered by the monitor software (Ref 29). Second, the operational technique (module interface, software application, etc.) was selected and verified as operative in the MCZ-1/25. Third, the actual implementation and testing of the technique was accomplished in the UNID. This general approach was effective in minimizing unknown variables and increasing confidence in the UNID as each iteration of these steps built on previously verified operations.

UNID Software

The desire for a flexible and maintainable product demanded the use of high order language (HOL) whenever possible. Since the MCZ-1/25 is supported by PLZ/SYS, it was decided that this HOL would be the language standard for the UNID.

PLZ/SYS. PLZ is a family of Zilog supported languages designed to satisfy the requirements of microcomputer applications. It includes: PLZ/SYS, a PASCAL-like compiler that produces an intermediate code file (Z-code); PLZCG, a second compiler, which accepts a file of Z-code as input and produces a file of relocatable Z80 object code; PLZ/ASM, a

low level assembler for Z80 object code; and PLINK, an object module linker capable of producing a single executable module (Ref 27). This family of software tools provided a robust environment for program development.

In attempting to apply the PLZ software tools to the UNID, several problems were encountered including software load difficulties, input/output interface troubles, and assembler linking complexities.

Load Anomoly. Baker had written software to enable transfer of UNID programs from the MCZ-1/25 to the UNID via a serial I/O port (Ref 1). This process required a set up in the MCZ-1/25, and the execution of a load command from the UNID local processor monitor. The file identified would be transferred to the UNID and loaded at location 1000, hexadecimal. The software could then be executed or manipulated via other UNID monitor commands.

The first problem noted with PLZ programs dealt with a loading anomoly. PLINK enables programs to be created from multiple modules as they are "linked" together. In this process, both the program memory load location (where the program is to be loaded in memory for execution), and the entry point location (where the program is to start), are identified and stored with the resultant file information. With PLZ, buffers and other data will be stored first in the file, and the entry point generally resides internal to the starting point of the file.

Baker's load software is designed to use the entry

point address for a load address in the UNID . With the assembly language routines that Baker implemented, the developed entry point and load position was always 1000 hexadecimal, thus a problem did not exist. With PLZ routines, the load command will load the software starting at the entry point address, leaving the true entry point offset from that position. For example, a PLZ program linked to be loaded at 1000 hexadecimal with a developed entry point of 1034 would be loaded at 1034 in the UNID and its actual entry point would be at 1068.

Once this situation was recognized, it was felt to be a problem that should be dealt with procedurally until the monitor software in the UNID EPROM could be modified. The technique used to load software is as follows: first, PLINK is used to link the object code at the desired hexadecimal location with the appropriate entry point module indicated normally; second, the entry point location is noted from the file information via a CAT command (Ref 30); third, the entry point value is changed to the desired hexadecimal location via the SET command (Ref 30); and fourth, the software is loaded via the UNID load command with the starting jump address as noted in step two. This procedure is easy to accomplish, and has little operational impact.

I/O Interface. The second problem encountered with PLZ was the absence of I/O interface in the UNID environment. The PLZ family is supported in the MCZ-1/25 with a variety of I/O packages developed in concert with the MCZ Operating

System (RIO). When PLZ programs are operating in the UNID, these I/O tools cannot be used due to their dependence on RIO and several Zilog proprietary modules.

An evaluation of this situation identified the requirement for a UNID system routine library capable of supporting the PLZ software. The nature of these routines also required that they be written in assembly language. The initial library (U.LIB) included routines for byte sequence input, output, and memory transfer. It is expected that this library will expand and be fine tuned with future UNID applications. Appendix A contains specific information concerning the library.

Parameter Passing. The requirement for library assembly routines to be linked to PLZ software presented the third area of concern. It is necessary at times, especially with input/output interface, for modules to communicate with each other. The PLZ parameter-passing schema is a technique used to enable information to be passed between PLZ modules. It can also be used between PLZ and assembly modules. This technique uses the system stack for a communication buffer, and requires housekeeping by the assembly module.

While straightforward in its application, the area is noted in this report because of limited documentation. The calling PLZ module builds a series of word-sized data areas on the stack containing input parameters and the return address. The assembly routine is obliged to save the IX

register, use the data as appropriate, and deallocate the passed parameter area prior to returning. The PLZ module then expects any returned parameters to be on top of the stack. It is recommended that the interested reader review section seven of the PLZ User Guide (Ref 27) and several examples of applicable UNID software to clarify this technique.

Module Interface. While persuing the previous problem, a fourth area of concern was resolved. With the local operating system evolving as a series of PLZ and assembly modules, and data movement requirements extending throughout both local and shared memory, concern was felt over the organization of such a system. The PLINK processor, combined with PLZ data-only modules, provided the necessary organizational capability.

PLINK possesses the ability to link multiple modules together as a single program. Additionally, the load location, the entry point or starting location, and any additionally desired module locations can be specified.

Also, PLZ/SYS enables a PLZ module to exist without any executable code. A module can consist of data buffers defined as global in scope, making them available for access by other modules through external references.

Using these capabilities, a UNID operating system can be created from multiple PLZ and assembly modules with specified locations. The operating system itself can be identified to reside in low memory (just above the EPROM)

while the data buffers can be distributed as desired over both system and shared memory.

This technique was tested in both the MCZ and the UNID and found to be exceptionally effective for memory mapping. With this technique, both PLZ and assembly modules access data and procedures via logical identifiers while the physical positioning of the software remains transparent to the programmer. Also selective global and external definitions enable the "hiding" of software from modules lacking a need for interface.

These two techniques provide a flexible tool for memory management, and enable organizational modifications with a single PLINK command execution. This capability provided additional flexibility with respect to UNID hardware development. Memory development difficulties were temporarily shunted without restricting parallel development by simply mapping around the sensitive memory segments.

I/O Interrupt. The fifth problem area dealt with enabling an interrupt-driven I/O interface on the local side of the UNID. The interested reader can find detailed information on the Z80 interrupt process in discussions by Leventhal and Barden (Ref 2, 13). Basically, the UNID local processor operates in interrupt mode two. This mode uses the I register combined with a sensing mechanism to point to a location containing the address of the routine that will service the I/O interrupt. With the local card configuration, the I register provides the high 8 bits of

the address while a Pricrity Interrupt Controller (PIC) senses the USART's ready indications and generates the low 8 bits. The PIC also has the capability of programmable priority. With a command input to the PIC, a priority of interrupts can be established among all the I/O being monitored by the PIC.

A typical receive interrupt would result in PIC identification, Z80 interrupt, I/O controller addressing via I register and PIC bit combination, controller execution with a PIC reset, and finally interrupt reestablishment with program continuation.

The transmit differs slightly with the local operating system identification of a transmit requirement, a transmit set-up procedure call to enable the communications at the desired port, PIC identification of the interrupt, Z80 interrupt, I/O controller addressing via I register and PIC bit combination, controller execution with a PIC reset, and finally interrupts reestablishment with program continuation.

This interrupt mechanism required the development of an initialization process, the creation of the I/O vector address organization, and the development of a series of I/O interrupt controllers for servicing the interrupts. An assembly code module (L.VINT) was created to meet these requirements. This module contains the I/O vector address table (IOVCTB) with preset I/O controller identification defined via DEFW commands (Ref 31). Each USART receive and

transmit interrupt has an associated procedure call which will service that interrupt. The requirement that the I register provide the high 8 bits of the vector address table demands that the IOVCTB be located on an even memory boundary (1600, 1700 etc.). This is accomplished by use of the Origin statement (ORG). With the ORG, the table can be located as desired, and the PIC-provided 8 lower bits will index into the table for the correct I/O controller address.

Following the IOVCTB table in L.VINT is an initialization procedure. The procedure will initialize all four local USARTS, set the I register, initialize the PIC, set the Z80 to interrupt mode two, and enable interrupts. The PIC is programmed for equal priority among all interrupts. This decision is arbitrary and can easily be changed when operational conditions warrant. The initialization routine is additionally supported by a USART initialization subroutine.

The remainder of the module is comprised of I/O controller routines to service the channel interrupts. A potential for code savings exists in this area with the fact that within the receive class, these routines are largely identical. The difference rests in the I/O port addressing differences. The decision to generate nearly duplicate routines was the result of two considerations. First, the flexibility issue is served with a simple, easily modifiable arrangement. Combining the four receive handlers into a

generalized receive handler would not be difficult, but would present challenges to any unique channel requirements. The unknown future applications facing the UNID warrant a quest for simplicity.

Second, on a conceptual level, the UNID may support more than a communication concentrator role. The potential for a logical host interface shift inside the UNID physical boundary warrants concern for a unique channel interface. For example, a Packet Assembler/Disassembler (PAD) could enable simple console interface to the UNID and require a much more complex role for the I/O controller. These considerations favored a simple, unique I/O interface for each local channel.

UNID I/O Ports. The sixth area of concern, the development of the local processor I/O interface, required confirmation of the UNID CTC alignment and USART porting addresses. The standard MCB board "console" or "TTY" CTC configuration and port address is the same as the MCZ-1/25. The arrangement on the local and network card was developed to support hardware convenience. It should be noted that the USART configuration on the MCB board differs substantially from the USART configuration on the local board. The MCB uses an 8251 USART configured with a Counter Timer Circuit (CTC) for baud rate control. The local board uses four 2651 USART's with two CTC's for baud rate control. There are significant differences between the two types of USART's. Additionally, the local card uses an 8214

Priority Interrupt Controller (PIC) for I/O interrupt generation. The interested reader can find details about this area in reports by Baker and Papp, and the component reference by Osborne (Ref 1, 14, 15). Table I shows the pertinent interface information.

Finally, a demonstration of the full local side hardware and software capability was performed. This was accomplished by a UNID configuration with ADM-3 consoles on each local channel and a Plessey PT-100 connected as the local processor console. The software used was a small demonstration program (L.DEMO1) which enabled 30-byte packet insertion at the ADM-3's with destination determination by packet contents. All channel transfer combinations were executed and a simulation of network transfer was accomplished via shared memory buffer insertion.

Summary

The process of creating an operational environment with the UNID presented several challenges. The full complement of hardware enabled application of the high order language family of PLZ. The problem of loading PLZ-linked software into the UNID was solved by procedure. The lack of I/O interface tools outside of the MCZ-1/25 system was met with the creation of an assembly library for byte sequence input, output, and memory transfer. Module interface and operating system organization was handled with the use of the PLZ parameter passing schema, the PLZ data-only module build capability, and the application of PLINK command options.

TABLE I

UNID CTC and USART Configuration

ADDRESS	COMPONENT	COMMENT
00	USART 1 Data	
01		
02		
03	USART 1 Command	
04	USART 2 Data	
05		
06		
07	USART 2 Command	
08	USART 3 Data	
09		
0A		
0B	USART 3 Command	
0C	USART 4 Data	
0D		
0E		
0F	USART 4 Command	
10	CTC1 CH 0	
11	CTC1 CH 1	Drives USART 1
12	CTC1 CH 2	Drives USART 2
13	CTC1 CH 3	
14	CTC2 CH 0	
15	CTC2 CH 1	Drives USART 3
16	CTC2 CH 2	Drives USART 4
17	CTC2 CH 3	
18		
•	•	•
•	•	•
•	•	•
D5	MCB CTC CH 1	Drives MCB USART
DE	MCB USART Data	Console or TTY
DF	MCB USART Command	Console or TTY

The proper porting and CTC timer associations were identified for operational use, and the requirement for interrupt driven I/O was supported with the development of a vector initialization and I/O interrupt controller module. With the successful completion of the local demonstration, the UNID was deemed capable of supporting operational host-to-node development.

VII. UNID Software Design and Implementation

Introduction

With a sound UNID environment, the process of developing operational software could be addressed. The UNID's job is three phase: it must concentrate the input communication from four "local" channels; it must distribute local network communication to the identified destination; and it must store and forward network traffic not addressed to the connected host computers. The purpose of this chapter is to address the development of UNID software required to support the host-to-node and host-to host area. Specifically, it concerns the development of the local and network operating systems.

Data Flow Organization

For a general concept, the UNID is required to be able to concentrate and distribute data elements. The elements it is required to handle are based on the X.25 packet (Ref 9). This packet has a variety of configurations and different associated meanings. To the UNID, however, these packets are without meaning and represent chunks of data to be pushed through the system.

With four local channels and one network port, the UNID can be thought of as having five sources and five destinations for data. The local side must be capable of accepting input from any local communication channel, and

routing that data either to another local channel or to the network. Parallel to that operation, the network side must be capable of accepting information from the network port and routing it to either a local channel or back out on the net. Additionally, each side must be capable of accepting data routed to it from the other side (local to net, net to local).

These data flow requirements suggest an organization for data buffers. On the local side, individual channel input buffers are required for I/O interrupt service at the byte level. A single local-to-local transmission buffer is required for packet transmission back out to a local channel. Additionally, two interface buffers are required for shared communications to and from the network.

On the network side, a single input buffer is required for I/O interrupt service. A single network transmission buffer for network to network communication parallels the local side organization. Finally, the network operating system must share the two buffers for communications to and from the local side.

This data flow enables reasonably simple table processing techniques. Each table can be flexibly defined in terms of the X.25 packet or frame. Constants can be combined to specify the size of the tables. Three variables defined for each table, XXXXNS, XXXXNE, and XXXXSZ, can provide the pointers to the next-to-be-serviced byte, the next-empty byte, and the byte size of the table. With these

ingredients, individual bytes are accessable, processing decisions can be accomplished by pointer disparity, and data movement and transmission can be transparently handled in terms of the constant-defined entities. Figure 7 presents a general picture of the UNID buffer organization.

Local Operating System

For the L.OS, the typical packet would flow from a local channel to an input buffer. There it would be examined for destination either to the net or back to a local channel. If sent to the net, the network operating system (N.OS) would accept the packet through the shared tables and process it. If sent to a local channel, it would be transmitted to that host. Packets could also "originate" from the network, in which case they would be examined for destination and routed to a local channel.

This data flow must be tempered with the realities of the UNID environment and some subtle processing concerns. The first concern is for I/O. Since the UNID must react in real-time with interrupt mode I/O, both input and output must be accomplished at the byte level. In this fashion, small I/O interrupt controllers are required to avoid interrupt timing conflict.

Second, while I/O is at byte level, routing is at the packet level. The data handling must be in terms of packet sized entities. This can be accomplished through table definitions and process thresholds based on a prescribed and constant-defined entity size.

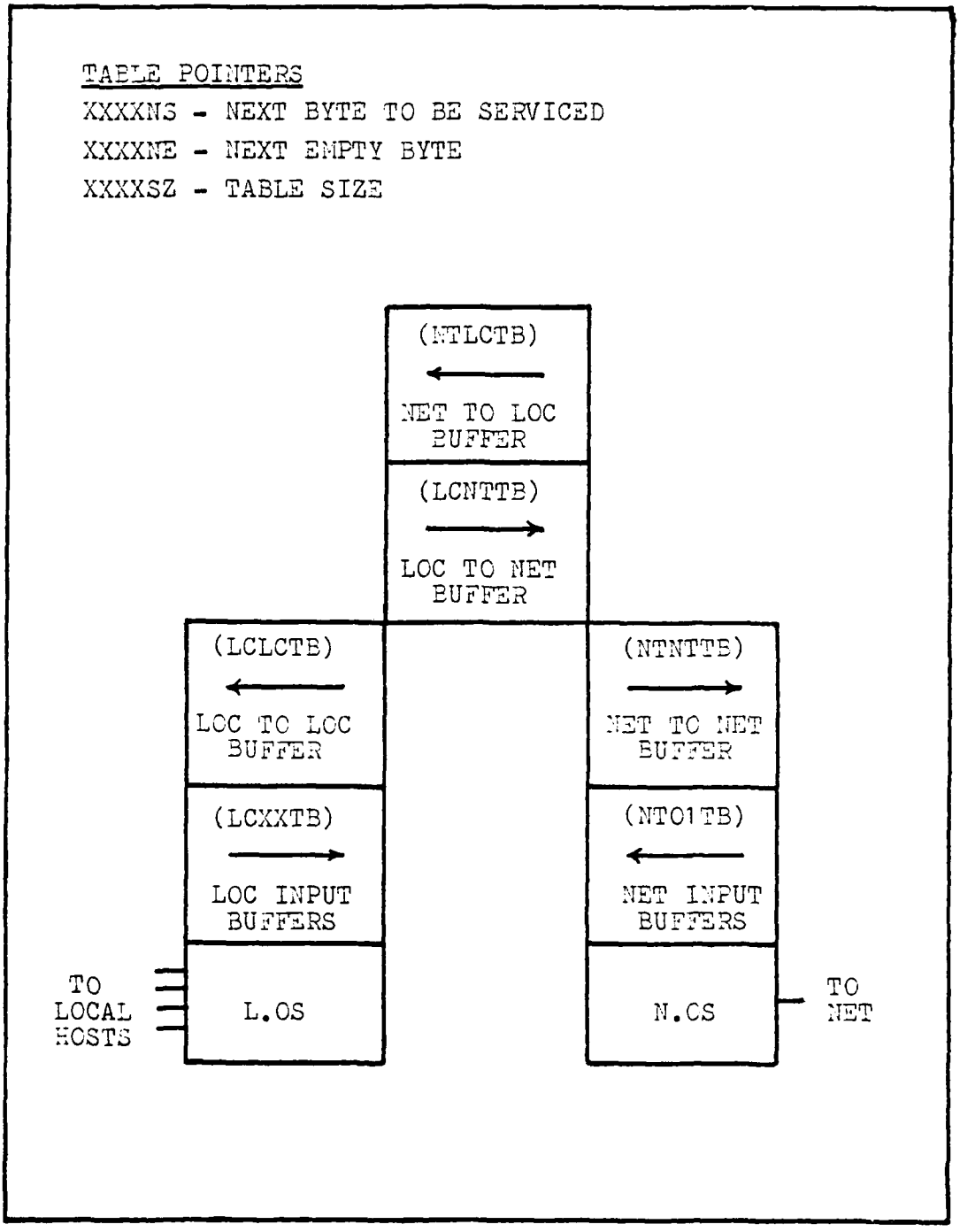


Figure 7. UNID Buffers

Third, while both input and route processing may be random, output transmission may be polled. With only two sources for output data (the local-to-local and net-to-local tables), each source can be examined at intervals thus eliminating a concern for random or conflicting output transmissions.

This analysis was translated into the DFD's shown in Figures 8-12. Figure 8 is the overview diagram and each transform bubble is expanded for detail in the following four figures. Using the DFD's, a structure chart for the L.OS was developed as shown in Figure 13. Main is the controlling driver with a one time call to U_SHTAB_INIT, L_TAB_INIT, and INVINT (an assembly support module) for initialization, and an endless loop of calls to ROUTE_IN and ROUTE_OUT. Both ROUTE_IN and ROUTE_OUT call DET_DEST for determining the destination of the packet. ROUTE_IN uses MOVSEQ (an assembly library routine) to move a packet in memory, while ROUTE_OUT uses TRNMIT to output a packet to a local channel. Procedures LD_TAB_HSKP and SRVC_TAB_HSKP are used to housekeep the buffer tables after a packet load or service (removal).

This organization was implemented in a PLZ environment with skeleton destination processing. X.25 provides a methodology for routing, and is quite complex. Since a demonstrated network capability was the primary goal of this investigation, a simplified packet routing technique was used to suffice until full X.25 implementation.

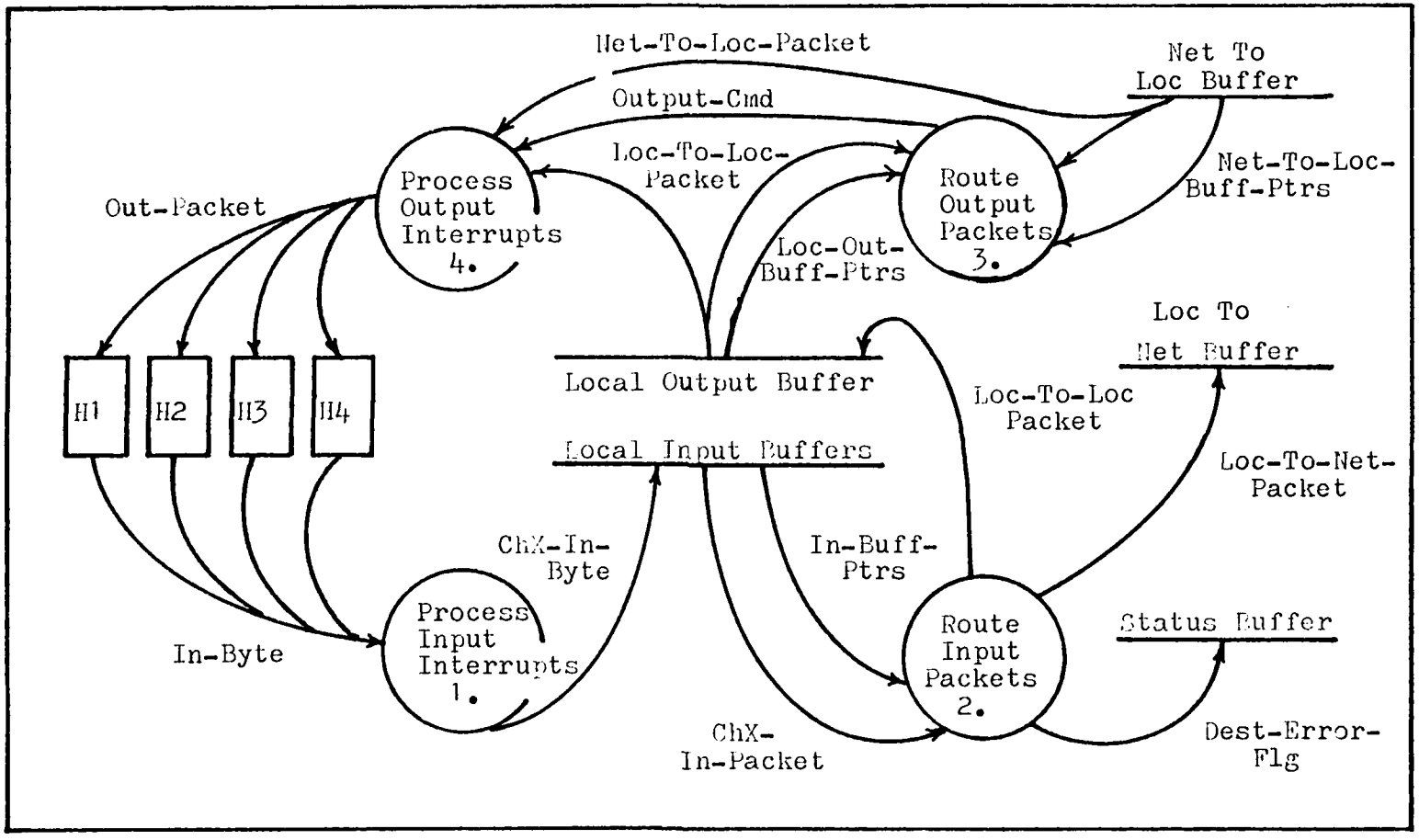


Figure 8. L.OS DFD Level 0

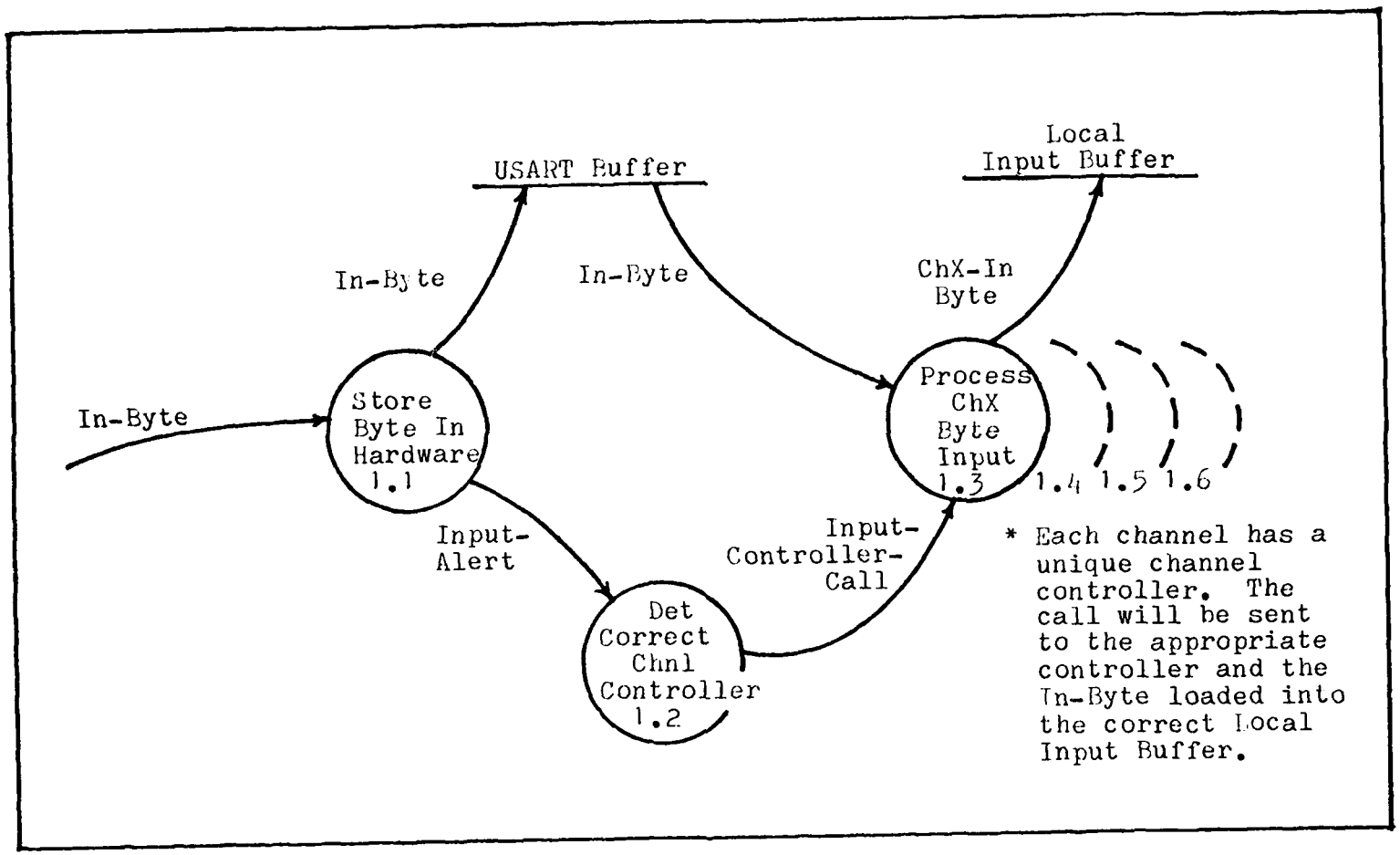


Figure 9. L.OS DFD Level 1

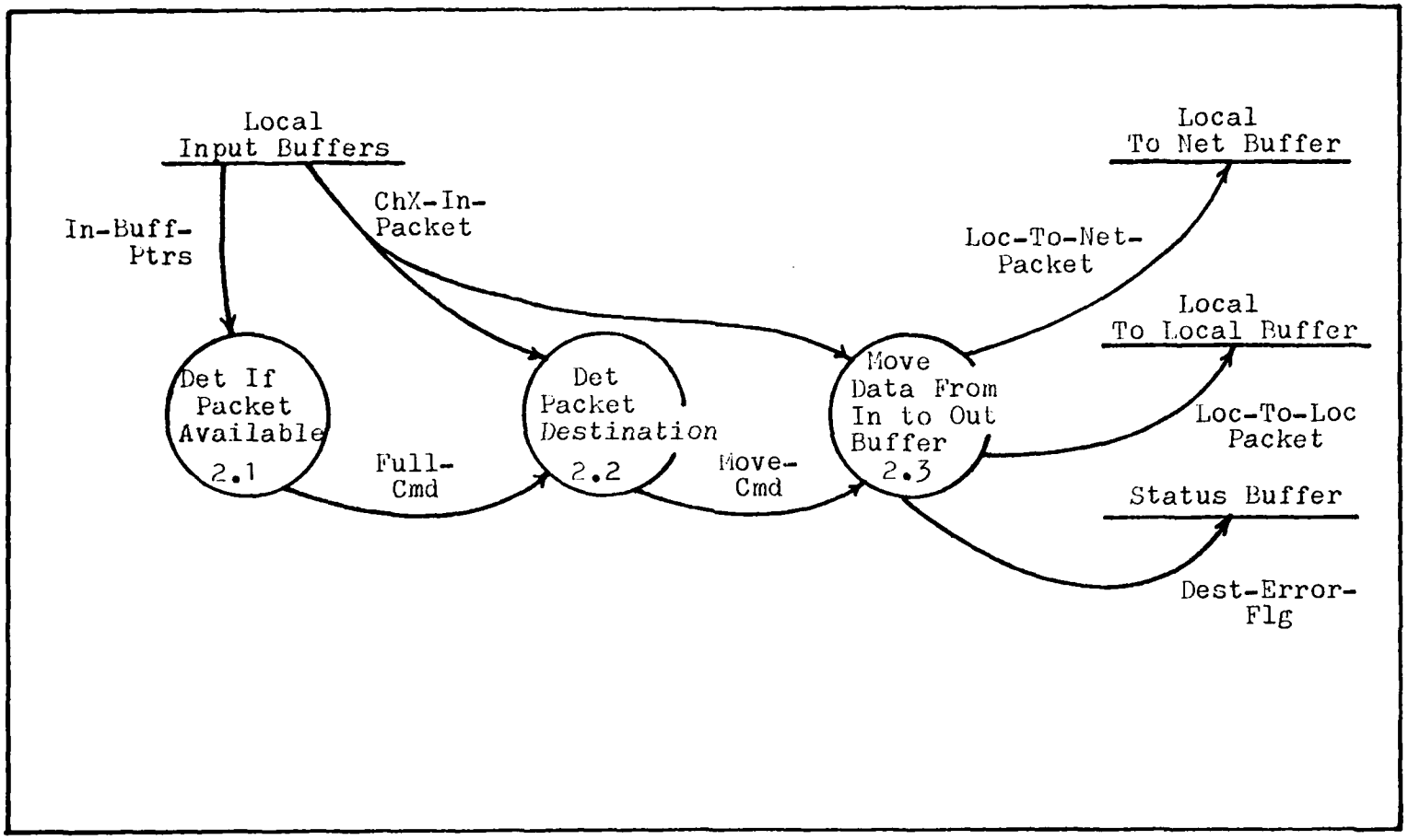


Figure 10. L.OS DFD Level 2

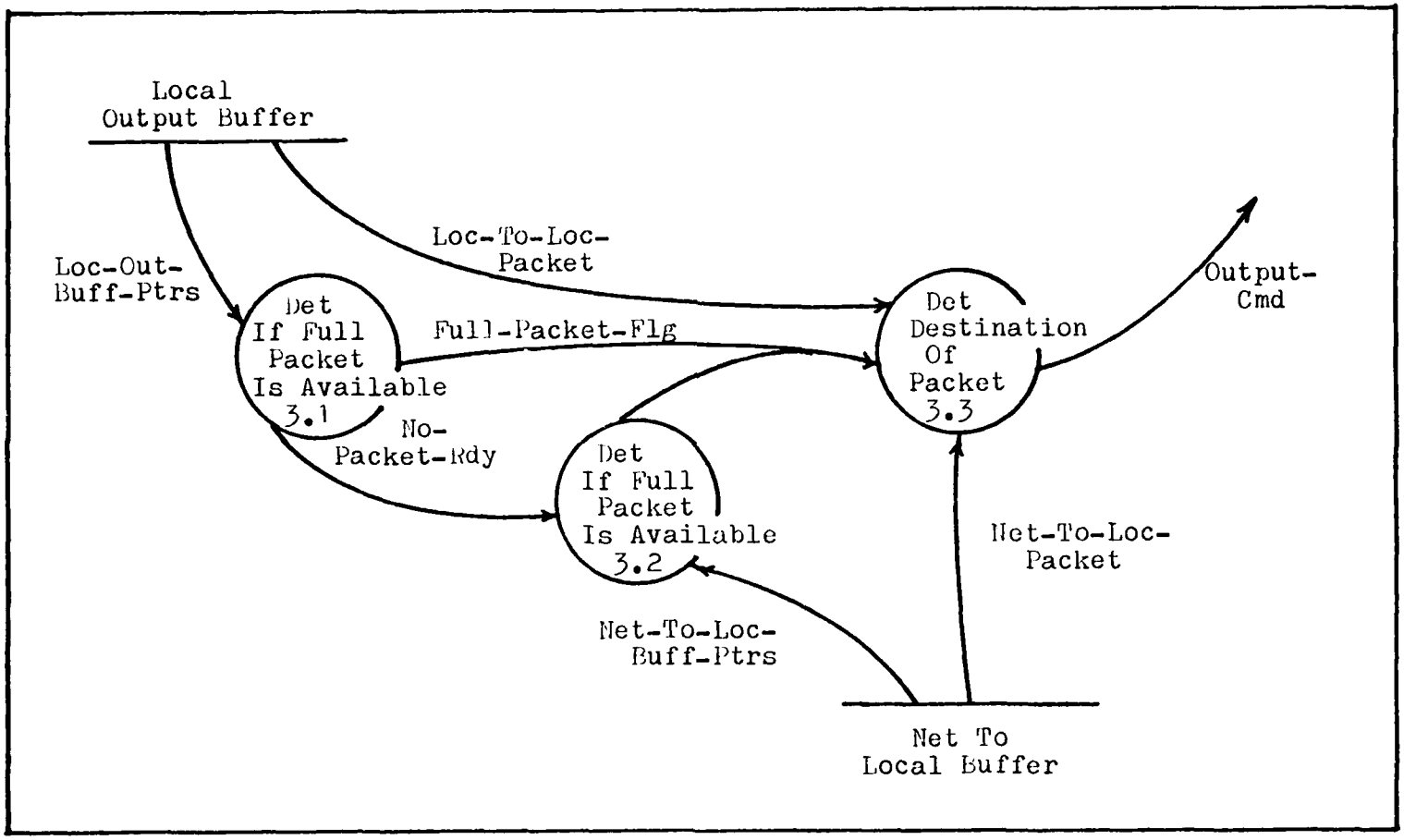


Figure 11. L.OS DFD Level 3

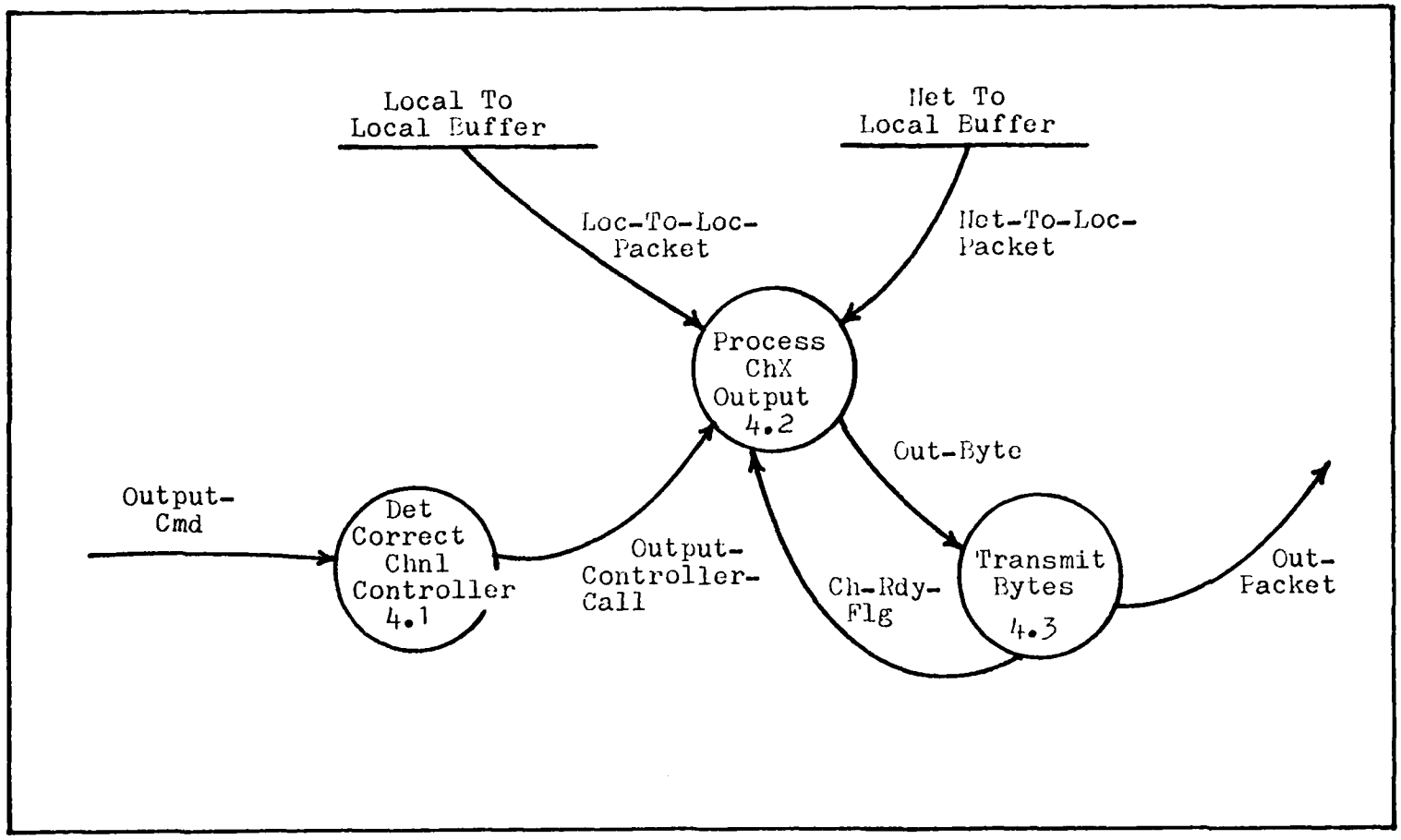


Figure 12. L.OS DFD Level 4

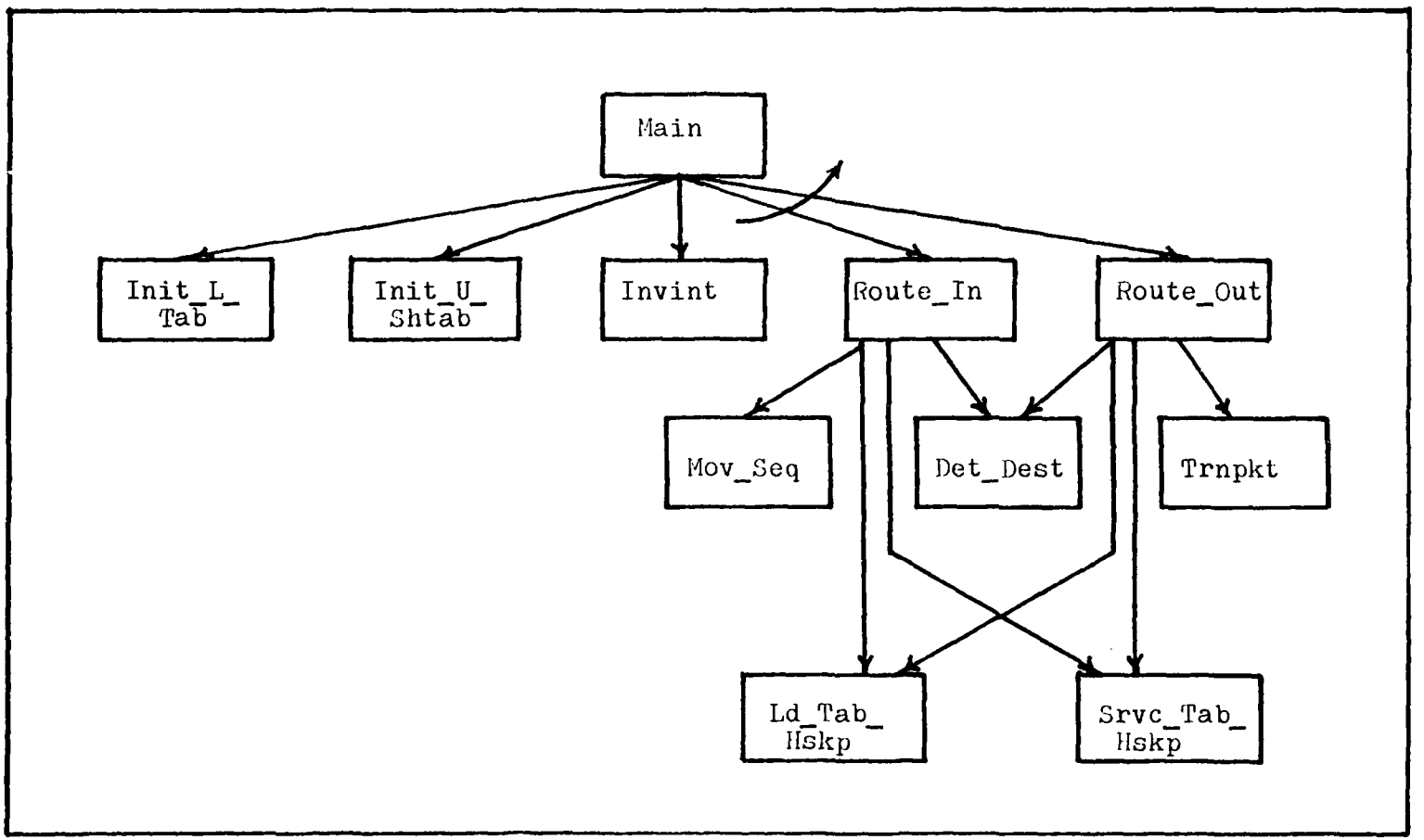


Figure 13. L.OS Structure Chart

Additionally, the packet size was set for a nominal 30 bytes instead of the 128 byte X.25 standard to facilitate testing. This organization was successfully tested using ADM-3 terminals for hosts. Four channel receipt and transmission, buffer table flow, and network buffer routing was verified as operative. Appendix B contains specific information concerning the local operating system.

Network Operating System

The network operating system (N.OS) is similar to the L.OS in operation. A typical packet could flow from the network channel to an input buffer. There it would be examined for destination either to a local channel or back out to the net. If sent to a local channel, the L.OS would accept the packet through the shared memory tables and process it. If sent to the net, it would be queued for transmission back to the network. Packets could also "originate" from the local side in which case they would be routed to the network.

Like the L.OS, network I/O must be interrupt driven at the byte level with data flow at the packet level, while network transmission may be polled. Unlike the L.OS, network transmission must be framed according to X.25 level 2 protocol. This additional duty requires header and trailer handling for frame identification (Ref 9).

Once again, this analysis was translated into the DFD's shown in Figures 14-17. Figure 14 is the overview diagram and each transform bubble is expanded for detail in the

following three figures. Using the DFD's, a structure chart for the N.OS was developed as shown in Figure 18. Main is the controlling driver with a one time call to N_TAB_INIT and INSIO (an assembly support module) for initialization, and and endless loop of calls to ROUTE_IN and ROUTE_OUT. Only ROUTE_IN calls DET_DEST for determining the destination of the packet. ROUTE_IN uses MOVSEQ (an assembly library routine) to move a packet in memory, while ROUTE_OUT uses TRNMIT to output a packet on the network. Procedures LD_TAB_HSKP and SRVC_TAB_HSKP are used to housekeep the buffer tables after a packet load or service (removal).

This organization was implemented in a PLZ environment with skeleton destination processing as noted in the L.OS discussion. This organization was sucessfully tested for network routing and buffer table flow. Appendix C contains specific information concerning the network operating system.

Implementation Notes

During the implementation of the L.OS and N.OS, two problems were encountered. First, and by far most significant, were PLZ compiler/linker failures. The initial problem was noted when PLINK failed to complete a link and terminated with a console output of "m?". This output appeared to indicate an undefined external reference. The source of the problem was the use of two constant definitions. Using constant defined parameters for table definition, a table size was being computed as a

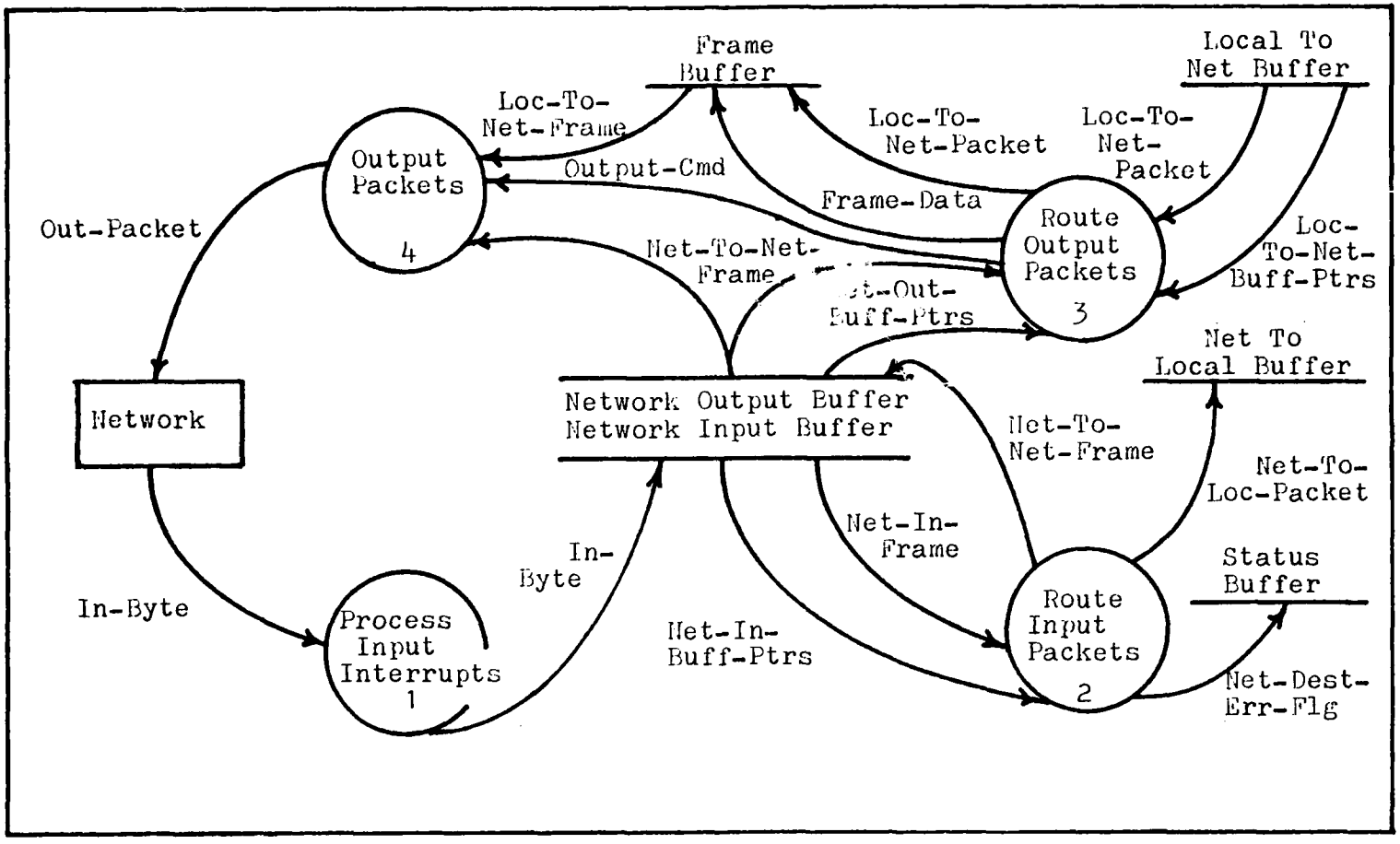


Figure 14. N.OS DFD Level 0

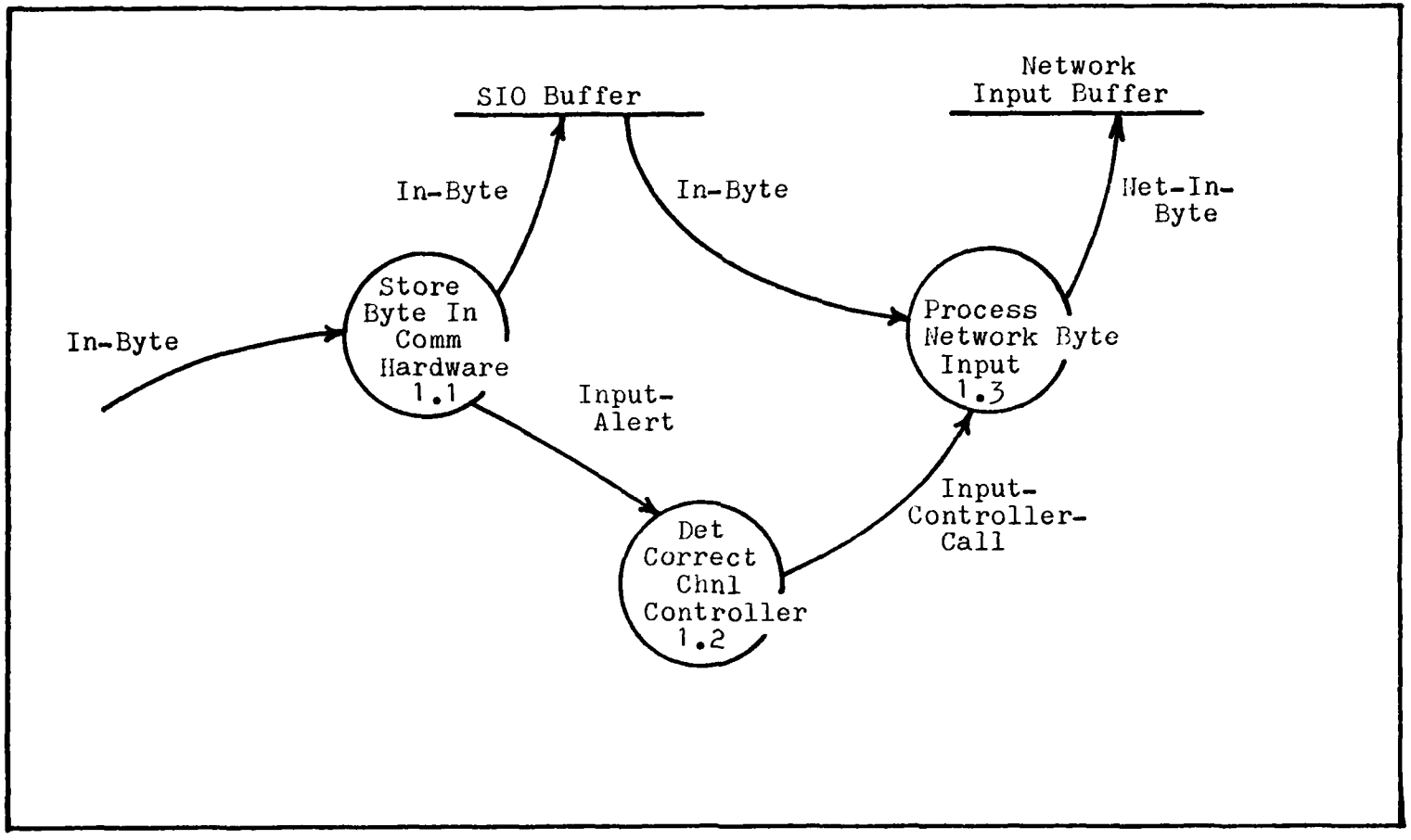


Figure 15. N.OS DFD Level 1

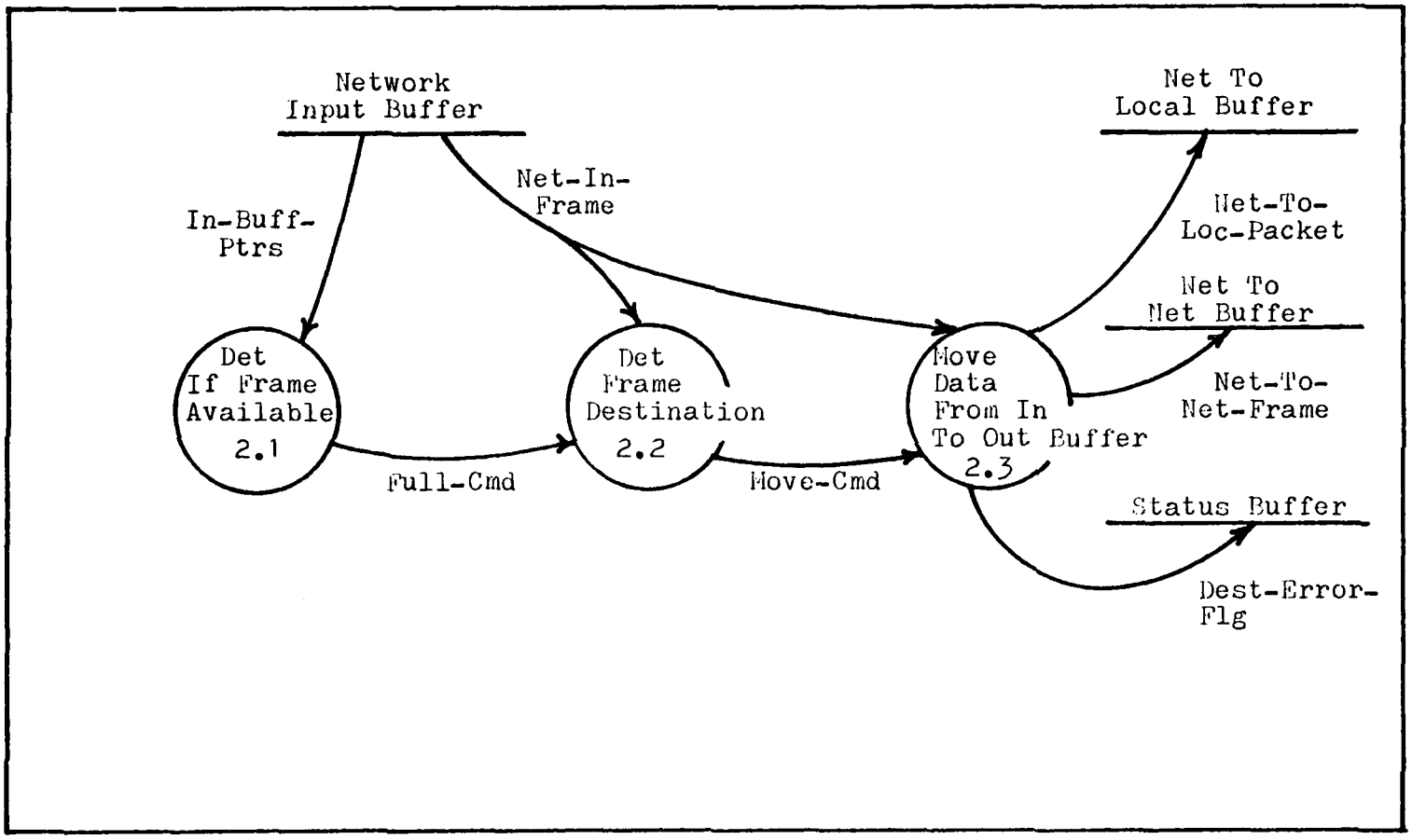


Figure 16. N.OS DFD Level 2

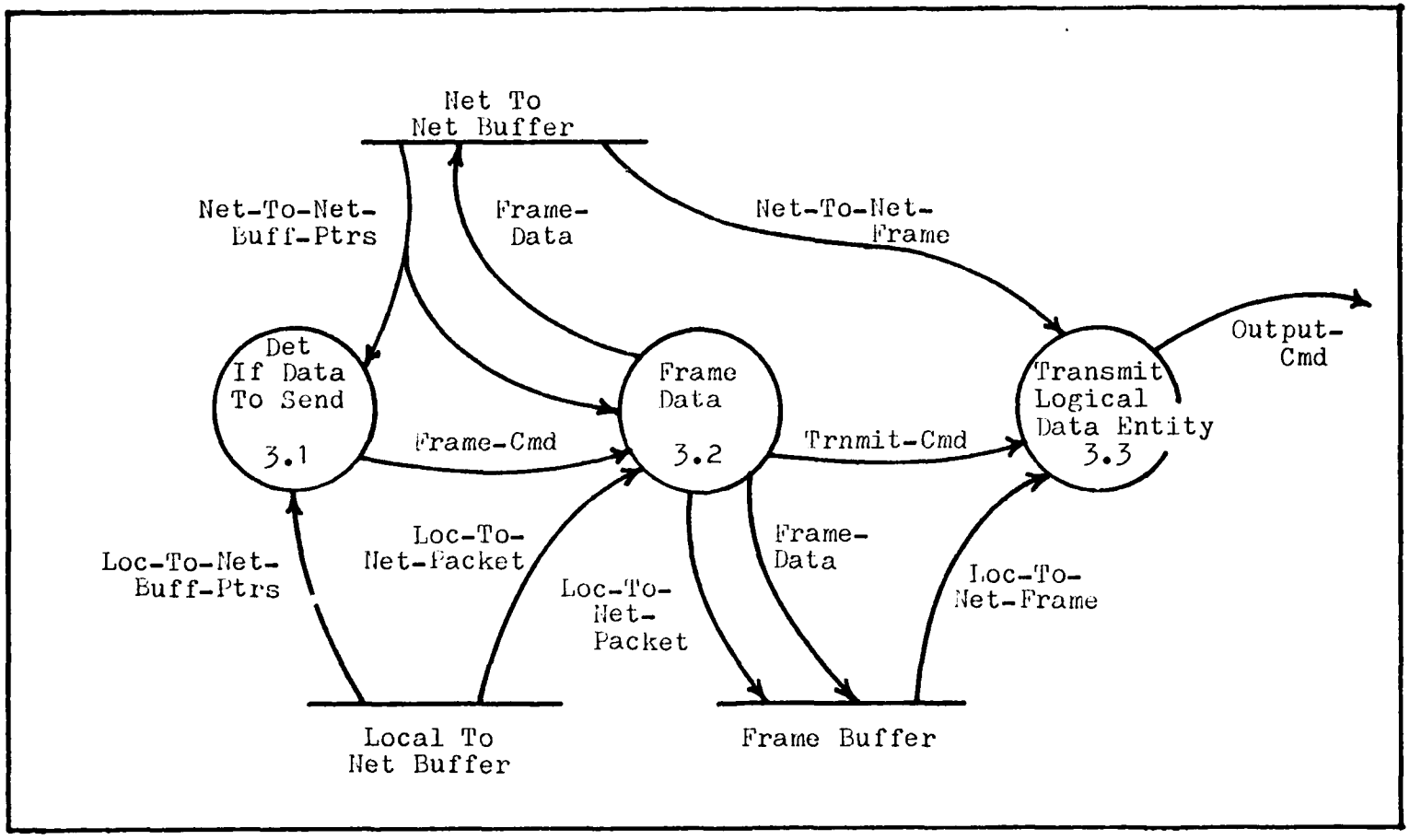


Figure 17. N.OS DFD Level 3

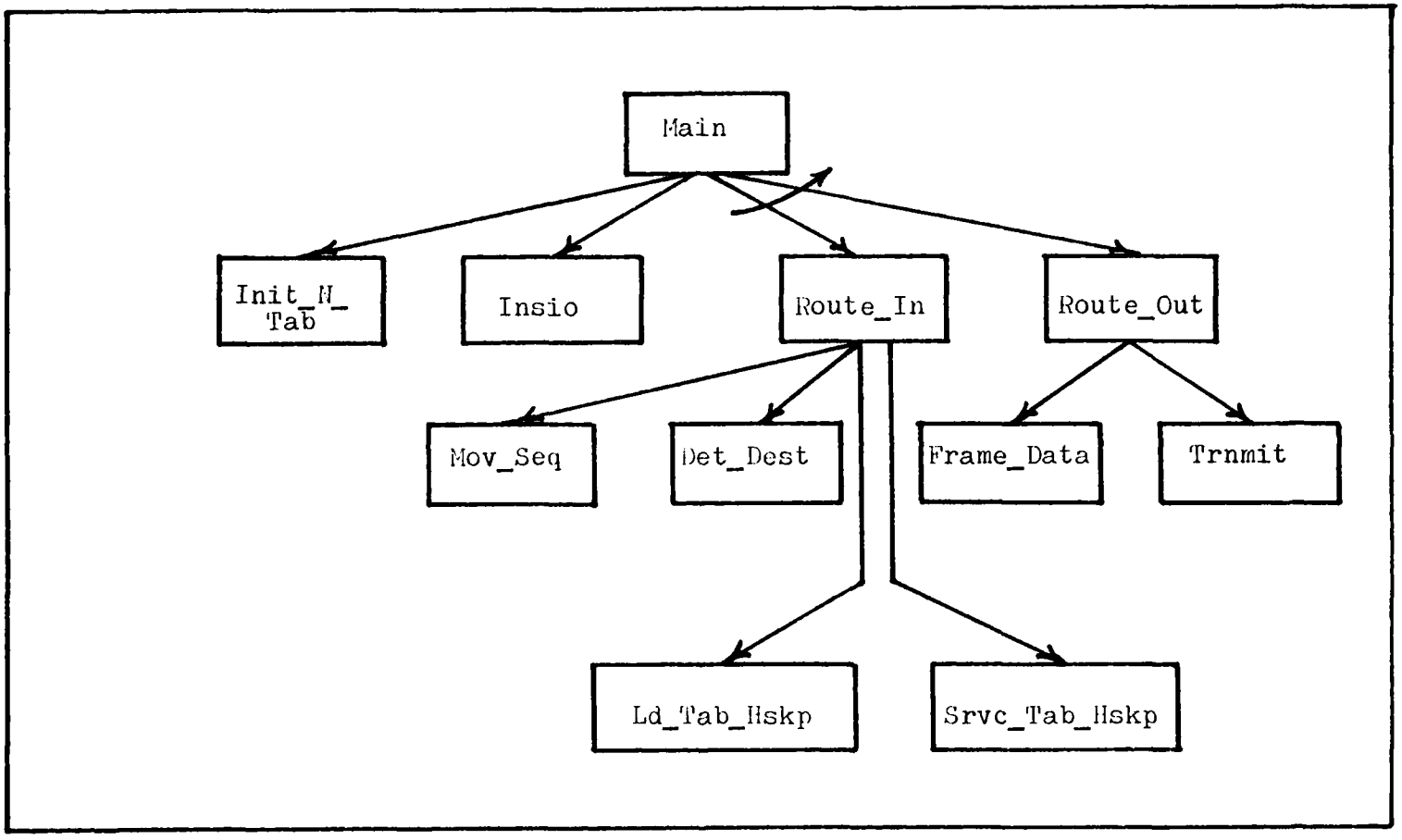


Figure 18. N.OS Structure Chart

constant-times-a-constant number of bytes. By creating a new constant, performing the multiplication under the constant definition, and using the new constant in the table definition, the problem was avoided. PLZ apparently does not allow arithmetic actions on two or more constants within a table definition.

The second problem noted was the incorrect generation of a entry point address after the link process. The symptom was a developed entry point indicating a higher address than what was correct. This problem gave no indication other than incorrect execution of the resultant program. The source of the problem was found to be a table definition. By reducing the size of the table, the problem was avoided. The word "avoided" should be stressed as no permanent resolution of this problem was achieved and future encounters are likely.

In both of these cases, the problems appear to be compiler/linker failures. The constant definition problem may be a limitation of PLZ, but the compiler does not flag an error. The second error appears to be the transparent overwriting of an area of link-used data. Action has been taken to identify these problems to Zilog.

A second problem area surfaced with the shared memory implementation. It is quite easy for the processors to conflict with each other's memory or with the EPROM's stack location. The results of this conflict are unpredictable and often misleading. It is recommended that careful

attention be paid to the EPROM stack area and the PLINK maps as reflected in L.OS.MAP and N.OS.MAP.

Finally, an error noted in table implementation pointed out a design opportunity. Initialization of the buffer tables was originally in the "main" module of the operating systems. By splitting the initialization process and moving the code to the "data-only" modules where the tables are actually defined, two advantages can be gained. First, concern for module temporal cohesion is lessened. A single large initialization process is eliminated as each buffer definition module will contain its own initialization action. Second, and most important, definitions will reside closely with the initialization process. This increase in visibility for initialization will serve to clarify future software maintenance action.

Summary

The data flow of UNID processing is one of packet concentration or distribution. The UNID has five sources of data (four local and one network). This flow suggested a single buffer organization with minimum pointer requirements. Both the local and network operating system structures were developed through data flow diagram analysis and transformation into module structure charts. Both operating systems require buffer and communication initialization and loop endlessly by routing data in and out. Simplification of routing and certain packet qualities was required to enable this investigation's progress. In

each case of simplification, a generalized application technique or modular interface was used to enable easy expansion to a full capability in the future. With the completion of this area of development, both operating systems were verified operative as designed.

VIII. Host Environment

Introduction

The purpose of this chapter is to present an analysis of the host environment associated with the DELNET. Hobart presented the first global requirement's picture with detail at protocol level three and below (Ref 10). The weakness in that picture rests with functional definition and clarity. For example, his table 6 is a hierarchy for "host-to-host" processes (Ref 10: 43). This table, while relatively complete from the viewpoint taken by the author, contains nearly fifty actions spanning four layers of the ISO model. The application of the ISO model can reduce this very complex, time-ordered sequence, to a series of service layers enabling both a clearer operational concept and a logical development sequence to emerge. This chapter reevaluates the host environment in terms of the model, and provides an outline for future implementation.

Communication Interface

Before the host processes can be addressed, a communication requirement must be satisfied. Any computer system selected as a DELNET host must be capable of providing communications interface at two ports. The first port is the operator interface for interaction with the user. The assumption is that the host will use its standard operator console to perform this task. The second port is

required for interface to the network through the UNID. A standard serial RS-232 connection at 19.2 Kbaud is specified, although lower speeds are software selectable at the UNID, and may be sufficient for selected operations.

The design importance of these two communications requirements rests with their operation in real-time. With the DELNET's distributed network control and the random nature of user input, the host computer must respond in a timely fashion to I/O on both ports. Communication processing can be infinitely complex and varied from system to system. Similar I/O schemes are generally available on the systems proposed for the DELNET, and a standard recommendation was achieved through the following evaluation of three techniques: block I/O, sleep/wake, and interrupt I/O.

Block I/O. A variety of block input/output mechanisms exist and are generally language specific. The technique enables an I/O process to be started while other processes continue in parallel. This is similar to a common forked mechanism. The parallel action must return periodically to check a "busy" flag to identify the completion of the I/O activity. With this capability, the input of a network packet could be started ahead of its arrival and host processing could continue until the busy indicator noted a completion. An example of this type of technique is the UNITREAD/WRITE functions available with UCSD PASCAL (Ref 21).

While effective in a great many applications, this technique would be unsatisfactory in the DELNET environment. The major problem is its inabliity to deal with the full range of random events. For example, a lengthy host process could delay the busy flag check until another input packet started to arrive. With no input process operative, a portion of the packet would be lost. This situation could be relieved with physical level control (auxiliary use of modem controls, for instance), but this is an additional protocol effort to design, develop, and maintain. Sequencing events is another solution, but this leads to a logical layer of protocol that restricts the freedom and independence of host operations. While solutions are available, their long term side effects suggest block I/O avoidance.

Sleep/Wakeup. A formalized extension of the sequencing possibility is a sleep/wakeup technique. With this idea, certain host processes are restricted from time to time. For example, a host that is awaiting a certain response may have its communication resricted to only that type of response. This data block pending action can be managed at a variety of levels. Tanenbaum suggests an EnableHost/DisableHost possibility to be managed by the network node (Ref 20: 324-385). X.25 control packets could achieve the same option at the network layer for control by transport services.

The major deficiency with this option rests with the

additional protocol requirement, the artificial sequencing nature of the process, and the restrictive effect on the asynchronous host processes. Additionally, this artificial layer adds to the maintenance burden with its distributed residence in both the host and node environments.

Interrupt I/O. Most computers have procedures to allow the central processor to be interrupted for priority events. The events may range from emergency diagnostic conditions to simple system services. The service of I/O devices is generally included among these options. With interrupt I/O, software service routines are identified and executed when the indicated interrupt occurs. The program executing will interrupt at a convenient instruction boundary, the service routine called and executed, and the interrupted program continued. These I/O handlers can perform short, time-critical I/O functions while leaving the central program free from the encumbrance of I/O processing.

Interrupt driven I/O is advantageous on several counts. First, it is reasonably simple to use. The complexities of interrupt operations vary from system to system. At one end of the spectrum is the micro computer where a single command alerts the processor to branching requirements for interrupts. The mini or main frame computer may have a sophisticated interlacing of system services. In either case, once interrupt conditions are established, the I/O routines appear transparent to the

using software. The operating software "sees" input only after it has arrived and "knows" its output will leave without real-time impact.

Second, full asynchronous operations are enabled at the host. Operator console instructions, multiple host connect operations, network operating system functions, and DELNET application processes can all occur "concurrently" as each process maintains its own sequence of events without concern for the overall I/O process. This freedom of operations places additional burdens on host processes for error-free action. But with the ISO model, error-free specialization by the protocol layers is the cornerstone of design. Each layer provides requests of the lower adjacent layer and error-free service to the higher layer. Interrupt I/O frees the layers from the burden of an artificial protocol to maintain the I/O processes.

These advantages, coupled with the general availability of interrupt I/O, lead to the conclusion that the DELNET hosts must have, at a minimum, a receive I/O interrupt capability. Data transmission could also be interrupt driven, but since packet output will always be a known event capable of being interrupted itself, interrupt driven output transmission is not required. In fact, to lessen the I/O system interface at the host, it may be simpler to proceed with polled transmission.

Network Layer

The function of the network layer is to manage the flow

of packets. In chapter three, this layer was specified as X.25. In the following discussion, the specifics of that recommendation are identified and an implementation scheme is suggested.

CCITT Terminology. X.25 is a provisional recommendation for interface between Data Terminal Equipment (DTE) and Data Circuit-Terminating Equipment (DCE) for systems operating in packet mode (Ref 9: 243-284). DTE refers to a host computer while DCE generally refers to a type of carrier equipment, for example, a modem. Hobart, in order to simplify the application of X.25 to DELNET topology, allowed DCE to mean node or UNID (Ref 10: 46-48). Unfortunately, CCITT departs from that alignment and refers to a node as Data Switching Exchange (DSE) equipment. While not a major problem, this confusing terminology can be minimized by the recognition that the UNID serves functionally as both a DCE and a DSE. The reader should be aware that future references will follow Hobart's lead and refer to the UNID as a DCE while assuming its dual role.

X.25 Specification. The network layer of X.25 is the subject of lengthy discussion in current literature (Ref 6, 12, 20, 34). Basically, it consists of control and data packet definitions with procedure and flow requirements. The packets include fields for the variety of control information required for flow control including packet identification, logical channel assignment, and sequence identification. An endless variety of possibilities exist,

but a typical application operation would include: a Call Request packet sent from a calling host to a called host; a Call Accepted packet returned with agreement on facilities and logical channel number; data exchange via sequenced data packets providing the necessary application information (this process may be controlled with control packets such as Interrupt, Receive Ready/Not Ready, Restart, etc.); and with application completion, a Clear Request and Clear Confirmation to terminate the virtual circuit. Table II presents a list of typical X.25 packets and their definitions.

The detailed implementation of X.25 is well beyond the scope of this study, and will not be persued beyond DELNET design considerations. It is important to recognize that the network layer of X.25 is a service providing link in the ISO model chain. With the structural confidence of the model, development of the network layer can proceed oblivious to operational concerns outside its boundary. This functional delineation lessens the complexity of the job and provides for development clarity.

Network Function Alignment. In the case of the DELNET, the first concern with the network layer is functional alignment. The question of where to locate elements of the layer can be addressed by refining the role definitions of the host and the UNID. As was indicated in Figure 3, the network layer spans the physical boundary of both the host and the UNID. The question exists as to the

TABLE II

TYPICAL X.25 PACKET TYPES

Data	- Data packet
Call Request	- Packet sent by calling DTE to called DTE indicating a request for connection
Call Accept	- Packet sent by called DTE to calling DTE indicating acceptance of a connection
Clear Request	- Request to disconnect and free a logical channel interface
Clear Confirm	- Acknowledgement of a request to disconnect
Interrupt	- High priority packet
Interrupt Confirm	- High priority packet acknowledgement
Receive Ready	- Packet used to indicate willingness to accept n packets within a window
Receive Not Ready	- Packet used to indicate temporary inability to accept packets
Reset Request	- Packet used to reinitialize a virtual call (logical channel)
Reset Confirm	- Acknowledgement of reset
Restart Request	- Packet used to reinitialize all virtual calls (logical channels)
Restart Confirm	- Acknowledgement of restart

job mix of the functions assigned to either the host or UNID.

With the UNID, as each new processing requirement is levied, it departs from the pure concentrator/distributor role and becomes an extension of the host. The advantages of this approach are primarily in the maintenance area. With host specific interfacing capable of removing protocol requirements from the host, the host environment becomes less complicated. On the other hand, each new responsibility absorbed by the UNID has a quadruple impact since each new function must be operative at all four local channels. While some generalization may lessen the impact, absorbing host operations and combining channel processes will lessen future flexibility and create a less desirable design.

Additional concern may be felt from the limitations of the UNID. While 32K of system memory and 32K of shared memory appear adequate for the proposed role of the UNID, memory and processing capacity have not been operationally demonstrated. This unknown warrants a conservative approach.

Finally, the maintenance problem with the hosts may be lessened with the realization that language commonality will be prevalent at least with the initial implementation. PASCAL, in particular, is available on the VAX-11/780, LSI-11, and Eclipse. While problems of consistency are sure to exist between systems, this language is currently

available on more DEL computer systems than two UNID's can support.

All of this suggests a minimized role for the UNID. With that in mind, the primary job for the UNID at the network level will remain packet routing with implied network status communications at the UNID-host and UNID-NOS level. Accomplishment of this task must begin at the UNID with routing/status information available to both the local and network processors. The shared memory capability enables a single copy of this information to be available to the processors. Specific details are left to the implementor, but with the data stored containing both routing and status, simplified LOGIN and security procedures could be developed. As each new host is logged in, the UNID's status could be updated by broadcast control packets from the NOS. A typical process would include a LOGIN packet from host to UNID; UNID routing of the packet to the NOS host; authorization processing by the NOS; and broadcast control packet generation from NOS to all UNID's identifying the new network participant. Without the host indicated as operative by the NOS, the routing/status information would not be updated to include the host, and the UNID would be enabled to forward LOGIN packets only, thus restricting network access until NOS authorization.

The remainder of the network layer must be incorporated at the host. The recommendation is for a series of procedures capable of packet recognition, formation, and

transfer matching the services outlines in Table II. The implementor should be careful to retain the service philosophy of the ISO model. By avoiding the complications of interaction, the network services can be combined at the transport layer to form the decision-making, sequencing, and timing macro operations requires for applications.

Transport Layer

Three major reasons exist for a transport layer in the DELNET. First, as was indicated in the previous discussion, the network layer will be a better constructed product if developed as discrete services. As such, their individual functions are limited and must be combined to form logical processes. The transport layer can fill the role with formation of macro functions to meet the needs of higher applications.

The second reason stems from the X.25 requirement. X.25 was devised for communications systems. As such, it has some weaknesses in terms of virtual addressing. An example of a weakness is the service provided for RESET and RESTART. Tanenbaum notes that the network can send either of these network commands to a host at will (Ref 20: 324-326). After a RESET, the host is left without information concerning the status of packets still outstanding. Recovery must be accomplished at a higher level. The transport layer can guarantee data is recovered and not duplicated.

The third reason for the transport layer deals with

specific DELNET addressing. The X.25 addressing scheme is quite complex, providing for four bits of group and eight bits of channel addressing for both permanent and virtual address assignments. The design capabilities are well beyond the DELNET's potential use. The opportunity exists to shift addressing out of the complicated network layer into a simpler scheme managed by the transport layer. There is no ISO concept violation with this function shift and the transport addressing may still use the X.25 structure, but in a limited fashion.

Transport Services. As can be expected, a wide variety of transport services can be found in current applications. Few systems are alike as each fine-tunes its protocol to the expected applications. In the global scale arena, ARPANET blurs the ISO layer definitions. Its original concept contained node-to-node control working in concert with host-to-host protocol. Today, the system has not one, but two transport protocols: the original network control protocol (NCP), and the newer transmission control protocol (TCP) (Ref 20).

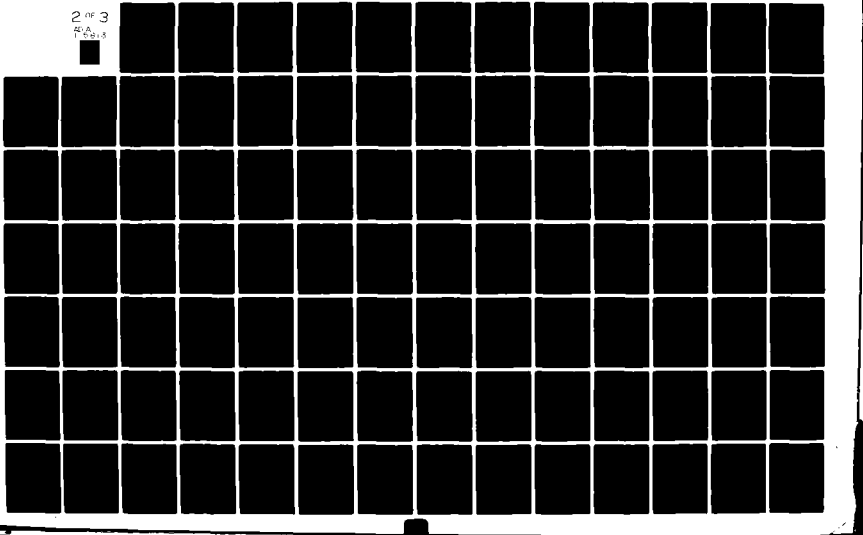
IBM's System Network Architecture (SNA) is another example of an evolutionary system. Originally conceived with limited distributed control, the system has seen a variety of changes since its inception in 1974. Its parallel to the ISO transport layer is divided between path control and transmission control facilities.

DECNET was one of the initial commercial products

AD-A115 613 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 9/2
DEVELOPMENT OF THE DIGITAL ENGINEERING LABORATORY COMPUTER NETW--ETC(U)
DEC 81 J W GEIST

UNCLASSIFIED AFIT/GCS/EE/81D-8 NL

2 OF 3
AD A 115 613



provided to enable private networks. Like the previous examples, this Digital Equipment Corporation (DEC) product has evolved. As with SNA, the evolution has moved in the direction of distributed control with nodes and hosts operating more independently. DECNET's transport layer, as well as the lower three layers, is identical with the ISO model. Departure from ISO appears above the transport protocol with no session layer, and a large application layer absorbing ISO levels six and seven.

DECNET can provide an example of transport commands. DEC has five transmission subsystem commands available to DECNET users: Connect, Disconnect, Receive, Transmit, and Transmit Interrupt. Additionally, these commands can be used with different levels of transparency, requiring more or less system knowledge as the application warrants.

Tanenbaum provides another suggested transport command set very similar to DECNET. His services include: Connect, Listen, Close, Send, and Receive (Ref 20). These commands were developed with emphasis on the ISO model, and in particular, X.25 applications.

Both of these examples provide a good basis for DELNET applications. The transport layer must provide the ability to: connect and disconnect with a distant host; send and receive data; listen for connect operations; and transmit high priority control data. The recommendations proposed to fill these needs stem from the previous two examples and are listed in Table III.

TABLE III

TYPICAL TRANSPORT SERVICES

Channel = CONNECT (From, To)

This service accepts two inputs, the calling and called hosts, and returns a logical channel or status for control.

Status = DISCONNECT (Channel)

This service accepts the logical channel as input and returns a status after disconnection.

Status = SEND (Channel, Buffer, Bytes)

This service sends a number of bytes from the buffer specified, over the channel identified. The status is used for control.

Status = RECEIVE (Channel, Buffer, Bytes)

This service receives a number of bytes into the buffer specified, over the channel identified. The status is for control.

Status = LISTEN (Channel)

This service is an option providing for a host's suspension pending communications on a specified channel.

Status = INTERRUPT (Channel, Type)

This service enables priority interrupt action over the specified channel.

Each of these services can be implemented as HOL functions with passed parameters and returned status/data. As such, layer interface can be well defined by the parameter values. The network layer must manage the input specified, and return the output. The application layers must accept the output, and process accordingly. Changes within a protocol layer must simply meet the highly visible parameter specifications. This mechanism provides for a clearly drawn interface that may be sufficiently documented through program documentation rather than an artificial layer of interface control documents.

Application Layers

As can be expected, the closer to network applications, the more blurred become the layer distinctions. The ISO model proposes three layers above the transport services: session, presentation, and application. These layers reflect a functional separation based more on structural and maintenance considerations than with logic boundries. Current network applications reflect a curious mix of ideas, more often with the session layer missing and a single application layer dealing with presentation concerns. The DELNET can start small, while maintaining a corporate perspective to the ISO model.

Session Layer. Session protocol refers to an extension of the connection process of the transport layer. With this layer, each host may negotiate the services to be provided during the session or connection from host to host. This

binding of options can manage host dialogue at a higher level than transport while relieving the application software from concern for connection details. Initial DELNET operations do not indicate a need for this service since each host will be required to place its entire inventory of services at the disposal of the network. Nevertheless, this layer offers potential considerations for future development, both in terms of physical as well as logical system entities. With multiple processes concurrently operating on different logical channels, a single host may want to selectively bind a process to the required elements prior to process initiation. This classic deadlock potential can be addressed through the session layer, and must be considered once the DELNET advances towards more sophisticated distributed functions.

Presentation Layer. The presentation layer also lacks specific definition. Its purpose is to provide common services to the application layer. Tanenbaum identifies a number of services including: text compression, character code conversion, file format conversion, and virtual terminal processing (Ref 20). This catch-all of services has limited application with the initial DELNET operation, but is potentially valuable. The key to its use depends upon host applications. With HOL use, a core library of presentation services can be developed and linked as appropriate for the host. Data conversion processes and virtual terminal applications offer the most immediate

possibilities. With these services, it can be seen that the goal of the presentation layer is not only development ease, but system transparency. Whether it is a transparent file transfer of ASCII data from a 16-bit computer to an 8-bit micro, or the consistent operator interface among all DELNET participants, the presentation layer can provide a discrete, manageable environment to meet those demands.

Application Layer. The final layer is application - the layer that interfaces with the user for the desired function. At this layer resides the specific applications to be available to the network. The initial DELNET applications include: NOS functions, message generation, remote job entry, and file transfer. Leaving the NOS aside for right now, the remaining applications can be achieved in an infinite variety of ways. It is this flexibility, and the accompanying fine tuning of the network to specific needs, that contributes heavily to the uniqueness between networks.

The most complete approach to an application returns to the protocol concept. With any application, the desired function or command can be thought of as another entity in the protocol chain. For example, a remote job entry protocol could be established on the DELNET providing commands for job submission, transfer, and output direction or retrieval. Day provides a good overview of this concept with a presentation of both RJE and file transfer concerns (Ref 12: 78-116).

While the application concept should be kept discrete, its implementation lends itself to a simple sequence of operator interface, data transfer, operation or command file creation, execution, and output routing. Application interface with the user can range from pure transparency of services associated with distributed processing, to a detailed dialogue of system specifics. While transparency is a fine goal, the costs of that quality in terms of overhead are significant. On the other hand, requiring DELNET users on the VAX-11/780 to know LSI-11 operation details is absurd. The initial list of applications, plus the quality of the expected user, indicate a compromise with a bias towards providing a practical, usable tool. Simple, interactive dialogue, similar to Hobart's demonstration software, can specify elements and variables sufficient to create a valid DELNET operational environment.

With data transfer handled by the lower level protocol, the application layer is left to build and execute host computer command files. Use of standard host system services can provide the core of application executions. For example, HELP commands can be accomplished by file requests from the NOS. Once obtained, this data may be simply output to the operator console as optionally formatted by the presentation layer. Standard system services such as Copy, Type, or Print can provide immediate, nearly maintenance-free delivery of HELP or message functions to the operator.

Remote job entry can be accomplished in a similar fashion. Most systems enable command file execution. With these files, command language operatives are built within the file and the entire file is submitted for execution. File building, data transfer, and formatting, followed by system execution can provide remote job execution. It should be obvious that while these processes are simplistic in nature, their operation within the network will require not only interface with the host systems to guarantee correct operation, but protocol overhead to guarantee that the network will not degrade given process failure. It is that aspect that is critical and is dependent upon the quality of the subordinate layers of protocol.

A note of caution is also required concerning dependence on host system services. While most standard services are reasonably stable, they do change from time to time. Emphasis on a well defined and highly visible interface technique is a must. Priority must be given to the potential for change at the host and the consequences of those changes. A simple command format change at one host could disable that host, and potentially the network, if the NOS is present and not portable.

This illustrates the design concerns for quality. The DELNET will be able to survive initially with incomplete protocols. As the applications advance and the picture grows more complicated, the performance of the network will grow more dependent upon the lower level groundwork.

Without a global viewpoint for quality at all layers, the network will degrade radically as applications are heaped on an unsound foundation.

Network Operating System. Departing from pure host concerns, the concept of a NOS stems from the need for a central control to coordinate network operations. In a distributed design like the DELNET, the NOS functions should be minimized. Specifically, they should include such corporate duties as network user authorization, security, and status monitoring; as well as centralized processes such as help commands. The driving philosophy is simplicity. The NOS should be assigned duties that will require central control or can be better maintained as a single network entity.

Hobart envisioned a heavier used NOS. For example, his design recommended file transfer operations deal through the NOS with a series of file manipulation commmands at the NOS. While there may be some software savings with the "single copy" emphasis, this approach restricts host freedom, blurs protocol layers, and artificially increases communication traffic to the NOS host. By allowing the hosts to manage their own affairs once they are operating in the net, the NOS is left with few network services. This serves to advance free flow host to host interaction, and lessen the overhead burden of the NOS host.

With a small series of NOS functions, an additional benefit can be gained through an appended host design. The

NOS can be effected through process calls similar to any host process. With a core host design as a basis, the NOS could be appended to a host with the insertion of command packet recognition software, and the linking of the appropriate service packages. The advantage to this approach is twofold. First, there is a consistency of modularity among processes. The NOS functions would be packaged, operated, and maintained in an identical fashion as the normal host software. Conceptually, it would exist as another element of application protocol. Second, the NOS functions would be portable. Within the hosts using the same HOL, moving the NOS would be a matter of software insertion and module linking at the host, and network adjustment of the NOS address if not transparently designed. This is a key addition to DELNET flexibility, substantially reducing NOS dependency on specific hardware.

Design Alternatives. During the investigation of the role definition for the UNID, a design alternative for the DELNET was encountered. As was mentioned previously, the UNID is primarily tasked as a communications switch. This assignment is due to the concern for both functional simplicity and future maintainability. Specific concerns were felt over positioning substantial processes at the UNID because of the required application at all host ports. However, a variation of this idea could enable some unique DELNET operations not envisioned in the original concept.

The option is a variation of a Packet

Assembler/Disassembler (PAD). The purpose of a PAD is to provide a mini host interface to a network. There is some controversy concerning the definition and conceptual use of a PAD. The ARPANET TIP qualifies as a PAD, but purists rely on the CCITT definition that considers the PAD to be an interface between two DTE's (Ref 6).

Ignoring the definitional problems, a PAD concept could provide the DELNET with some interesting alternatives. For example, designing a PAD to be resident in the UNID and functional at one port, could allow specialized operations at that port. By shifting a subset of the host overhead into the PAD, the UNID port could directly interface with entities lacking the power of a host computer. Simple operator interface consoles, graphic scopes, mass storage devices, or network gateways are potential services for the PAD. With this opportunity, entire computer systems need not be tied to the network to gain access to a device. With the myriad of devices available in the DEL, this option could increase operational possibilities and result in more effective use of equipment.

There are, however, costs involved with this concept. PAD's are generally primitive. Services provided will be limited by device interface and the software impact on the UNID. Each service provided by a PAD would be unique. Generalizational of PAD software would be limited by the unique operations of the chosen devices. Additionally, it may be difficult to implement a PAD completely within the

protocol framework outlined in this report. Conceptual differences and implementation imperatives may make the PAD a concept anomoly. This does not have to be the case, however, as CCITT standards exist for PAD development (Ref 9). If developed properly, the concept offers significant oportunities for additional network applications within the DEL.

Summary

This chapter presented an initial look at the host environment in terms of ISO model application. It presented alternatives for I/O interface, and concluded that an interrupt driven technique is the optimal choice. The network layer for the DELNET was examined for terminology clarification, function alignment with topology, and elementary host structure. Transport layer motivation was discussed, along with evaluation and selection of transport services. The three ISO application layers were compared to both commercial and private examples. Layer applicability to DELNET operations was discussed, and a proposed role for each layer was established. A DELNET network operating system concept was prescribed with limited scope and a flexible, host-appended, design. Finally, a modified PAD design alternative was presented for consideration as a device interface mechanism for the network.

IX. Conclusions and Recommendations

The objective of this investigation was to advance the protocol development of the DELNET and demonstrate a network capability. While good progress was accomplished in the protocol area, operational implementation was limited by the initial state of the UNID and memory inconsistencies (Ref Papp). With the products of this report - a clarified protocol structure, a viable development methodology, and a sound node and host organization - coupled with a reliable UNID memory system, it is felt that operational results are achievable in the very near future.

Conclusions

The foremost conclusion of this investigation deals with the value of the ISO protocol model. Since encountering the model in the initial literature search stage of this investigation, it has grown in prominence, becoming a key element in newly released protocol texts and network article publications. This study confirms that the literature attention the model is currently receiving is warranted, and concludes that its use as the basis for DELNET development is paramount. It is not enough to follow structured practices as Hobart exhibited. Application of the model provides sound logic for process distinctions in the enormously complex environment of a computer network. System clarity, effective maintainability, and future

development ease, rest squarely on a quality application of the model.

A second conclusion is that the UNID is a viable network node for DELNET operations. The weakness noted in this investigation stemmed from its incomplete status prior to this report. Processor operation, memory size and organization, communication interface, and interactive monitor control, demonstrated an adequate performance for anticipated operations. The continuing problems with memory stability, while an operational deficiency, do not appear to be catastrophic or insolvable. The problem elimination, either through modification or redesign, would be a small investment compared to the operational potential of the UNID.

Extending the UNID conclusions to the area of software development, a qualified approval of the development facilities is a third conclusion. The MCZ-1/25 and its interface with the UNID, plus the variety of services offered by Zilog's PLZ software family, provide a robust environment for targeting software to the UNID. The qualification is a reflection of the concern for the two undocumented and highly troublesome software failures encountered with PLZ. Future development must be cognizant of the spurious behavior of PLZ in the past, and be prepared for a probable recurrence.

The final conclusion rests primarily with the design issues faced by the host environment. Only the one

technical issue of interrupt I/O remains to be demonstrated to enable full development to proceed. Within that development rests a number of goals including X.25 implementation, transport layer macro command definition, and application layer development. The conclusion drawn from this investigation is that these layers should be structurally developed to absorb full complexity, but implemented to meet the less complex DELNET operational needs and research objectives. Future efforts should provide global structure while minimizing complexity in favor of operational simplicity. The returns of an operational DELNET will be substantial providing both optimized equipment use, and advanced pedagogical applications in a local network environment. The operational parameters of a local network can enable less operational transparency, reduced network complexity, and lower maintenance costs, without degrading a global structure capable of supporting the most sophisticated applications and research.

Recommendations

From a technical viewpoint, the remaining detail to be demonstrated for network operation, is interrupt driven I/O. It is necessary to develop a dependable interrupt capability, including error recovery, at each potential host system. With the assembly language and system interface nature of this job, it is recommended that these processes be addressed as course project material or as a subordinate

element of a thesis project.

From a network viewpoint, recommendations for future work can be addressed via the protocol layers. The link level layer of the network can be provided largely through the hardware services of the SIO. Development of that area during this investigation was minimal, and the link layer (effectively SLDC protocol) may be addressed as class material or secondary thesis work. A note of caution should be made since this investigation (as well as other concurrent work) found the SIO to be remarkably complex and poorly documented. It would be a mistake to underestimate the complexity of this problem, particularly in the area of synchronous operations.

Implementation of X.25 at the network layer is a substantial problem. This investigation, as well as commercial costing estimates, indicate X.25 to be quite a challenge. With some modification of complexity as suggested in the conclusions, this area of development would provide an excellent thesis topic.

Additionally, the transport definition is ripe for development in parallel with initial application layer activity. The host development provides numerous opportunities for system integration and network structure work associated with protocols.

There is no standard recommendation concerning subject mix or sequence. Several of the areas can, and should, be developed concurrently. The restriction in sequence is

simply that subject packaging should be in adjacent areas of the ISO model, and that the lower levels should be accomplished first to enable higher application operations. With that in mind, the interrupt I/O and link level SIO work must be addressed initially, followed by progressively higher protocol layers. Coordination of development could enable a series of parallel projects to progress concurrently. Table IV summarizes the recommended action.

Regardless of the choice, the cornerstone of DELNET development must be quality. The nature of a layered environment, with each layer dependent on subordinate processes, requires complete and thorough development. Protocols are deceptively complex with extensive design, test, and implementation overhead associated with even the most modest objective. Future DELNET efforts must emphasize quality over quantity, and any development plan should reflect that goal.

TABLE IV

FOLLOW-ON RECOMMENDATIONS

A. Interrupt Input/Output

1. DELNET I/O Standard Evaluation
2. Host Specific Development
   a. User Software Package
   b. Error Recovery Package

B. Link Level Protocol

1. SIO Synchronous Implementation
2. UNID Operating Systems Augmentation
3. Network Communications Upgrade to Fiber Optic

C. X.25

1. DELNET Standard Services Design
2. PASCAL Implementation for VAX and LSI
3. UNID Operating Systems Augmentation

D. Transport Services

1. DELNET Standard Services Design
2. PASCAL Implementation for VAX and LSI

E. Application Services

1. DELNET Standard Services Design
   a. Host Structure
   b. NOS Function
   c. Remote Job Entry
   d. File Transfer
2. PASCAL Implementation for VAX and LSI

Bibliography

1. 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.

2. Barden, William Jr. The Z-80 Microcomputer Handbook. Indianapolis: Howard W. Sams and Co., Inc., 1979.

3. Berglund, Ralph G. "Comparing Network Architectures," Datamation, 24(2): 79-85 (Feb 1978).

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

5. Clark, David D. et al. "An Introduction to Local Networks," Proceedings of the IEEE, 66(11): 1497-1517, (Nov 1978).

6. Day, John D. "Resource Sharing Protocols," Computer, 12(9): 47-55 (Sept 1979).

7. Digital Equipment Corporation. LSI-11, PDP-11/03 User's Manual. Manufacturer's Data. Maynard, Mass.: Digital Equipment Corporation, 1975.

8. Digital Equipment Corporation. VAX-11/780. Manufacturer's Data. Maynard, Mass.: Digital Equipment Corporation, 1979.

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

10. 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.

11. Intel Corporation. Intellec Series II. Manufacturer's Data. Santa Clara, Ca.: Intel Corporation, 1978.

12. Kuo, Franklin F. Protocols and Techniques for Data Communication Networks. Englewood Cliffs, N.J.: Prentice-Hall Inc., 1981.

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

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

15. 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.

16. Pouzin, Louis and Hubert Zimmermann. "A Tutorial on Protocols," Proceeding of the IEEE, 66(11): (Nov 1978).

17. Schneider, G. Michael. "Computer Network Protocols: A Hierarchical Viewpoint," Computer, 12(9), (Sept 1979).

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

19. 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.

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

21. UCSD PASCAL. System II.0 User's Manual. La Jolla, Ca.: Institute for Information Systems, 1979.

22. Walden, David C. and Alexander A. McKenzie. "The Evolution of Host-to Host Protocol Technology," Computer, 12(9): 28-38, (Sept 1979).

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 Electronic Engineering Group, Richards-Gebaur AFB, Mo, 31 Oct 77.

24. Weinberg, Victor. Structured Analysis. New York: Yourdon Press, 1979.

25. Yourdon, Edward and Larry Constantine. Structured Design. New York: Yourdon Press, 1978.

26. Zilog, Inc. MCZ-1/20,25 Hardware User's Manual, Manufacturer's data. Cupertino, Ca.: Zilog, Inc., 1977.

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

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

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

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

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

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

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

34. 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

Shared Software Components

This appendix contains the software source listing of the shared components for the UNID operating systems. It includes the shared library routines for I/O (U.LIB), and the shared buffer module (U.SHTAB).

```
;
;*******************************************************************************
;*******************************************************************************
;PROLOGUE -      MODULE U.LIB                          DATE 09 NOV 81
;
;               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

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

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

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

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

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

```
!*****************************************************************************
 *****************************************************************************
 MODULE      U.SHTAB      UNID SHARED TABLE MODULE    DATE 09 NOV 81

       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
        PACKETS_IN_TABLE := 10
        STAT_NBR := 20
        TABLE_SIZE := PACKET_SIZE * PACKETS_IN_TABLE

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

        NTLCTB ARRAY [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 := TABLE_SIZE

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

!*************************************************************************
!
END U_SHTAB

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

Appendix B

Local System Software Components

This appendix contains the software source listing of the components specifically associated with the UNID Local Operating System (L.OS). It includes the memory load map (L.OS.MAP), the local buffer module (L.TAB), the vector interrupt module (L.VINT), and the main driver (L.MAIN).

```
PLINK 4.0

LOAD MAP
MODULE          ORIGIN LENGTH

U.LIB             8000   0082
U.SHTAB           8100   02DA
L.VINT            0000   86A5
L.TAB             B000   065F
L.MAIN            B700   0990

LINKAGE INFO      C090   0000

GLOBAL          ADDRESS   MODULE

INIT_L_TAB        B5FA    L.TAB
INIT_U_SHTAB      8378    U.SHTAB
INVINT            8520    L.VINT
LC01NE            B12E    L.TAB
LC01NS            B12C    L.TAB
LC01SZ            B130    L.TAB
LC01TB            B000    L.TAB
LC02NE            B260    L.TAB
LC02NS            B25E    L.TAB
LC02SZ            B262    L.TAB
LC02TB            B132    L.TAB
LC03NE            B392    L.TAB
LC03NS            B390    L.TAB
LC03SZ            B394    L.TAB
LC03TB            B264    L.TAB
LC04NE            B4C4    L.TAB
LC04NS            B4C2    L.TAB
LC04SZ            B4C6    L.TAB
LC04TB            B396    L.TAB
```

```
LCLCNE          B5F6    L.TAB
LCLCNS          B5F4    L.TAB
LCLCSZ          B5F8    L.TAB
LCLCTB          B4C8    L.TAB
LCNTNE          822E    U.SHTAB
LCNTNS          822C    U.SHTAB
LCNTSZ          8230    U.SHTAB
LCNTTB          8100    U.SHTAB
MAIN            C061    L.MAIN
MOVSEQ          8000    U.LIB
NTLCNE          8360    U.SHTAB
NTLCNS          835E    U.SHTAB
NTLCSZ          8362    U.SHTAB
NTLCTB          8232    U.SHTAB
RECSEQ          8023    U.LIB
SNDSEQ          8053    U.LIB
STATTB          8364    U.SHTAB
TDAADD          B700    L.MAIN
TPRADD          B702    L.MAIN
TRNMIT          858A    L.VINT

PROGRAM L.OS -- C090 BYTES
ENTRY: C061
```

```
!*****************************************************************************
 *****************************************************************************
 MODULE      L.TAB        LOCAL OS TABLES      DATE 09 NOV 81

       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
        PACKETS_IN_TABLE := 10
        TABLE_SIZE := PACKET_SIZE * PACKETS_IN_TABLE

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

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

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

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

        LCLCTB ARRAY [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 := TABLE_SIZE

            LC02NS := 0
            LC02NE := 0
            LC02SZ := TABLE_SIZE

            LC03NS := 0
            LC03NE := 0
            LC03SZ := TABLE_SIZE

            LC04NS := 0
            LC04NE := 0
            LC04SZ := TABLE_SIZE

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

        END INIT_L_TAB

!*************************************************************************************
!
END L_TAB
```

```
;
;*****************************************************************************
;*****************************************************************************
;PROLOGUE -      MODULE L.VINT           DATE 09 NOV 81
;
;               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

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

```
;*****************************************************************************
;*****************************************************************************
;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.
;
;*******************************************************************************
```

```
                        ORG 8500H       ; **** 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
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

                        ORG 8520H       ; **** 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 ****

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
!*****************************************************************************
!
!*****************************************************************************
!
!PROLOGUE -      MODULE L.MAIN        DATE 09 NOV 81

        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 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
        PACKETS_IN_TABLE := 10
        STAT_NBR := 20                      ! NBR OF STATUS ENTRIES IN STATTB !
        TABLE_SIZE := PACKET_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 !

    EXTERNAL                                ! IN L.VINT !
        INVINT PROCEDURE
        TRNMIT PROCEDURE

    EXTERNAL                                ! IN L.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 [TABLE_SIZE BYTE]
    LC01NS INTEGER
    LC01NE INTEGER
    LC01SZ INTEGER

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

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

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

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

EXTERNAL                                    ! IN U.SHTAB !
    INIT_U_SHTAB PROCEDURE

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

    NTLCTB ARRAY [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 24 AUG 81%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 -   THIS PROC IS A SKELETON.  IT SIMPLY EVALUATES THE
    FIRST CHARACTER OF THE PACKET FOR DESTINATION IF INPUT
    LOCALLY.  IF INPUT FROM THE NET, THE SECOND CHARACTER IS
    EVALUATED.  A ERROR IS NOTED ON 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 -    THIS PROC MUST BE DEVELOPED TO MATCH X.25 PROTOCOL
    REQUIREMENTS.
 *************************************************************************
```

```
!
    INTERNAL

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

            IF TABLE
                CASE '01'
                    THEN
                    BYTE_01 := LC01TB[LC01NS]
                CASE '02'
                    THEN
                    BYTE_01 := LC02TB[LC02NS]
                CASE '03'
                    THEN
                    BYTE_01 := LC03TB[LC03NS]
                CASE '04'
                    THEN
                    BYTE_01 := LC04TB[LC04NS]
                CASE 'LL'
                    THEN
                    BYTE_01 := 'T'
                    BYTE_02 := LCLCTB[LCLCNS]
                CASE 'LN'
                    THEN
                    BYTE_01 := 'T'
                    BYTE_02 := NTLCTB[NTLCNS+1]
                ELSE
                    BYTE_01 := 'E'
            FI
```

```
            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'
                    FI
                ELSE
                    DESTINATION := 'ER'
            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 + PACKET_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  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.

 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
                    MOVSEQ( LC01TB[LC01NS],
                            LCNTTB[LCNTNE],
                            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 !
    IF DESTINATION
        CASE 'LL'                        ! LOCAL DESTINATION !
            THEN
            MOVSEQ( LC02TB[LC02NS],
                    LCLCTB[LCLCNE],
                    PACKET_SIZE)
            LD_TAB_HSKP('LL')

        CASE 'LN'                        ! NETWORK DESTINATION !
            THEN
            MOVSEQ( LC02TB[LC02NS],
                    LCNTTB[LCNTNE],
                    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
            MOVSEQ( LC03TB[LC03NS],
                    LCNTTB[LCNTNE],
                    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
                    MOVSEQ( LC04TB[LC04NS],
                            LCNTTB[LCNTNE],
                            PACKET_SIZE)
                    LD_TAB_HSKP('LN')

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

            SRVC_TAB_HSKP('04')

        FI

    END ROUTE_IN
```

```
!*****************************************************************************
 *****************************************************************************
 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.
    AND SRVC_TAB_HSKP.
 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) >= PACKET_SIZE)      ! IF NET PACKET 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], ! TRANSMIT PKT !
                                U01DAT)
                    SRVC_TAB_HSKP('NL')          ! HSKP NET TO LOC TABLE !
                CASE '02'                        ! CH 2 DESTINATION !
                    THEN
                    TRNMIT_PKT( NTLCTB[NTLCNS], ! TRANSMIT PKT !
                                U02DAT)
                    SRVC_TAB_HSKP('NL')          ! HSKP NET TO LOC TABLE !
                CASE '03'                        ! CH 3 DESTINATION !
                    THEN
                    TRNMIT_PKT( NTLCTB[NTLCNS], ! TRANSMIT PKT !
                                U03DAT)
                    SRVC_TAB_HSKP('NL')          ! HSKP NET TO LOC TABLE !
                CASE '04'                        ! CH 4 DESTINATION !
                    THEN
                    TRNMIT_PKT( NTLCTB[NTLCNS], ! 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

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

Appendix C

Network System Software Components

This appendix contains the software source listing of the components specifically associated with the UNID Network Operating System (N.OS). It includes the memory load map (N.OS.MAP), the network buffer module (N.TAB), the SIO interrupt module (N.INSIO), and the main driver (N.MAIN).

```
PLINK 4.0

LOAD MAP
MODULE       ORIGIN LENGTH

U.LIB          8000   0082
U.SHTAB        8100   02DA
N.INSIO        0000   C1E0
N.TAB          E000   02CF
N.MAIN         E300   0406

LINKAGE INFO   E706   0000

GLOBAL      ADDRESS   MODULE

INIT_N_TAB     E2A0   N.TAB
INIT_U_SHTAB   8378   U.SHTAB
INSIO          C120   N.INSIO
LCNTNE         822E   U.SHTAB
LCNTNS         822C   U.SHTAB
LCNTSZ         8230   U.SHTAB
LCNTTB         8100   U.SHTAB
MAIN           E6DA   N.MAIN
MOVSEQ         8000   U.LIB
NT01NE         E14C   N.TAB
NT01NS         E14A   N.TAB
NT01SZ         E14E   N.TAB
NT01TB         E000   N.TAB
NTLCNE         8360   U.SHTAB
NTLCNS         835E   U.SHTAB
NTLCSZ         8362   U.SHTAB
NTLCTB         8232   U.SHTAB
NTNTNE         E29C   N.TAB
NTNTNS         E29A   N.TAB
NTNTSZ         E29E   N.TAB
NTNTTB         E150   N.TAB
RECSEQ         8023   U.LIB
SNDSEQ         8053   U.LIB
STATTB         8364   U.SHTAB
TRNMIT         C17C   N.INSIO

PROGRAM N.OS -- E706 BYTES
ENTRY: E6DA
```

```
!*****************************************************************************
 *****************************************************************************
 MODULE      N.TAB        NETWORK OS TABLES      DATE 09 NOV 81

       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
        NTOlTB ARRAY [F_TABLE_SIZE BYTE]
        NTOlNS INTEGER
        NTOlNE INTEGER
        NTOlSZ 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
```

```
;
;*****************************************************************************
;*****************************************************************************
;PROLOGUE -      MODULE N.INSIO          DATE 09 NOV 81
;
;               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

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

```
;*****************************************************************************
;*****************************************************************************
;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.
```

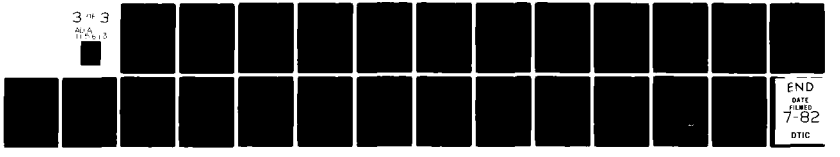
```
;               THE Z80 INTERRUPT MODE IS SET TO 2
;       AND INTERRUPTS ENABLED.  AT THIS POINT, THE UNID
;       IS CONFIGURED FOR INTERRUPT DRIVEN COMMUNICATIONS ON THE
;       NETWORK SIDE.
;               FINALLY, THE IX IS RESTORED, THE RETURN ADDRESS
;       RECOVERED, AND A RETURN EXECUTED.
;               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 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.
;
;*******************************************************************************
```

```
                ORG 0C100H      ; **** 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 0C120H      ; **** 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 ****

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
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 O.ERATING MODE
OUTI               ; SET TIME CONSTANT

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

AD-A115 613 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 9/2
DEVELOPMENT OF THE DIGITAL ENGINEERING LABORATORY COMPUTER NETW--ETC(U)
DEC 81 J W GEIST

UNCLASSIFIED AFIT/GCS/EE/81D-8 NL



```
                    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              NOP             ; CTC TIME OUT STUB. NOT CURRENTLY USED.
                    RET             ; RETURN

                    ;EQUATES

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
```

```
                ;DEFINES

CTCLST  DEFB    CTC_0                   ; CTC CH 0 ADDRESS
        DEFW    CTCI01                  ; CTC INTERRUPT VECTOR ADDRESS
        DEFB    01000011B               ; CH 0 - RESET, INTERRUPTS OFF
        DEFB    00000001B
        DEFB    01000111B               ; CH 1 - CTR MODE, CNST NXT, RESET
        DEFB    10000000B               ;      - 9600 BAUD
        DEFB    01000011B               ; CH 2 - RESET, INTERRUPTS OFF
        DEFB    00000001B
        DEFB    01000011B               ; 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    00000100B               ; REG 4 - X1, ASYNC 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

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

```
;*****************************************************************************
;*****************************************************************************
;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
TRNJ30          IN A,(A_STAT)   ; READ PORT A STATUS REG
                BIT 2,A         ; AND WAIT UNTIL TX BUFFER IS EMPTY
                JP Z,TRNJ30

                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

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

```
!*****************************************************************************
!
!*****************************************************************************
!
!PROLOGUE -      MODULE N.MAIN        DATE 09 NOV 81

        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, FRAME_DATA, ROUTE_IN, AND ROUTE_OUT.
!
!*****************************************************************************
!
!*****************************************************************************
!
```

```
MAIN MODULE

    TYPE
        PBYTE ^BYTE

    CONSTANT                                    ! GENERAL CONSTANTS !
        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
        PACKETS_IN_TABLE := 10
        FRAME_SIZE := PACKET_SIZE + 3
        TABLE_SIZE := PACKET_SIZE * PACKETS_IN_TABLE
        STAT_NBR := 20                          ! NBR OF ENTRIES IN STATUS TABLE !

    CONSTANT                                    ! CONSTANTS USED BY FRAME_DATA !
        HDR00 := 00                             ! FRAME HEADER BYTE 00 !
        HDR01 := 01                             ! FRAME HEADER BYTE 01 !
        DAT00 := 02                             ! FRAME DATA START LOCATION !
        TRL00 := HDR01                          ! FRAME TRAILER BYTE 00 !
                 + PACKET_SIZE
                 + 1

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

    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 [TABLE_SIZE BYTE]
    NT01NS INTEGER
    NT01NE INTEGER
    NT01SZ INTEGER

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

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

    NTLCTB ARRAY [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 !

INTERNAL                                    ! INTERNAL TABLES USED !
                                            ! THROUGHOUT MODULE !
    STARTUP_HDR ARRAY [* BYTE] :=           '%R%R%L%L'
                                            'UNID NETWORK OS%R%L'
                                            'VERSION 24 AUG 81%R%L'
                                            'EXECUTING%R%L}'

    FRMETB ARRAY [FRAME_SIZE BYTE]          ! FRAMING 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 PROC IS A SKELETON. IT SIMPLY FORCES ALL NETWORK
    TRAFFIC TO THE LOCAL SIDE FOR TERMINATION.

 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 -     THIS PROC MUST BE DEVELOPED TO MATCH X.25 PROTOCOL
    REQUIREMENTS.
 *****************************************************************************
!
    INTERNAL

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

            DESTINATION := 'NL'          ! FORCE ALL NET TRAFFIC TO LOCAL !

        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 + PACKET_SIZE   ! ADV NEXT EMPTY PNTR !
                IF NTLCNE >= NTLCSZ              ! IF TABLE WRAP !
                    THEN                         ! THEN SET PNTR TO 0 !
                    NTLCNE := 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 + PACKET_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  FRAME_DATA      FRAME DATA FOR NET TRANSMISSION

       THE PURPOSE OF THIS PROC IS TO ESTABLISH THE CORRECT
    FRAME AROUND DATA PACKETS FOR NETWORK TRANSMISSION.

 INPUT -    THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING
    THE TABLE CONTAINING THE DATA TO BE FRAMED.

 PROCESSING -   THIS PROC IS A SKELETON. FOR NETWORK TO NETWORK
    DATA RESIDING IN NTNTTB, THE PROC MODIFIES THE CURRENT FRAME
    HEADER BYTES TO 'N' AND 'T'. NOTE THAT THE NTNTTB CONTAINS
    DATA WITH THE FRAME. FOR LOCAL TO NETWORK DATA RESIDING IN
    LCNTTB, THE PROC BUILDS THE FRAME WITH A HEADER, THE PACKET,
    AND A TRAILER, AND LOADS IT INTO THE FRAME TABLE.

 OUTPUT -   SEE PROCESSING.

 INTERFACE -    THIS PROC IS CALLED BY ROUTE_OUT.

 NOTES -    1. THIS PROC MUST BE DEVELOPED TO MATCH X.25 PROTOCOL
    REQUIREMENTS.
 *****************************************************************************
!
```

```
INTERNAL

    FRAME_DATA PROCEDURE(TABLE WORD)
    ENTRY
        IF TABLE
            CASE 'NN'                   ! IF CALLED FOR NET TO NET TABLE !
                THEN                            ! THEN !
                NTNTTB[NTNTNS] := 'N'           ! SET HDR TO 'NT' !
                NTNTTB[NTNTNS+1] := 'N'

            CASE 'LL'                   ! IF CALLED FOR LOCAL TO NET TABLE !
                THEN                            ! THEN !
                FRMETB[HDR00] := 'L'            ! SET HEADER AND TRAILER !
                FRMETB[HDR01] := 'C'
                FRMETB[TRL00] := 'Z'

                MOVSEQ( LCNTTB[LCNTNS],         ! AND MOVE DATA TO FRAME TBL !
                        FRMETB[DAT00],
                       PACKET_SIZE)

        FI

    END FRAME_DATA

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

```
!*****************************************************************************
 *****************************************************************************
 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.

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

 PROCESSING -    THE PROC CHECKS THE INPUT BUFFER'S POINTERS FOR
    PACKET/FRAME ARRIVAL.  IF DATA 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.

 OUTPUT -    A PACKET OR 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-NTC1NS) >= FRAME_SIZE)       !IF NET PACKET IS RDY !
        ORIF (NT01NS > NT01NE)
            THEN                                 ! THEN !
            DESTINATION := DET_DEST('01')        ! DETERMINE DESTINATION !
                                                 ! MOVE DATA AND HSKP TBLS !
            IF DESTINATION
                CASE 'NL'                        ! LOCAL DESTINATION !
                    THEN
                    MOVSEQ( NT01TB[NT01NS+2],
                            NTLCTB[NTLCNE],
                            PACKET_SIZE)
                    LD_TAB_HSKP('NL')

                CASE 'NN'                        ! NETWORK DESTINATION !
                    THEN
                    MOVSEQ( NT01TB[NT01NS+2],
                            NTNTTB[NTNTNE],
                            FRAME_SIZE)
                    LD_TAB_HSKP('NN')

                ELSE
                    STATTB[NET_RI_DEST_ERR] += 1! ERROR, COUNT DEST ERROR !
            FI

            SRVC_TAB_HSKP('01')

        FI

    END ROUTE_IN
```

```
!*****************************************************************************
 *****************************************************************************
 PROCEDURE  ROUTE_OUT        ROUTE PACKETS OUT

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

 INPUT -     DATA PACKETS OR FRAMES ARE ROUTED VIA EVALUATION
    OF NTNTTB AND LCNTTB POINTERS AND INTERNAL PACKET ROUTING INFORMATION.

 PROCESSING -    THE PROC CHECKS EACH INPUT BUFFER'S POINTERS FOR
    PACKET/FRAME ARRIVAL. IF READY, THE DATA IS FRAMED AND THEN
    TRANSMITTED VIA PROC TRNMIT.
    THE TABLE THAT WAS SERVICED (PACKET REMOVED) HAS ITS POINTERS
    HOUSEKEPT BY SRVC_TAB_HSKP.
    AND SRVC_TAB_HSKP.
 OUTPUT -    A FRAME OF DATA IS TRANSMITTED TO THE NETWORK CHANNEL.

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

 NOTES -     NONE.
 *****************************************************************************
!
```

```
INTERNAL

    ROUTE_OUT PROCEDURE
    ENTRY

        IF ((NTNTNE-NTNTNS) >= FRAME_SIZE)      ! IF NET PACKET IS RDY !
        ORIF (NTNTNS > NTNTNE)
            THEN                                ! THEN !
            FRAME_DATA('NN')                    ! UPDATE THE FRAME !
            TRNMIT( NTNTTB[NTNTNS],             ! TRANSMIT THE FRAME !
                    FRAME_SIZE)
            SRVC_TAB_HSKP('NN')                 ! AND HSKP THE TABLE !
        FI

        IF ((LCNTNE-LCNTNS) >= PACKET_SIZE)     ! IF LOCAL PACKET IS RDY !
        ORIF (LCNTNS > LCNTNE)
            THEN                                ! THEN !
            FRAME_DATA('LN')                    ! FRAME THE PACKET !
            TRNMIT( FRMETB[HDR00],              ! TRANSMIT THE FRAME !
                    FRAME_SIZE)
            SRVC_TAB_HSKP('LN')                 ! AND HSKP THE TABLE !
        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

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

Vita

Captain John W. Geist was born on November 29, 1946 in Columbus, Ohio. In 1964, he graduated from St. Mary High School. In late 1964, he entered the United States Air Force and served as an electronics technician on assignments both within the United States and abroad. In 1973, he was selected for the Airmen's Education and Commissioning Program and attended The Ohio State University, where he received a Bachelor of Science degree with a major in Computer Science. Following his commission at the Air Force Office Training School in 1975, he served as a systems analyst for the North American Air Defense Command in Colorado Springs, Colorado. He entered the Air Force Institute of Technology in June 1980.

Permanent address: 335 Forest St.
Columbus, Ohio 43206

AD-A115613

UNCLASSIFIED

Dean for Research and
Professional Development
Air Force Institute of Technology (ATC)
Wright-Patterson AFB, OH 45433

4 JUN 1982

Lynn Wolaver

LYNN E. WOLAVER
Dean for Research and
Professional Development

[redacted]

APPROVED FOR PUBLIC RELEASE IAW AFR 19[redacted]

[redacted]


FILME
8