DECsystem-10
Data Communications Products

For Software Specialists

Course Number: SWS-10-DC-SG

Digital Equipment Corporation
Educational Services
Marlborough, Massachusetts
June 1978

Student Guide

Introduction

This course is designed for the experienced specialist who has a need for information on the products and workings of DECsystem-10 communications products and the workings of DECsystem-10 networks.

In modules 1 and 2, definitions, network concepts, and a catalogue of the various communications products are presented. Material covered includes the specific node to node protocol used, the monitor calls and UUO's which are network specific, inter-task (TSK:) communications, the internal data structures of both the DECsystem-10 and the communication device. Also addressed is the assembly of communications device code, the loading of the code, debugging and the one-time diagnostic.

In modules 3-10, features of the DECsystem-10 networks are examined. The latter modules may be studied independently of one another.

Prerequisites and Student Preparation

DECsystem-10 Monitor Internals, Communications Fundamentals, Introduction to the PDP-11.

Student preparation should include coverage of the "DDCMP" and "DAP" modules from the DECNET Architecture self-paced package.

It is strongly recommended that the Specialist have at least three months of "hands-on" field experience with the DECsystem-10.

Course Goals

The goal of this course is to familiarize the student with the workings and capabilities of the various DECsystem-1Ø data communications products. More specifically, achievement of this goal includes

1. Reviewing communications and DEC network terminology and networking concepts.

2. Providing the student with the specifications and capabilities of each of the DECsystem-1Ø data communications products.

3. Presenting, if needed, the ANF-1Ø protocol.

4. Examining the network related monitor calls and commands.

5. Acquainting the student with the abilities and the uses of the pseudo-device TSK: in terms of interprocess communication in a network environment.

6. Examining the internal structure and code of TOPS-1Ø Operating System and the DN8Ø Series processors.

7. Acquainting the student with the procedures used in the assembly and installation of network software.

8. Presenting the specialized tools available for testing and debugging in the support of the DECsystem-1Ø networks.

Course Outline

MODULE 1 - INTRODUCTORY CONCEPTS

I. Introduction to Networks
   A. What is a Computer Network?
   B. Why a Network?
   C. How a Network - Techniques

II. Terminology - General Definitions

MODULE 2 - NETWORK CONCEPTS

I. DECsystem-1Ø Networks Definition and Discussion
   A. Asynchronous Communications
   B. Synchronous Communications
   C. Comparison of These Two
   D. Remote Communications
   E. Topologies
      1. Simple
      2. Complex
         a. Route-Through
         b. Multi-path

II. DECsystem-1Ø Communications Products Overview
   A. DN8Ø Series
      1. DN87 (s)
      2. DN85
      3. DN8Ø, DN81, DN82
   B. DN92, DC72NP
   C. DN61, (DN62, DAS78)
   D. DC76NP, DC75
   E. DN98

MODULE 3 - PROTOCOLS

I. General Protocols
   A. General Discussion Protocols
   B. Types of Protocols
      1. Byte Oriented (BISYNC, BSC)
      2. Byte Count (DDCMP)

3. Bit Oriented (SDLC)

II. DECsystem-1Ø Networks Protocols
A. NCL
B. DAP

MODULE 4 - USING DECsystem-1Ø NETWORKS

I. DECsystem-1Ø Networks Uses and Capabilities

II. Network Devices, Device names

III. Network Commands
A. ASSIGN
B. SEND
C. SET HOST(ESS)
D. NODE
E. WHERE
F. LOCATE

IV. Network Monitor Calls (UUO's)
A. LOCATE
B. GTNTN.
C. GTXTN.
D. NODE.
E. WHERE
F. (CALL1.)

MODULE 5 - INTERTASK COMMUNICATIONS

I. Tasks - Definition (TSK:)

II. File Transfer
A. TECO
B. PIP

III. Intertask Communication
A. MACRO - Example
B. FORTRAN - Example
C. COBOL - Example

MODULE 6 - TOPS-10 NETWORK INTERNALS

I. How Monitor handles Networks
   A. SCNSER
   B. NETSER
   C. RDXSER
   D. (DTESER)

MODULE 7 - DN80 SERIES INTERNAL TASKS

I. What They Are

II. How They Work
   A. Scheduling
   B. System Calls

III. Sample

MODULE 8 - INSTALLATION

I. Assembly of Network Software
   A. Configuration Files
   B. Source Files
   C. Control (.CTL) Files

II. Loading
   A. NETLDR
   B. BOOT11
   C. BOOTDT

MODULE 9 - TESTING AND DEBUGGING

I. CHK11 - Testing (Hardware)

II. Debugging
   A. DDT11
   B. DUMPING
   C. Reading DUMPS

MODULE 1Ø - DN8Ø SERIES INTERNAL MATERIALS

I. Flow
   A. Structure
   B. Table Definitions
   C. Routines

Course Resources

Technical Aspects of Data Communications - John E McNamera
Digital Press 1977 Documentation No. JB002-A

Introduction to Minicomputer Networks -
Digital Equipment Corporation
EK 05300 75 09A/20 0945

PDP11 Peripherals Handbook - Digital Equipment Corporation
EB 05961 76 05A/20 D 09/02 60

DECsystem-10 Technical Summary -
Digital Equipment Corporation
EH 07127 63/77 010 03 10

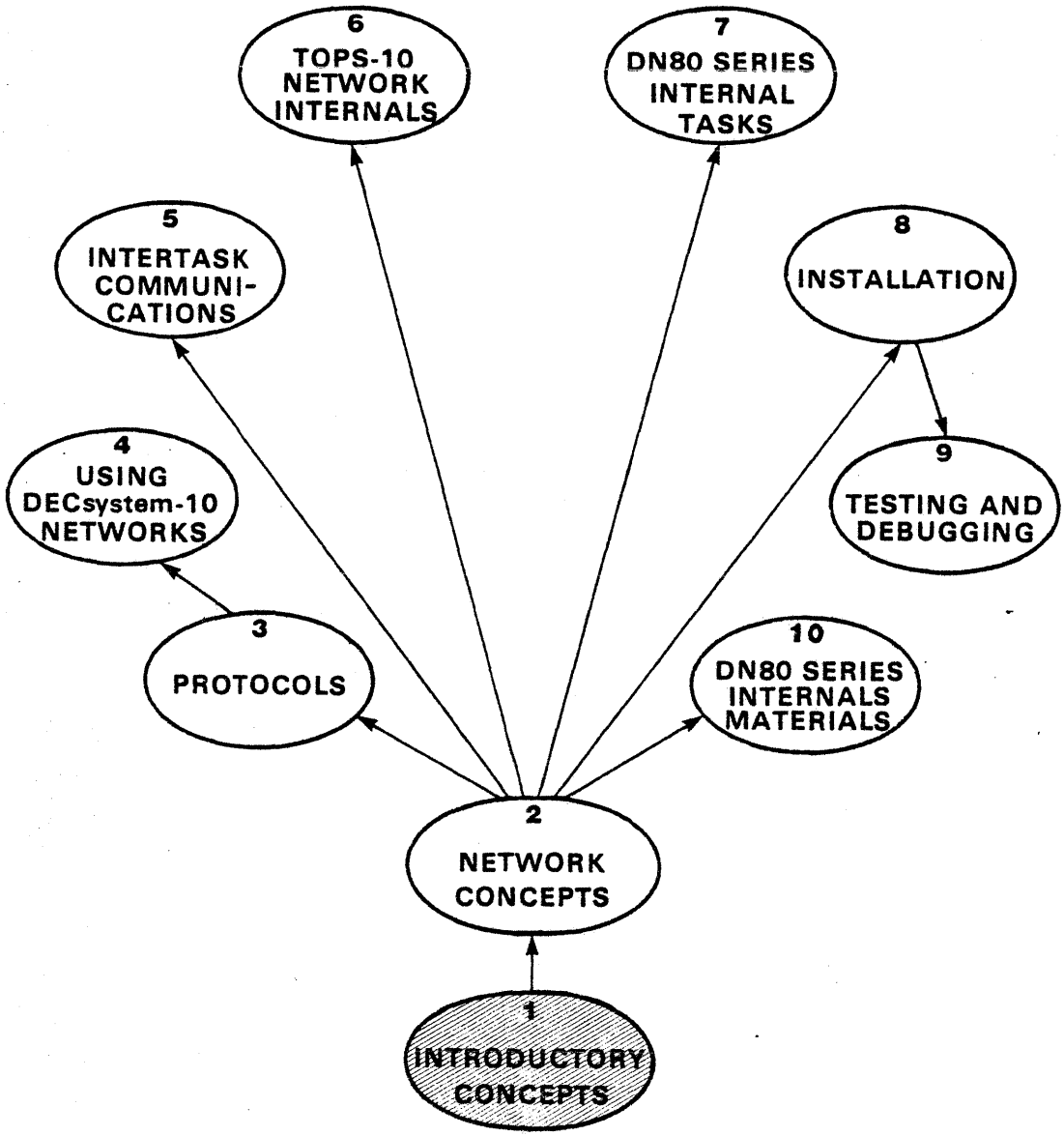
DECsystem-10 Software Notebooks
AD-5483A-RB

DECsystem-10 Networks Programmers Guide and Reference Manual
DEC-10-ONPGA-A-D

Networks Software Installation Guide -
In Volume 12 of the Software Notebooks
and appears as module 8 of this course

Introductory Concepts
Module 1

Course Map



M8 0277

Introduction

Before studying the specific communications products supported by the DECsystem-10, it is imperative that basic data communications concepts be understood. Your previous experience, either prerequisite courses or actual communications work, should have introduced much of the generic terminology and many of the technical principles. The first task of this course is to make sure you can "speak the language" of data communications.

Following the Terminology section is a brief description of the whys and wherefores of networks.

Objectives

Upon completion of this module, the student will be able to

1. Define "network."

2. Describe 3 situations where a network might be useful.

Additional Resources

Technical Aspects of Data Communications by John McNamara, Document # JB 002 A

PDP-11 Peripherals Handbook

Introduction to Networks

A computer network is a group of computers which are interconnected to permit communications from one processor to another. This interconnection may be a simple wire from one machine to another as in a factory process control system where one machine simply puts a voltage on that wire to indicate the completion of a job. Or the interconnection may be a telephone line over which data is transferred in large quantities among several computers. The computers in a network are generally called nodes.

WHY A NETWORK?

There are many reasons why a network is appropriate. For example:

1. Process control systems often have several small computers, each primarily concerned with a specific part of a large job. Each of these small computers may be controlled by one central system which keeps track of the entire process.

2. A large organization may have several regional processing centers which handle the computer needs of each region. If the home office should require reports or data from these regions, it would be advantageous to have this information directly available over communications lines (as opposed to mailing tapes).

3. A company may have one central computer which handles orders or general computing. There may be many remote terminals which are spread over a large geographical area. A communication network which is capable of concentrating the lines from each of these terminals into regional centers could greatly reduce the cost of communications lines.

4. An organization may have a large data base, or set of programs which they wish to have shared among a group of users on various other computers. A network would make this resource sharing possible.

5. A corporation may have several data collection points, e.g., factories, warehouses, sales offices, which need reports based on the data generated both locally and remotely. A network will allow these management reports to be generated more quickly, thereby more accurately representing the assets of the corporation.

HOW A NETWORK? - Various Protocols

There are several uses of networks. One of the more common uses is the Remote Job Entry (RJE) system in which jobs may be sent to a remote computer (through cards or terminals) for processing and the output returned to the RJE site. Another approach is Message Switching. In a message switching environment, full messages are sent intact from the source machine to the destination machine. One long message, then, may tie up a line while other (perhaps more important) traffic must wait for it. Since messages are often of variable length, the control of message handling often leads to inefficient line usage. Packet Switching is another technique in computer networks. The processing of this type of system entails breaking the (variable length) messages into hunks (packets) of fixed format and (generally) length. These packets are then sent individually over various routes to be re-assembled at the ultimate destination. Generally, then, the concern in networks is the efficiency of processor and line utilization.

Terminology/General Definitions

ACU - Automatic Calling Unit

Amplitude Modulation (AM) - A method of transmission whereby the amplitude of the carrier wave carries the information.

ANSI - American National Standards Institute, a standards-making body.

Asynchronous Transmission - Transmission in which time intervals between transmitted characters may be of unequal length. Transmission is controlled by start and stop elements at the beginning and end of each character.

Bandwidth - The range of frequencies assigned to a channel or system; the difference expressed in Hertz between the highest and lowest frequencies of a band. This corresponds to the information-carrying capacity of the channel, system or band.

Baud - A unit of signaling speed equal to the number of discrete signals per second. Example: If the signaling unit interval is 20 milliseconds, the signaling speed is 50 baud. One signal usually carries one bit of information, but it can carry more or (rarely) less.

BSC (BISYNC) - Binary Synchronous Communications protocol, IBM's half-duplex, character-oriented protocol.

CCITT - Consultative Committee on International Telegraph and Telephone, a standards-making body.

Channel - A communications path.

Coaxial Cable - A cable consisting of an outer conductor concentric to an inner conductor, separated from each other by insulating material. It can carry a much higher bandwidth than a wire pair.

Concentrator - A communications device that provides communications capability between many low speed, usually asynchronous channels and one or more high speed, usually synchronous channels. Usually different speeds, codes, and protocols can be accommodated on the low speed side. The low speed channels usually operate in contention and thus require buffering. The concentrator may have the capability to be polled by a computer, and may in turn poll terminals.

Cyclic Redundancy Check (CRC) - An error detection scheme in which the check character is generated by taking the remainder after dividing all the serialized bits in a block of data by a predetermined binary pattern.

DAP - Data Access Protocol.

DDCMP - Digital Data Communications Message Protocol, byte-oriented line protocol used in DECnet.

Duplex - Simultaneous two-way independent transmission. Also referred to as full-duplex.

EIA - Electronics Industries Association, a standards-making body.

Four-wire Circuit - A circuit using two pairs of conductors - one pair for the "go" channel and the other pair for the "return" channel. A telephone circuit is basically duplex (it carries voice signals both ways). In the local network this is achieved over two wires because the waveforms travelling each direction can be distinguished. In the trunk network, where amplifiers are needed at intervals and multiplexing is common, it is easier to separate the two directions of transmission and use (effectively) a pair of wires for each direction. At this point, it is a four-wire circuit.

Frequency Division Multiplexing (FDM) - Dividing the available transmission bandwidth into narrower bands each of which is used for a separate channel.

Frequency Modulation (FM) - A method of transmission whereby the frequency of the carrier wave carries the information.

Half-Duplex - A circuit designed for transmission in either direction but not both directions simultaneously.

Hertz - A unit of frequency equal to one cycle per second. Cycles are referred to as Hertz in honor of the experimenter Heinrich Hertz. Abbreviated Hz.

ISO - International Standards Organization, a standards-making body.

Leased Line - A line reserved for the exclusive use of a leasing customer without interexchange switching arrangements. Also called a private line, or conditioned line.

Longitudinal Redundancy Check (LRC) - An error checking technique based on an accumulated exclusive OR of transmitted characters. An LRC character is accumulated at both the sending and receiving stations during the transmission of a block. This accumulation is called the Block Check Character (BCC), and is transmitted as the last character in the block. The transmitted BCC is compared with the accumulated BCC character at the receiving station for an equal condition. An equal comparison indicates a good transmission of the previous block.

MARK - Presence of a signal. In telegraphy, MARK represents the closed condition or current flowing. Equivalent to a binary one condition; opposite of SPACE.

MCB - Multi-system Communication Base, Digital's company-wide network front end.

Modem (Modulator-Demodulator) - A device which converts analogue signals to digital signals (and vice versa) to be sent over communications lines. Also called a data set when line control module is included.

Multiplexing - The division of a transmission facility into two or more channels either by splitting the frequency band transmitted by the channel into narrower bands, each of which is used to constitute a distinct channel (frequency-division multiplex); or, by allotting this common channel to several different information channels one at a time (time-division multiplexing).

Multipoint Line - A single communications line to which more than one terminal is attached. Use of this type of line normally requires some kind of polling mechanism, addressing each terminal with a unique ID. Also called Multi-Drop.

Parallel Transmission - Method of data transfer in which many bits of a character or byte are transmitted simultaneously either over separate communication lines or on different carrier frequencies on the same communication line.

Phase Modulation (PM) - A method of transmission whereby the phase of the carrier wave is varied in accordance with the signal.

Protocol - A formal set of conventions governing the format and relative timing of message exchange between two communicating processes. See also Control Procedure.

Serial Transmission - A method of transmission in which each bit of information is sent sequentially on a single channel rather than many bits simultaneously as in parallel transmission.

Simplex Mode - Operation of a channel in one direction only, with no capability of reversing.

SPACE - Absence of a signal. Equivalent to a binary 0; opposite of MARK.

Switched Line - A communications link for which the physical path may vary with each usage, e.g., the dial-up telephone network.

Synchronous Transmission - Transmission in which the data characters and bits are transmitted at a fixed rate with the transmitter and receiver synchronized. This eliminates the need for start-stop elements, thus providing greater efficiency. Compare Asynchronous Transmission.

Throughput - Linespeed in terms of the amount of user's data per unit time.

Time Division Multiplexing (TDM) - A means of obtaining a number of channels over a single line by connecting terminals one at a time at frequent intervals. Compare FDM

Two-wire Circuit - A circuit formed of two conductors insulated from each other, providing a "go" and "return" channel in the same frequency. Compare Four-wire.

Vertical Redundancy Check (VRC) - A check or parity bit added to each character in a message such that the number of bits in each character, including the parity bit, is odd (odd parity) or even (even parity).

COMMUNICATIONS HARDWARE COMPONENTS

DH11	16 lines, ASYNC; TTY lines for DC20 and DN80 series
DL11-C	1 line, ASYNC; local CTY, current loop for KL10's and DN80 series
DL11-E	1 line, ASYNC; KLINIK line, diagnostic link to DN20s
DL11-W	1 line, ASYNC, with real time clock; connects the DN20 to a DL11-E on the primary front-end for diagnostic purposes
DMC11	interconnects -11's, DDCMP in hardware, SYNC, NPR, used in DN20
DN11	ACU
DQ11	1 line, SYNC, NPR, used in DN80 series
DUP11	1 line, bit/byte SYNC, used with KMC11 in DN20
DV11	8 or 16 lines, SYNC, NPR, not used by DECnet.
DZ11	8 lines, ASYNC, used in DN20
KG11-A	Computes CRC, used in DN80 series
KMC11	microprocesssor - helps DN20 drive DUP11s (may be associated with a DZ11 or DUP11 to make them NPR devices)

Not all of the above are directly involved in the communications environment. But, they are the types of devices with which you may come in contact.

This page intentionally left blank

Module Test

1. From your experience, give an example where a computer network might be useful within DIGITAL. (Give an instance where no network now exists.)

2. Match the numbered terms with the lettered definitions.

1. Half-Duplex
2. BISYNC
3. DDCMP
4. Leased Line
5. Modem
6. Simplex Mode
7. Two-wire circuit
8. Synchronous Transmission
9. Asynchronous Transmission
10. MARK
11. SPACE

a) is controlled by start and stop elements at the beginning and end of each character. The time intervals between transmitted characters may be of unequal length.

b) is IBM's half-duplex, character-oriented protocol.

c) is the byte-oriented protocol used in DECnet.

d) is a circuit designed for transmission in either direction but not both directions simultaneously.

e) is reserved for the exclusive use of a leasing customer without interexchange switching arrangements.

f) represents the closed condition or current flowing. It is also equivalent to a binary one condition.

g) is a device which converts analogue signals to digital signals, and vice versa, to be sent over communications lines. It is also called a data set.

h) is the operation of a channel in one direction only with no capability of reversing.

i) is the absence of a signal and is equivalent to a binary 0.

j) eliminates the need for start-stop elements, thus providing greater efficiency for sending data characters and bits at a fixed rate with the transmitter and receiver synchronized.

k) is formed of two conductors insulated from each other, providing a "go" and "return" channel in the same frequency. It is metallic and not grounded.

Evaluation Sheet

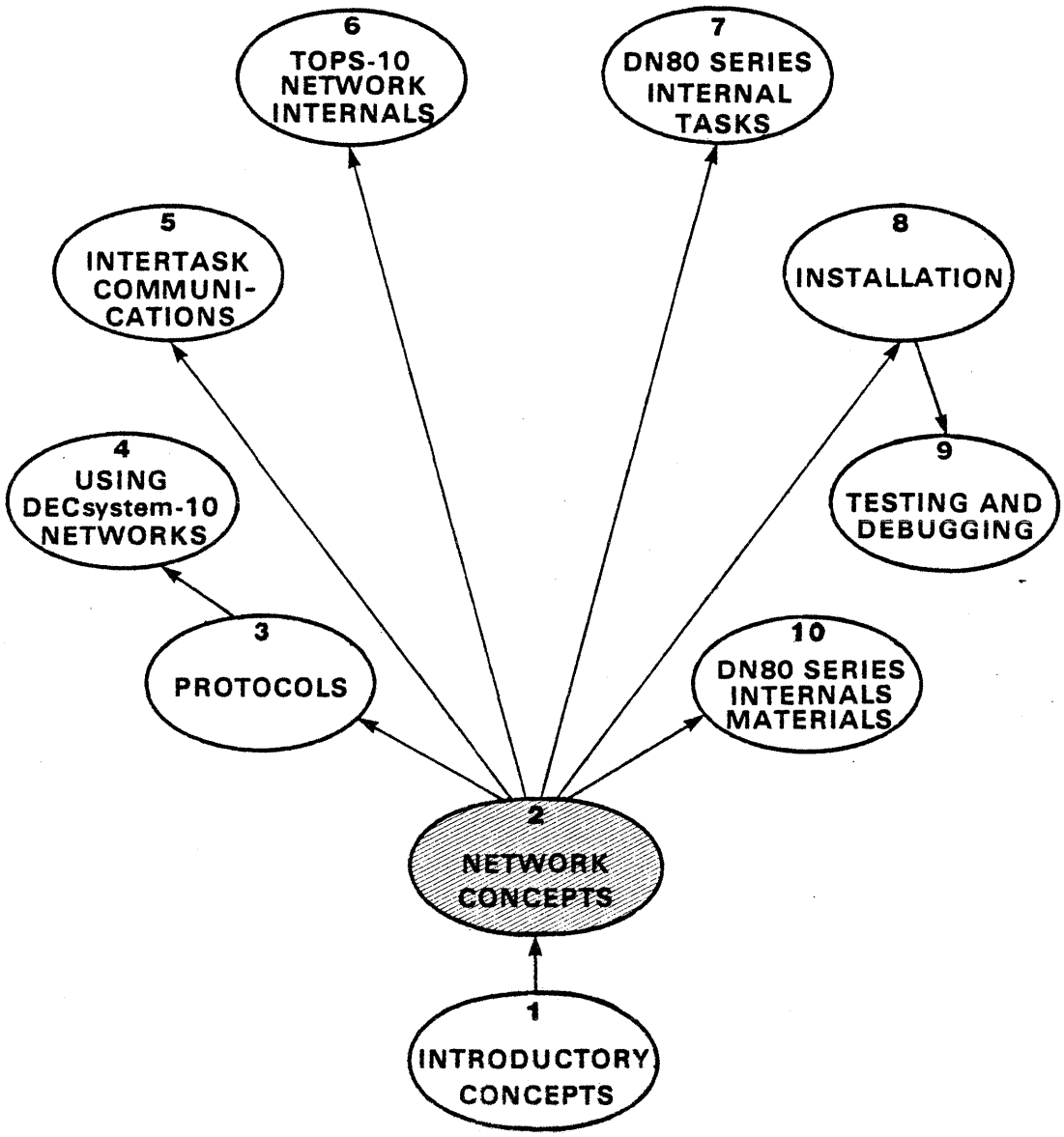
1. Answers may vary. Consult instructor with your answer.

2. 1 - d
   2 - b
   3 - c
   4 - e
   5 - g
   6 - h
   7 - k
   8 - j
   9 - a
   10 - f
   11 - i

This page intentionally left blank

Network Concepts
Module 2

Course Map



M8 0277

Introduction

This module covers the current line of DECsystem-10 communications products. The module opens with a general discussion of DECsystem-10 networks; then, presents in catalogue form the configuration and capabilities of these communications products.

Objectives

Upon completion of this module, the student will be able to

1. Recognize and describe the capabilities of the various DECsystem-10 communications products.
2. Describe the various topologies of networks.
3. Describe the major functions of remote computing.
4. Describe differences and relative advantages of synchronous and asynchronous communication techniques.

Additional Resources

DECsystem-10 Technical Summary

DECsystem-10 Networks Definition and Discussion

Asynchronous communications equipment, with associated terminals, serves a broad spectrum of user needs within the DECsystem-10 interactive environment. These needs include interactive program development, operator control of the system, program production and control, data entry, program and data preparation, interactive problem solving, student instruction, and information storage and retrieval, to name a few. Asynchronous communications is used to link the DECsystem-10 with terminal equipment of three types: hard copy such as the LA30 or LA36 DECwriters, CRT terminals such as the VT52 DECscope, and buffered terminals such as the VT61.

Local, current loop terminals within 1500 feet of the computer site can be connected over dedicated hard-wired lines. Remote terminals located beyond this distance are connected over dedicated or dial-up EIA telephone lines.

Speeds of asynchronous hard-copy terminal equipment generally range from 10 to 120 characters per second. CRT terminals range in speed from 10 to 9600 characters per second, depending upon their communications interface and type of line used for transmission.

The goal of an interactive terminal system is to match all users to the system in such a way that they feel the terminal is identified closely with the computer. Consequently, the machine becomes an easy and natural extension of each application. To do this, the computer must be a willing partner in the interchange so that the user feels that he is in control of the terminal, and not the reverse. Typically, a user types input sporadically, occasionally makes mistakes, and reads fast on output. The Data Communications system for the DECsystem-10 has been engineered to deal with these user characteristics.

The system, which includes hardware and software elements, accepts characters at uneven rates, allows for single character or line correction of mistakes, and allows the user to type at his own speed - even typing ahead of the computer's response to his characters. When users are entering large volumes of data and need no prompting, the system accumulates data and anticipates ongoing entries so that the user can continue to type despite fluctuations in the system's response.

Features of DECsystem-10 asynchronous communications include the ability to delete one or more characters, delete entire lines, retype lines and characters, interact on a character or complete line basis (depending upon the application), suppress unwanted

output, and make use of prompting messages to call for input. Terminals may communicate with the system administrator or operator at the central or remote sites as well as with other terminals.

Synchronous Communications

DECsystem-10 synchronous communications provides essentially error-free, high-speed paths between the central computer and remote stations or other computer systems. DECsystem-10 synchronous interface equipment controls transmission over high-speed synchronous lines. This transmission is on a message basis in contrast with the character-by-character basis of lower-speed, asynchronous transmission.

Full duplex protocol software supplied with the DECsystem-10 makes efficient use of high-speed transmission in both directions simultaneously on a full-duplex line. Front end hardware and software handles hardware control, message formatting, and message acknowledgements. The data transmission is "pipelined," a technique which increases line efficiency by fully overlapping the acknowledge-continue signals.

Transmission errors are detected using CRC-16. Data errors are corrected through retransmission of the erroneous block. Hardware is available to interface with a broad range of communications modems.

Synchronous versus Asynchronous Transmission

Asynchronous transmission is employed to transmit data on a character-by-character basis; synchronous transmission is preferred for moving large blocks of data. Asynchronous transmission, at 2400 bits/sec, is limited to conveying no more than 240 eight-bit characters per second; synchronous transmission, at 2400 bits/second, can convey 300 eight-bit characters per second. This difference comes about because synchronous transmission uses block-oriented clock synchronization, while asynchronous transmission re-establishes its character-oriented clock synchronization with each character. Thus, in the asynchronous case, each eight-bit character requires, minimally, one start bit and at least one stop bit in addition to its data bits.

The most important distinction between asynchronous and synchronous methods of transmitting data is the means of error detection and recovery. Error detection for asynchronous devices

is normally accomplished by insertion of a parity bit into the data field of each character transmitted. (Note that DEC DDCMP uses 8 bits for data in asynchronous mode.) Half-duplex (local character echo) receivers will typically check this parity bit and notify the sender of the existence of an error. In full-duplex systems, such as the DECsystem-10, the character echo is generally provided by the communications computer (DN87, DN80, etc.) to which the user is immediately attached. In this case, transmission error detection and corrective action are thus the responsibility of the sender, who simply compares what was sent with what the computer says it received. The communications computer which echoes the character to the user then undertakes to deliver that character, without error, through the network, to the appropriate DECsystem-10 host.

This mode of error detection is highly efficient and fully acceptable for interactive, terminal-oriented systems, but is less efficient and generally not acceptable for message-oriented communications with other computers. Synchronous communication normally uses a block-oriented error detection technique, such as the Cyclic Redundancy Check (CRC-16) polynomial. This technique is most efficient for block-oriented data transmission and, for this reason, communications with remote computers is accomplished using synchronous data transmission. Note that DDCMP uses CRC-16 in both asynchronous and synchronous modes.

Remote Communications

Prior to the introduction of DECsystem-10 remote communication concentrator products, each remote user had been individually linked to the computer center by separate long-distance telephone lines. The only device that the remote user had available at his location was the terminal. This user was able to utilize available devices at the central station, but could not obtain high volume output at the remote site. Either the user had to travel to the central station to obtain a listing, or had to have the listings delivered. However, with remote communications hardware and software, listing files and data can be sent via a single synchronous full-duplex line to a small remote computer. That remote computer then services its terminals, card readers and line printers. The remote computer and its associated peripherals constitute a remote station.

Remote station use of the central computer is essentially the same as local use. All programs and peripherals available to local users at the central computer station are also available to remote users. In addition, the remote user can access devices located at the remote station or at another remote station. Local

users at the central station can also make use of the peripherals at remote stations. Therefore, by specifying a station number in appropriate commands to the operating system, each user of the DECsystem-10 is given considerable flexibility in allocating system facilities and in directing input and output to the station or DECsystem-10 of their choice.

The DECsystem-10 allows for simultaneous operation of multiple remote stations. Software provisions are incorporated in the operating system to differentiate one remote station from another. By utilizing peripheral devices at various stations, the user is provided with increased capabilities. For example, data can be collected from various remote stations, compiled and processed at the central station, and then the results of the processing can be sent to all contributors of the data.

Simple Topologies

The most basic network topology is a point-to-point topology. It is best illustrated by the following diagram.

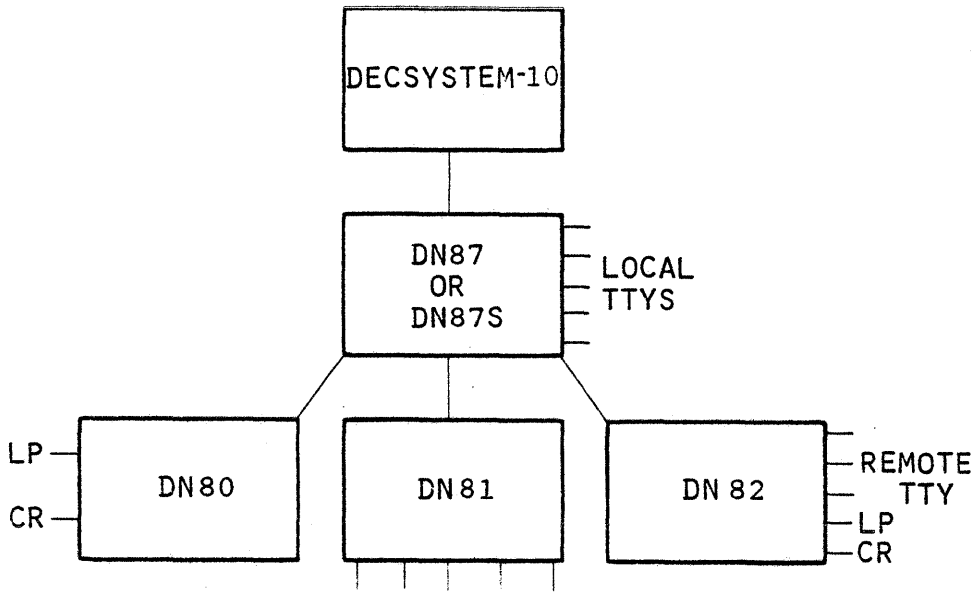


Figure 2-1

M8 0251

Complex Topologies

This is the name given to a group of general network building capabilities which, when put together, allow the user to make many different cost, performance and high availability trade-offs. These building block capabilities are:

Route-through

This is the ability for a Remote Station to send information, via the DN87(S), to a DECsystem-10 to which it is only indirectly attached - i.e., indirectly through another Remote Station or Universal Front End. This is most easily illustrated in the following diagram:

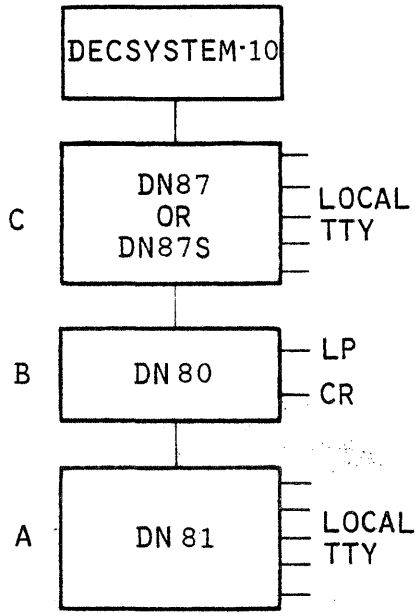


Figure 2-2

M8 0114

Information flows from A to C only by being routed through B. This is a very useful price performance trade-off since the cost of a communications line from A to B and from B to C may be substantially cheaper than the cost of lines from A to C and from B to C.

Multi-pathing

This is the ability to have more than one path between nodes in the network. In general, these links are automatically load leveled depending upon the relative speed and error rate of the lines. This is best illustrated by the following diagram:

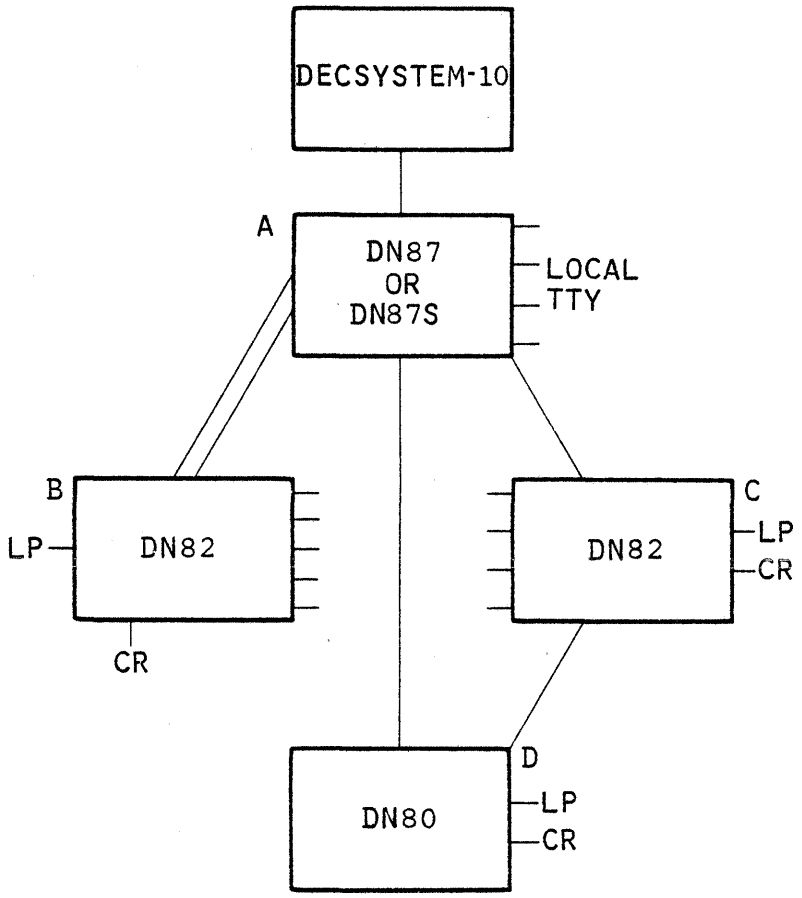


Figure 2-3

M8 0113

Note that multi-pathing is not supported in Release 6.03 of the TOPS-10 monitor; however, any DN8x* may communicate with any other DN8x, or DC72NP on a network.

* DN8x is the generic name for the -80 series systems which are DC75NP, DN80, DN81, DN82, DN85, DN87, and DN87S.

DECsystem-10 Communications Products Overview

The DECsystem-10 data communications products are based on the DN87 Universal Synchronous/Asynchronous Communications Front End Subsystem. The DN87 is configurable in asynchronous only mode (up to 112 asynchronous lines), synchronous only mode (up to 12 synchronous lines) or with a mixture of synchronous and asynchronous lines in the same DN87 with an aggregate up to 40.8Kb. These three modes allow the DN87 to be configured cost effectively in a wide variety of customer specific configurations. This flexibility is the essence of the DN87 Communications Subsystem.

The DN87 is capable of synchronous communication with the DN80 series Remote Stations, the DN92 Remote Station and the DC72NP Remote Station. These allow Remote Job Entry, Remote Concentration of interactive terminal lines or a combination of the two. The DN87, with these Remote Stations, supports complex topologies such as multi-pathing, route-through, and multiple host support. With DECnet-10, it is possible to have a DECsystem-10 communicate with any DECnet system, for example, RSX-11M, RSX-11D, RSX-11S, IAS, RSTS/E, RTS-8, DECsystem-10 or DECSYSTEM-20 if equipped with DECnet software (not currently all available).

The DN87S has the same functionality as a DN87 except that it is attached to the DECsystem-10 via the DTE20 interface rather than the DL10. Up to three DN87S's may be attached to the DTE20 interface on a KL10, one DN87S per DTE. The DN87 and the DN87S require TOPS-10 version 6.03 or later monitor release.

Asynchronous Functionality

On an eight-line group basis, the DN87/87S is capable of terminating 20mA current loop, local EIA or EIA with full modem control type of asynchronous line/terminal interfaces. On an individual line basis, the DN87/87S asynchronous lines can be,

* ASCII Teletype-compatible code or 2741 EBCD or Correspondence Code.
* Full-duplex with echoplex (i.e., echo generated by computer) or full-duplex with local copy (simulated half-duplex).
* Program selectable line speeds from 50 through 9600 baud.

* Split transmit/receive speeds.

* Automatic baud rate detected for 110, 134.5, 150 and 300 bits/second lines.

Some off-loading of the DECsystem-10 host is accomplished because the DN87/87S does the majority of the echoing for asynchronous lines. It does not echo special characters, nor does it echo in certain complex cases. The DN87/87S also does character fill generation.

Synchronous Functionality

The DN87/87S is capable of terminating EIA and/or current-loop type synchronous links. The line speeds may be 2400, 4800, 7200, 9600, 19.2K, 38.4K or 40.8K baud on an individual line basis. These links operate only in full-duplex with simultaneous bidirectional transmission. The synchronous links use DDCMP protocol for error checking and correction and for point-to-point link control.

These synchronous links communicate only with the DC72NP Remote Station (DC72NP is a software-only upgrade of the DC72), the DN80 Series Remote Stations, the DN92 Remote Station or another DN87/87S. On an individual synchronous line basis the DN87/87S can also communicate on a task-to-task basis with systems running DECnet software.

OPTIONS LIST

The DN87 is the Universal Synchronous/Asynchronous Front End Communications Subsystem. Software providing complex topology support is included with the DN87. the DN87 requires the addition of DN81-xx synchronous and/or asynchronous line options.

Interface Options

Component	Function
DN87-DA,DB	Including DL10-C port only. Requires DL10-A with availabe port-slot. Note that this is what would be ordered with a 1080 or 1060 System Pacakge which includes a DL10.
DN87-AA,AB	DN87 including DL10-A Communications Interface and one DL10-C port.
DN87S-AA,AB	DN87 with DTE20 interface, (requires a 1090; limit of 3 per KL10.)

Asynchronous Line Options

DN81-EA,EB	Asynchronous Expansion Cabinet including one DN81-EC 16-Line Asynchronous Expansion Group. Requires two DN81-Fx 8-Line Terminator Groups to activate the lines.
DN81-EC,ED	Asynchronous 16-Line Expansion Group. this requires two DN81-Fx 8-Line Terminators to activate the lines.
DN81-FA	8-Line Terminators each with 20 mA Current Loop Local Interfaces.
DN81-FB	8-Line Terminators each with EIA Local Interfaces.
DN81-FC	8-Line Terminators each with EIA Full Modem Control Interfaces.
DN81-FD	8-Line Terminators with Integral Auto Answer Modems. (These modems require customer supplied DAA's.)

Synchronous Line Options

DN81-EE,EF Synchronous Expansion Cabinet. Includes one DN81-H.

DN81-H Synchronous Line Controller Expansion Line. For data transmission speeds of up to 10K baud and for attachment to EIA RS232C compatible modems.

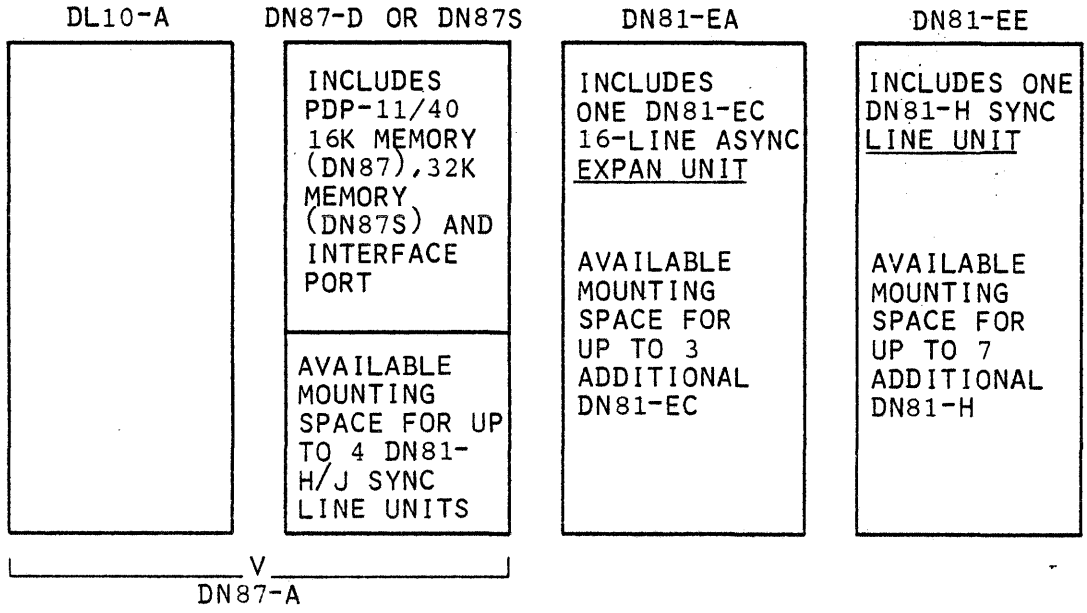
DN81-J Synchronous Line Controller. For data transmission speeds of up to 40.8K baud and for attachment to 303-type current mode modems.

NOTE

Option designators are such that, if there are two designations, the first is 115V/60Hz and the second is 230V/50Hz. If there is only a one option designation, that option is NOT power dependent.

Hardware Configuration Guidelines

A pictorial representation of the DN87 cabinet arrangements should help to simplify the explanation of the configuration rules.



LINE MAX PER DN87

MAX NO. OF SYNC LINES	MAX NO. OF ASYNC LINES
0	96
2	64
6	32
10	0

LINE MAX PER DN87S

MAX NO. OF SYNC LINES	MAX NO. OF ASYNC LINES
Ø	112
4	64
8	32
12	0

Figure 2-4

M8 0112

Asynchronous Configuration Rules

* The DN87 cabinet (DN87-A or DN87-D) has no available mounting space for asynchronous lines.

* If any asynchronous lines are required, a DN81-EA Asynchronous Expansion Cabinet must be ordered. This cabinet includes one DN81-EC Asynchronous 16-Line Expansion Group and has available mounting space for up to three additional DN81-EC units. Therefore, the DN81-EA is capable of containing up to 64 asynchronous lines.

* A maximum of two DN81-EA Asynchronous Expansion Cabinets are permitted per DN87.

* Each DN81-EC Asynchronous 16-Line unit requires two DN81-Fx (FA, FB, FC, or FD) 8-Line Terminators to activate the lines.

Synchronous Configurations Rules

* The DN87 cabinet (DN87-A or DN87-D) has available mounting space for up to four DN81-H/J synchronous line units.

* If more than four Synchronous Line Units are required, a DN81-EE Synchronous Expansion Cabinet must be ordered. This cabinet includes one DN81-H Synchronous Line Unit, and available mounting space for up to seven additional DN81-H units.

* Proper number of asynchronous lines are made by using the Asynchronous Configuration Rules from above.

* Proper number of synchronous lines are made up using the Synchronous Configuration Rules from above.

DN85 Synchronous Communications System (no longer offered)

The DN85 offers a general-purpose synchronous communications capability as well as support of the DN80 series remote stations.

Features:

* PDP-11/40 multiplexer controller.
* Direct-to-memory synchronous line interfaces.
* Full-duplex Digital Data Communications Message Protocol (DDCMP) support.
* Supports DN92, DC72NP and DN8x series remote stations, as well as general-purpose synchronous communications links.
* User programs in separate DECsystem-10 systems may communicate with each other through the DN85.
* Individual line speeds of up to 40.8K bits/second.
* Up to 16 lines may be added; total line speed of up to 40.8K bits/second supported.
* Provision for adding up to three additional DN85-D controllers, each with up to 16 synchronous lines and a total line speed of 40.8K bits/second.
* Uses CRC-16 Error Detection Polynomial.

The DN85 Synchronous Communications System is recommended for applications involving DN80-series remote stations and/or general purpose synchronous communications to remote computers. It uses high-performance single synchronous line interfaces which are direct-to-memory instead of the interrupt-per-character used by the DS11. This significantly lowers the processor overhead associated with servicing each line, enabling single line speeds of up to 40.8K bits/second.

The DN85 uses a PDP-11/40 central processor. Using the DN85, the DECsystem-10 fully supports the DN80-series of remote stations and DC72 stations having the DC72-NP option. With additional software, DN85 may be used for communications between remotely located DECsystem-10's. By employing the DN85, for example, a user program in one DECsystem-10 may pass data files, etc., to a user program in another DECsystem-10. The DN85 uses the full-duplex DDCMP protocol (Digital Data Communications Protocol)

supported by DIGITAL in many of its operating systems. This provides a highly flexible, easy to implement communications protocol for communications with general-purpose remote computers, as well as full support of DN80, 81, and 82 remote stations. A DDCMP field upgrade kit is also available for installations having DC72's (DC75-NP).

The DDCMP synchronous communications protocol supported in the DN85 is a highly efficient, full-duplex message oriented protocol which is well suited for support of communications links to interactive remote stations (such as DC72NP, DN80, 81, and 82) and remote general-purpose computers. Its full-duplex features allow true simultaneous two-way communications over dedicated four-wire communications lines with appropriate modems. This more than doubles the throughput on a communications link over that which is possible using more conventional half-duplex protocols. Compatible modems include the Bell System 201, Bell System 203, Bell System 208, Rixon PM24, ICC Modem 2200, or any synchronous modem which conforms to the Electronics Industries Association RS-232B or C Computer Interface Standards as well as CCITT V.24 (white book) standards.

Component	Function
DN85-A	Synchronous Communications Multiplexer. Provides a PDP-11/40 multiplexer controller and a DL10-A high-speed interface to DECsystem-10 memory. Individual synchronous lines, up to a maximum of 16 are added using DN85-F and -G line interfaces. Maximum single-line speed supported in 40.8K bits/second; maximum total line speed supported is 40.8K bits/second. Includes mounting space for the first three DN85-F and/or DN85-G single-line synchronous interfaces.
DN85-D	Synchronous Communications Multiplexer. Same as DN85-A except that only a DL10-C UNIBUS port for a DL10-A is provided. Prerequisite is a DC75-A, DC76-A, DN78-A, DN79-A, or DN85-A. Total line speed capacity per DN85-D is 40.8K bits/second; maximum single-line speed supported is 40.8K bits/second. Includes mounting space for the first three DN85-F and/or DN85-G single-line synchronous interfaces.
DN85-E	Expander Cabinet. Provides mounting space for up to 11 DN85-F and -G line interfaces. Basic DN85-A and DN85-D provide mounting space for the first three DN85-F and/or -G line interfaces.
DN85-F	Single-line Synchronous Interface. Provides a single full-duplex synchronous line interface for the DN85-A or DN85-D; up to 16 are supported. The DN85-F interfaces to EIA-RS232 or CCITT modems of the Bell Type 200, 201, 203, 208 or equivalent at speeds of up to 10K bits/second. Modem control is included. Includes a 25-foot (7.5 meter) cable to customer-supplied modem.
DN85-G	Single-line Synchronous Interface. Provides a single full-duplex synchronous line interface for the DN85-A or DN85-D; up to 16 are supported. The DN85-G interfaces to modems of the Bell Type 301, 303 or equivalent at line speeds of up to 40.8K bits/second. (Note that maximum single-line speed and total line speed capacity of DN85 is 40.8K bits/second.) Includes modem control and 25-foot (7.5 meter) cable to customer-supplied modems.

DN80 Series Remote Station

The DN80 Series Remote Stations are a family of remote stations with a wide range of functionality. The DN80 is a Remote Job Entry Station with a 300 cpm Card Reader, a 300 lpm Line Printer and an LA36 Console. The DN81 is a Remote Concentrator capable of concentrating up to 32 asynchronous lines. The DN82 is a combination of the DN80 and DN81, facilitating both RJE and Remote Concentration.

When linked to a DN87/DN87S Universal Front End the DN80 Series Remote Stations will support complex topologies such as route-thru, multipathing, and multiple host support.

Asynchronous Functionality

The asynchronous terminal lines attached to the DN80 Series Remote stations have the same functionality as the asynchronous lines attached directly to the DN87/DN87S.

Synchronous Functionality

The synchronous lines are used as a full-duplex dedicated line to a DC75NP, a DN92, another DN80 series remote station or the DN87/DN87S DECsystem-10 Front End. The DN80 Series Remote Stations are all downline loadable from the DECsystem-10 through the synchronous link.

Unit Record Devices

The Line Printer (LP) and the Card Reader (CR) on a DN80 or DN82 Remote Station are available from the DECsystem-10 host in a transparent manner. This means that any user at any terminal can use this remote LP or CR much like he would use a LP or CR attached directly to the host. This is accomplished by use of the DECsystem-10 Monitor Command LOCATE which sets the default LP and CR locations for a specific job.

The LA36 DECwriter, which is included with the DN80 Series Remote stations, can be used as the Operator Controlling Console for the spoolers that are running for the LP and CR at that station. This means that any operator-type requests (e.g., LP out of paper) appear at the remote location with the LP and CR rather than at the host.

Host Interfaces

The DN80 Series Remote Stations can only communicate with the DECsystem-10 via the DC75NP, DN92, or another DN80 series remote station or the DN87/DN87S Front Ends. The DN87/DN87S requires version 6.03 or a later version of the TOPS-10 monitor, whereas the DC75NP and the DN85 require the 6.02A or later monitor. The DC75 Front End will not run under 6.02 or later monitors unless it is equipped with the "NP" upgrade (additional 4K memory, KG11 CRC unit and software).

The DN81 Remote Concentrator provides remote concentration for up to 32 interactive asynchronous terminals for the DECsystem-10. It enables a significant savings in line-use costs by using a single leased or private full-duplex synchronous communication line to the DECsystem-10 instead of one asynchronous line per terminal. User terminals can employ local communications lines to connect to the DN81 concentrator. The DN81 concentrator in turn is connected to the remote DECsystem-10 with a single high-speed synchronous communications line.

The DN81 includes one DN81-EC 16-line group (a maximum of two are supported). Lines are activated in groups of eight using the DN81-FA, FB, FC, and FD line groups. The DN81 hardware and software support of asynchronous terminals is equivalent to that provided by the DC76 local asynchronous concentrator, except that total throughput is limited to 960 characters/second.

The DN82 Remote Station is the combination in a single station of the DN80 Remote Station and the DN81 Remote Concentrator. It can be ordered as a DN82; or, a DN80 or DN81 may be field-expanded at a later date to become a DN82.

All of the DN80 remote stations use the DDCMP synchronous communications protocol to support full-duplex communications with the DECsystem-10 via leased or private four-wire synchronous lines. DDCMP provides highly efficient, full-duplex, simultaneous bi-directional transmission of data to make optimum use of available communications-line band width. DDCMP uses the CRC-16 algorithm to provide automatic error checking and an efficient scheme for message re-transmission in the case of line errors.

Support of DN80-Series Remote Stations by the DECsystem-10 is provided by the DN85/DN87(S) Synchronous Line Multiplexer or the DC75NP Synchronous Line Multiplexer. The maximum synchronous line speed supported by the DN80-Series Remote Stations is 9600 bits/second, full duplex.

Compatible modems include the Bell System 201, Bell System 208, Rixon PM24, ICC Modem 2200, or any synchronous modem which operates full-duplex and conforms to the Electronics Industries Association RS-232B or C Computer Interface Standards as well as CCITT V.24 (white book) standards.

OPTIONS LIST

Component	Function
DN80	Remote Batch Station. Includes PDP-11/40 controller, KG11 CRC-16 arithmetic unit, 300-cards/minute card reader, 300-lines/minute, 64-character line printer, and full-duplex synchronous line interface for transmission speeds of up to 9600 bits/second. The DN80 requires a modem which must be end-to-end compatible with the modem on the DN75NP, DN85 or DN87(S).
DN81	Remote Concentrator. Includes a PDP-11/40 controller, KG11 CRC-16 arithmetic unit, a DN81-EC asynchronous 16-line group, and a full-duplex synchronous line interface for transmission speeds of up to 9600 bits/second. Asynchronous lines are added to the DN81 using DN81-FA, FB, FC, and FD eight-line groups and a DN81-EA 16-line expansion group if more than 16 asynchronous lines are required. A maximum of four DN81-FA, FB, FC, and FD eight-line groups are supported (with one DN81-EC and one DN81-EA). The DN81 requires a modem which must be end-to-end compatible with the modem on the DC75NP, DN85 or DN87(S).
DN82	Remote Station. Includes a PDP-11/40 controller, KG11 CRC-16 arithmetic unit, 300-card/minute card reader, 300-lines/minute, 64-character line printer, DN81-EC asynchronous 16-line group, and a full-duplex synchronous line interface for transmission speeds of up to 9600 bits/second. Asynchronous lines are added using DN81-FA, FB, FC, and FD eight-line groups and a DN81-EA 16-line group if more than 16 asynchronous lines are required. A maximum of four DN81-FA, FB, FC, and FD eight-line groups are supported (with one DN81-EC and one DN81-EA). The DN81 requires a modem which must be end-to-end compatible with the modem on the DC75NP, DN85 or DN87(S).

OPTIONS LIST

DN80 Series Basic Stations

Component	Function
DN80-AA,AB	Remote Batch Station. Includes 1 DN81-H, LA36 Console, DN80-CA/CB, DN80-LA/LB, PDP-11 and software.
DN81-AA,AB	Remote Concentrator. Includes PDP-11, 1 DN81-H, DN81-EA and software. One more DN81-EC can be added for a total of 32 asynchronous lines.
DN82-AA,AB	Remote Batch Station and Concentrator. Includes PDP-11, LA36 Console, DN80-CA/CB, DN80-LA/LB, DN81-81H, DN81-EA and software. DN81-Fx asynchronous interfaces are also required and must be ordered separately.

Unit Record Options

DN80-CA,CB	Card Reader, 300 cards per minute.
DN80-LA,LB	Printer, 300 lines per minute, 132 printing positions, 64 printing characters, EDP character set.
DN80-LC,LD	Printer, 230 lines per minute, 132 printing positions, 96 printing characters, EDP character set.

Asynchronous Line Options

Component	Function
DN81-EA,EB	Asynchronous Expansion Cabinet including one DN81-EC 16-Line Asynchronous Expansion Group. Requires two DN81-Fx 8-Line Terminator Groups to activate the lines.
DN81-EC,ED	Asynchronous 16-line group and expander cabinet (included in DN81 and DN82). Prerequisite is a DN80. Requires one or two DN81-FA, FB, or FD eight-line groups to activate lines.
DN81-FA	20-mA current loop line group. Provides eight 20-mA active local lines to a DN81-EA,B or DN81-EC,B. Includes eight M908 split-lug connectors to customer-supplied 4-wire cable from individual terminals.
DN81-FB	Local EIA line group. Provides eight local (data only) EIA lines to a DN81-EA,B or DN81-EC,B. Includes eight 25-foot cables to H312 null modems (included) which connect directly to local EIA-compatible terminals. May also be used for connection to modems if modem control is not required (not recommended).
DB81-FC	Full modem control EIA interface. Provides eight EIA lines with full modem control to a DN81-EA,B or DN81-EC,D. Includes eight 25-foot cables to customer-supplied modems. (See following list of compatible modems.)
DN81-FD	Integral answer-only modem interface, including modems. Provides eight automatic-answer, full-duplex, 300-baud, answer-only integral modems to a DN81-EA or DN81-EC, and 25-foot cables to customer-supplied Bell 1001A data access arrangements (USOC code "CBS") or equivalent for dial line use. These modems are end-to-end compatible with Bell System 103A2 modems and Bell System 103E modems in originate mode, Bell System 113A modems, and DEC DF11-BA originate-only modems. For long local circuits, the DN81-FD answer-only integral modem may be connected directly (or, when required, by using a Bell 1000A, USOC code "CDT" data access arrangement) to a terminal equipped with a DEC DF11-BA originate-only integral modem.

Synchronous Line Options

Component	Function
DN81-H	Synchronous Line Controller Expansion Line. For data transmission speeds of up to 9600 bits/sec and for attachment to EIA RS232C compatible modems.
DN81-J	Synchronous Line Controller. For data transmission speeds of up to 40.8K bits/sec and for attachment to 303-type current mode modems.

NOTE

These same line options (DN81-xx) are used on the DN80 series remote stations, as described here, on the DN87 and for DC76 add-ons.

Due to the large and ever increasing number of data sets available, it is not practical to list those which will work satisfactorily with the DN81 or DN82. The following data sets are among those supported (by the DN81-FC):

Bell System 103A-type

Bell System 103E-type data station

DN92 Remote Station

The DN92 Remote Station is a low cost remote job entry concentrator that is attached to the DECsystem-10 via one synchronous link to a DN87/DN87S front end. The DN92 may be configured in one of the following ways:

1. Up to 16 asynchronous terminal lines.
2. Up to 12 asynchronous terminal lines plus a card reader or line printer.
3. Up to 8 asynchronous terminal lines plus a card reader and line printer.

Only one synchronous interface is permitted on a DN92; therefore, the DN92 must be a sequential node. This means it may not do multipathing or route-through. Data to the DN92 may be routed from some other node, but no data may be routed through the DN92 since it is only capable of having one synchronous link.

Options

DN92-AA, AB Basic Unit, includes 16K, PDP-8/A processor, VT52 console terminal, desk, ROM for down-line loading, one synchronous line interface and software. Requires at least one, but a maximum of five optional units listed below:

(a) DN92-EA Asynchronous 4-line multiplexer, including line drivers. The DN92-EA Asynchronous 4-line Multiplexer will accomodate either 20 ma. or EIA lines in any mixture. One out of every four can be full modem control. The baud rate is switch selectable anywhere from 50 to 4800 baud. Maximum of four DN92-EA's per DN92-A.

(b) DN92-CA, CB Card Reader. The DN92-CA, CB Card Reader is a Digital CR8-F for 80-column (only) cards, and operates at 285 CPM. Maximum of one Card Reader per DN92-A.

(c) DN92-PA LA180 Printer. The LA180 operates at 180 characters per second and uses a 96-character set (upper and lower case). The carriage is 132 columns wide. Maximum of one LA180 per DN92-A.

(d) DN92-VA, VD LP05 Line Printer, (64-characters). This is the LP05 (LE8-V) and operates at 300 lines per minute using a 64-character set. The carriage is 132 columns wide. Maximum of one LP05 per DN92-A.

(e) DN92-WA, WD LP05 Line Printer, (96 characters). This is the same as the DN92-VA except that it uses the 96-character set (upper and lower case). Maximum of one LP05 per DN92-A.

DN61 IBM 2780/3780 FRONT END

The DN61 Front End allows IBM 2780 and 3780 emulation and termination. This means that terminals similar to IBM 2780's and/or 3780's can be used as RJE stations into the DECsystem-10 or that the DECsystem-10 can emulate (or simulate) an IBM 2780 or 3780 to an IBM 360/370 computer. The DN61 can handle a maximum of 12 synchronous lines each operating independently with any combination of 2780 or 3780 emulation or termination. The maximum aggregate throughput of a DN61 is 100,000 bits per second (100K baud). The DN61-D is interfaced to the DECsystem-10 via the DL10 and the DN61-S is interfaced via the DTE10. Multiple DN61's are supported per DECsystem-10.

OPTIONS LIST

Component	Function
DN61-DA,DB	IBM 2780 and/or 3780 Emulation and Termination Front End. Including DL10-C port and requiring DL10-A with available port. Requires addition of DN61-H and/or DN61-J synchronous line units.
DN61-SA,SB	IBM 2780 and/or 3780 Emulation and Termination Front End including DTE10 interface. Requires addition of DN61-H and/or DN61-J synchronous line units (1090).
DN61-EA,EB	Synchronous Expansion Cabinet for expansion past 4 synchronous lines. Includes no synchronous lines.

DN62 IBM HASP MULTI-LEAVING FRONT END - (not yet available)

The DN62 Front End combines all of the functionality of the DN61 with support of IBM HASP multi-leaving work stations as DECsystem-10 RJE stations and also allows for the DECsystem-10 to simulate a HASP multi-leaving work station to an IBM host computer. The DN62 can handle a maximum of 12 synchronous lines each operating independently with any combination of 2780 or 3780 or HASP multi-leaving emulation or termination. The maximum aggregate throughput of the DN62 is 100,000 bits per second (100K baud). The DN62-D is interfaced to the DECsystem-10 via the DL10 and the DN62-S is interfaced via the DTE10. Multiple DN62's are supported per DECsystem-10.

OPTIONS LIST

Component	Function
DN62-DA,DB	IBM 2780 and/or 3780 and/or HASP multi-leaving work station Front End. Including DL10-C port and requires DL10-A with available port. Requires addition of DN62-H and/or DN62-J synchronous line units.
DN62-SA,SB	IBM 2780 and/or 3780 and/or HASP multi-leaving work station Front End including DTE10 interface. Requires addition of DN62-H and/or DN62-J synchronous line units.
DN62-EA,EB	Synchronous Expansion Cabinet for expansion past 4 synchronous lines. Includes no synchronous lines.

DC76 Asynchronous Communications System - (no longer offered)

The DC76 system provides a flexible, large-capacity asynchronous data communications capability for the DECsystem-10. The fully expanded DC76 system allows users to connect up to 512 asynchronous terminals with a wide choice of transmission speeds, modem connections, and data codes.

Features

* Up to 128 full-duplex or full-duplex with local copy asynchronous lines per DC76 multiplexer, expandable to a maximum of four multiplexers and 512 lines.

* Individual line speeds program-selectable from 50 to 9,600 baud.

* Split-speed operation program-selectable on a per-line basis.

* Automatic baud detection for 110, 134.5, 150, and 300-baud lines.

* ASCII, EBCDIC, APL, or Correspondence codes independently selectable on each line.

The DC76 Asynchronous Communications System brings a large-capacity broad-spectrum communications capability to the DECsystem-10. The DC76 uses up to four DIGITAL PDP-11/40 processors for multiplexer control and front-end character handling. The PDP-11/40 processors are interfaced directly to the DECsystem-10 main memory via the DL10 PDP-10/PDP-11 Interface.

Each DC76 subsystem can handle up to 128 asynchronous communications lines with a total aggregate line speed of 1,500 characters/second. The maximum speed of any line is 9,600 baud. Input speeds above 2400 baud are not supported by standard software. The DC76 provides for automatic baud rate recognition among 110, 134.5, 150, and 300-baud lines. Other speeds from 50 to 9,600 baud and split (i.e., different) transmit and receive speeds can be obtained by monitor commands and/or by the system management presetting line speeds using monitor initialization features.

The DC76 will support asynchronous terminals which may be eight-level ASCII terminals or seven-level terminals compatible with the IBM 2741, using EBCDIC, APL or Correspondence character sets. ASCII terminals compatible with DIGITAL LT33, LT35, LA30, LA36, LA37, VT05, VT06, VT50, VT61, and VT52 are fully supported. ASR units must include X-ON/X-OFF paper tape reader control for full support; the support of cursor functions on displays is a function of the user program.

The DC76 does not support the "reader-run" control feature found on some Teletypes (LT33-D, LT35-D). ASCII terminals should be eight-unit asynchronous code with one stop unit, except at 110 baud where two stop units are assumed. The eighth data bit is transmitted to the terminal as even-character parity, but is ignored on reception. Five- and six-level codes are accomodated by the hardware (except six-level code with 1.5 stops bits) but are not supported by standard software.

The DC76-FC (DN81-FC) eight-line interface group, in conjunction with the DC76 software, will support any modem with EIA-RS232/CCITT V.24-signal interface and operating characteristics (control level sequencing) compatible with Bell System 103A or 103E modems, providing that the modem at the DC76 end of the circuit is end-to-end compatible with the modem at the user terminal end of the circuit.

The DC76 software supports full duplex (two-way simultaneous) line operation. Full duplex with local copy is also supported. Two-way alternate (the usual mode of Bell 202 modems) is not supported. However, the "almost two-way" alternate operation of 2741 and 2741-compatible terminals is supported over full-duplex lines. Polled operation (using Bell 103F-type modems) is not supported with either full-duplex or two-way alternate simplex modems. The "make-busy" feature provided by Bell 103E, 113B and other similar modems is not supported.

Due to the large and ever increasing number of data sets available, it is not practical to list those which will work satisfactorily with the DC76. The following data sets are among those supported by the DC76-FC (DN81-FC):

Bell System 103A-type

Bell System 103E-type data station

Bell system 103F-type (but not polled or multipoint operation)

Bell System 113B-type data station

Bell System 202 full-duplex (4-wire) mode only

DIGITAL DF11-BB-type full-duplex, receive-only integral modem.

OPTIONS LIST

Component	Function
DC76-A (DN81-A)	Asynchronous Communications Multiplexer. Includes a DL10-A high-speed interface to DECsystem-10 memory, PDP-11/40 multiplexer controller, and one DC76-E 16-line group. Requires up to two DC76-FA, FB, FC and/or FD 8-line groups to activate lines. Up to 7 additional DC76-E's may be added to the DC76-A for a total of 128 synchronous lines (over 64 lines require a DC76-EC). Total line speed capacity is 1500 character/second.
DC76-D (DN81-D)	Asynchronous Communications Multiplexer. Same as the DC76-A, except that only the DL10-C UNIBUS port for a DL10-A is included. Prerequisite is a DC75-A, DC76-A, DN78-A, DN79-A, or DN85-A. Up to three DC76-D, may be added to a DC76-A for a total of 512 asynchronous lines. Total line speed capacity per DC76-D is 1500 characters/second.
DC76-E (DN81-E)	16-line Group. Provides up to 16 additional lines for the DC76-A or the DC76-D. Up to 7 DC-E's may be added per DC76-A or DC76-D, DC76-FA, FB, FC, and FD 8-line group line interfaces are required to activate DC76-E lines. The DC76-E does not increase DC76-A or DC76-D line speed capacity. The basic DC76-A and DC76-D provide mounting space for the first four DC76-E line groups (64 lines). The fifth DC76-E for lines 65-80 must be the DC76-EC below.
DC76-EC (DN81-EC)	DC76-E 16-line Group and Expansion Cabinet. Required for over 64 lines. Includes DC76-E 16-line group and mounting space for three additional DC76-E 16-line groups (lines 81-128).
DC76-FA (DN81-FA)	20-mA Current-loop Line Group. Provides eight 20-mA active local lines to a DC76-E. Includes 8 M908 split-lug connectors to customer-supplied cable from individual terminals (four wires or two pairs per terminal).

DC76-FB (DN81-FB) Local EIA Line Group.
Provides eight local (data only) EIA lines to a DC76-E. Includes eight 25-foot cables to H312 null modems (included) which connect directly to local EIA-compatible terminals. May also be used for connection to modems if modem control is not required (not recommended).

DC76-FC (DN81-FC) Full Modem Control EIA Interface.
Provides eight EIA lines with full modem control to a DC76-E. Includes eight 25-foot cables to customer-supplied modems. (See above for partial list of compatible modems.)

DC76-FD (DN81-FD) Integral Answer-only Modem Interface (including modems).
Provides eight automatic-answer, full duplex, 300-baud, answer-only integral modems to a DC76-E and 25-foot cables to customer-supplied Bell 1001A data access arrangement (USOC code "CBS") or equivalent for dial line use. These modems are end-to-end compatible with Bell System 103A2 and 103E modems in originate mode, Bell System 113A modems, and DEC DF11-BA originate-only modems. For long local circuits, the DC76-FD answer-only integral modem may be connected directly (or, when required, by using a Bell 1000A, USOC Code "CDT" data access arrangement) to a terminal equipped with a DEC DF11-BA originate-only integral modem.

DC76 Baud Rates

50	150*	1200
75	200*	1800
110*	300*	2400
134.5*	600	4800**
		9600**

* Supported by automatic baud rate detection.

** May be program selected, but are supported by standard software for output only. Use split speed (300 input, 4800 output) if desired.

DC75-NP Synchronous Communications System - (no longer offered)

The DC75-NP Synchronous Communications System provides a highly reliable, high-speed path between the central DECsystem-10 computer and DC72-NP Remote Stations and/or other computer systems.

Features

* Eight full-duplex synchronous lines, expandable to 16 lines per DC75.

* Individual line speeds of 2400, 4800, or 9600 bits/second.

* Total line speed of 40K bits/second per DC75NP.

* Provision for adding up to three additional DC75-D controllers, each with 16 synchronous lines and a total line speed of 40K bits/second in full duplex.

The DC75-NP is recommended for applications involving up to 8 synchronous lines to DC72-NP remote stations. Each DC75-NP Controller handles a total line speed of up to 40.8K bits/second including error checking, formatting and line control. Individual lines may operate at speeds up to 9,600 bits/second.

The DC75-A consists of a high-speed interface to the DECsystem-10 memory bus (DL10), PDP-11 multiplexer controller, and multiple line synchronous multiplexer (DS11). The DC75-NP multiplexer controller packs and unpacks characters directly into the DECsystem-10 memory and can execute instructions from the DECsystem-10 memory for bootstrap operations. The DS11 multiple line synchronous interface handles 8 full-duplex lines and can be expanded to handle up to 16 full-duplex lines.

The basic DC75-A is capable of supporting up to three additional PDP-11 multiplexer controllers, which may be any combination of DC75-D's or DC76-D's, DN61-D's, DN85-D's or DN87-D's. Compatible modems include the Bell System 201, Bell System 208, Rixon PM24, ICC Modem 2200, or any synchronous modem which conforms to the Electronics Industries Association RS-232B or C Computer Interface Standards as well as CCITT V.24 (white book) standards and operates full-duplex.

OPTIONS LIST

Component	Function
DC75-A	Synchronous Communications Multiplexer. Provides a PDP-11 multiplexer controller, a DL10-A high-speed interface to DECsystem-10 memory, and a DS11 eight-line synchronous line multiplexer. Total line speed is 40.8K bits/second; maximum single line speed is 9600 bits/second.
DC75-D	Synchronous Communications Multiplexer. Same as the DC75-A except that only a DL10-C UNIBUS port for a DL10-A is included. Prerequisite is a DC75-A, DC76-A, DN78-A, DN61-A, DN85-A, or DN87-A. Total line speed capacity per DC75-D is 40.8 bits/second.
DC75-E	Incremental Eight-line Group for DC75-A or DC75-D. Provides up to 8 additional synchronous lines on a DC75-A or DC75-D, for a total of 16 lines. The DC75-E does not increase the total line speed capacity of the DC75-A or DC75-D. It remains at 40K bps; i.e., 16 lines at 2400 bps. Only one DC75-E can be added to a DC75-A or DC75-D (Prerequisite is a DC75-A or DC75-D).
DC75-NP	Upgrade Option for installed DC75-A and -D systems. Implements DDCMP synchronous communications protocol compatibility with the DN85/DN87(S).

DC72-NP Remote Station - (no longer offered)

The DC72-NP Remote Station and options make DECsystem-10 peripherals available to any remote site that can be connected to a DECsystem-10 by a leased synchronous communication line. The remote peripherals supported include 110- to 2400-baud asynchronous ASCII terminals (e.g., the LA30 DECwriter, VT05 Video Display Terminal,) line printers, and card readers. These remote peripherals are functionally equivalent to their local counterparts.

Features

* Full duplex, bi-directional, simultaneous transmission.

* Card reader, 300 cards/minute.

* Line printer, 132 columns, speeds from 165 characters/second up to 300 lines/minute. * Concentrates up to 16 local terminals at speeds up to 2400 bits/second.

* Uses leased synchronous lines at speeds up to 4.8kB.

* Meets EIA-RS232-C or CCITT V.24 (white book) modem interface standards.

The DC72-NP Remote Stations provide remote users of a DECsystem-10 with a full set of user-oriented input/output peripherals at prices comparable to a conventional RJE terminal. In addition to the RJE capability, the DC72-NP provides an interactive terminal for operator control, dedicated applications, and general interactive use. The DC72-A, B and C basic stations include a 10-character/second operator console, a 300-card/minute reader and a 132-column line printer.

The DC72-A features a 165-character/second, 64-character-set printer which gives dot matrix printing and a two-channel vertical-format control. The DC72-B has a faster drum printer with a 64-character set. The speed of this printer is 300 132-column lines per minute. The vertical format unit is a single-channel unit switch-selectable for page sizes of 3, 3.5, 4, 5.5, 6, 7, 8, 8.5, 11, 12, and 14 inches. The DC72-C is similar to the DC72-B, but offers a 96-character set line printer that includes lower-case letters and additional symbols; the printing speed is 230 132-column lines/minute.

The DC72-L asynchronous eight-line group is used to add eight asynchronous terminal interfaces to any of the DC72 series. The DC72-L enables line speed to be individually selected for each line. Terminal output rates of 110 to 2400 baud and transmission rates of 110 to 300 baud are supported; 134.5-baud terminals of the IBM 2741 types are not supported on the DC72-NP. User programs which use cursor movement commands must supply fill characters at 600 baud and above.

Two DC72-L's (16 lines maximum) can be added to each DC72-A, B, or C. Terminals may be connected to the DC72-L either locally using 20-mA current loops or through EIA RS-232C or CCITT V.24 (white book) standard asynchronous modems. Modem control is not available on the DC72-L, however.

The supporting software for the DC72-NP is an evolution of the DC71 software first delivered in 1971. It takes full advantage of full-duplex communication with the DECsystem-10 to run both reader and line printer simultaneously, while at the same time providing good interactive response for up to 16 asynchronous terminals.

For communication with the DN85/DN87, the DC72-NP uses the standard DDCMP communication protocol (requires DC72-NP). Compatible modems include the Bell System 201, Bell System 208, Rixon PM24, ICC Modem 2200, or any synchronous modem which conforms to the Electronics Industries Association RS-232B or C Computer Interface Standards as well as CCITT V.24 (white book) standards and operates full-duplex.

OPTIONS LIST

Component	Function			
	Console	Card Reader	Line Printer	Terminals
DC72-A	10 characters /second	300 cards /minute	64-character set 165 characters /second dual-channel vertical forms control	Up to 16 with DC72-L asynch-ronous line groups
DC72-B	10 characters /second	300 cards /minute	64-character set 300 lines/minute, preset single-channel vertical forms control	Up to 16 DC72-L asynch-ronous line groups
DC72-C	10 characters /second	300 cards /minute	96-character set 230 lines/minute, preset single-channel vertical forms control	Up to 16 with DC72-L asynch-ronous line groups

DC72-L 8 asynchronous-line group for DC72-A,B, or C. Mounts in DC72-NP cabinet. Up to two DC72-L's are supported on each DC72-NP.

DC72-NP Implements DDCMP communications protocol for a DC72-NP remote station. Supported by the DN85/DN87 communications concentrator and DC75 communications concentrator equipped with the DC75-NP option.

This page intentionally left blank

Module Test

1. Draw an example of a computer network using

   A. Simple Topology

   B. Complex Topology

2. Explain the major difference between simple and complex topologies.

3. Explain "multi-pathing" and give two reasons why it might be used.

4. Give three differences between synchronous and asynchronous data transmission techniques.

5. Give two advantages of remote computing.

6. Configure the minimum network for a user with a KI10 who needs the processor plus 64 terminals in Boston, a line printer and 16 asynchronous lines in Cleveland, 8 asynchronous dial-in lines in Baltimore, and a card reader/line printer plus 1 terminal (LA36) in New York City. How would you route the telephone lines?

7. What questions would you ask the customer in 6?

This page intentionally left blank

Evaluation Sheets

1. An example of a Simple Topology (A) and a Complex topology (B).

   A. See Fig (2-1)

   B. See Fig (2-2)

2. Simple topologies use point to point communications. Consequently, each remote station has direct communication with the destination mode. In a simple topology, a given node can communicate only with those nodes to which it has a direct line.

   Complex topologies are those which allow communication between nodes which may not be directly connected, that is, some nodes in the network can perform "route-through." These "route-through" nodes can take messages from one remote node and re-transmit it to another such that eventually the message gets to its destination.

3. Multi-pathing is the term which applies to the complex topology in which there is more than one possible path (or route) from one node to another. See Figure (2-3).

   Reasons for multi-path connection:

   A. Load-leveling - provides high availability.

   B. Redundancy - if one line goes down, another may still be usable.

   C. Reliability - if a given node is down, alternate routes may be used.

   D. Versatility (efficiency) - with greater connectivity, the line usage may be tailored dynamically.

4. Differences between synchronous and asynchronous data transmission techniques.

   A. Synchronous - more information per unit time.

   B. Synchronous - requires message buffering in transmitter, and receiver.

C. Asynchronous - character oriented, not block/message oriented.

D. In terminals, asynchronous need be less sophisticated.

E. Synchronous - easier to use CRC error detection schema.

5. Advantages of remote computing

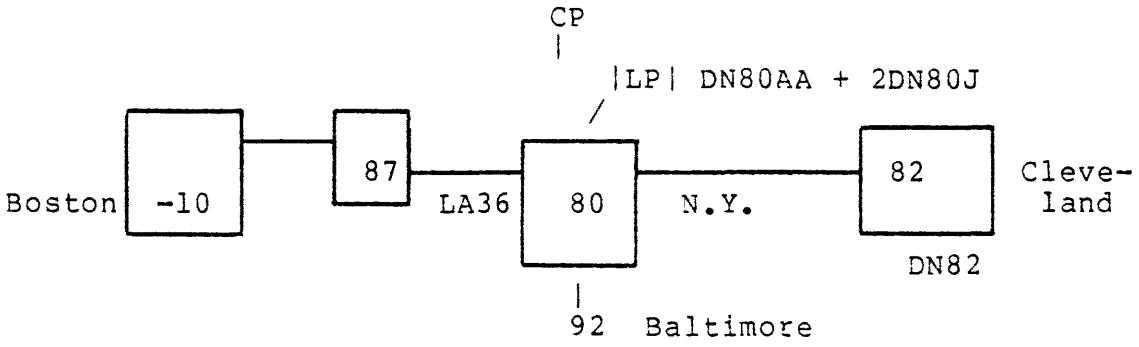
A. Costs - line charges are reduced.

B. Convenience - listings etc., are available on site.

C. Time Saved

D. Flexibility

6.

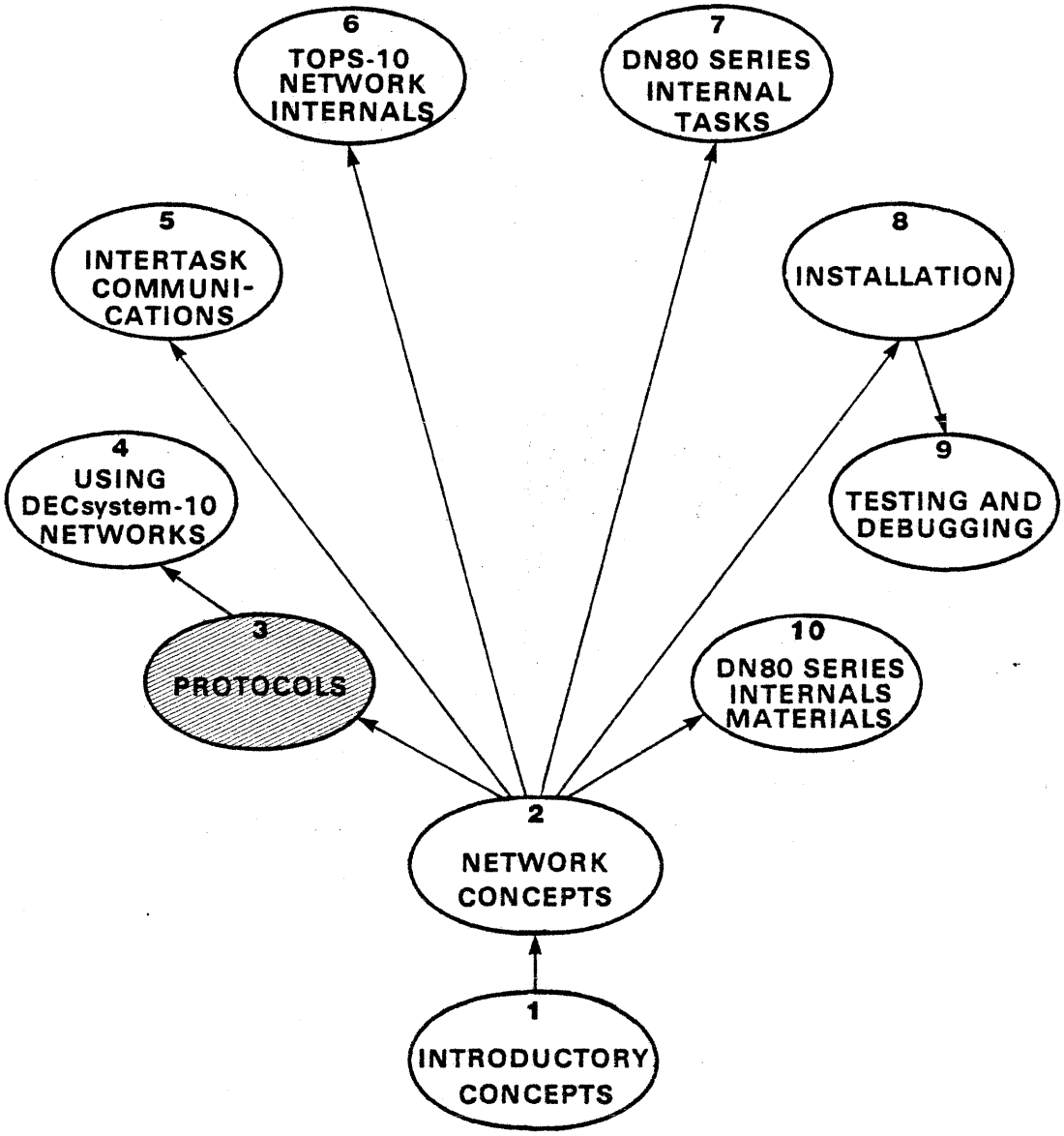


DN87AA
DN81FA
3 DN81EC
8 DN81FB
1 DN81J

7. (E.G.) EIA or 20 mil?
Line speeds?
Can we put a CDR in Cleveland?
Throughput for Baltimore?

Protocols
Module 3

Course Map



M8 0277

Introduction

This module covers the message protocols used in DECsystem-10 networks. The first portion is a brief discussion of protocols in general; then, Network Control Language (NCL) is covered in detail. A review of DDCMP is also included so that complete protocol imformation is available in one package. The protocol break-downs indicate what each field of each type of message (in both NCL and DDCMP) means.

Objectives

Upon completion of this module, the student will be able to

1. Identify NCL message types (using the supplied material).
2. Format NCL messages (using the supplied material).

General Protocols

In Data Communication, a protocol is a set of rules which defines the format of data transferred over the communication line. In a Data Communication system there are generally several levels of protocols. From lowest to highest, they are line level, message level, and task/process level.

Line level protocols address the problems of

1. Framing (or Synchronization)
   - where a message starts and ends.
2. Error Detection
   - was a message received correctly (CRC calculation).
3. Transparency
   - what to do if the data portion of a message contains characters which might be confused with control characters.

Message level protocols address the problems of

1. Sequence checking/control
   - were messages out of sequence.
   - was a message lost.
2. Error Control
   - what to do if a bad message was received (CRC error, out of sequence, wrong length, etc.).
   - when should re-transmission be requested or initiated.
   - what happens if there is no response to a message (line down, etc.).
3. Acknowledgements/negative acknowledgements
   - when should ACK or NAK be sent.
   - what to do when received.

4. Routing

   - what path to take from sender to receiver.

Task/Process level protocols address the problem of

Task to task communication

- how to arrange for a task to get data from/to another.
- how to handle task to task acknowledgements.
- how to format data.

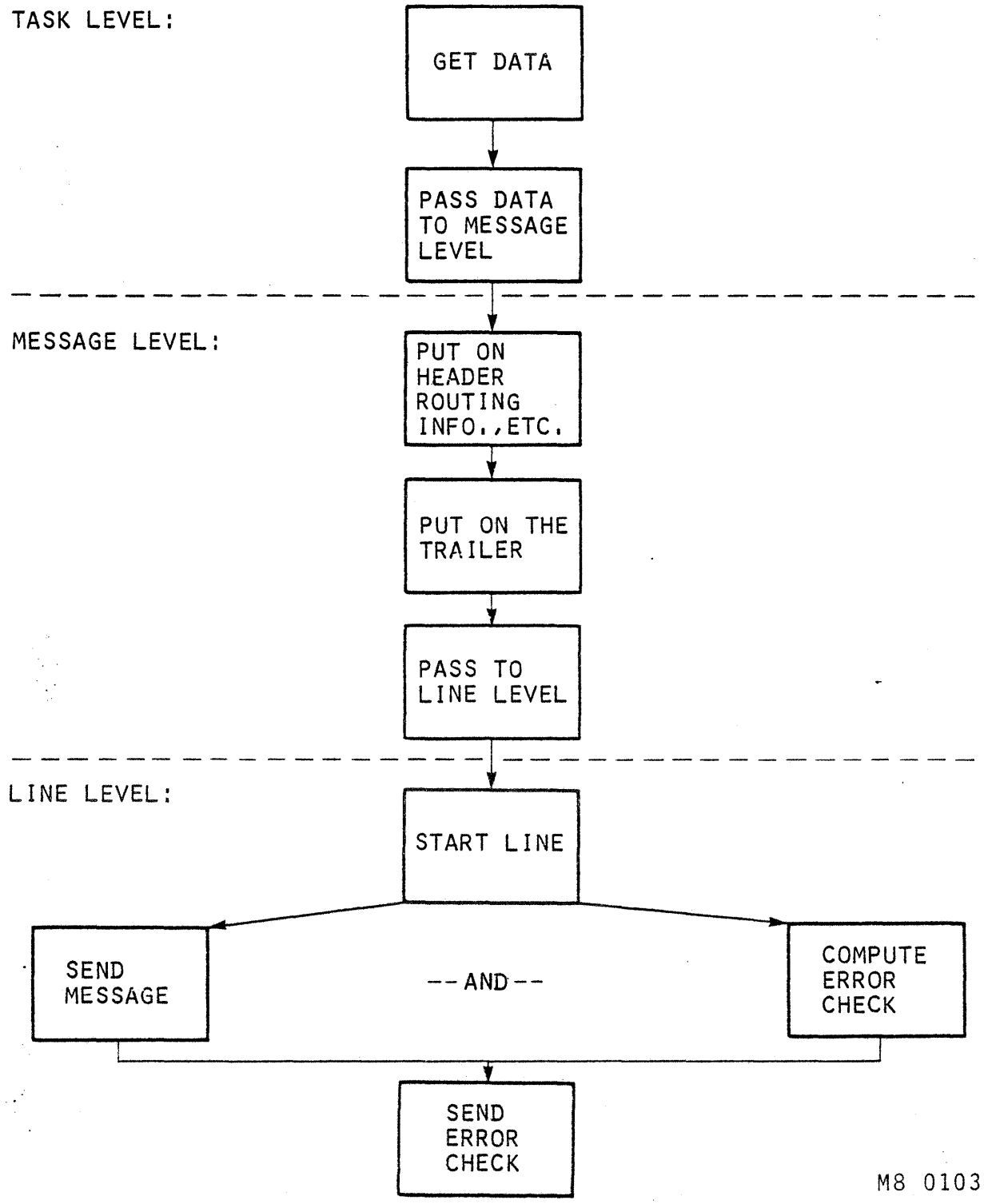

TASK LEVEL:
GET DATA
PASS DATA TO MESSAGE LEVEL
MESSAGE LEVEL:
PUT ON HEADER ROUTING INFO.,ETC.
PUT ON THE TRAILER
PASS TO LINE LEVEL
LINE LEVEL:
START LINE
SEND MESSAGE
--AND--
COMPUTE ERROR CHECK
SEND ERROR CHECK
M8 0103

Figure 3-1

There are three major types of line protocols: 1) byte-oriented, 2) byte-count, and 3) bit-oriented (Note: byte-count may be considered an extension of byte-oriented.) These three represent different approaches to the problem of transparency, that is, how to ensure that the content of a message does not get confused with line control information. The simplest solution to this problem would be to disallow certain characters from a message, but it is often the case that the system has no control over this.

BYTE-ORIENTED PROTOCOL

IBM's Binary Synchronous Communications protocol (BISYNC) is an example of a byte-oriented protocol. The overall format of a BISYNC message is shown in Figure 3-2.

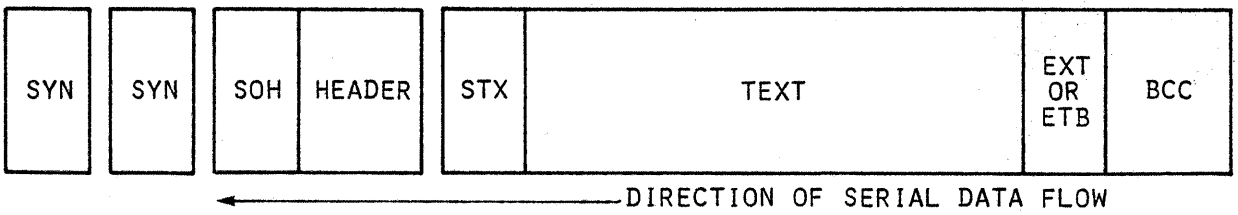


Figure 3-2

The header is optional, but if one is used it begins with SOH (Start of Header) and ends with STX (Start of Text). SOH, STX, and ETX are special characters. It is easy to see that if the text portion of the message happened to contain an ETX, the receiver would be confused. BISYNC avoids this problem by using "transparent mode." "Transparent mode" is entered by putting the character DLE before the STX, and is exited from by a DLE ETX sequence. The recipient of the message then takes all characters between the DLE STX and the DLE ETX as text. The problem of the occurrence of a DLE in the text is solved by a technique called "DLE doubling" or "byte stuffing." This is accomplished by having the transmitter "stuff" an extra DLE into the outgoing data stream immediately preceding the DLE in the text. The receiver, upon encountering an incoming DLE examines the next character. If it is another DLE, it is ignored. A transparent message is illustrated in Figure 3-3.

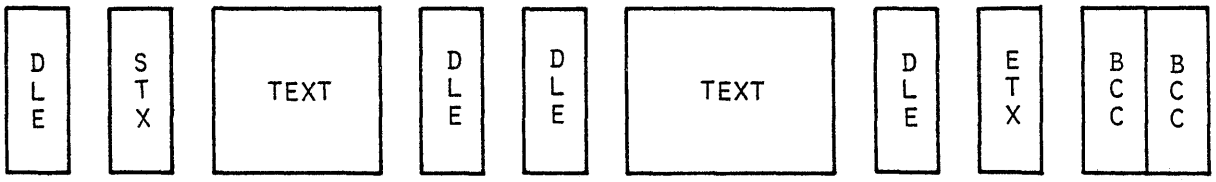


Figure 3-3

BYTE-COUNT PROTOCOL

It is possible to devise a protocol which, by keeping track of byte-count, solves the transparency problems without the use of DLE or other control characters. One of the widely used protocols that does this is Digital Equipment Corporation's DDCMP protocol, the format for which is shown in Figure 3-4.

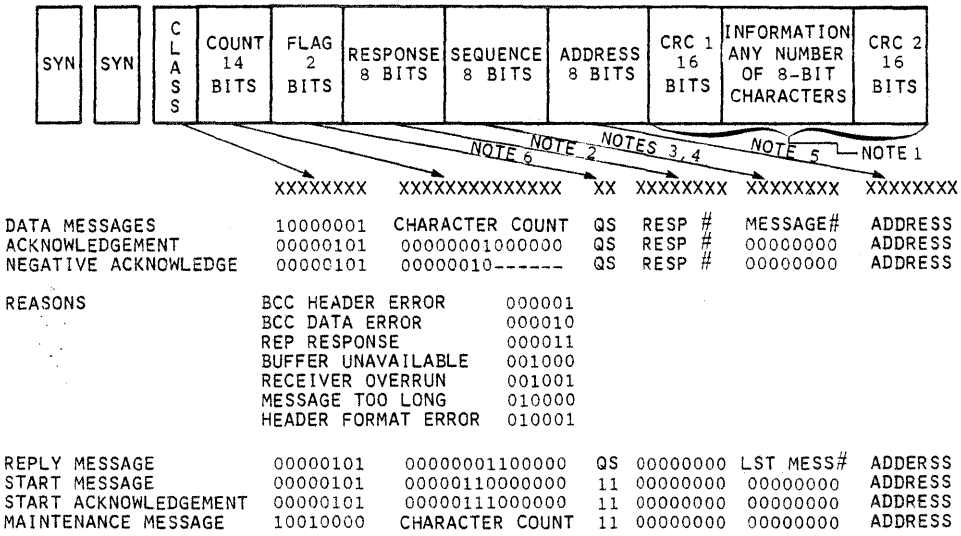


Figure 3-4

Note that the header is fixed length (8 characters) so the receiver always knows where the text begins.

BIT-ORIENTED PROTOCOL

SDLC is an example of a bit-oriented or "bit-stuffing" protocol. The basic format for SDLC is a "frame," shown in Figure 3-5. The information field is not restricted in format or content and can be of any reasonable length (including zero). Since the SDLC protocol does not use characters of defined length, but rather works on a bit by bit basis, the 01111110 flag may be recognized at any time.

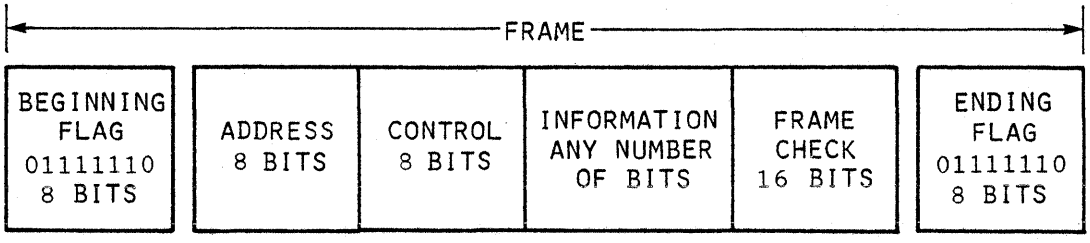


Figure 3-5

In order that the flag not be sent accidentally, SDLC procedures require that a binary 0 be inserted by the transmitter after any succession of five continuous 1's. The receiver then removes a 0 that follows a received succession of five 1's. Inserted and removed zeros are not included in the transmission error check.

An in-depth discussion of these protocols is found in Chapters 16-19 of Technical Aspects of Data Communication by John E. McNamara published 1977 by Digital Equipment Corporation, document number JB002-A.

DDCMP

Included for reference here is a breakdown of DDCMP message types and messages.

> **NOTE**
>
> NCL messages comprise the DATA portion of DDCMP data messages.

Table 3-1

"DDCMP FORMATS"

DATA	-- SOH	CC1	CC2	MSG#	NMSG	A0	BCC1	N*DATA	BCC2
ACK	-- ENQ	<001>	FILL	MSG#	FILL	A0	BCC1		
NAK	-- ENQ	<002>	RNAK	MSG#	FILL	A0	BCC1		
REPLY	-- ENQ	<003>	FILL	FILL	NLST	A0	BCC1		
RESET*	-- ENQ	<004>	FILL	FILL	NNXT	A0	BCC1		
RESACK*	-- ENQ	<005>	FILL	NEXP	FILL	A0	BCC1		
START	-- ENQ	<006>	FILL	FILL	NBEG	A0	BCC1		
STACK	-- ENQ	<007>	FILL	NREC	NXMT	A0	BCC1		
BOOT	-- DLE	CC1	CC2	<000>	<000>	A0	BCC1	BOOTDATA	BCC2

BOOTDATA= <000> == BOOT
<001> == EXAMINE
<002> == DEPOSIT
<003> == GOTO
<004> == CLEAR
<005> == DEBUG
<011> == ACCEPT
<012> == EXAMINE DATE
<013> == REJECT
<014> == REQUEST LOAD
<015> == REQUEST BOOT

* No longer used

"DDCMP MESSAGES"

ALL BUT "DATA" ARE PRECEDED BY A SYNCHRONIZATION SEQUENCE.
"SYNCHRONOUS LINE PROTOCOL"

"DDCMP FORMATS"

DATA	-- SOH	CC1	CC2	MSG#	NMSG	A0	BCC1	N*DATA	BCC2
ACK	-- ENQ	<001>	FILL	MSG#	FILL	A0	BCC1		
NAK	-- ENQ	<002>	RNAK	MSG#	FILL	A0	BCC1		
REPLY	-- ENQ	<003>	FILL	FILL	NLST	A0	BCC1		
RESET*	-- ENQ	<004>	FILL	FILL	NNXT	A0	BCC1		
RESACK*	-- ENQ	<005>	FILL	NEXP	FILL	A0	BCC1		
START	-- ENQ	<006>	FILL	FILL	NBEG	A0	BCC1		
STACK	-- ENQ	<007>	FILL	NREC	NXMT	A0	BCC1		
BOOT	-- DLE	CC1	CC2	<000>	<000>	A0	BCC1	BOOTDATA	BCC2

DATA SOH=201 START CHARACTER FOR DATA MESSAGES.
CC1 LOW ORDER 8 BITS OF THE CHARACTER COUNT OF THE DATA PORTION.
CC2 HIGH ORDER 8 BITS OF THE CHARACTER COUNT OF THE DATA PORTION.
1.MULTIPOINT-2 HIGH ORDER BITS ARE FLAGS.
2.POINT-POINT-2 HIGH ORDER BITS ARE ALWAYS ZERO.
MSG# NUMBER OF LAST GOOD MESSAGE RECEIVED: IMPLIES ACK OF ALL LOWERED NUMBERED MESSAGES.
NMEG THE NUMBER OF THIS MESSAGE.
A0=1 STATION NUMBER, ALWAYS ONE FOR POINT TO POINT.
BBC1 16 BITS OF BCC COMPUTED ON THE FIRST 6 BYTES OF THE MESSAGE.
N*DATA N=THE NUMBER OF DATA BYTES, A 16 BIT QUANTITY MADE UP OF CC1 AND CC2.
BCC2 16 BITS OF THE BCC COMPUTED ON THE "N" DATA BYTES.

****

ACK ENG=005 THIS IS THE STARTING CHARACTER FOR CONTROL MESSAGES.
<001>
FILL=0 FILLER, IS CHECKED AND MUST BE ZERO.
MSG# MESSAGE NUMBER OF THE LAST GOOD MESSAGE RECEIVED.
FILL=0 FILLER, IS CHECKED AND MUST BE ZERO.
A0=1 STATION NUMBER, ALWAYS ONE FOR POINT TO POINT.
BCC1 16 BITS OF BCC COMPUTED ON THE FIRST 6 BYTES OF THE MESSAGE.

****

```
NAK     ENQ=005 THIS IS THE STARTING CHARACTER FOR CONTROL
                MESSAGES.
        <002>
        RNAK    REASON FOR NEGATIVE ACKNOWLEDGEMENT:
                1. HEADER BCC INCORRECT
                2. DATA BCC INCORRECT.
                3. THE LAST REP MESSAGE RECEIVED INDICATES
                   WE LOST ONE OR MORE MESSAGES.
                   10-BUFFER SPACE TEMPORARILY UNAVAILABLE.
                   11-RECEIVE OVERRUN (DATA LOST).
                   20-DATA MESSAGE IS TOO LONG.
                   21-HEADER FORMAT ERROR (E.G. NON-ZERO FILL).
        MSG#    MESSAGE NUMBER OF THE LAST GOOD MESSAGE
                RECEIVED
        FILL=0  FILLER, IS CHECKED AND MUST BE ZERO.
        A0=1    STATION NUMBER, ALWAYS ONE FOR POINT TO POINT.
        BCC1    16 BITS OF BCC COMPUTED ON THE FIRST 6 BYTES
                OF THE MESSAGE.

*****************************************************************

REP     ENQ=005 THIS IS THE STARTING CHARACTER FOR CONTROL
                MESSAGES.
        <003>
        FILL=0  FILLER, IS CHECKED AND MUST BE ZERO.
        FILL=0  FILLER, IS CHECKED AND MUST BE ZERO.
        NNXT    NEXT NUMBERED MESSAGE TO BE TRANSMITTED
                (I.E., LOWEST MESSAGE THAT HAS NOT BEEN ACK'ED.
        NLST    NUMBER OF LAST TRANSMITTED MESSAGE.
        A0=1    STATION NUMBER; ALWAYS ONE FOR POINT TO POINT.
        BCC1    16 BITS OF BCC COMPUTED ON THE FIRST 6 BYTES
                OF THE MESSAGE.

*****************************************************************
```

```
RESET*  ENQ=005 THIS IS THE STARTING CHARACTER FOR CONTROL
MESSAGES
        <004>
        FILL=0  FILLER, IS CHECKED AND MUST BE ZERO.
        FILL=0  FILLER, IS CHECKED AND MUST BE ZERO.
        NNXT    NEXT NUMBERED MESSAGE TO BE TRANSMITTED
                (I.E., LOWEST MESSAGE THAT HAS NOT BEEN ACK'ED.
        A0=1    STATION NUMBER; ALWAYS ONE FOR POINT TO POINT.
        BCC1    16 BITS OF BCC COMPUTED ON THE FIRST 6 BYTES
                OF THE MESSAGE.

* NO LONGER USED

*****************************************************************

RESACK* ENQ=005 THIS IS THE STARTING CHARACTER FOR CONTROL
                MESSAGES.
        <005>
        FILL=0  FILLER, IS CHECKED AND MUST BE ZERO.
        NEXP    MESSAGE NUMBER EXPECTED TO BE SENT NEXT;
                USUALLY "NNXT" FIELD OF "REP" MESSAGE.
        FILL=0  FILLER, IS CHECKED AND MUST BE ZERO.
        A0=1    STATION NUMBER; ALWAYS ONE FOR POINT TO POINT.
        BCC1    16 BITS OF BCC COMPUTED ON THE FIRST 6 BYTES
                OF THE MESSAGE.

* NO LONGER USED

*****************************************************************

START   ENQ=005 THIS IS THE STARTING CHARACTER FOR CONTROL
                MESSAGES.
        <006>
        FILL=0  FILLER, IS CHECKED AND MUST BE ZERO.
        FILL=0  FILLER, IS CHECKED AND MUST BE ZERO.
        NBEG    FIRST MESSAGE NUMBER THIS STATION WILL TRANSMIT
                AFTER STARTUP IS COMPLETED.
        A0=1    STATION NUMBER; ALWAYS ONE FOR POINT TO POINT.
        BCC1    16 BITS OF BCC COMPUTED ON THE FIRST 6 BYTES OF
                THE MESSAGE.

*****************************************************************
```

```
STACK   ENQ=005 THIS IS THE STARTING CHARACTER FOR CONTROL
                MESSAGES.
        <007>
        FILL=0  FILLER, IS CHECKED AND MUST BE ZERO.
        NREC    NEXT MESSAGE NUMBER FOR RECEPTION; USUALLY
                "NBEG" FIELD OF THE "STRT" MESSAGE.
        NXMT    ???
        A0=1    STATION NUMBER, ALWAYS ONE FOR POINT TO POINT.
        BCC1    16 BITS OF BCC COMPUTED ON THE FIRST 6 BYTES
                OF THE MESSAGE.

**********************************************************************

BOOT    DLE=220 THIS IS THE STARTING CHARACTER FOR STATION
                MANAGEMENT MESSAGES.
        CC1     LOW ORDER 8 BTS OF THE CHARACTER COUNT OF THE
                DATA PORTION
        CC2     HIGH ORDER 8 BITS OF THE CHARACTER COUNT OF THE
                DATA PORTION.
                1. MULTIPOINT-2 HIGH ORDER BITS ARE FLAGS.
                2. POINT-POINT-2 HIGH ORDER BITS ARE ALWAYS ZERO.
        <000>
        A0=1    STATION NUMBER, ALWAYS ONE FOR POINT TO POINT.
        BCC1    16 BITS OF BCC COMPUTED ON THE FIRST 6 BYTES
                OF THE MESSAGE.
```

```
            ************"BOOTDATA"**************************

           "BOOTDATA"

BOOT
        SNA     EXTENSIBLE BINARY; THE NODE NUMBER OF THE
                STATION WHICH ORIGINATED THE BOOT MESSAGE.
        <000>
EXAMINE
        SNA     EXTENSIBLE BINARY; THE NODE NUMBER OF THE
                STATION WHICH ORIGINATED THE BOOT MESSAGE.
        <001>
        <ADR1>
        <ADR2>
DEPOSIT
        SNA     EXTENSIBLE BINARY; THE NODE NUMBER OF THE
                STATION WHICH ORIGINATED THE BOOT MESSAGE.
        <002>
        <ADR1>
        <ADR2>
GO TO
        SNA     EXTENSIBLE BINARY; THE NODE NUMBER OF THE
                STATION WHICH ORIGINATED THE BOOT MESSAGE.
        <003>
        <ADR>
```

CLEAR		
	SNA	EXTENSIBLE BINARY; THE NODE NUMBER OF THE STATION WHICH ORIGINATED THE BOOT MESSAGE.
	<004>	
DEBUG		
	SNA	EXTENSIBLE BINARY; THE NODE NUMBER OF THE STATION WHICH ORIGINATED THE BOOT MESSAGE.
	<005>	
	<ADR1>	
	<ADR2>	
ACCEPT		
	DNA	EXTENSIBLE BINARY; NODE NUMBER THE BOOT STRAP SHOULD BE ROUTED TO. ZERO MEANS DEFAULT.
	<011>	
	<ADR>	
EXAMINE DATE		
	DNA	EXTENSIBLE BINARY; NODE NUMBER THE BOOT STRAP SHOULD BE ROUTED TO. ZERO MEANS DEFAULT.
	<012>	
	<ADR>	
	<DATA>	

```
REJECT
        DNA     EXTENSIBLE BINARY; NODE NUMBER THE BOOT STRAP
                SHOULD BE ROUTED TO. ZERO MEANS DEFAULT.
        <013>
REQUEST BOOT
        DNA     EXTENSIBLE BINARY; NODE NUMBER THE BOOT STRAP
                SHOULD BE ROUTED TO. ZERO MEANS DEFAULT.
        <014>
        <TYPE>  EXTENSIBLE BINARY; CODE FOR THE TYPE OF NODE
                REQUESTING LOAD:
                1=DC71 (PDP8I WITH DP01)
                2=DC72 (PDP8E WITH DP8E)
                3=PDP11/40 WITH DU11
                4=DAS82 (PDP11/40 WITH DQ11)
        <SERIAL> EXTENSIBLE BINARY; THE SERIAL NUMBER FOR THE
                 NODE BEING BOOTED.
        <DESCRIPTION> EXTENSIBLE ASCII; TEXT WHICH DESCRIBES
                      PROGRAM TO BE LOADED, USUALLY A FILE
                      DESCRIPTION.

REQUEST LOAD
        DNA     EXTENSIBLE BINARY; NODE NUMBER THE BOOT STRAP
                SHOULD BE ROUTED TO. ZERO MEANS DEFAULT.
        <015>
        <TYPE>  EXTENSIBLE BINARY; CODE FOR THE TYPE OF NODE
                REQUESTING LOAD:
                1=DC71 (PDP8I WITH DP01)
                2=DC72 (PDP8E WITH DP8E)
                3=PDP11/40 WITH DU11
                4=DAS82 (PDP11/40 WITH DQ11)
        <SERIAL> EXTENSIBLE BINARY; THE SERIAL NUMBER FOR THE
                 NODE BEING BOOTED.
        <DESCRIPTION> EXTENSIBLE ASCII; TEXT WHICH DEXCRIBES
                      PROGRAM TO BE LOADED, USUALLY A FILE
                      DESCRIPTION.

                *********END OF "BOOTDATA"*********************

        BCC2    16 BITS OF THE BCC COMPUTED ON THE "N"
                DATA BYTES
```

DECsystem-10 Networks Protocols

NCL Messages

There are three types of NCL messages: Unnumbered Control Messages, Numbered Control Messages, and Data Messages. NCL defines the data and control messages over a logical link, that is, the connection between processes as opposed to the DDCMP connection over a physical link between machines. Many of the DDCMP functions (for the physical link) are duplicated by NCL (for the logical link).

Unnumbered Control

In NCL, the messages which are unnumbered are those for which there is some mechanism other than numbering. This mechanism will cause the message information to be repeated if the message gets lost in the network. For example, if an ACK gets lost, the sender will still get the information from the next ACK message.

The ACK, NAK, REP, Start and Stack functions of DDCMP are duplicated in NCL (as unnumbered messages). DAP/Data and Node ID messages complete the list of unnumbered message types. See Table 3-2 for a breakdown of the fields in each of these messages.

The unnumbered control messages are:
- DAP/DATA
- ACK
- NAK
- REPLY
- START
- STACK
- NODE ID

Numbered Control Messages

The Numbered Control Messages consist of The Connect, Disconnect, Neighbors, Request Configuration, Data Request and Station Control messages. See Table 3-3 for a breakdown of the fields in each of these messages, noting that the last field (CM) is further broken down in the DAP Formats table, Table 3-4.

The numbered control messages are:
- CONNECT
- DISCONNECT
- NEIGHBORS
- REQUEST-CONFIGURATION
- CONFIGURATION
- DATA REQUEST
- STATION CONTROL

Sample Message

The following is a DECsystem-10 Networks DATA message, the data is A, the message is from node 26 to node 22.

201	11	100	135	25	0	BCC	DDCMP Header
Start of Header	Char count 2 bytes		of last good msg	This message number	zero	2 bytes BCC of header	
110	26	22	224	112	3		NCL Header
Data Msg	From	To	NCL ACK	Msg #	Data Link Addr		
2	1	101					the data
Len	Type (data)	A					
BCC							
BCC of message							

201 11 100 135 25 0 x x 110 26 22 224 112 3 2 1 101 x x

Table 3-2

"NCL FORMATS"

```
UNNUMBERED CONTROL -- NCT  SNA  DNA  NCA  NCN  OPD
NUMBERED CONTROL   -- NCT  SNA  DNA  NCA  NCN  0    CM
DATA               -- NCT  SNA  DNA  NCA  NCN  DLA  DEV CONTROL
                                                     (DAP)
```

"NCL NETWORK MESSAGES" == "CM"

```
CONNECT         --  CNT  <001>  DLA  SLA  DPN  SPN  MML  FEA
DISCONNECT      --  CNT  <002>  DLA  SLA  RSN
NEIGHBORS       --  CNT  <003>  (NNM LVL)
REQ CONFIG      --  CNT  <004>
CONFIGURATION   --  CNT  <005>  OPD
DATA REQUEST    --  CNT  <006>  DLA  DQR
STATION CONTROL--   CNT  <007>  OPD
```

"UNNUMBERED CONTROL MESSAGES"

```
0       1       2       3       4       5       6       7

0=NCT.DM (DATA)         SNA     TRACE  INTERRUPT  SEQ.   EXT.
1=NCT.AK (ACK)           &              MESSAGE    NODE   BIT
2=NCT.NK (NAK)          DNA
3=NCT.RP (REPLY)     PRESENT
4=NCT.ST (START)
   'CPD' IS 'NNM'
            'SNM'
            'SID'
5=NCT.SK (STACK)
6=NCT.ID (NODE ID)
   'CPD' IS 'NNM'
            'SNM'
            'SID'
```

```
SNA     SOURCE "NNM"
DNA     DESTINATION "NNM"
NCA     NETWORK CONTROL ACK, LAST NETWORK MESSAGE RECEIVED OK.
NCN     NETWORK CONTROL MESSAGE NUMBER, ONE BYTE BINARY FIELD.
OPD     OPTIONAL DATA
```

Table 3-3

```
NUMBERED CONTROL
      NCT         NETWORK CONTROL MESSAGE TYPE AND FLAGS,
                  EXTENSIVE FIELD.
                  NCT.TP (B0,B1,B2) BITS 0-2=TYPE FIELD
                  NCT.DM 0=DATA MESSAGE
                  NCT.AK 1=ACK
                  NCT.NK 2=NAK
                  NCT.RP 3=REP
                  NCT.ST 4=START, "OPD" IS "NNM" "SNM" "SID"
                  NCT.SK 5=STACK
                  NCT.ID 6=NODE ID, "OPD", IS "NNM" "SNM" "SID"
                  NCT.RH (B3) 3="SNA" AND "DNA" PRESENT
                  (B4)    4=TRACE
                  NCT.IT (B5) INTERRUPT MESSAGE, "NOP" "DRQ"
                         COUNT.
                  NCT.SQ (B6) SEQUENTIAL NODE.
                  NCT.EX (B7) 7=EXTENSIBLE BIT
      SNA         SOURCE "NNM"
      DNA         DESTINATION "NNM"
      NCA         NETWORK CONTROL ACK, LAST NETWORK MESSAGE
                  RECEIVED OK.
      NCN         NETWORK CONTROL MESSAGE NUMBER. ONE BYTE
                  BINARY FIELD.
      0
      CM    **************"CONTROL MESSAGE"*********************

      ********
      CONNECT
      ********
      CNT       COUNT OF REMAINING BYTES IN MESSAGE
      <001>     NC.CNT==<001>
      DLA       DESTINATION MESSAGE LINK ADDRESS, I.E., THE
                INDEX INTO THE NODES CONNECTION DATABASE.
                EXTENSIBLE FIELD, MAXIMUM OF 12 BITS, ZERO
                IS ILLEGAL.
      SLA       SOURCE LINK ADDRESS (1 OR 2 BYTE EXTENSIBLE
                BINARY NUMBER).
      DPN       DESTINATION "PN".
      SPN       SOURCE "PN"
      MML       MAXIMUM "DDCMP" MESSAGE LENGTH.
```

```
FEA       FEATURES: "DCM"+"RLN"+"DVT".
          1. DCM=DATA CODE AND MODE.
                    DCM.AS,B0 ;ASCII
                    DCM.EB,B1 ;ABCDIC
                    DCM.IM,B2 ;IMAGE
                    DCM.HO,B3 ;HOLLERITH (CDR ONLY)
                    DCM.DI,B4 ;DEC IMAGE (CRD ONLY)
                    DCM.RV,B5 ;RESERVED
                    DCM.CP,B6 ;COMPRESSED FORMAT
          2. RLN=RECORD LENGTH (MAX IF VARIABLE):
                    EXTENSIBLE BINARY.
          3. DVT=DEVICE SPECIFIC ATTRIBUTES:
                "DCD","DLP","DTY","DRX"
                    1. DCD=ATTRIBUTES FOR CARD READER.
                          BITS 0+1=SPEED:
                             0=DONT CARE ABOUT CPM RATE.
                             1=LESS THAN 300CPM.
                             2=BETWEEN 300 AND 600 CPM.
                             3=GREATER THAN 600 CPM.
                    2. DTY=ATTRIBUTES FOR TELETYPES.
                             DTY.MC,B0 ;MODEM CONTROL.
                             DTY.AS,B1 ;AUTO-BAUD
                             DTY.SB,B2 ;HANDLER CAN SET BAUD
                                        RATE.
                             DTY.27,B3 ;2741
                             DTY.BD,B4 ;BAUDOT
                             DTY.AD,B5 ;AUTO DIAL LINE
                                        (BELL 801).
                    3. DVT=DEVICE SPECIFIC ATTRIBUTES:
                             "DCD","DLP","DTY","DRX"
                             1.DCD=CARD READER ATTRIBUTES.
                                  (SEE ABOVE)
                             2.DLP=LINE PRINTER ATTRIBUTES.
                                    DLP.S0-0 ;SPEED (SEE DCD)
                                    DLP.S3-1 ;1^
                                    DLP.S6-2 ;2^
                                    DLP.S9-3 ;3^
                                    DLP.LL-B2 ; LOWER CASE
                                                REQ.
                                    DLP.RC-B3 ; REMOVE CHAR.
                                                SET REQ.
                                    DLP.MP -B4 ; MULTI-PART
                                                 PAPER REQ.
                                    DLP.CS-B5 ;12 CHANNEL
                                      SKIPPING REQ.
                                          B7 ;B7=1.
                                    DLP.SK-B8=B9  ;SKIP REQ:
                                           0=DONT CARE.
                                           1=CHANGEABLE
```

```
                    FROM HANDLER.
                  2=CHANGEABLE AT
                    SITE.
                  3=CHANGEABLE
                    BUT DONT CARE
                    HOW.
          DLP.OP=B10 ;REQUIRES
                      OVERPRINT.
          DLP.68-B11+B12
                     ;6/8 LINES
                      PER INCH
          DLP.CF-B13 ;CHANGEABLE
                      FORM WIDTH.
3.DTY=TELETYPE ATTRIBUTES:
       (SEE DTY ABOVE)
4.DRX="REMOTE DATA ENTRY DEVICE"
        ATTRIBUTES:
        DRX.MD,B0 ;MULTI DROP
                   LINE DATA
                   ENTRY LINE.
        DRX.PL,B1 ;LINE ACCEPTS
                   A POLL
                   SEQUENCE.
```

```
**********
DISCONNECT
**********
CNT       COUNT OF REMAINING BYTES MESSAGE
<002>     NC.DSC==<002>
DLA       DESTINATION MESSAGE LINK ADDRESS, I.E., THE
          INDEX INTO THE NODES CONNECTION DATABASE.
          EXTENSIBLE FIELD, MAXIMUM OF 12 BITS, ZERO
          IS ILLEGAL.
SLA       SOURCE LINK ADDRESS (1 OR 2 BYTE EXTENSIBLE
          BINARY NUMBER).
RSN       REASON:
          RSN.OK,0 ;NORMAL DISCONNECT.
          RSN.OT,1 ;OBJECT TYPE NOT AVAILABLE.
          RSN.XN,2 ;TOO MANY CONNECTS TO PROCESS.
          RSN.NP,4 ;PROCESS DOES NOT EXIST AT THIS NODE.
**********
NEIGHBORS
**********
CNT       COUNT OF REMAINING BYTES IN MESSAGE
<3>       NC.NBN==<003>
NNM       NODE NAME, A BINARY EXTENSIBLE FIELD, MAXIMUM
          OF 12 BITS, IDENTIFYING NODE NAME. ZERO MEANS
          NEXT NODE OVER SYNCHRONOUS LINE.
LVL       LINK VALUE IS A ONE-BYTE VALUE USED TO
          DETERMINE THE PREFERRED PATH; 0 MEANS LINK
          IS DOWN. (PREFERRED PATH IS THAT WHOSE SUM
          OF LINK VALUES IS LOWEST).
**********
REQ CONFIG
**********
CNT       COUNT OF REMAINING BYTES IN MESSAGE
<004>     NC.RCF==<004>
**********
CONFIGURATION
**********
CNT       COUNT OF REMAINING BYTES IN MESSAGE
<005>     NC.CNF==<005>
OPD       OPTIONAL DATA.
**********
DATA REQUEST
**********
CNT       COUNT OF REMAINING BYTES IN MESSAGE
<006>     NC.DQR==<006>
DLA       DESTINATION MESSAGE LINK ADDRESS, I.E., THE
          INDEX INTO THE NODES CONNECTION DATABASE.
          EXTENSIBLE FIELD, MAXIMUM OF 12 BITS, ZERO
          IS ILLEGAL.
DQR       NUMBER OF REQUESTS
```

```
            ************
            STATION ID
            ************
            CNT       COUNT OF REMAINING BYTES IN MESSAGE
            <007>     NC.CTL==<007>
            OPD       OPTIONAL DATA.

                      *NC.MAX==NC.CTL ;LAST NC MESSAGE TYPE.

DATA
             NCT          NETWORK CONTROL MESSAGE TYPE AND FLAGS,
                          EXTENSIBLE FIELD.
                          NCT.TP (B0,B1,B2) BITS 0-2=TYPE FIELD
                          NCT.DM 0=DATA MESSAGE
                          NCT.AK 1=ACK
                          NCT.NK 2=NAK
                          NCT.RP 3=REP
                          NCT.ST 4=START, "OPD" IS "NNM" "SNM" "SID"
                          NCT.SK 5=STACK
                          NCT.ID 6=NODE ID, "OPD", IS "NNM" "SNM" "SID"
                          NCT.RH (B3) 3="SNA" AND "DNA" PRESENT
                          (B4)    4=TRACE
                          NCT.IT (B5) INTERRUPT MESSAGE, "NOP" "DRQ"
                                  COUNT.
                          NCT.SQ (B6) SEQUENTIAL NODE.
                          NCT.EX (B7) 7=EXTENSIBLE BIT
             SNA          SOURCE "NNM"
             DNA          DESTINATION "NNM"
             NCA          NETWORK CONTROL ACK, LAST NETWORK MESSAGE
                          RECEIVED OK.
             NCN          NETWORK CONTROL MESSAGE NUMBER. ONE BYTE
                          BINARY FIELD.
             DLA          DESTINATION MESSAGE LINK ADDRESS, I.E., THE
                          INDEX INTO THE NODES CONNECTION DATABASE.
                          EXTENSIBLE FIELDS, MAXIMUM OF 12 BITS, ZERO
                          IS ILLEGAL.
```

Table 3-4

```
**************************************************************

               "DAP LINE PROTOCOL"

               "DAP FORMATS"

DATA                  -- CNT <001>  (DATA)
DATA WITH EOR         -- CNT <002>  (DATA)
DATA REQUEST          -- CNT <003>  DRQ
STATUS                -- CNT <004>  STC   STD
CONTROL               -- CNT <005>  DCT   CDT
USER ID               -- CNT <006>  PPN   PSWD  UNAME ACCT  GROUP
FILE SPECIFICATION    -- CNT <010>  FST   FEA   FDES

**************************************************************

DATA
       CNT    LENGTH OF WHAT FOLLOWS IN BYTES (EXT. BINARY).
       <001>  DC.DAT==<001>
       <DATA>

**************************************************************

DATA WITH EOR   EOR=END OF RECORD
        CNT     LENGTH OF WHAT FOLLOWS IN BYTES (EXT. BINARY).
        <002>   DC.DAR==<002>
        <DATA>

**************************************************************

DATA REQUEST
        CNT     LENGTH OF WHAT FOLLOWS IN BYTES (EXT. BINARY).
        <003>   DC.STS==<003>
        DRQ     NUMBER OF DATA REQUESTS

**************************************************************
```

```
STATUS
        CNT     LENGTH OF WHAT FOLLOWS IN BYTES (EXT. BINARY).
        <004>   DC.CTL==<004>
        STC     STATUS CODE: BINARY VALUES
                STC.ER,0        ;DEVICE ERROR.
                STC.SB,1        ;SET BITS.
                STC.CB,2        ;CLEAR BITS.
                STC.XA,3        ;TRANSFER ABORTED.
        STD     DEVICE SPECIFIC STATUS: SCD,SLP,STY
                SCD--STATUS FOR CARD READER:
                  SCD.ME,0 ;MASTER ERROR (NOT SET BY EOF)
                  SCD.HE,1 ;HOPPER EMPTY.
                  SCD.RE,2 ;REGISTRATION ERROR.
                  SCD.IP,3 ;INVALID PUNCH FOR CHARACTER SET.
                  SCD.SF,4 ;STACKER FULL.
                  SCD.JF,5 ;JAM ON FEED.
                  SCD.JP,6 ;PICK FAILURE ON FEED
                            B7=1 (EXTENDED FIELD).
                  SCD.CZ,7 ;EOF CARD DETECTED.
                  SCD.HZ,8 ;HDW EOF.
                  SCD.OR,9 ;OVERRUN
                  SCD.OF,10 ;OFF LINE.
                  SCD.SR,11 ;STOP READING.
               STY--STATUS FOR TELETYPES:
                  STY.DE,0 ;DEFERRED ECHO.
                  STY.CV,1 ;CONVERT LC ON INPUT.
                  STY.XS,2 ;OUTPUT FROZEN BY ^S.
                  STY.II,3 ;IMAGE INPUT.
                  STY.IO,4 ;IMAGE OUTPUT.
                  STY.TP,5 ;TTY PAGE.
                  STY.TT,6 ;TTY TAPE.
                  STY.HT,7 ;HDW TABS.
                  STY.FF,8 ;HDW FORM FEED.
               SLP--STATUS FOR LINE PRINTERS:
                  SLP.FE,0 ;FATAL ERROR.
                  SLP.OL,1 ;OFFLINE (+OTHERS).
                  SLP.NP,2 ;OUT OF PAPER.
                  SLP.PJ,3 ;PAPER JAM.
                  SLP.OP,4 ;OPERATOR OFFLINE.
                  SLP.SQ,5 ;SLEW.
                  SLP.HM,6 ;HAMMER.
                            ;BIT 7=1.
                  SLP.LP,8 ;LOW ON PAPER.
                  SLP.SF,9 ;PAPER STACKER FULL.
                  SLP.NI,10 ;NO INK.
                  SLP.PQ,11 ;UNACCEPTABLE PRINT QUALITY.

**************************************************************
```

```
CONTROL
        CNT     LENGTH OF WHAT FOLLOWS IN BYTES (EXT. BINARY).
        <005>   DC.CTL=<005>
        DCT     DEVICE SPECIFIC CONTROL INFORMATION:
                  DCT.EP,0 ;ECHO PIPELINE MARKER (NO CDT FIELD)
                  DCT.CG,1 ;CHARACTER GOBBLER (NO CDT FIELD)
                  DCT.TC,2 ;TELETYPE CHARACTERISTICS.
                  DCT.AD,3 ;AUTO DIAL.
                           ;CDT = ASCII DIGITS OF NUMBER TO
                                  BE DIALED.
                 "FOR FILES"
                  DCT.SI,0 ;SET RECORD POINTER FOR INPUT.
                  DCT.SO,1 ;SET RECORD POINTER FOR OUTPUT.
                 "FOR LINE PRINTER"
                  DCT.LS,0 ;LOAD SKIP CHANNEL TAPE.
        CDT      CONTROL DATA, THIS OPTIONAL FIELD CONTAINS
                 CONTROL DATA FOR A DEVICE.
                 1.TTY CHARACTERISTICS:
                  # OF MILLISECONDS AFTER BACKSPACE <010>
                  # OF MILLISECONDS AFTER HORIZONTAL TAB <011>
                  # OF MILLISECONDS AFTER LF <012>
                  # OF MILLISECONDS AFTER VERTICAL TAB <013>
                  # OF MILLISECONDS AFTER FORM FEED <014>
                  # OF MILLISECONDS AFTER CARRIAGE RETURN <015>

                  RECEIVE SPEED (BITS/SEC, 134=2741)
                  TRANSMIT SPEED
                  WIDTH OF TTY CARRIAGE
                  AUTO CRLF POSITION
                  ELEMENT NUMBER

                  2741 BIT:
                   CDT.CB,0 ;"DEBREAK"FEATURE PRESENT
                   CDT.PL,1 ;APL MODE
                   CDT.TD,2 ;TIDY MODE

************************************************************

USER ID
        CNT       LENGTH OF WHAT FOLLOWS IN BYTES (EXT. BINARY).

        <006>     DC.UID==<006>
        PPN       PROJECT PROGRAMMER NUMBER (EXTENSIBLE ASCII).
        PSWD      PASSWORD
USER NAME (EXTENSIBLE ASCII)
        ACCT      ACCOUNTING CODE (EXTENSIBLE ASCII).
        GROUP     GROUP CODE (EXTENSIBLE ASCII).

************************************************************
```

```
FILE SPECIFICATION
        CNT         LENGTH OF WHAT FOLLOWS IN BYTES (EXT. BINARY).
        <010>       DC.FSP==<010>
        FST         FILE STATUS:
                     FST.IN,0 ;OPEN FOR INPUT.
                     FST.OU,1 ;OPEN FOR OUTPUT.
                     FST.AP,2 ;OPEN FOR APPENDING.
                     FST.UP,3 ;OPEN FOR UPDATING.
                     FST.DL,4 ;DELETE.
      FEA       FEATURES: "DCM"+"RLN"+"DVT".
                1. DCM=DATA CODE AND MODE.
                         DCM.AS,B0 ;ASCII
                         DCM.EB,B1 ;ABCDIC
                         DCM.IM,B2 ;IMAGE
                         DCM.HO,B3 ;HOLLERITH (CDR ONLY)
                         DCM.DI,B4 ;DEC IMAGE (CRD ONLY)
                         DCM.RV,B5 ;RESERVED
                         DCM.CP,B6 ;COMPRESSED FORMAT
                2. RLN=RECORD LENGTH (MAX IF VARIABLE):
                         EXTENSIBLE BINARY.
                3. DVT=DEVICE SPECIFIC ATTRIBUTES:
                       "DCD","DLP","DTY","DRX"
                         1. DCD=ATTRIBUTES FOR CARD READER.
                               BITS 0+1=SPEED:
                                  0=DONT CARE ABOUT CPM RATE.
                                  1=LESS THAN 300CPM.
                                  2=BETWEEN 300 AND 600 CPM.
                                  3=GREATER THAN 600 CPM.
                         2. DTY=ATTRIBUTES FOR TELETYPES.
                                  DTY.MC,B0 ;MODEM CONTROL.
                                  DTY.AS,B1 ;AUTO-BAUD
                                  DTY.SB,B2 ;HANDLER CAN SET BAUD
                                             RATE.
                                  DTY.27,B3 ;2741
                                  DTY.BD,B4 ;BAUDOT
                                  DYT.AD,B5 ;AUTO DIAL LINE
                                             (BELL 801).
```

```
                    3. DVT=DEVICE SPECIFIC ATTRIBUTES:
                            "DCD","DLP","DTY","DRX"
                            1.DCD=CARD READER ATTRIBUTES.
                                 (SEE ABOVE)
                            2.DLP=LINE PRINTER ATTRIBUTES.
                                  DLP.S0-0 ;SPEED (SEE DCD)
                                  DLP.S3-1 ;1^
                                  DLP.S6-2 ;2^
                                  DLP.S9-3 ;3^
                                  DLP.LL-B2 ; LOWER CASE
                                              REQ.
                                  DLP.RC-B3 ; REMOVE CHAR.
                                              SET REQ.
                                  DLP.MP -B4 ; MULTI-PART
                                               PAPER REQ.
                                  DLP.CS-B5 ;12 CHANNEL
                                    SKIPPING REQ.
                                         B7 ;B7=1.
                                  DLP.SK-B8=B9  ;SKIP REQ:
                                          0=DONT CARE.
                                          1=CHANGEABLE
                                            FROM HANDLER.
                                          2=CHANGEABLE AT
                                            SITE.
                                          3=CHANGEABLE
                                            BUT DONT CARE
                                            HOW.
                                  DLP.OP=B10 ;REQUIRES
                                              OVERPRINT.
                                  DLP.68-B11+B12
                                             ;6/8 LINES
                                              PER INCH
                                  DLP.CF-B13 ;CHANGEABLE
                                              FORM WIDTH.
                            3.DTY=TELETYPE ATTRIBUTES:
                                  (SEE DTY ABOVE)
                            4.DRX="REMOTE DATA ENTRY DEVICE"
                                   ATTRIBUTES:
                                   DRX.MD,B0 ;MULTI DROP
                                              LINE DATA
                                              ENTRY LINE.
                                   DRX.PL,B1 ;LINE ACCEPTS
                                              A POLL
                                              SEQUENCE.
FDES    FILE DESCRIPTION, EXTENSIBLE ASCII FIELD OF
        FORM: "DEV:[P,PN]FILE.EXT".
```

This page intentionally left blank

Module Test

For this module test, feel free to consult with other members of the class.

1. Using the sample message supplied by your instructor and the module text, determine the message type and the message content, the source and destination, and the message numbers.

2. Using the text supplied by your instructor, write down (in octal) the message which would be sent from the source to the destination. Also, indicate whether an ACK (with what number) would be returned to the source node.

This page intentionally left blank

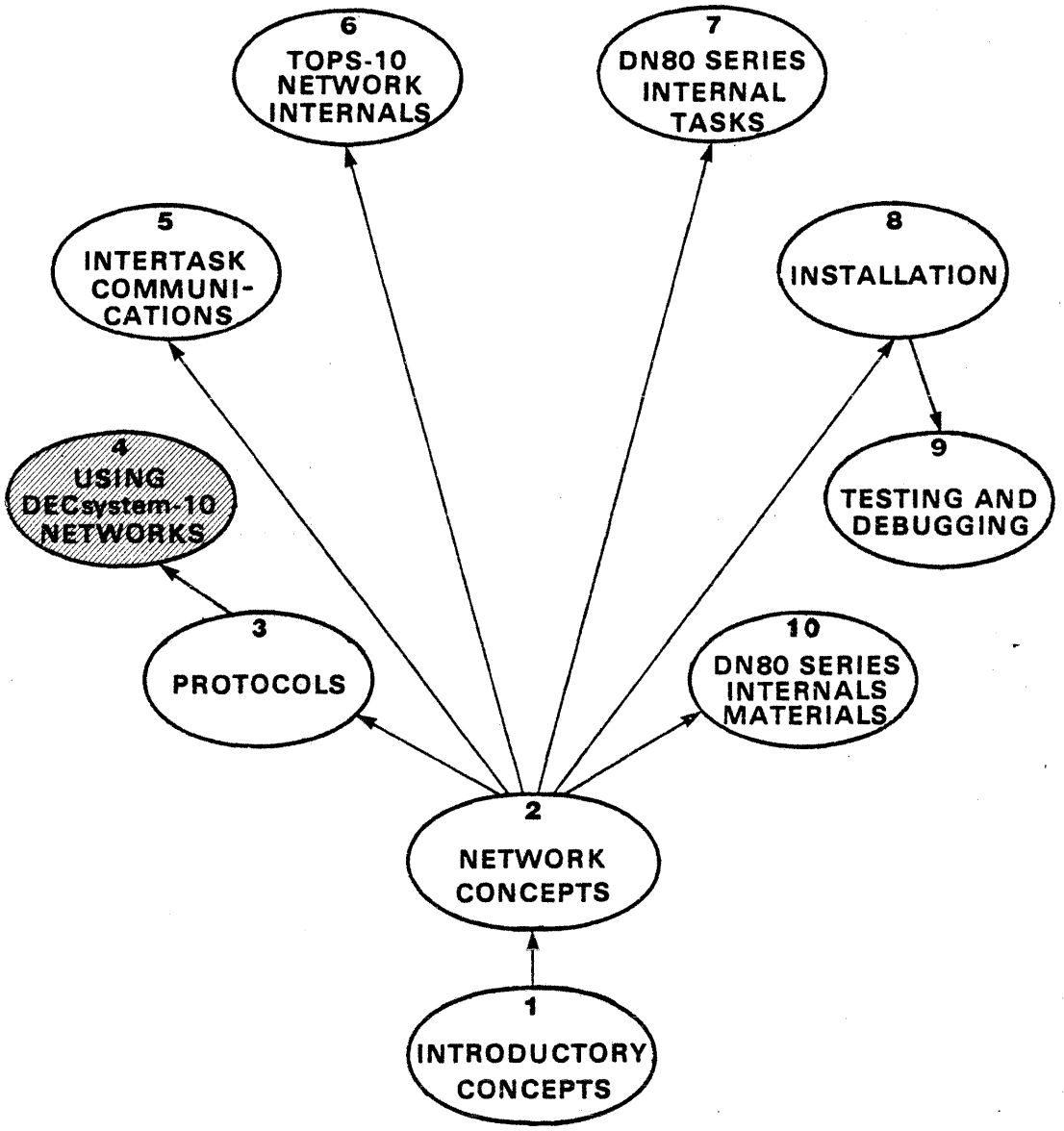
Evaluation Sheet

Consult with your instructor for the correct messages for your particular problems. You may also have a fellow student check your answers.

This page intentionally left blank

Using DECsystem-10 Networks
Module 4

Course Map



M8 0277

Introduction

This module covers the use of DECsystem-10 networks. Monitor commands and monitor calls (UUO's) which are network specific are introduced.

Objectives

Upon completion of this module, the student will be able to

1. Use the various network commands.
2. Use the network monitor calls.

Additional Resources

DECsystem-1Ø Software Notebooks
DECsystem-1Ø Networks Programmers Guide and Reference Manual

Overhead 4-1

NETWORKS

INTERCONNECTION OF SYSTEMS

* PROCESSING SYSTEMS

* COMMUNICATIONS CONTROL SYSTEMS

* REMOTE STATIONS

DECsystem-10 Networks Uses and Capabilities

DECsystem-10 Networks is the collective name for the set of software products that extend the facilities of various DIGITAL operating systems so they may be interconnected to form computer networks. The various systems interconnected in a computer network fall into three general categories:

1. Processing systems

2. Communications control systems

3. Remote stations

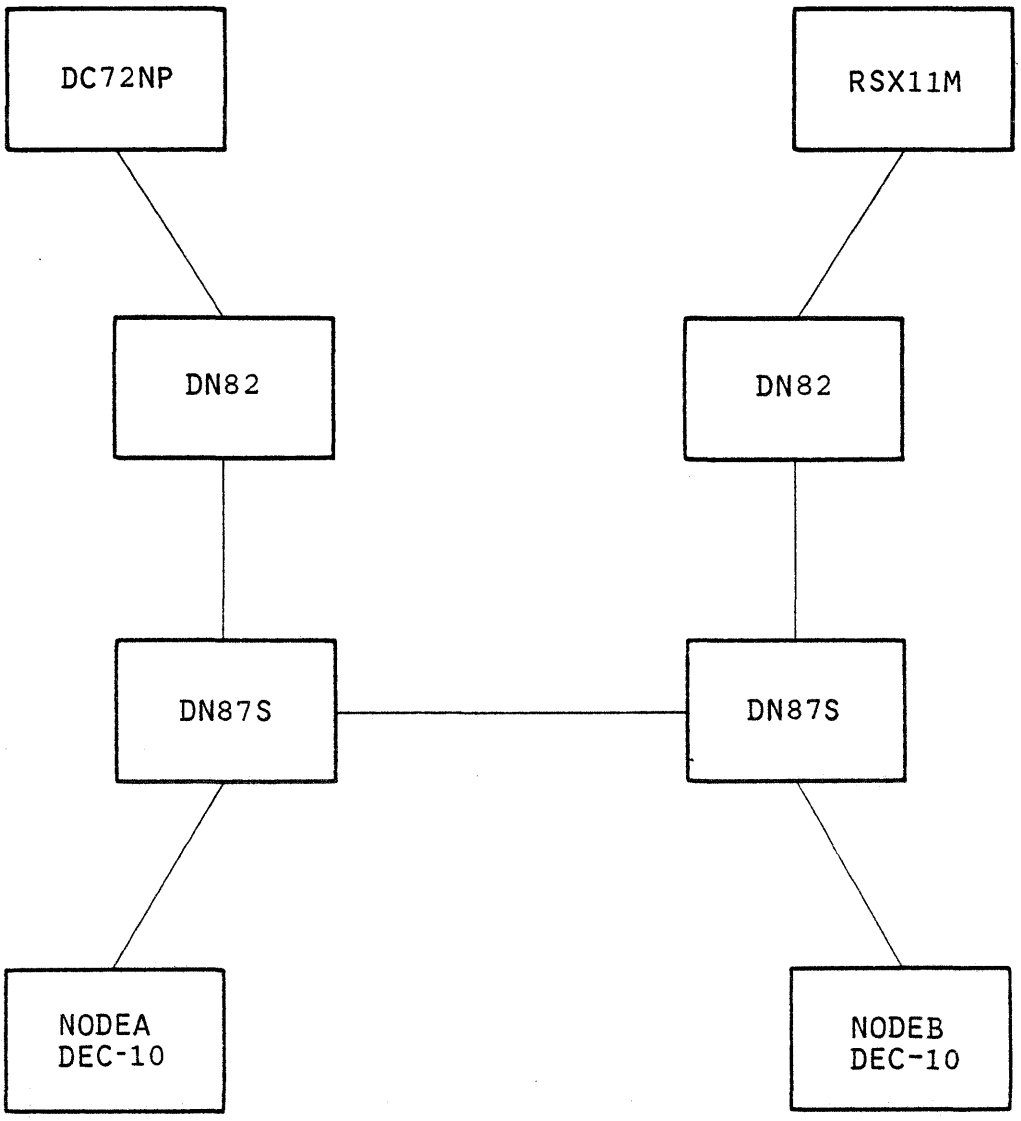
A processing system is a computer with an operating system that executes user software, and optionally provides additional facilities for program development. Examples: DECsystem-10 and RSX-11M.

A communications control system is a general-purpose communications facility that acts as an interface between a processing system and a network. A communications control system has dedicated software and supports the communications protocol of the network. A communications control system is required to act as a network front-end for each DECsystem-10 in a network. Examples of these systems are the DC75-NP and DN87-S Synchronous Communication Systems.

A remote station is a system that allows access to the network from locations that may be distant from a processing system. Remote stations can usually support a variety of input/output devices such as terminals, line printers, and card readers. Typical computers in this category are the DC72-NP and DN80, DN81, and DN82 Remote Stations.

In a network environment, each computer system is called a "node."

Overhead 4-2



M8 0104

Overhead 4-3

CAPABILITIES

Resource sharing

Distributed access to data base

Parallel processing of jobs

Load balancing between systems

Centralized processing of distributed data

Overhead 4-4

Network Devices/Device Names

gggnnu

where:

ggg = the generic device name (e.g., LPT, CDR).

nn = the 2-digit node number (e.g., 01, 10, 77).

Leading zeros are required

u = the 1-digit physical unit number in the range 0 through 7.

Overhead 4-5

EXAMPLES:

CDR123 - Card reader number 3 at node number 12.

LPT010 - Line printer number 0 at node number 1.

DEFAULTS:

LPT02 - Any line printer at node 2.

CDR12 - Any card reader at node 12.

Overhead 4-6

Network Commands

ASSIGN
SEND
SET HOST
NODE
WHERE
LOCATE

The search for the default node generic device is conducted according to the following rules:

1. The system searches for the named device at the node where the user is logically located. If the device exists and is available, the user is given access to the device.

2. If the device does not exist at the user's logical node, a search is conducted at the user's processing node. If the device exists and is available, the user is given access to the device.

3. If both of the above rules fail, an appropriate message is displayed on the terminal and control is returned to the monitor.

For example, if the requesting user is located at node number 2, and the processing node is number 1, specifying LPT1 causes a search for the device LPTØ21 at the user's node 2. If the device does not exist, the system searches for device LPTØ11 at the processing node number 1. If this search also fails, an error message is displayed on the user's terminal.

ASSIGN

Function:

The ASSIGN command allocates an I/O device to the user's job for the duration of the job or until a DEASSIGN command is given. When an assignment is performed, the system displays a message indicating the physical device that is assigned.

Command Formats:

ASSIGN node-id_phys-dev log-dev

where node-id = a node identifier of the node on which the device is to be assigned. This identifier can be either the node-name or the node-number.

phys-dev = any physical network assignable device listed in Table 4-1. The device specification must be of the form gggnnu. The node and unit numbers may be omitted and the default search rules applied. If the device is not available in accordance with the search rules, the monitor prints an error message.

If a node-number is specified in the nn field of the phys-dev specification and the number does not correspond with the node-id specified, an error message is printed.

An underscore (_) must separate the node-id and the physical device name. The "back-arrow" character is equivalent to underscore.

Overhead 4-7

ASSIGN NODE-ID_PHYS-DEV LOG-DEV

```
.ASSIGN DALLAS_LPT
LPT030:  ASSIGNED

.ASSIGN BOSTON_LPT LPT
LPT040:  ASSIGNED
```

Assign any card reader on node 22.

```
.ASSIGN CDR22
CDR220:ASSIGNED
```

Overhead 4-8

ASSIGNABLE DEVICES

DEVICE	GENERIC NAME	NETWORK ASSIGNABLE?
CARD PUNCH	CDR	NO
CARD READER	CDR	YES
CONSOLE TERMINAL	CTY	NO
DISK	DSK	NO
LINE PRINTER	LPT	YES
MAGNETIC TAPE	MTA	NO
OPERATOR TERMINAL	OPR	NO
PAPER TAPE PUNCH	PTP	NO
PAPER TAPE READER	PTR	NO
SYSTEM LIBRARY	SYS	NO
TASK	TSK	YES
TERMINAL	TTY	YES

TABLE 4-1

Overhead 4-9

SEND

OPR - operator of node where user is LOCATEd

OPRnn - operator at node nn

TTYnn - terminal nn

JOBnn - job nn

Overhead 4-10

SET HOST xxx

xxx = node name

or

node number

SET HOST

Function:

The SET HOST command enables the user on a DN81, DN82, or DN87(S) remote station to connect to the command decoder of another network system. If logged-in, the user's job is detached at the node upon which the command is issued, and the terminal is then connected to the node specified in the command string.

The node specified must be a network node on which users can perform LOGIN or an equivalent function.

Using the SET HOST command, the user may remain at one terminal and have the logical terminal connection switched from one system to the command decoder of a different system.

After issuing the SET HOST command, the user can perform the LOGIN process or use the ATTACH command to obtain processing capabilities at the new host system.

Command Format:

SET HOST node-id

where node-id = an identifier of the node to which the user wishes to connect.

Characteristics:

The SET HOST command

Requires the terminal to be on a DN81, DN82, or DN87(S) remote station.

Leaves the terminal in monitor mode.

Does not require user to be logged-in.

Does not destroy the user's core image.

NODE

Function:

The NODE command prints the network configuration information. If a node-ID is not specified, the node-ID and related information for each known node is displayed. If the user specifies a node-ID, only the node information for the specified node is displayed.

Command Format:

NODE node-id

where node-id = a node identifier of a node in the network, or null to list entire network.

Overhead 4-11

NODE

Examples:

```
.NODE
NODE KI592(1)NETMON#592 0.21 02-26-75
        MCR[1],TTY[40],CDR[1],LPT[1]
NODE DALLAS(3)
        TTY[33],CDR[1],LPT[1]
NODE BOSTON(2)
        TTY[1]

.NODE DALLAS
NODE DALLAS(3)
        TTY[33],CDR[1],LPT[1]
```

WHERE COMMAND

Function:

The WHERE command enables the user to determine

1. the node-name

2. the node-number

3. the software I.D. (monitor name)

4. the creation date of the monitor software

of the node at which a specific peripheral device is located. If the node of a particular terminal is requested, the information returned is that of the physical location of the terminal. The information may or may not be that of the controlling job, depending on whether the user changed the job's logical location with the LOCATE command. This command is particularly useful to determine where the user is within the network, especially if he has been switching from node to node with the LOCATE command.

Command Format:

WHERE devn

dev = any physical device and n is the unit number

The output is in the form:

NODE node-name(node-num) software-id creation-data

If the specified device is not recognized by the system, the following message is printed:

?NO SUCH DEVICE

Overhead 4-12

WHERE

```
.WHERE LPTØ1Ø:
NODE NYC(1) NETMON #999 Ø.14 Ø2-19-77

.WHERE TTY:
NODE LAB22(4) NETMON #596 Ø.22 Ø3-24-77

.WHERE OPR:
NODE NOME(1) NETMON #323 Ø.32 Ø3-Ø3-77

.WHERE CTY:
NODE KI434(5) NETMON #443 Ø.14 Ø3-24-77
```

LOCATE COMMAND

Function:

The LOCATE command logically establishes the user's job at a specified network node. When the job is initiated, the logical location corresponds to the user's physical location; therefore, this command is needed only if the user wants to change the logical node.

This command is used to alter the default device search list of the job. For instance, to perform all I/O via the devices at remote station PARIS, specify

.LOCATE PARIS

and the default I/O devices will be those on node PARIS. The LOCATE command does not change the physical location of the job; the job runs on the node on which it is initiated.

Command Format:

LOCATE node-id

where node-id = an identifier of the node upon which the job is to be logically located. If node-id is 0, the job is located at the node of the job's command decoder. If node-id is omitted, the job is located at the node of the user's terminal, i.e., the physical node.

This command also changes the location of the line printer which receives spooled output. For example, LOCATE 7 causes the PRINT command to send data to the printer on node 7. This feature of LOCATE can also be used to print data on an IBM 3780-like terminal using a DN61.

NOTE

A batch job which was initiated from cards, is initially located based on the card reader on which the job was read in.

Overhead 4-13

USING NETWORK DEVICES

```
.ASSIGN NODEA CDR
   CDR120:   ASSIGNED
     .COPY WEX.MAC=CDR:



.ASSIGN LPT12  FOO
   LPT122:   ASSIGNED
     .COP FOO:=WEX.MAC
```

Overhead 4-14

Network Monitor Calls (UUO's)

LOCATE

GTNTN.

GTXTN.

NODE.

WHERE

(CALL1.)

Overhead 4-15

MONITOR CALLS

DECsystem-10 monitor calls are programmed operators whose functions are not prespecified by the hardware. Monitor calls are used to effect a system command function from a running program. The systems programmer may use these monitor calls as software-implemented instructions. For an in-depth discussion of monitor calls see the DECsystem-10 Monitor Calls Manual.

CALL and CALLI Monitor Operations

CALLI	CALLI Mnemonic	Call	Function
CALLI AC,62	LOCATE	MOVEI AC,node-num LOCATE AC, error return normal return	Change the job logical node
CALLI AC,165	GTNTN.	MOVE AC,[XWD 0, terminal-number] GTNTN. AC, error return normal return	Return the line number and the station number or the terminal number specified
CALLI AC,164	GTXTN.	MOVE AC,[XWD node-number, line-number] GTXTN. AC, error return normal return	Return the physical TTY name of a terminal attached to a specified line number

CALLI AC,157	NODE.	MOVE AC,[XWD function-code,E] NODE AC, error return normal return E:XWD 0,count argument-list	Perform the NODE function on the node specified
CALLI AC,63	WHERE	MOVEI AC,channel no. or MOVE AC,[SIXBIT /dev/] WHERE AC, error return normal return	Return the physical node of the device

Overhead 4-16

LOCATE AC, or CALLI AC,62

The LOCATE monitor call changes the logical node associated with the user's job. The call is

```
MOVEI AC,node-num    ;or MOVE AC,[SIXBIT/node-name/]
LOCATE AC,           ;or CALLI AC,62
  error return
normal return
```

The node number requested is contained in AC as follows:

-1 changes the job's location to the physical node of the job's terminal.

Ø changes the job's location to that of the job's command interpreter node.

n changes the job's location to node n.

The normal return is taken if the monitor call is implemented and the specified node is a valid one. Subsequent generic device specifications are at the new node if the devices exist and are not busy. The error return is taken if the monitor call is not implemented.

EXAMPLES:

1. Logically locate the job at node number 3. If the monitor call fails, transfer control to the statement at label ERROR. If successful, transfer to label WINNER.

```
MOVEI AC,3
LOCATE AC,
   JRST ERROR
JRST WINNER
```

2. Logically locate the job at node NODEA. If the monitor call fails, go to label TOOBAD. If successful go to label OK.

```
MOVE AC,[SIXBIT/NODEA/]
LOCATE AC,
   JRST TOOBAD
JRST OK
```

Overhead 4-17

GTNTN., or CALLI 165

The GTNTN. monitor call returns the line number and the station number of a specified terminal number. Its calling sequence is shown below.

```
MOVE AC,[XWD 0, terminal-number]
GTNTN. AC,
  error return

  normal return
```

where terminal-number is the SIXBIT number received as a result of a TRMNO. monitor call. This number is the physical name of the terminal received when a user issues the INITIA command (e.g., TTY107).

On a normal return, the monitor returns the following information in the AC.

```
node (station) number,,line-number
```

The node number is the number of the node at which the specified terminal number is located. The line-number, on non-network systems, is the equivalent of the terminal number. On a networks system, line-number's value is not equal to the terminal-number and is dependent on the issuance of the SET HOST command.

On an error return, the monitor returns an error code in the AC. Table 4-2 lists the possible error codes.

Table 4-2
GTNTN. Error Codes

Code	Mnemonic	Meaning
0	NTNSD%	No such device as the one specified.
1	NTNAT%	Device specified is not a TTY.
2	NTTNC%	The specified TTY is not connected.

For an example of the GTNTN. and GTXTN. monitor calls see the next section.

Overhead 4-18

GTXTN., or CALLI 164

The GTXTN. monitor call returns the physical TTY name of a terminal attached to a specified line number. Its calling sequence is shown below.

```
MOVE AC,[XWD node-number,line-number]
GTXTN. AC,
  error return
normal return
```

where node-number and line-number are the results obtained from the GTNTN. monitor call.

On a normal return, the monitor returns the terminal's physical TTY name in the AC (e.g., TTY107).

On an error return, the monitor returns an error code in the AC. Table 4-3 lists the possible error codes.

Table 4-3
GTXTN. Error Codes

Code	Mnemonic	Meaning
0	XTUNT%	The device specified is an unknown network TTY.
1	XTNLT%	The device specified is not a local TTY.

Overhead 4-19

EXAMPLE:

AN EXAMPLE OF THE GTXTN. and GTNTN. monitor calls is shown below.

```
          SEARCH UUOSYM

T1 =  1
T2 =  2
T3 =  3
T4 =  4
P  = 17

START:    PJOB      T1,                    ;GET JOB NUMBER
          MOVEM     T1,JOB                 ;SAVE IT
          TRMNO.    T1,                    ;GET TERMINAL NUMBER
            JRST    DET                    ;CHECK TO SEE IF DETACHED
          SUBI      T1,.UXTRM              ;SUBTRACT OFFSET
          MOVE      T2,[%FTPER]            ;GET FEATURE TEST WORD FOR
                                           ;NETWORKS
          GETTAB    T2,
            POPJ    P,                     ;SHOULD NOT HAPPEN
          TRNN      T2,F%REM&777777        ;IS THERE A NETWORK?
          POPJ      P,                     ;NO, LINE NUMBER IS IN T1
NET:      MOVEI     T4,3                   ;YES, GET LINE NUMBER
NET1:     IDIVI     T1,10                  ;FIRST CONVERT NUMBER TO
                                           ;SIXBIT
          LSHC      T2,-6
          SOJG      T4,NET1
          HLRZS     T3,                    ;PUT INTO RIGHT HALF
          ADD       T3,[SIXBIT/TTY000/]    ;AND MAKE IT SIXBIT
          GTNTN.    T3,                    ;RETURNS NODE NUMBER,,LINE
                                           ;NUMBER
            JFCL
          GTXTN.    T3,                    ;NOW GET TTYNNN IN SIXBIT
            JFCL
          POPJ      P,
DET:      MOVN      T1,JOB                 ;GET TTY STATUS WORD
          JOBSTS    T1,
            POPJ    P,                     ;ERROR
          JUMPL     T1,DETJOB              ;JOB IS DETACHED IF BIT 0
                                           ;IS ON, TAKE APPROPRIATE
                                           ;ACTION
                                           ;TAKE SOME ACTION
          JRST      NOJOB

JOB:      BLOCK 1
          END START
```

Overhead 4-20

NODE. AC, or CALLI AC,157

The NODE. monitor call performs the following functions:

1. assign a logical device name to a specified physical device.
2. return the node number associated with a specified node name.
3. return the node name associated with a specified node number.
4. perform special station control messages.
5. request bootstrap messages.
6. return system configuration information.

The calling sequence for the NODE. monitor call is shown below.

```
MOVE   AC,[XWD function-code,addr]
NODE.  AC,
  error return
normal return
```

addr:argument-list

where function-code specifies which one of the possible functions this NODE. call is to perform. Refer to Table 4-5 for a list of the possible functions.

addr is the address of the argument block. The contents of the argument block are different depending on which function code is specified.

argument-list is the first word of the argument block. The argument block is three to five words long, depending on the function code specified. The argument block formats are described in Table 4-5.

Overhead 4-21

On a normal return, the monitor leaves the AC unchanged. If the block for the device does not exist at the time of the monitor call, a device block is built by the monitor and communication is established with the remote device.

The error return is taken when one of the following conditions occurs:

1. The argument list specified is not correct.
2. An illegal node name was specified.
3. An illegal node number was specified.
4. Remote node control was requested but is not available.
5. The calling job is not locked in core.
6. The monitor cannot perform the logical device name assignment that was requested.

See table 4-5 for the NODE. error codes.

Overhead 4-22A

Table 4-4
NODE. Function Codes

Code	Mnemonic	Argument Block	Function
1	.NDALN	XWD 0,#-of-args SIXBIT/node-name/ or SIXBIT/node-number/ SIXBIT/devicename/ SIXBIT/logicalname/	Assign logical device name specified in the argument block to the physical device specified in the AC. If the logical name is omitted from the argument list, the monitor will assume that the logical name is the same as the physical name specified.
2	.NDRNN	XWD 0,#-of-args SIXBIT/node-name/or SIXBIT/node-number/or XWD 0,node-number	Return the node number in the AC, given the node name. or, return the node name in the AC, given the node number.
3	.NDSSM	XWD time,#-of-args SIXBIT/node-name/or SIXBIT/node-number/or XWD 0,node-number XWD #-of-bytes,addr XWD #-of-bytes,addr	Perform special station control message. the argument in addr+2 specifies the number of bytes that are to be transmitted, and the address of these bytes. The argument in addr+3 if present, specifies the number of bytes that are to be received, and the

address of the location that is to receive them. The time (bits 9-17) is the number of seconds before a time-out error.

Overhead 4-22B

Code	Mnemonic	Argument Block	Function
4*	.NDRBM	XWD 0,#-of-args 0 0 XWD #-of-bytes,addr	Request bootstrap messages from a remote node. The monitor returns the node number in addr+1; but addr+2 is unused. The argument specified in addr+3 indicates the number of bytes to be received and the location that is to receive them. This is a privileged function. The calling job must be a privileged job and locked in core.
5	.NDRCI	XWD 0,#-of-args SIXBIT/node-name/ or SIXBIT/node-number/ 0 XWD #-of-devs,dev-type . . . XWD #-of-devs,dev-type	Return the number and type of devices on a specified node. Your program fills in words 0, 1, and 2 of the argument block. the monitor returns information, beginning with word 3.Note you set word 2 to zero.

* This is a privileged function, calling job must be privileged and locked in core.

Overhead 4-23

Table 4-5
NODE. Error Codes

Code	Mnemonic	Error
1	NDIAL%	You did not set up the argument block properly.
2	NDINN%	You specified either an illegal node name or an illegal node number.
3	NDPRV%	Your job is not a privileged job, and the function you specified requires privileges.
4	NDNNA%	The node you specified is not available.
5	NDNLC%	Your job is not locked in core and it should be in order to perform the specified function.
6	NDTOE%	A time-out error occurred.
7	NDRNZ%	Your program specified function code 5, but word 2 of your argument block was not zero.

Overhead 4-24

This example, upon normal return, causes all references to logical device LPTA to be associated with physical device LPT on node NODEA.

```
        MOVE AC,[XWD 1,ADR]
        NODE AC,
           JRST NOGOOD
        JRST GOOD
        .
        .
        .
ADR:    XWD Ø,4                 ;four arguments in list
        SIXBIT /NODEA/          ;6-bit node-ID
        SIXBIT /LPT/            ;6-bit device name
        SIXBIT /LPTA/           ;6-bit logical name
```

Overhead 4-25

WHERE AC, or CALLI AC,63

The WHERE UUO returns the physical node number of the specified device. When the UUO is called, AC must contain either the channel number of the device as a right-justified quantity, or the logical device name as a left-justified SIXBIT quantity. The call is

```
MOVE AC,[SIXBIT/dev/]     ;or MOVEI AC, channel no.
WHERE AC,                 ;or CALLI AC,63
   error return
normal return
```

If OPR is specified as the device name, the node number at which the job is logically located is returned in the right-half of AC. If zero is specified, the node number of the job's command decoder is returned in the right-half of AC. If TTY is specified, the node number at which the job's terminal is physically located is returned.

On a normal return, the left half of AC contains the status of the node, and the right half of AC contains the node number associated with the device. The status of the node is represented by the following bits:

```
Bit 13 = 1  if the node is dial-up (.RMSDU).
Bit 14 = 1  if the node is loaded (.RMSUL).
Bit 15 = 1  if the node is in the loading procedure (.RMSUG).
Bit 16 = 1  if the node is down (.RMSUD).
Bit 17 = 1  if the station is not in contact (.RMSUN).
```

The error return is taken if the UUO is not implemented, the channel specified is not INITed, or the requested device does not exist.

Overhead 4-26

EXAMPLES:

1. Get the number of the node where logical device LPT0 is located. If successful, go to WINNER. If UUO fails, go to ERROR.

```
MOVE AC,[SIXBIT/LPT0/]
WHERE AC,
   JRST ERROR
JRST WINNER
```

2. Get the number of the node where the job's command decoder is located. If successful, go to CONTIN. If the UUO fails, go to label NOMCR.

```
MOVEI AC,0
WHERE AC,
   JRST NOMCR
JRST CONTIN
```

EXAMPLES:

Overhead 4-27A

```
.NODE
NODE    KL1026(26)      SCLIKB KL10 SYS#1026 09-05-77
        MCR[1] CDR[1] LPT[3] PTR[1] PTP[1] MTA[9] DTA[8] TSK[16]
NODE    DN8703(3) DN87 V11(23) 05-APR-77
        TTY[17] CDR[1] LPT[1] PTR[1] PTP[1]
NODE    DAS92(72) DAS92 V0.3 31-MAR-77
        TTY[13] LPT[1]
NODE    KL1242(10)      RS300  KL10 SYS#1242 09-28-77
        MCR[1] LPT[1] PTR[1] PTP[1] PLT[1] MTA[12] DTA[2] TSK[128]
NODE    CTCH22(22) DN82 V13(27) 22-AUG-77
        TTY[33] CDR[1] LPT[1]
NODE    DS401A(2) DN87 V11 02-AUG-76
        TTY[17]
NODE    NOVA(31) DN87S V13(27) 22-AUG-77
        TTY[49]
NODE    ENCORE(32) DN87S V13(27) 10-SEP-77
        TTY[5] RDA[1]
NODE    EJWMN(16) DN87 V13(27) 22-AUG-77
        TTY[65]
NODE    CYNIC(66) DN82 V13(27) 22-AUG-77
        TTY[17] CDR[1] LPT[1]
NODE    KI514(14)      RX3A1A SYS #514/546 05-11-77
        MCR[1] CDR[2] LPT[3] PTR[1] PTP[1] PLT[1] MTA[18]
        DTA[9] TSK[16]
NODE    IT(15) DC75NP

NODE    NEXT(27) DN87 V13(27) 22-AUG-77
        TTY[1]
```

Overhead 4-27B

```
.WHERE TTY
NODE    NOVA(31) DN87S V13(27) 22-AUG-77 TTY327 LINE # 45

WHERE OPR
NODE    NOVA(31) DN87S V13(27) 22-AUG-77 TTY260 LINE # Ø

WHERE CTY
NODE    KL1Ø26(26)      SCLIKB KL1Ø SYS #1Ø26 Ø9-Ø5-77
                        CTY LINE #421

.WHERE OPRØ
NODE    KL1Ø26(26)      SCLIKB KL1Ø SYS#1Ø26 Ø9-Ø5-77
                        TTY4 LINE # 4

.WHERE LPT
NODE    KL1Ø26(26)      SCLIKB KL1Ø SYS#1Ø26 Ø9-Ø5-77 LPT262

.SET HOSTESS 14

.
     RX3A1A SYS #514/546 11:Ø9:31

.WHERE TTY
NODE   NOVA(31) DN87S V13(27) 22-AUG-77 TTY327 LINE # 45

.WHERE OPR
NODE   KI514(14)  RX3A1A SYS #514/546 Ø5-11-77 CTY LINE # 451

.WHERE CTY
NODE   KI514(14)   RX3A1A SYS #514/546 Ø5-11-77 CTY LINE # 451

.WHERE OPR31
NODE   KI514(14)   RX3A1A SYS #514/546 Ø5-11-77 CTY LINE # 451

.WHERE OPRØ
NODE   KI514(14)    RX3A1A SYS #514/546 Ø5-11-77 CTY LINE # 451

WHERE LPT
NODE   KI514(14)    RX3A1A SYS #514/546 Ø5-11-77LPT14Ø
```

Overhead 4-28

```
.SET HOSTESS 26


.
    SCLIKB KL10 SYS#1026 11:14:18

.WHERE OPR
NODE   NOVA(31)  DN87S V13(27) 22-AUG-77 TTY260 LINE # 0

.WHERE LPT 140
NODE   KL1026(26)   SCLIKB KL10 SYS#1026 09-05-77 LPT262
```

This page intentionally left blank

Module Test

1. What monitor command would you use to send the message "HELLO" to the CTY of node CYNIC? (Refer to the sample NODE output below.)

```
                              MODULE TEST
.NODE     KL1026(26)          RL333A KL10 SYS #1026 09-28-77
          MCR[1] CDR[2] LPT[3] PTR[1] PTP[1] MTA[9] DTA[8]
          TSK[16]
NODE      CTCH22(22) DN82 V13(27) 22-AUG-77
          TTY[33] CDR[1] LPT[1]
NODE      ENCORE(32) DN87S V13(27) 10-SEP-77
          TTY[5] RDA[1]
NODE      NOVA(31) DN87S V13(27) 22-AUG-77
          TTY[49]
NODE      EJWMN(16) DN87 V13(27) 22-AUG-77
          TTY[65]
NODE      CYNIC(66) DN82 V13(27) 22-AUG-77
          TTY[17] CDR[1] LPT[1]
NODE      KI514(14)           RX3A1A SYS #514/546 05-11-77
          MCR[1] CDR[2] LPT[3] PTR[1] PTP[1] PLT[1] MTA[18]
DTA[9]    TSK[16]


NODE      IT(15) DC75NP

NODE      NEXT(27) DN87 V13(27) 22-AUG-77
          TTY[1]
```

2. Using the NODE output above, how many LPT's are there in the network?

3. If your program was running on NODE 14 and you wanted output on NODE 66, what are 2 ways of causing that to happen?

4. How would you assign the card reader on node CTCH22 if you were LOCATEd on node 66?

5. Given the sequence

```
.SET HOST 26
.LOCATE CYNIC
.ASSIGN PTR
.ASSIGN LPT
```

What devices would be assigned?

6. Write (on paper or an editor) a MACRO routine which will use the network UUO's to

Determine where your job is logically located;

Determine where your terminal is physically located if these are different LOCATE to the node where your job is physically located;

Determine the node name where you are now located;

Determine the TTY name of line 0 on this node.

Note: It is permissable to ignore normal macro overhead for this code; that is, just write the relevant code as though that code was part of a larger program.

Overhead 4-29

Evaluation Sheets

1. SEND OPR66: HELLO<CR>

2.

NODE	LPTS
26	3
22	1
32	0
31	0
16	0
66	1
14	3
15	0
27	0
TOTAL	8

3. a. LOCATE 66

   b. LOCATE CYNIC

4. ASSIGN CDR221

5. a. ASSIGN PTR
      PTR260 ASSIGNED
      (THERE IS NO PTR ON NODE CYNIC (66))

   b. ASSIGN LPT
      LPT660 ASSIGNED

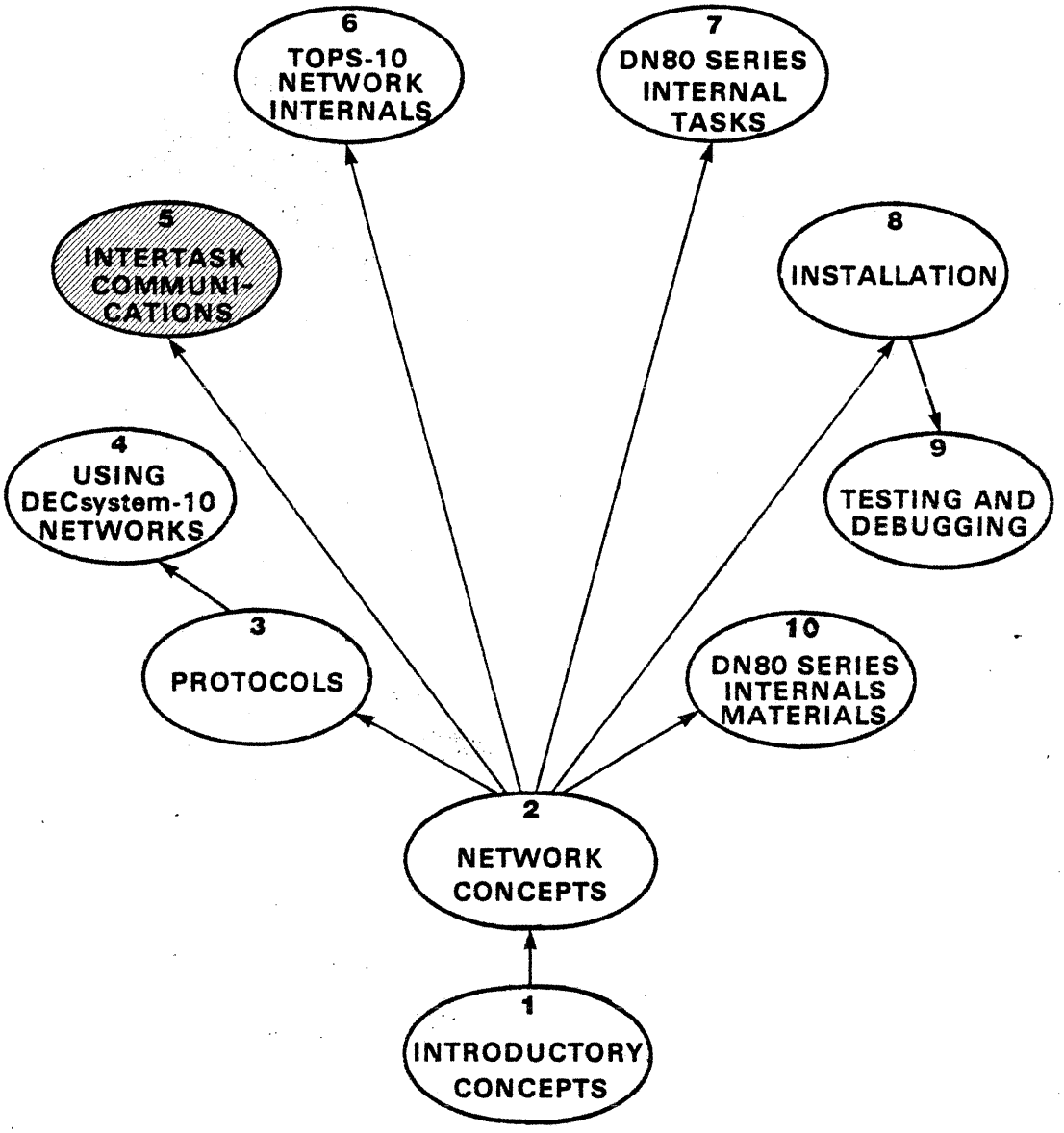
6.

```
P6:     MOVE    T1,[SIXBIT /OPR/]       ;SPECIFY DEVICE OPR
        WHERE   T1,                     ;GET LOGICAL LOCATION
          JRST NONET                    ;NO NETWORK SUPPORT IMPLEMENTED

        HRRZ    T2,T1                   ;SAVE THIS NODE# IN T2

        MOVE    T1,[SIXBIT /TTY/]       ;SPECIFY CONTROLLING TERMINAL

        WHERE   T1,                     ;GET PHYSICAL LOCATION
          JRST NONET
        HRRZ    T3,T1                   ;SAVE THIS NODE # IN T3

        CAMN    T3,T2                   ;ARE THESE TWO NODE #'S THE
                                        ;SAME?
        JRST    P6A                     ;YES, SKIP LOCATE CODE
        MOVEI   T1,-1                   ;NO, SPECIFY PHYSICAL NODE
        LOCATE  T1,                     ;LOCATE THERE
          JRST NONET

P6A:    HRRZI   T2,2                    ;MAKE ARGUMENT BLOCK FOR NODE.

                                        ;REMEMBER T3 STILL HAS NODE
                                         #
        MOVE    T1,[XWD 2,T2]           ;SET UP FUNCTION CODE,,ARG
                                         ADDR
        NODE.   T1,                     ;GET NODE NAME
          JRST NODERR                   ;NODEL. ERROR
        MOVEM   T1,NODNAM               ;SAVE NODE NAME

        HRLZ    T1,T3                   ;GET NODE,,0 INTO T1
        GTXTN.  T1,                     ;NOW GET THE TTY NAME
          JRST STERR                    ;GTXTN. ERROR
        JRST DONE
```

Intertask Communication
Module 5

<<For Internal Use Only>>

Course Map



M8 0277

Introduction

Transferring files between nodes in a network provides a means for remote data acquisition and efficient file storage and maintenance. Since files on one network node can be accessed from any other node, only one copy of a file need be stored and maintained in the network. Thus, the problem of maintaining up-to-date multiple copies of a file is eliminated. Note that the problem of multiple users updating copies is not addressed. Files may be transferred in two ways: by a system utility program such as PIP or by a user program.

Files can be conditionally transferred by a user program. For example, to find the latest version of a file and print it, the programmer can write a program that performs a LOOKUP of that file on each network node and transfers the most recent version of the file to the line printer. To perform these LOOKUPs and transfers, cooperating processes must be set up.

This module provides the information needed to understand the interface, from a user level, between tasks on various nodes.

Objectives

Upon completion of this module, the student will be able to

1. transfer a file from one node to another on a DECsystem-1Ø network.
2. describe the information necessary for inter-task communication in a higher level language.

Additional Resources

DECsystem-1Ø Software Notebooks

DECsystem-1Ø Networks Programmer's Guide and Reference Manual

Tasks - Definition (TSK:)

Intertask communication is the transfer of data from one running task to another on the same node or different nodes. Communication between a task running on the DECsystem-10 and a task running under RSX-11 is accomplished by using the DECnet Compatible Port. Once the DECnet Compatible Port software is installed, the user can perform intertask communications between the systems in the same manner as he would communicate between two DECsystem-10s. For a description of the DECnet Compatible Port software, see the DECsystem-10 Networks Software Installation Guide (DECsystem-10 Software Notebooks).

The two tasks involved in the data transfer must cooperate to initiate the connection. During initiation, one task assumes a passive role and the other assumes an active role. File transfers from user programs are easily implemented using READ and WRITE statements. Internode file transfers may be performed using PIP, TECO, and any language that allows logical file access for reading and writing.

The first example below, the TECO File Transfer, is used to illustrate that a task (TSK:) may be considered to be an I/O device as the physical I/O devices are.

File Transfer

TECO FILE TRANSFER

The Passive Task

Node name = NODEA Node number = 03

.SET HOST NODEA	The task to receive the file is to be run on node NODEA; therefore, attach to the command decoder of node NODEA.
.LOGIN...	Log a job into NODEA.
.R TECO	Run the TECO program.
*ERTSK:TECO$$	Perform a read from device TSK. The job will go into I/O wait state until connected to.
*EWDSK:OUTFIL.MAC$$	After reading from device TSK, write a file called OUTFIL.MAC onto the disk.
*EX$$	Exit to monitor after copying the input file to the output file.
^C ^C	Return to monitor.
.CCON	Continue the TECO job running.
.DETACH	Detach from the passive task.

The Active Task

Node name = NODEB Node number = 04

.SET HOST NODEB	The task to send the file is to be run on node NODEB; therefore, attach to the command decoder of node NODEB.
.LOGIN...	Log a job into NODEB.
.R TECO	Run the TECO program.
*EWTSK03:TECO$$	Perform a write to device TSK on node number 03. The job will go to node number 03 and connect to the job under the same PPN.
*ERDSK:INFIL.MAC$$	Read the file INFILE.MAC from device DSK.
*EX$$	Exit to monitor after copying the input file from the disk to the output device TSK. Thus the file transfer is performed. After completion, TECO returns to monitor.

PIP FILE TRANSFER

The passive task

Node name = NODEA Node number = 03

.SET HOST NODEA	The task to receive the file is to be run on node NODEA; therefore, attach to the command decoder of node NODEA.
.LOGIN...	Log a job into NODEA.
.R PIP	Run the PIP utility.
*DSK:OUTFIL.MAC=TSK:PIP	Copy a file fron device TSK and write it onto the disk as a file called OUTFIL.MAC. The job will go into I/O wait state until connected to.
^C ^C	Return to monitor.
.CCON	Continue the PIP job running.
.DETACH	Detach from the passive task.

The Active Task

Node name = NODEB Node number = 04

.SET HOST NODEB	The task to send the file is to be run on node NODEB; therefore, attach to the command decoder of node NODEB.
.LOGIN...	Log a job into NODEB.
.R PIP	Run the PIP utility.
*TSK03:PIP=DSK:INFILE.MAC	Copy the file INFILE.MAC from the disk to device TSK on node number 03. The job will go to node number 03 and connect to the job under the same PPN.
^C	After completion of the transfer, return to the monitor.

The passive task declares that it is willing to accept connection for intertask communication. Below is a description of the steps taken by the passive task in establishing a connection. The active task specifies that it wants to perform intertask communication and also specifies the node, name, and account number of the passive task with which it wants to communicate.

The passive task waits until an active task requests to be connected to it and then a connection is established. The active task then sends identifying information to the passive task. Therefore, the first operation in establishing intertask communication should be the active task sending information and the passive task receiving it.

After the connection process is completed, either task may send or receive information. In addition, either task may break the connection.

Intertask communication is similar to writing to and reading from any network device. For this purpose, a new pseudo-device called TSK has been defined. The pseudo-device TSK maps into an actual task name and may be accessed like any file-structured device.

Intertask Communication

A task can establish a connection with another task only if the other task is willing to communicate. Thus a task running in the network need not concern itself with other tasks performing connects to it. If a task does not declare its willingness to connect, any attempts to connect to it are ignored.

The Passive Task

In establishing a connection, the passive task must declare its willingness to perform intertask communication, its task name and the channel to be used. Below is a description of how connections are established for passive tasks written in MACRO, FORTRAN, and COBOL.

MACRO - A MACRO task can declare its willingness to communicate by performing an ASSIGN system command, the corresponding NODE UUO, or an OPEN/INIT UUO sequence. The passive task's ASSIGN system command is in the form:

```
.ASSIGN TSK log-dev
```

where log-dev is a logical device identifier.

To declare itself as a willing passive task, the MACRO-10 program must associate a channel to be used for the I/O with an OPEN UUO. See the DECsystem-10 Monitor Calls Manual for a description of the UUOs referenced in this module.

The OPEN UUO for a task has the form:

```
OPEN ch,[mode
        SIXBIT /TSK/
        XWD out,in]
   error return
   normal return
```

where ch specifies a channel number.

When the channel is open, the passive task must declare its task name for connection purposes. This declaration is performed using a LOOKUP UUO. For example:

```
LOOKUP   ch,[SIXBIT/tsknam/    ;passive task name
                Z              ;task extension
                Z              ;not used
                Z]             ;PPN
```

where ch is the same channel number as specified in the OPEN UUO and tsknam is the name of the passive task. The extension field may be included but it is ignored. Word 4, the PPN specification, is applicable only for privileged users. This word defaults to the PPN of the user if zero or if issued from a non-privileged task.

After the return of the LOOKUP UUO, an INPUT UUO can be issued. At this point, the passive task enters a wait state until an active task connects to it.

FORTRAN - To declare itself available for intertask communication, a FORTRAN task should use the OPEN statement (see the DECsystem-10 FORTRAN-10 Language Manual). The following arguments to the OPEN statement are applicable for a passive FORTRAN task.

UNIT=u — Defines the FORTRAN I/O unit number to be used. The unit number (u) must be a decimal integer within the range 1-63; however, UNIT may be assigned by an integer variable or constant.

DEVICE='TSK'	Specifies that the logical device is a task.
FILE='filename'	Defines the name by which the task is identified. The name is used by the other task as the destination for messages. The filename can be up to six characters. The name defaults to a form of FORxx where xx is the unit number.
ACCESS='SEQIN'	Declares that the task will be receiving data sent to it.
MODE='ASCII'	Defines the character set of the information that may be received. Only ASCII data transmission is supported.

These arguments define the passive task as a willing receiver for intertask communication. Arguments that can be specified in a standard FORTRAN OPEN statement, but are not applicable to intertask communication, are ignored. Thus the OPEN statement:

```
OPEN(UNIT=20,DEVICE='TSK',ACCESS='SEQIN',MODE='ASCII')
```

causes the task to be considered a passive task named FOR20 (default name) that uses unit 20 for receiving intertask communication.

After the OPEN statement is issued, the first READ or WRITE statement on the specified channel causes the task to enter a wait state until an active task connects to it.

COBOL - A COBOL task can declare itself a passive task by a SELECT statement in the ENVIRONMENT DIVISION FILE-CONTROL section of the source program. (See DECsystem-10 COBOL User's Guide.) The form of the SELECT statement is:

```
SELECT filename ASSIGN TO device-name.
```

where the device-name is either TSK or a logical name defined prior to execution by an ASSIGN command of the form:

```
.ASSIGN TSK device-name
```

The device-name in the ASSIGN system command must be the same as the device-name in the SELECT statement.

To define the external name of the task, the passive task should use the VALUE OF IDENTIFICATION IS clause in the file-description. For example:

```
VALUE OF IDENTIFICATION IS "RECVR"
```

In COBOL, only the first six characters specified are used in intertask communication. This is because the last three characters in the specification are an extension and intertask communication does not use the extensions.

To complete its part in establishing the connection, the passive task must issue an OPEN statement in the PROCEDURE DIVISION. Thus, a statement of the form

```
OPEN INPUT filename
```

causes the filename associated with the TSK in the SELECT statement to be opened for input. Thus, the issuing task can read from the specified file.

The Active Task

The active task initiates the connection to a passive task that is in a wait state. The active task must specify the passive task to which it wants to be connected. The following information is needed:

1. Task name (default is the name of the active task)
2. Node-ID (default is the local node)
3. Account number (default is the PPN of the active task)

If the active task wants to connect to a remote task with the same name and the same account number as the active task, an ASSIGN system command can be entered. This command is of the form:

```
.ASSIGN node-id TSK log-dev
```

where node-id and log-dev are as specified in the ASSIGN command.

MACRO - The active MACRO task must perform all the steps of a passive MACRO task, but it must also perform an ENTER UUO. A complete active task specification must use the NODE. UUQ. The NODE. UUO is of the form:

```
     MOVE  AC,[XWD 1,address]
     NODE. AC,
       error return                    ;AC contains the error
                                         ;number
     normal return

     .

     .

     .

address: XWD 0,4
     SIXBIT /node-id/                  ;node-id of the other
                                         ;task
     SIXBIT /TSK/                      ;pseudo-device TSK
     SIXBIT /log-dev/                  ;the logical name
                                       ;assigned by
                                       ;the user
```

The actual channel for intertask communication is specified by the OPEN UUO. For example:

```
OPEN ch,[mode
                SIXBIT /TSKnn/
                XWD out,in]
```

where ch specifies a channel number.

The task then performs a LOOKUP UUO to associate the active task name with the channel specified in the OPEN UUO. For example:

```
LOOKUP ch,[SIXBIT/tsknam/     ;active task name
                Z
                Z
                Z]
```

where ch is the same channel specified in the OPEN UUO.

To complete the connection, the active task issues an ENTER UUO. The ENTER UUO is of the form:

```
ENTER ch,[SIXBIT /tsknam/
                 Z
                 Z
             XWD ppn]
  error return
normal return
```

where ch is the channel number as specified in the LOOKUP and OPEN UUOs, and tsknam is the name of the passive task.

At this point, the connection is established and user-level identification can occur.

FORTRAN - The active task issues the connect to the passive task via an OPEN statement similar to that of the passive task. The active task can also specify the PPN of the passive task with which it wants to be connected. The PPN is specified using the DIRECTORY = argument of the OPEN statement. For example, to connect to the task running under PPN [350,1234], use an OPEN statement of the form:

```
OPEN(UNIT=20,DEVICE='TSK',ACCESS='SEQOUT',MODE='ASCII',
1DIRECTORY='350,1234')
```

The node of the passive task is defined in an ASSIGN system command of the form:

```
.ASSIGN node-id_TSK log-dev
```

where log-dev is a FORTRAN logical unit number.

For example:

```
.ASSIGN BOSTON_TSK 20
```

In addition, the ACCESS of the active task should be specified as 'SEQOUT' if it is to send information to the passive task. After the OPEN statement has been executed successfully, the two tasks are connected. To perform 2-way intertask communication, two OPEN statements should be issued on separate channels. One OPEN statement should specify 'SEQOUT' on the sending channel; the other should specify 'SEQIN' on the receiving channel.

COBOL - The active COBOL task can specify the node and PPN of the passive task to which it is to be connected. The node specification is performed using the ASSIGN system command. The PPN of the passive task is specified in the file description by the USER-NUMBER IS clause. For example, the task "RECVR" running under PPN [123,456] can be specified by the file-description clauses:

```
VALUE OF IDENTIFICATION IS "RECVR"
USER-NUMBER IS "123,456".
```

In the PROCEDURE DIVISION, the task should perform an OPEN statement on the file specified for intertask communication. For example:

```
OPEN OUTPUT filename.
```

At this point, the communication is established.

Breaking the Connection

The intertask communication connection may be broken by either task. If either task closes the channel (file), the closing of the file appears as an end-of-file to the other task. The end-of-file (EOF) causes the connection to be broken.

In MACRO, the task can use the CLOSE and RELEASe UUOs to break the connection. FORTRAN uses the CLOSE statement with arguments. COBOL tasks use the CLOSE statement, specifying either the filename or all files.

Sending data

Data is transferred from one task to another by issuing I/O statements to the channels associated with the connection. MACRO provides the INPUT UUO and the OUTPUT UUO; FORTRAN and COBOL provide the READ and WRITE statements.

The following FORTRAN task accepts characters from the terminal and sends them to a remote task called BUCKET on node 45.

```
     DIMENSION IST(72)
     OPEN(UNIT=20,DEVICE='TSK45',FILE='BUCKET',ACCESS=
     1'SEQOUT',MODE='ASCII')
     OPEN(UNIT=5,DEVICE='TTY',MODE='ASCII',ACCESS=
     1'SEQIN')
10        READ(5,99,END=20) IST
     WRITE(20,99) IST
     GO TO 10
99        FORMAT(72A1)
20        CALL EXIT
     END
```

This page intentionally left blank

Module Test

1. With the configuration specified in class for this problem, execute either a PIP or TECO file transfer.

2. If you are familiar with FORTRAN or COBOL, describe, from memory, the parameters necessary for intertask communication in that language. Otherwise, do the same for MACRO.

This page intentionally left blank

Evaluation Sheet

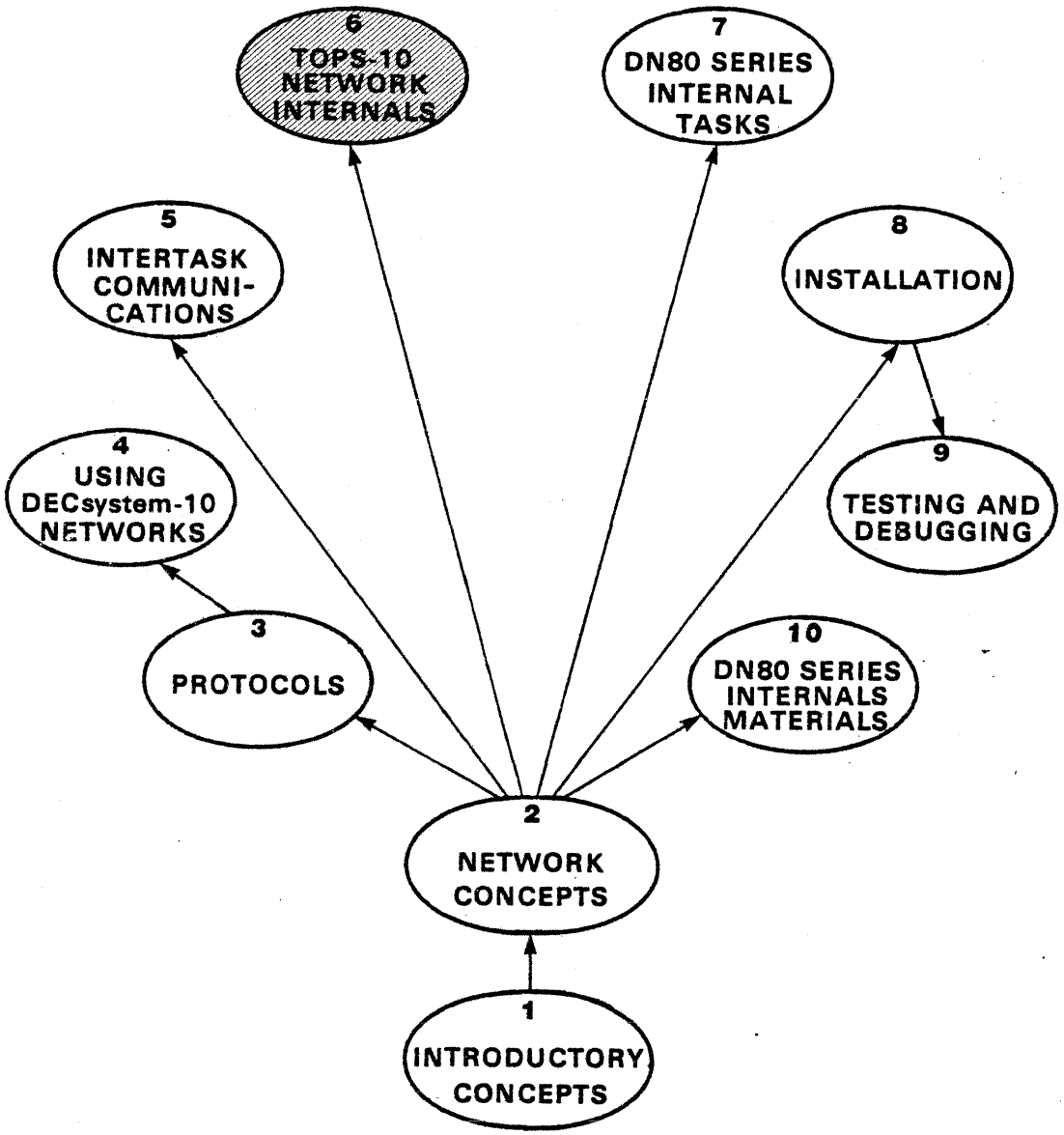
1. The correct action will be related to the specific configuration available to the class. Competence will be determined by successful transfer of a file as per configuration data supplied by the instructor.

2. See the text of this module covering intertask communication for the language which you selected.

This page intentionally left blank

TOPS-10 Network Internals
Module 6

Course Map



M8 0277

Introduction

This module describes most of the code in TOPS-1Ø related to networking as of the 6.Ø3 release and notes some of the changes to be made for announce 6.Ø4 functionality.

This page intentionally left blank

Monitor Modules That Comprise Tops-10 Networks

COMNET - Configuration Database

This module defines the format of all control blocks used by the network and has code to find TTYs that need servicing.

NETSER - Primitives and NCL Protocol

This module is where the NCL implementation lives which takes up the bulk of the module. Also included is all of the terminal interface to SCNSER and code that manipulates the network data base (building/freeing control blocks, linking/delinking them, etc.)

NETCDR - Remote Card Reader Interface

Actually part of NETSER, this is the device dependent interface to remote card readers.

NETLPT - Remote Card Reader Interface

Like NETCDR, except for LPTs.

TSKSER - Intertask Communication

This has all the interface for intertask communication, for both inter-system and intra-system tasks. The latter interface doesn't really use the network, it shortcuts around it.

RDXSER - Remote Data Entry

This handles all interfacing to Remote Data Entry devices.

D85INT - DL10 Interface

This handles the transmittal of messages between front end and NETSER via the DL10.

D8SINT - Queued Protocol (DTE-20) Interface

This is an analogue to D85INT that communicates with its front end via DTESER and its DTE-20 queued protocol.

NETWORK CONTROL BLOCKS

After the name of each control block is the AC that is usually used to reference the control block.

FEK(J) - Front End Kontroller

This has the common information needed to let NETSER and D85INT or D8SINT communicate with each other. A word in it is reserved for use by the specific interrupt service module which is used for state recording by D85INT and points to a private little control block by D8SINT. (Also used (ING.Ø4) for certain non-network functions.)

DTK(W) - DTE2Ø Kontrol Block

The FEKUNI word in the FEK only allocates a single word for front end dependent functions which is insufficient for the data D8SINT has to keep track of. Therefore, DN87S's use this word to point to a DTK which contains all the DTE specific information. DTKs are three words long and contain a semaphore to prevent a race condition in D8SINT's usage of the queued protocol, a location which is the DTE number associated with the FEK, and a byte pointer used when the 87S has to send a message in several fragments. The queued protocol is the mechanism used to transfer data across DTE2Øs and is implemented in DTESER.

NDB(W) - Node Data Block

There is one of these for each node presently online. They contain data necessary to run NCL to that node, i. e. message counters, state information, message lists, and the number and types of devices on the other end.

NDP - Network Dispatch Table

There is one of these in each network device driver. Similar to device dispatch tables, they consist of instructions executed to process network events. Presently they are three words long and handle these events:

1. NDPIDT - called when a data message for the device arrives. The action here is to copy the data into monitor buffers, stripping DAP information and expanding the data.

2. NDPIST - called when a status message for the device arrives.

3. NDPODN - called when a message has been sent to the front end. The action is to advance output buffers and try to send the next message.

NDT - Network Device Table

This provides information that defines what sort of remote devices this system supports, initial data that has to go into DDBs, and points to the NDP (network dispatch table) for that device.

NETLAT - Link Address Table

This is a vector that maps logical links (there is one for each connection) to the DDB or LDB connected. DDBs are entered for network devices like TSK, LPT, CDR, etc. LDBs are entered for remote terminals and have the sign bit set to signify this entry points to a LDB. Data messages and several control message types contain a device link address field which is used to index this table to determine the device the message refers to.

PCB(U) - Protocol Control Block

PCBs contain control information about each message processed by NETSER, be it generated locally, sent to this -10 from elsewhere, or routed through the -10. This information includes things like byte pointers for the data and link words for the various message queues.

NCL PROTOCOL

NCL Functions

NCL is a protocol layer that sits between the physical link protocols (DDCMP, DTE20 queued protocol, etc.) and the utility protocols (DAP is the only one the TOPS-10 network uses). NCL can be broken into several layers itself:

Internode Startup - When NETSER learns of the existance of a new network node by either being told a front end has just come up or by receiving a NEIGHBORS message from a node referencing another node we don't know about, it has to initiate communication with that node to reset message counters. This is accomplished via the unnumbered messages below. See a later section (6.8.1) for a description of the interactions.

NODEID - This is sent only to neighbors. (In NETSER's case, that means front ends.) The main thing this message does is exchange node numbers with the front end to let other startup messages work.

START - Analogous to DDCMP's START message. (DDCMP is documented in #130-959-007-05.)

STACK - Analogous to DDCMP's STACK message except that the receiver doesn't have to send an ACK for message #0. (Nothing bad happens if he does.)

Message acknowledgement - The other use for unnumbered messages is to ACK/NAK messages between nodes. Incidentally, unnumbered messages are those that may be lost in the network with no harm to internode communication except for possibly a short pause in message flow (since the messages get implicitly acked on succeeding messages if they were received). No harm occurs because they will be regenerated if necessary.

ACK - Analogous to DDCMP's ACK message. The receiver may release any messages this message acknowledges.

NAK - Analogous to DDCMP's NAK message except it is only sent in response to a REP message and contains no reason field.

REP - Analogous to DDCMP's REP message and is sent whenever a node hasn't received an ACK for some time after sending one or more data messages.

ROUTING

NEIGHBORS - This is the sole means of determining the topology of the network beyond a node's neighbors. Each node sends a NEIGHBORS message to all other nodes in the network whenever NCL is started with a node or whenever the node's neighbors change. As a node receives these from all others in the net it builds up its topology tables. The contents of the NEIGHBORS message is a list of pairs of node number and link cost, one for each neighbor. The link cost is used in evaluating the cost of sending messages down each path in the network. For details on the process, see section 10.

Logical link control - Like many other packet switching systems, NCL communicates over things called logical links. A good analogy is the time domain multiplexing used by the telephone people to carry several conversations over a single microwave link. Unlike many other packet switching systems, NCL communicates with devices, not just between processes. In some respects, the difference is moot since device drivers receiving NCL messages could just as well be a process scheduled like any other.

REQ CONFIG - This is sent by NETSER soon after NCL communication starts. The node that receives it replies by sending a CONFIG message, below.

CONFIG - This message contains a list of device types (TTY, LPT, TSK, etc.) and the number of each present on that node. NETSER uses this information to do a little preprocessing of user generated connection requests and if the target node doesn't have that device in its configuration, then NETSER refuses to try to connect it.

CONNECT - This initiates the establishment of a logical link. The CONNECT message references a generic or physical device and the receiver is expected to return a CONNECT to complete the connection or a DISCONNECT to reject it if the device is restricted or in use.

DISCONNECT - This is sent due to three possible events:

1. In response to a CONNECT from a node trying to access a device or task it may not (e. g. the device is unavailable or the system is stand alone).

2. The user no longer needs the connection (RELEASE UUO, DEASSIGN command, etc.) This initiates the breaking of an existing logical link.

3. In response to a DISCONNECT INITIATE.

Flow Control and Data

DATA REQ - To prevent flooding destination nodes with data, this message is used to implement flow control on all logical links. This message notifies the receiver that it can send "X" more data messages, where "X" is the parameter of the DATA REQ. Currently, the typical value of "X" is 1.

DATA - These messages may only be sent while the data request count for the link is positive. Each transmission decrements the count. As the receiver processes them it replys with more DATA REQs to let the sender send more DATA messages.

Utility

STATION CTRL - This message provides support for various utility functions in the network that need not go through the expense of creating a connection. This message has a subtype field to specify the particular function of the message. Examples include examine, deposit, start, and load request.

NCL Message Formats

Below is the description of all the NCL message formats, taken from the working document used during development of DN80-series code. Each message is a concatenation of the specified fields.

```
unnumbered control -- NCT  DNA  SNA  NCA  NCN  OPD
numbered control --   NCT  DNA  SNA  NCA  NCN  0    CM
DATA --               NCT  DNA  SNA  NCA  NCN  DLA  DEVCTL
```

DEVCTL=device control.
DLA=destination message link address, i.e. the index into the node's connection database. Extensible binary field, maximum of 12 bits, zero is illegal.
DNA=destination NNM
NCA=Network Control Ack; last network message received ok.
NCN=Network Control message Number. One byte binary field.
NCT=network control message type and flags, extensible field.
bits 0-2=type field
0=data message
1=ack.
2=nak.
3=rep.
4=start. OPD is NNM SNM SID.
5=stack. OPD is NNM SNM SID.
6=node id. OPD is NNM SNM SID.
bit 3=SNA and DNA present.
bit 4=trace
bit 5=interrupt msg(i.e. don't adjust data request count)
bit 6=Non-sequential node.
bit 7=extensible bit
NNM=node name, a binary extensible field, maximum of 12 bits, identifying node.
OPD=optional data.
SID=software identification, extensible ASCII with two subfields:
1) name and version of operating system and DEMOS software,
2) creation date.
SNA=source NNM.
SNM=station name is an extensible ASCII field.

CM = one of the following:

```
CONNECT --          CNT  <001>  DLA   SLA   DPN   SPN   MML  FEA
DISCONNECT --       CNT  <002>  DLA   SLA   RSN
NEIGHBOURS --       CNT  <003>  (NNM  LVL)
REQ CONFIG --       CNT  <004>
CONFIGURATION --    CNT  <005>  (OBJ  NDV   PID)
DATA REQUEST --     CNT  <006>  DLA   DRQ
STATION CONTROL --  CNT  <007>  STC
     CNT=count of remaining bytes in message.
     DCM=data code and mode:
          b0=ASCII
          b1=EBCDIC
          b2=Image
          b3=Hollerith(CDR only)
          b4=DEC image (CDR only)
          b5=reserved
          b6=compressed format
     DLA= (defined above).
     DPN=destination PN.
     DVT=device specific attributes:
        =attributes for card reader:
          c1,b0+b1=speed
                  0=don't care
                  1=<300
                  2=between 300 and 600
                  3=>600)
            ,b2=mark sense
            ,b3=hdw EOF required
            ,b4=suppress EOF card detection
        =attributes for line printer:
          c1,b0+b1=speed(see DCD)
            ,b2=lower case req
            ,b3=remov.  char set req
            ,b4=multi-part paper req
            ,b5=12 chan skipping req
            ,b7=1
          c2,b0+b1=skip requirements
                  0=don't care
                  1=changeable from handler
                  2=changeable at site
                  3=changeable but don't care how
            ,b2=req overprint
            ,b3+b4=6/8 lines/inch
            ,b5=changeable form width
        =attributes for teletypes:
          c1,b0=modem control
            ,b1=auto-baud
            ,b2=handler can set baud rates
            ,b3=2741
            ,b4=baudot
```

,b5=auto dial line.
FEA=features: DCM+RLN+DVT (RLN is remote line number)
LVL=link value is a one-byte binary value used to determine the preferred path. (Preferred path is that whose sum of link values is lowest.)
MML=maximum NCL message length for connection, including NCL header, but not including DDCMP.
OBJ=object type for process:
0=tty handler (MCR)
1=tty
2=card reader
3=line printer
4=paper tape reader
5=paper tape punch
6=plotter
7=magnetic tape
10=dectape
11=task (job)
12=RDX, data entry terminal.
200-377=reserved
OPD=optional data (extensible characters).
PID=process identification. For devices this is an extensible binary field, 177 means default choice, 0 - n means unit #. For tasks this is a single extensible ASCII string usually name and qualifier (e.g. UIC or PPN).
PN =process name, having 2 parts: 1) OBJ, 2) PID.
RSN=reason
0=normal disconnection
1=object type not available
2=too many connects to node
3=too many connects to process
4=process does not exist at this node
10=reassign, next ext field is dest node number
SLA=source message link address. Extensible binary field, maximum of 12 bits, zero is illegal.
SPN=source PN.
STC=station control.

MONITOR INTERFACE TO TOPS-10 NETWORKS

In general, the monitor uses the network instead of the network using the monitor, so the bulk of the text below will be for that direction.

Throughout the rest of this file, the number after the subroutine name is the approximate page number in the listing.

Command Level Routines

NODE.C (9) - NODE Command - The data printed comes from the NDBs. If no argument was passed, all the nodes are output. If a single node name or number was typed then only the info for that node or an error message (if that node doesn't exist) is printed.

HOST.C (114) - SET HOST Command (actually SET HOSTESS) - After verifying that this was typed on a virtual terminal and that the node named exists and has command terminal support, the job (if any) is detached from the TTY and a DISCONNECT message is created and sent to the node the terminal is physically attached to. A field in the message contains the address of the node to reconnect to and when the -11 receives the message it sends a disconnect confirm to the old node and a connect initiate to the new node.

NETASG (5) - ASSIGN Command - This is called whenever COMCON sees an ASSIGN command with an underscore in the device name. All the code does is convert the device name from the NODE_DEVu form to the GGGnnu form and pass it back to COMCON.

Clock level Routines

NET2ND (46) - NCL Once a Second Routine - This is called on a once per second per FEK basis. The main job it does is to send the messages to nodes other routines didn't have time or desire to. These include the neighbors message and all the startup messages except STACK. Additionally, boot requests from remote stations are timed out in case NETLDR can't get to them. (If they weren't timed out, then no new boot requests would be processed from that node and NETLDR would never be restarted for that node again.) The multi-pathing project will add code to this routine that should already exist. First, the -10 does not usually acknowledge the receipt of NCL messages without some prodding. TTY messages are acknowledged right away because of some special handling of TTY messages containing keyboard data. If we receive a REP message, we return an ACK if we can (protocol says we should) and messages are often acked indirectly via the ACK field of all NCL messages. This still leaves cases (like TSK and CDR input) where messages are not acked and results in the sender buffering the message so long it sends a REP to see what happened which will win an ACK and finally it can return those messages to

the buffer pool. Presently, REPs only come from -11s as the -10 doesn't bother to timeout messages itself so this code also will have to be added to NET2ND.

DL10 and DTE20 clock routines - These verify that the keep-alive timers are valid. If they timeout, device dependent cleanup is done and FEKINT is called to tell NETSER that one of its front ends just died.

UUO level routines

The UUO level interface to NETSER is extensive.

NODE. UUO - This UUO allows programs that know they are using a network to handle device dependent features they have to be concerned about. The functions of the 5 subUUOs are as follows:

1. Assign a remote device. This accepts both a node identification and a device name relative to that node and then goes out and assigns it to the program.

2. This converts between SIXBIT or node number and SIXBIT node name or binary node number.

3. This transmits and waits for reception of a station control message. It is a privileged function and exists for NETLDR (the downline loader) and DDT11 (a debugging program).

4. When the bootstrap ROM in a remote station is started, it sends a load request to the -10 which stores information about the request while waiting for the NETLDR job to start. The first thing NETLDR does is execute this subUUO which passes the bootstrap parameters to NETLDR whereupon dumping and reloading the node takes place.

5. This returns information about which devices are directly connected to any node in the network.

Terminal Routines

As with any terminal hardware interface (DLSINT, D76INT, etc.), communication with SCNSER is done via calls to a vector of interrupt service routines (ISRs). The useful ones (those that do more than a POPJ) are listed below, how they work is documented in "NETSER FUNCTIONS".

1. D85TYP - "Types" a character on a virtual terminal.

2. D85CHP - Notifies node controlling the terminal that "hardware" parameters for that line may need to be changed.

3. D85LPC - Notifies node that a "line" parameter has changed. Currently the only thing checked is the unlock keyboard request.

4. D85ELE - Notifies node that the interpretation of the typeball element is to be changed.

5. D85REM - "Special calls from SCNSER". The only special condition NETSER is interested in is the CTRL-O function.

6. D85OFL - Offline check. The virtual terminal is considered offline if it is not connected to a physical TTY.

COMMUNICATIONS BETWEEN NETWORK MODULES

Front-End Interface Drivers and NETSER Most of the data passed between the front-end controller and NETSER is kept within the FEK control blocks. The three routines described below are NETSER routines that interface to the device drivers. Two device drivers are presently supported and exist in separate modules. D85INT interfaces to DL10s and D8SINT interfaces to DTE20s via DTESER.

NETWRT(51) - Whenever a message must be sent to a front end, the PCB pointing to it is passed to this routine. From data within the PCB that points to the destination and the best front end to use for messages that are destined for that node, NETWRT picks up the right FEK. The NCL fields in the message for numbering and acknowledgement are filled and, if the front end is not already receiving a message [FEKBSO(J)=-1], the message is sent right away [XCT FEKWRT(J)]. If the output side is already busy, the message is put on the end of a list whose header is FEKOAD. When the current message is shipped out, the service routine will delink the first message in the list and send it.

NETRDD - The service routines are not allowed to accept messages from a front end until NETSER explicitly asks for it. NETSER does this by calling NETRDD with the address of a FEK to get a message from. If there is not already a PCB associated with that FEK [FEKIAD(J)=0], MAKPCB is called to get one from monitor free core then NETRDD gets an additional 128 words of message buffer to store the message. This PCB is passed to the FEK [XCT FEKRDD(J)] and for some perverse reason NETRDD goes to SCNTTO to process any TTY activity.

When saying that a PCB may be associated with a FEK. This means that during normal operation, NETSER receives a message (see FEKINT below), processes it, and then goes off to NETRDD to read in the next one. To avoid all of the overhead of calling GIV4WD to return a PCB and then GET4WD to get another, PCBs for input are recycled. After a message is processed, another message is read into the same PCB.

FEKINT - Whenever a service routine has detected an event that deserves NETSER's attention, FEKINT is called with several status bits in the LH of J. These bits note input done, output done, front end down, and error log request. Details on the first three are as follows:

Input Done (INPINT(48)) - This is the general NCL message decoding routine. Details on processing each message type are presented later, but for now let's look at the code up to the message dispatch. First, the destination of the message is determined. If it is not for us [NEQ OURNNM], then the message is passed on its way by going off to NETHRU which sends it to another front end. Otherwise (the message is for us) the acknowledgement field is picked up from the message and passed to CHKNCA who frees any messages that the message received just acknowledged. A dispatch to INCTUN (unnumbered) or INCTNM (numbered) is done to process the data portion of the message. Unfortunately, NETSER does not yet order incoming messages, one of many things that have to be changed for the complex topologies project of 6.04.

Output Done (OUTINT(50)) - Once the message has been transmitted to the front end, NETSER still has a fair amount of processing to do. If the message is unnumbered, it will never need retransmission and is returned to free storage immediately. Otherwise, it is placed on a list of messages sent to that destination that have been transmitted but not yet acknowledged (list header is NDBQUE(W), RH). If there is another message to send out to that front end, it is delinked and passed to the service routine. If some data was in user address space (true for devices like LPT and TSK) then the device service routine is called to advance buffers.

Front End Down (KONDWN(49)) - When a front end goes down, much of NETSER's data base has to be diddled to remove the control blocks and data allocated to nodes accessible via that front end. First, if there is an outstanding input request for that node (which is usually true) it is freed by calling RMVPCB. Next, if there are messages waiting to be sent via that FEK, they are delinked from FEKOAD and freed. This is another problem with complex topologies. There will often be cases where a front end will go down, but messages destined to be sent by it point to destinations that are still reachable by other front ends. The messages must not be freed at this point, but should be routed via a different route if possible. Finally, all the NDBs are checked to see if messages for them are routed by that front end [J=NDBFEK(W)]. If they are, RMVNDB is called to remove knowledge of them from the data base. Again, this doesn't work with complex topologies. The present code makes no attempt to determine if the node may still be accessible by a different route.

NDB management

MAKNDB(20) - Whenever NETSER learns about new nodes in the network by receiving a NODEID message from a front end or a NEIGHBORS message from a node that references a node not in the data base, MAKNDB is called to add it to the data base. Space is allocated for the NDB, the node number of the new node is stored in NDBNNM, the FEK where the NODEID or NEIGHBORS message came from is put in NDBFEK, and the new NDB is added to the list of NDBs. Note that the FEK that told NETSER about the new node will be the one that NETSER will use to send all messages through. This is the entire routing algorithm, which while low in overhead will not be adequate for complex topologies.

RMVNDB(22) - Whenever a node becomes unreachable due to a front end dying or links elsewhere in the net breaking (as reported via NEIGHBORS messages) RMVNDB is called to remove that node from the data base. First, the offline message is typed on OPR, PSISER is called to note the change in topology, resources tied up waiting on the node by a station control message (generated by a NODE. UUO) are freed, connections to that node are broken and devices freed (TTYs are freed by calling CLRTTY and RMVTTY), space used to store the date and name of the node is freed, PCBs hung on NETQUE are freed, the neighbors list is updated if the node going down is a neighbor, and finally the NDB itself is freed.

SRCNDB(26) - It is often useful to map a node number into its NDB address, and that's what SRCNDB does. Passed either the binary node number or the sixbit node name, all the NDBs are searched and the success return is taken if there is a match with the NDB address put in W.

Device Interface (CDR, LPT, RDX, TSK)

The device interface between NETSER and the device drivers is fairly straightforward. Most important is the "NDP", described in section 2.4. In somewhat more detail, the service routines do the following:

NDPIDT - When a data message arrives for a network device (CDR or RDX), this dispatch is taken. The service routine copies the data from the PCB into one of the monitor buffers assigned to that connection. It then calls AVIMBF to point to the next monitor buffer for when the next message arrives and decrements the data request count. NETSER will wake up the job if it is waiting for some data to arrive. Eventually the input UUO routine will be called which will notice there is data, copy it to user memory, and send a data request to the sender of the data to let it send another message.

NDPIST - This routine handles the DAP status message. The service routine reads the status code and may act upon it as it desires. This information is highly device dependent, but usually has an on/offline bit that may be the only thing checked. In the case of network card readers, a card reader going offline results in no action in the service routine except to save the status bits in the DDB. Only when CDRINP (the input UUO) is called when there is no data in the monitor buffers will the service routine get serious about the offline CDR. At some point CDRSRQ, the routine that sends data requests to the CDR, will be called when there is no data in the monitor buffers. It will notice that the CDR is offline and call DEVERR (at UUO level). Sometime after that, CDRINP will discover that the CDR is offline and call HNGSTP to wait for it to come back online.

When the CDR comes back online, the next read attempt will cause NETSER to try to start the remote CDR which will result in receiving a STATUS message saying the CDR is online again which will clear some internal flags. New data will arrive and IO continues.

NDPODN - When a message from the -10 gets out to the front end, the -10 gets an interrupt to that effect. The device service module is called to try to advance buffers and send the next message.

NETSER FUNCTIONS

DDB Construction/Destruction

Many of the following routines are called from UUOCON by the DDB search routines when they think they might be looking at a network device.

TSTNET(6) - The only caller of this routine is UUOCON's DDBSRC routine. When DDBSRC has run through the whole DDB chain without finding a DDB to match on, it then calls TSTRDX, TSTNET, and TSTMPX in hopes of one of them doing a skip return meaning that they created a DDB to use. If all of those fail, a check for "TTY" or "TTYnnn" is done.

TSTNET only works when called by the OPEN UUO (M&777B6=OPEN) where it makes sure the fifth character of the device name is non-zero (LPT0 is a local device, LPT66 and LPT660 are network devices), converts the device name into node number and three or four character device name and calls GENNET.

GENNET(7) - Besides the call from TSTNET, GENNET is called from UUOCON's DVSTAS routine which searches for a generic device at a specific station. The call is made only if the device is not found on the DDB chain.

GENNET verifies that the node exists (SRCNDB), verifies that the device is on that node (SRCNDT), makes a DDB (MAKDDB), and connects the device (NCSCNT).

MAKDDB(17) - MAKDDB's main call is from GENNET, but is also used by the NODE. UUO function 1 and TSTRDX (and maybe others). It allocates space for a DDB, calls GETSLA to get a source link addr. to be stored in NETLAT, and calls LNKDDB to add it to the list of network DDBs.

UNLDDB(21) - This is called when NETSER has a DDB that is now idle and deserves to be thrown back to free core.

LNKDDB(25) - This is called after MAKDDB has allocated the space for a new DDB. This routine links the new DDB onto the DDB list right after NETDDB.

ZAPNET(25) - ZAPNET is called from UUOCON after every UUO to a network device (sigh) and returns the DDB to free core if it appears to be idle. If the device is still connected but no longer has ASSPRG or ASSCON set, then the device is disconnected before freeing.

SRCNDT(26) - This takes a generic device name and checks to see if it is a device this monitor knows about. The information for this comes from the block of NDTs in COMNET. If the other system has not reported that it supports that device the error return is taken.

GETSLA(2Ø) - This enters a DDB or LDB in the NETLAT vector of addresses of devices that will be connected to the net. It scans from the top looking for a zero entry and if it finds one then stores the address in that slot and returns the index in T1 which will be used as the source link address for future operations with that device.

Monitor Buffer Usage

Input devices have data storage holders that temporarily hold incoming network data until it can be sent to a user's buffers. NETSER calls these monitor buffers (not to be confused with the file system's monitor buffers).

BLDMBF(12) - This is called from the service routine after the CONNECT during the first input request of a device to build some monitor buffers. Typically two buffers are made to allow higher throughput. The number of buffers allocated determine the maximum number of data requests that will be outstanding.

AVIMBF(19) - As data messages arrive for network devices, the device service routines copy them into monitor buffers and call this after each message is copied to point to the next buffer.

AVOMBF(19) - When UUO level gets around to taking data out of the monitor buffers, it calls this to step to the next buffer. If no more buffers containing data are available, the error return is taken to tell the UUO level code there is nothing to copy.

DRQMBF(19) - This routine, called by the service routine whenever it wishes, tries to send some data requests to the other end to keep the data flowing. The number of messages requested equals the number of monitor buffers allocated for this device [MBFMXC] minus the number full [MBFBFC] minus the outstanding data requests [NETDRQ(F) LH]. What this means is that NETSER tries to allow the remote system to keep monitor buffers full at all times.

Message Generation

MAKPCB(20) - This is called whenever NETSER wants to send a message or ask that a message be read in. This allocates the core required and fills in the standard information. It is up to the caller to allocate space for the actual data and put a pointer to it in PCBOAD or PCBIAD (which turn out to be the same location). (Also, output data may be in two fragments, the second fragment pointed at by PCBOA2.)

RMVPCB(21) - This returns a PCB to monitor free memory. Space used by any data pointers will also be returned.

NCSHDR(27) - When NETSER has to send an NCL message, NCSHDR is called and passed a PCB to use. Space is allocated for the message, P2 gets a byte pointer that will be used for storing bytes in the message and P3 is cleared and will be used to count the number of bytes stored in the message. The six fields of the header are filled as follows:

1. NCT (type) - An argument to the routine is what type message this will be (START, NODEID, etc.) that plus the routing header and sequential node bits are stored in the first field. I'm not sure why the sequential node bit is turned on, it may get turned off later on?

2. DNA (destination node address) - the next field is the node number of the node to receive the message. It is picked up from the NDB passed to the routine.

3. SNA (source node address) - That's us, and this value is picked up from our own NDB.

4. NCA (NCL ack) - This is filled with a 0 for now. When we actually send the message this field will be filled in with the number of the last message we have processed from the destination of this message. (That way we stand a better chance of acknowledging as many messages as possible.)

5. NCN (NCL number) - Also a 0, this will be filled in along with the NCA field. This field is the number of this message.

NCS??? - These are the routines that make the specific types of NCL messages. Nearly all of these are called with W pointing to the destination NDB and F pointing to the DDB involved or set to Ø if none is involved. All of these return CPOPJ if memory for PCBs and data is not involved, and CPOPJ1 if the message was sent (well, queued) okay. The following are the unnumbered message types:

1. NCSSTR (START) - When NETSER hears about the existance of a new node in the net (by receiving a NODEID message from a neighbor or a NEIGHBORS message from anyone) it sends a START to initiate communication.

2. NCSSTK (STACK) - When NETSER receives a START from someone, it acknowledges it with a STACK.

3. NCSNID (NODEID) - This routine is not called with W pointing to anything because the NODEID message is sent to a neighbor to provoke it into returning a START. We don't know who the neighbor is yet so we can't have an NDB for it. The NODEID message is the only message ever sent without a routing header. (Perhaps a routing header with DNA=Ø could be used.)

4. NCSACK (ACK) - If the last ACK sent [NDBLAS] is not the same as the last message processed [NDBNCA], then an ACK will be sent. There is a problem here regarding complex topologies. NETSER does not timeout messages so it is quite common to see several messages sent from one -1Ø to another that are waiting for acknowledgement. The sender never sends a REP to force acking and the receiver never gets around to calling NCSACK. Fortunately the lists generally don't get very long since acknowledgement does take place via the NCA field of any NCL message. However, in a multi-pathing environment, the present operation will never detect, let alone recover from, a lost NCL message. In our present environment this is not a problem since NCL messages are supposedly only lost when an intervening node goes down which means that all communication between the two systems will cease.

5. NCSNAK (NAK) - Very similar to NCSACK, only the NCT field is different and the message is always sent.

NCMHDR(31) - This writes NCL headers for numbered NCL messages, messages that cannot be lost without impairing protocol operation. This writes the last byte of the header [DLA] with a 0 to denote that this is a control message. It also leaves a hole for the next byte which will become the length of the control message.

NCS??? revisited - The routines that handle numbered control messages are:

1. NCSCNT(32) CONNECT - This routine handles both connect initiates and connect confirms. For a connect confirm this is usually called at interrupt level from the routine that handles incoming connect inits. For a connect initiate (caused by something like accessing a remote device), this must be called at UUO level since NETHIB is called to wait for a rejection or connect confirm from the other side.

2. NCSDSC(34) DISCONNECT - This handles all of disconnect initiates and confirms and connect rejects. For the confirm and reject messages, which do not have replies, accumulator F must be 0. When those are sent, the routine returns immediately and the caller must release the DDB if there is still one in memory. When the disconnect init is sent, NETHIB is called until the confirm returns and lights NT.DSC.

3. NCSNBN(35) NEIGHBORS - Routing in NCL is keyed off of NEIGHBORS messages which are sent from each node to all others in the net. Whenever a neighbor comes up or goes down, NEIGHBORS messages will be sent to everyone in the net. The data in the message consists of pairs of node number and link value (a measurement of the cost to get to that node). There appears to be a problem here with networks bigger than 63 nodes (something we presently don't support). The code masks the node number to only 6 bits, only enough to handle the present maximum.

4. NCSRCF(36) REQUEST CONFIGURATION - Knowing what sort of devices exist on other nodes is interesting in its own right, but is also used to do a little preprocessing on access requests for devices - if they don't exist on that node, a connect won't be sent. This message requests that the receiver return a CONFIG message that will contain that system's configuration.

5. NCSCNF(37) CONFIG - This sends a configuration message to a node that requested one with a REQ CONFIG message. The data consists of pairs of device type and number, this routine gets that data from the NETCNF vector in COMNET.

6. NCSDRQ(38) DATA REQ - This requests that the other node send more data on the addressed logical link. It is the caller's responsibility to keep track of the number of requests made. A single DATA REQ message may request several messages to be sent. There is no way to retract requests.

7. NCSCTL(39) STATION CTRL - This routine is only called from one of the NODE. UUO's subUUOs. A parameter to the routine is a message block in user's memory that forms the body of the message.

NETHRU(53) - When INPINT receives a message that is not for this node, it calls NETHRU to route it out another front end. Routing isn't really done in this code, it just sends it to whoever is in NDBFEK.

Message Decoding

CHKNCA(57) - Once communication between two nodes is established, every message received acknowledges messages previously transmitted. CHKNCA deallocates messages that have been waiting on NDBQUE since the front end told us they had been successfully transmitted. It works by scanning the list pointed to be NDBQUE looking for the first message ACKed, freeing it if it is there, and looping if there are more to free. In a multi-pathing environment, it is possible to retransmit several messages and have a message ACKed before its retransmission is complete. I think the code will handle this, though not optimally. There is a check to make sure the ACK number is less than the last message sent. If it isn't (this special case), the ACK is simply ignored. A later ACK will get everything straightened out.

INC??? - These are the unnumbered message processors. Numbered messages flow through here to their dispatch routine. All of these are called with W pointing the NDB of the source node or 0 if there is none (I think NODEID is the only instance of this).

1. INCTNM(58) - This is called to process the next sequential message in order for the sending node. (Ordering should have been done by INPINT.) All this does is twiddle some bytes and jump to INCTDM.

2. INCTUN(58) - This is called on the receipt of an unnumbered message. All this does is dispatch on the message type to one of the routines below.

3. INCTID(59) NODEID - A neighbor node sends this message to initiate communication. If we already know of the existance of this node, NETSER tries to restart communication with it. For complex topologies, this is probably a bug since communication could have been started as a non-neighbor node. If we don't know about the node, an NDB is created for it and we will initiate communication with it at clock level (NET2ND). The node is added to our list of neighbors and SETNBN is called to notify the rest of the net that we have a new neighbor.

4. INCTST(59) START - This is part of the startup sequence for NCL. NETSER will ignore start messages from nodes it hasn't heard about by a NODEID message from a neighbor or a NEIGHBORS message from the sender's neighbor because it doesn't know how to route messages back to it. When we receive this, all of the message counters are reset and a STACK is returned to complete startup.

5. INCTSK(59) STACK - This completes startup. After this is received, NETSER may send messages to that node.

6. INCTAK(60) ACK - This processes the ACK message, but since the acknowledgement field was already processed earlier, this is simply a NOP.

7. INCTNK(60) NAK - This handles NAK messages (which have the capability to ACK messages too!). All transmitted but not acked messages (the list headed by NDBQUE RH) are retransmitted.

8. INCTRP(61) REP - What is supposed to happen is that if the last message sent by the sender of the REP has not yet been processed, we must send a NAK specifying the last message we have processed and the other system will retransmit everything it thinks we should have processed. Presently, the only path the code takes is to discover that we don't have anything on the input queue (NDBQUE LH, used to hold out of order messages) and send an ACK. If we did order messages, INCTRP would discard all unprocessed messages and send a NAK requesting all those and the lost messages be retransmitted.

RDOPDD(62) - This is used by INCTID, INCTST, and INCTSK to read the extra data that comes with their messages (node name, system name, creation date). It also resets the message counters used between the systems but doesn't clear out NDBQUE which is part of the problem discussed in INCTRP.

ICM??? - These are the processing routines for numbered control messages. These messages have to be numbered or else improper operation will occur if they are lost. Several of these can be concatenated in a single numbered control message but I don't believe any of our network systems do that.

1. ICMCNT(64) CONNECT INITIATE and CONNECT CONFIRM - If this is a connect initiate message (a connect with a DLA [destination logical address] field of 0), then we reach ICMCNN(65) where we dispatch to TTYCNZ if for a TTY and to TSKCNT if to a TSK. Otherwise, we return a disconnect reject message.

   For the connect confirm message, we ensure we sent a connect (non-zero NETLAT entry for that DLA), store the unit number in the DDB if we asked for a generic device, light the IOSCON bit and call NETWAK to wake the job that sent the connect initiate.

2. ICMDSC(66) DISCONNECT INITIATE, DISCONNECT CONFIRM, CONNECT REJECT - If this disconnect message is for a TTY, we dispatch down to that code to handle it. Otherwise we make sure we know about the device, light the NT.DSC bit (a magic bit TSKSER needs), clear IOSCON and call TSKSER (TSKDSC) if this is a disconnect initiate for a TSK. The reason for the disconnect is saved in NETRSN in case this is a CONN REJECT caused by calling NCSCNT earlier.

3. ICMNBN(67) NEIGHBORS - This rebuilds the topology table for that node then scans all the NDBs looking for unreachable nodes. Any that are found are declared offline. The scan is done by SRCNBN(67), a recursive routine that starts the scan with the -10's NDB. Each NDB scanned has a found bit (NDB.TP) set, then all its neighbors without the bit already on are scanned. When the first call to SRCNBN finally returns, all accessible nodes will have NDB.TP set and ICMNBN traces the NDB list clearing the bit if it is on and passing the NDB to RMVNDB if it is off. As a side effect of SRCNBN, new nodes in the network are identified and NDBs for them are created. Communication with them will be started by NET2ND if they don't do so first.

4. ICMRCF(69) REQUEST CONFIGURATION - Each node has several network devices that other nodes may access. This routine handles the REQ CONFIG message and simply calls NCSCNF to send a CONFIG message back to the requester.

5. ICMCNF(69) CONFIGURATION - The old configuration (if any) is cleared out and the new one stored. The data comes in pairs of device type and quantity. If the node has any TTYs on it, TTYCNT is called to start trying to connect them all.

6. ICMDRQ(70) DATA REQUEST - This message allows the -10 to send more data on a connection. After the increase in the number of message we can send is stored, if the device on this logical link is not a TTY (LPT, TSK), then NETWAK or SETIOD is called to tell the job the good news. If the device is a TTY, then TTYDRQ is called to make sure IO is active to that TTY. These are called even if the counts were non-zero. It might be useful to only wake the job if the count was zero to reduce the number of spurious IO done interrupts we get.

7. ICMCTL(71) STATION CONTROL - There are two ways to receive a STATION CTRL message. First, a node in the network may have received a load request from one of its neighbors. The data in this DDCMP maintenance message is copied into a STATION CTRL message and sent to one of the -10's in the network (so determined by having an MCR device). The other way is from the NODE. UUO station control function which sends a STATION CONTROL message to an -11 requesting that the -11 do something (usually examine or deposit memory). Once the -11 processes the request, it sends another back to return data, or success/fail indication.

Once a Second Interface

NET2ND(46) - In 6.Ø3, this was called once a second. Presently, it is called once a second for each FEK for the DN6Ø code. For each FEK, if the kontroller is online, we check to see if a NODEID has been sent; if not, one is. If we do not have a PCB set up to read a message from that FEK, NETRDD is called to set one up. Each NDB on that FEK is checked and START, REQ CONFIG, and NEIGHBORS messages are sent as required. Finally, outstanding boot requests are timed out in case NETLDR doesn't get far enough to request them (i.e., doing another load).

DAP interface

IDC??? - These routines process incoming data messages.

1. IDCCTL(73) - This is the dispatch for all data messages. First the DLA and SNA are checked for reasonableness and different dispatches are taken for TTYs and anything else. All the TTY dispatches are described later.

2. IDCDAT(75) DATA WITHOUT END OF RECORD - The NDPIDT dispatch is taken into the device's NDP. This calls the service routine to move the data into a monitor buffer. NETWAK or SETIOD is called to wake the job.

3. IDCDAR(75) DATA WITH END OF RECORD - Same as above.

4. IDCSTS(76) STATUS - The NDPIST dispatch is taken to have the service routine handle online/offline conditions, errors, etc.

5. IDCCMN(77) CONTROL - NOP. (Only used for TTYs).

6. IDCUID(78) USERID - NOP. (No code to handle.)

7. IDCFSP(78) FILE SPEC - NOP, same as above.

DAPHDR and DAPHDI (79) - These are called when NETSER wants to build a DAP message for eventual transmission to the net. These will make the PCB, store the NCL header (with the interrupt bit on if DAPHDI was called) and the link address, and return with T1 set to the byte pointer for the DAP data.

DAPWRB, DAPWRT, DAPWRC (79) - These are called when NETSER has a message ready to be sent. The various entry points control the type of manipulation the front end will do to the message as it copies the message: binary, image, or LPT compression.

DLYDRQ (81) - This is called from UUO level when we cannot output to the net because we have run out of data requests. The nonskip return is taken if the channel has been opened in nonblocking IO mode; the success return is taken after a call to NETHIB. A successful return does not mean that output can occur; it may signify that the connection is broken or nothing at all. The nonskip return does not indicate an error; it merely means that the caller has to return to UUOCON instead of waiting for IO to complete.

CHKDRQ (81) - This is called by output routines to decrement the pending data request count. If there are no data requests, the error return is taken and the count not decremented.

NETRLS, NETHNG, NTDONL (83) - These are common routines used by the device drivers. NETRLS handles part of the RELEASE UUO, NETHNG is called when the device goes offline, and NTDONL is a test to see if it is still offline.

Terminal Interface

"SCNSER" level - The SCNSER level of the network are those routines called via LDBISR. They all merely save information and trigger the interrupt level code to do the real stuff.

D85TYP (86) - This starts off the process of getting characters from SCNSER to the network, something that deserves a lot of reworking. As each character is passed to NETSER, a check is made of important status bits that may have changed. If differences are found, then the LRRCHP bit is set for that LDB and will be taken care of later. After marking the station and TTY as busy, the routine exits if it is at interrupt level (this means that NETSER had called XMTINT and it was trying to give us the next character to send). If we are at UUO level, a software interrupt is forced to get us to interrupt level to avoid race conditions and the interrupt routine finds an active TTY (it may not be this one) that needs output done to it and starts it running.

D85CHP (88) - This marks this line as possibly needing new characteristics sent to it.

D85LPC (84) - This handles line parameter changes. Presently the only one is "unlock keyboard" and that's only used IBM for 2741's.

D85ELE (84) - This handles requests to change the 2741 element installed.

D85REM (87) - This handles "special request" from SCNSER, I guess for network-only terminals. The only one that is handled is the "CTRL/O received" function. This initiates the sending of a character gobbler, a function that tells the remote station to erase the output buffer. I believe the code at TTYWAT makes sure that the line is left idle.

D85OFL (84) - The offline check is to see if the LDB is not connected to a TTY at an -11.

Interrupt Level - NETSER interrupts are software caused, just like scheduler interrupts are. When NETINT is entered, the routine that caused the interrupt has left the address of the NDB requiring service in NETFLG, the interrupt flag location that is analogous to done and error CONI bits of a real device.

NETINT falls into SCNTTO which asks COMNET for a active TTY to service. Since it will probably be sent a message, a TTY PCB is obtained from COMNET from a special pool of small PCBs that never go through CORE1 since they're accessed so often. SCNTT7 is called to service the request, then a message is sent if one was needed, the busy bit is cleared. After NCSACK is called (I'm not sure why), then the interrupt is dismissed.

SCNTT7 looks at request bits in LDBREM for the LDB passed to it. Each bit has an associated processing routine. The presently used bits, routines are:

1. LRRSCG, TTXGBL (108) - This sends a character gobbler to flush the output buffer on the remote node. While I think the remote should be able to do the CTRL/O processing itself, this function will still be necessary for ^C^C, the CLRBFO TTCALL, etc.

2. LRRCHP, TTYCHP (91) - This inspects 13 SCNSER bits and, if any are changed, sends a STATUS message to the remote to correct its impression of the connection. The bits checked are things like page mode, hardware tabs, free CRLFs, etc.

   Next, several byte values and a few more bits are compared against the old values and if differences are noticed a CHARACTERISTICS message is sent to straighten those out.

3. LRRTTO, TTXDAT (110) - This is the data output routine. If we have exhausted the data requests, LRRTTW (waiting) is set and LRRTTO (output available) is cleared and the the routine exits and will be recalled when some data requests arrive. Otherwise, the data request count is decremented and TTYHDR is called to write the NCL header for the data. The character saved by the SCNSER level code is retrieved and put in the buffer and XMTINT is called for more characters as long as there is data in the TTY chunks and space in the NCL message. (If the code looks funny, remember that XMTINT doesn't return a character, it calls the ISRTYP dispatch, D85TYP, which stuffs the character in the LDB and returns to XMTINT which returns to TTXDAT.)

4. LRRADL, TTX801 (109) - This sends an autodial request to the remote via a DAP CONTROL message. The number to be dialed is stored in location TELNUM and is the main reason why only a single line can be dialed at once. Removal of this restriction will probably require extending the LDB by at least a word.

5. LRREPW, TTXEPM (90) - Echo pipeline markers are sent from the remote when the connection is in host-echo mode, i. e. when the -10 is doing the echo. After every character sent to the -10 from the -11, an echo pipeline marker is included. These have a serial number and when the -10 receives one it sends it back if the -11 can restart echoing. If the -11 has sent some characters before it receives the returning pipeline marker, it cannot yet restart echoing since some unechoed data is still in the pipe. If it hasn't sent any new characters, then the serial number of the pipeline marker it read will match the number of the one last sent and the -11 will send a status message to the -10 saying it will echo future characters until the user types another character it can't deal with.

Link Construction/Destruction

TTYCNT (104) - When NETSER learns of a new node in the network, it tries to connect all the TTYs on that node, one at a time. This starts when the CONFIG message arrives from the node specifying how many TTYs are on the node. If the system isn't stand alone, TTYCNT is called to send out the first connect message. When the connect is refused (DSCTTY(104)) or confirmed (TTYCNX(105)), TTYCNT is called again to try for the next.

Message Generation

TTYHDR, TTYHDI (112) - These write the NCL header into the TTY message area the TTY PCB points to. Like DAPHDI, TTYHDI writes an interrupt message.

Message Decoding

TTYDRQ(94) - This is called when we receive data requests from the other node. The data request counter is increased and if the TTY was waiting for some data requests, the wait bit (LRRTTW) is cleared, the output available bit is set, and service is requested for the node. Aha! Service is requested by calling D85RQS which won't request an interrupt because it's already at interrupt level. SCNTTO is still reached because when TTYDRQ exits, NETSER will eventually get to INPDIS which goes to NETRDD which sees that .RMSUI, the NDB service request bit, is set and calls SCNTTO directly.

TTYCTL(95) - This processes the DAP CONTROL message which comes in four different flavors:

1. TTYEPM(95) - Echo pipeline marker. This is received after data is sent by the remote when it is relying on the -10 for echoing. The code simply records its serial number and sets a flag to remind itself that it has to send one back when the -10 is willing to let the -11 echo again.

2. TTXGBL(95) - Character gobbler. While one could define a function for this, this presently is an output only function.

3. TTYCHR(96) - Characteristics. This is the opposite of TTYCHP and stores most of the TTY parameters into the LDB.

4. TTYATO(95) - Auto dial. Virtual terminals don't have auto-dial units on them, this message is ignored.

TTYDAR, TTYDAT(97) Data - When data is received from a remote terminal, the data is simply passed to RECINT for processing.

TTYSTS(98) Status - This handles changes in all the TTY states the characteristics message doesn't.

Miscellaneous

GETZWD (40) - This is the main interface between NETSER and CORE1. To avoid tying up a lot of memory whenever the networks hang (they've been improving), it has an internal check to prevent it from using more than 25% of the monitor's free core. If it can get the memory, then it zeros it before giving a successful return to the caller.

SLPZWD (40) - This calls GETZWD until it succeeds, sleeping between calls.

NETSLP (52) - This is what SLPZWD calls to sleep for 2 seconds.

GIVZWD (40) - This is how the memory is returned to the monitor.

NETHIB (52) - This goes into event wait to wait for an interesting event to occur (generally something like the arrival of an answer to our connect or the arrival of some data requests when we're out).

NETWAK (52) - This is how we get out of event wait.

TSKSER

This section describes the routines that TSKSER has and how they relate to the IO UUOs. Even the routines called from NETSER at interrupt level are described in regard to the UUOs they eventually relate to.

The TSK is an odd network device in that it is a directory device. For all other devices (CDR, LPT, etc.) a connection is made by the time the OPEN UUO returns whereas TSKSER delays until an ENTER or the first INPUT is done.

The directory that TSKSER uses is kept solely in core (it can't very well live on a TSK like DECtape and disk directories live on DTAs and DSKs!). There is one entry for each connection allowed for TSKs (a NETCNF parameter). The directory entry consists of a 6 character task name, a 3 character extension (presently unused), and a PPN (only [1,2] can specify one). Furthermore, the address of the DDB associated with the task

directory entry is stored.

The name stored in the directory is the task name associated with this TSK. The name stored in the DDB is the name of the TSK connected to.

TSTTSK (18) - This is called from the device search routines to see if we're looking for a TSK. If we are, TSTTSK verifies that the referenced node exists and supports tasks then makes a DDB, assigns a directory slot and gives a success return back to UUOCON.

LOOKUP

TSKLKU (16) - A LOOKUP does not send a connect message, it merely tells TSKSER that the program is willing to talk to the task named in the LOOKUP block. The name, extension, and PPN are stored in the TSK directory.

ENTER

TSKENT (13) - This specifies that we should try to make a connection to the task named in the LOOKUP block. If a LOOKUP hasn't been done, the program name and the job's PPN are put in the directory, the extension isn't zeroed (it uses a SETZ!), and the ENTER block parameters are copied to DEVFIL, EXT, and PPN. If we are talking to a remote task, NCSCNT is called to send the connect; for a local task, we reach TSLENT which searches the directory for a match with the task name in the LOOKUP block. If it finds a passive task to connect to, it puts the name into the target's DDB, marks the DDBs as local tasks and links them with logical link addresses stolen from the real network. (That means that local tasks can't exist without a full network.) Finally, NETWAK is called to wake the passive task if it was waiting for a connection to be made.

TSKCNM (20) - This is called from NCSCNT to send the task names involved. It first picks up the task to connect to from the DDB and puts that in the connect message, then picks up its own name from the directory and sends that as the requesting task name.

TSKCNT (4) - The other system gets here when it receives the connect request. If there is a match in the task directory (TSKFND call), we try to return a connect confirm to complete the link. NETWAK is called in case the program is waiting on an INPUT UUO.

TSKFND (5) - This is used by TSKCNT and TSLENT to match a connect request with a waiting passive task. If it finds one, it returns the address of the TSK DDB.

INPUT

TSKDAT (7) - This is called from NETSER when data is received from a remote TSK. This copies the data into the monitor input buffers for the UUO level code to read. For local tasks, the same job is done in TSLOUT.

TSKINP (12) - This is the UUO level part of TSK input. If the link has been disconnected, we jump down to TSKINC to read any data that is still in the monitor buffers. Otherwise, if we are not connected, a connect confirm hasn't be received yet so we block until that happens and continue at TSKINC.

TSKINC is similar to other network input routines. If monitor buffers haven't been built, they are.

If there is no data in them and the connection is broken, IOEND is set and the user receives an end of file indication when the buffer ring empties. If the connection is open, DRQMBF is called to ensure that the other end can send data to us (though local tasks call TSLDRQ to fake some data requests). Since buffers are filled at UUO level and buffers are filled ahead for the efficiency that provides, while the monitor buffers have no data, the user buffers may. That case is checked, and if there is a full buffer, we return to UUOCON to give that to the user. If there wasn't, and the TSK was opened in asynchronous IO mode, we return via TSKOFF which performs some cleanup. Finally, we're stuck, so we call NETHIB until data does appear in the monitor buffers.

If (when) the monitor buffers have data, TSKIN3 copies as many buffers as it can into the user buffers. IOACT is used in a questionable manner to remember when we have to exit the copy loop, we never return to UUOCON with it set. After copying all the data we can, we try to send some data requests to keep things rolling and return to UUOCON.

OUTPUT

TSKOUT (10) - This waits for data requests if necessary, builds a DAP header and sends the data directly from the user buffer. IOACT is left on until the message makes it out of the -10 (see TSKINT). If the output is to a local TSK, TSLOUT is called which copies the data into the target task's monitor buffers.

TSKINT (8) - This is called when the output message makes it out of the -10 and into the front end. It advances the output buffers and sends the next one, if any.

RELEASE

RMVTSK (19) - This is called from UNLDDB when a DDB is being removed to also remove the entry in the task directory. If this DDB was for a local TSK, it wakes the other side to tell it that its connection has gone away.

TSKDSC (6) - If the remote TSK does a RELEASE, it will send a disconnect to us which will be processed here. Only some cleanup is done and the disconnect is confirmed. If the TSK was being used for output, IOBKTL is set to prevent future outputs, if it was being used for input, nothing is done but TSKINP will figure out what happened at set end of file.

SPECIFIC EXAMPLES

The following few sections take some common events and shows how the routines in NETSER interact to get the job done.

Node startup

NETSER communicates with the rest of the network via either a DN87 (which means a DL10 interface) or a DN87S (which means a queued protocol interface on a DTE20). Startup with the two systems is totally different until TOPS-10 realizes the link is available and gets NETSER into the act and is identical after that. For the DL10 interface, COMDEV tries to leave the DL10 window enabled at all times. If it thinks the -11 on the other side is down, it will field interrupts from that DL10 itself. On each interrupt it looks at a program name word the -11 has access to via a DL10 byte pointer in its window. If the name is "DC75" (other possibilities are presently "DC76" or "DN60"), it calls D75III in D85INT which does little more than set the FK.ONL bit in the FEK for that DL10 port.

On the other hand, the queued protocol startup begins when DTELDR tells DTESER to start primary protocol on any DTE20. DTESER calls D8SUP in D8SINT which immediately returns if it doesn't think there is a DN87S on the other end. If the DTE controls a DN87S, then FK.ONL is set and it returns.

NETSER now gets into the act at NET2ND. On every call it checks FK.ONL (the sign bit) and, when it first sees it set, it will not see FK.NID set so it sends the NODEID to start NCL startup and readys an input buffer for the front end to fill. NET2ND then scans all the NDBs associated with that FEK and normally will find none and exit.

Meanwhile, the front end will see it can transmit a message to the -10 and will send a NODEID to it. the message is delivered to D85INT or D8SINT which dispatches it to FEKINT which goes to INPINT which goes to INCTID which makes an NDB for the front end. To tell the rest of the net that the front end is adjacent to us, a NEIGHBORS message is sent to everyone we know.

Beyond this point things start behaving asynchronously and the order described below is only one possible. Unless it is very busy, the front end will probably make the next move and send a START message to the -10. When NETSER gets the message it resets the data base in the NDB pertaining to the queues of messages being processed and active TTYs, the pending station control stuff and the node's configuration. INCTST then calls RDOPPD to get the node name and other sundry stuff. This resets all the message counter data, resetting communication back to message 1. Finally it calls NCSSTK to send a STACK (start acknowledge) back to the front end.

When the front end receives the STACK it considers NCL to be started.

An alternate path would occur if the front end didn't send the START. NET2ND will be called again and this time it will find an NDB off the FEK and that it is not started. NCSSTR is called to send the START, eventually the front end will reply with a STACK. NET2ND notices this and sends a REQ CONFIG to get the node's configuration. When that arrives, the data is copied into the NDB and if the node has any TTYs on it and the system isn't stand alone, TTYCNT is called to try to connect one of them. When a reject or confirm comes back the next TTY is connected and the process loops until all of them have been touched.

A third path would be that both the -10 and front end send STARTs. When this happens, all of the code for the first two paths is executed and eventually everyone winds up in the same state.

Startup between -10 and nodes that are not front ends is similar, the major difference being that knowledge that another node exists comes from NEIGHBORS messages from nodes that have gone through NCL startup instead of NODEID messages.

There are a couple problems here. RDOPPD is called when a NODEID, START, or STACK message arrives. In the third case after we send a STACK we could start sending data messages. We will eventually receive the STACK from the other node and RDOPPD will zero the message counters which will cause data flow to that node to get fouled up. The cure is simply to make sure that this doesn't happen. Looks like the best thing is to not bother calling RDOPPD on a STACK. Fortunately, that case doesn't happen

very often. This will also handle a problem with complex topologies. If NETSER sends a STACK, it will think that it can now send messages and immediately do so. In a complex topologly environment, those data messages can arrive before the STACK and so must be queued for processing after the STACK. However, messages sent before the node sent its START could still be arriving and those must be discarded. Sigh. If we throw away all of the messages and rely on retransmission to get things straightened out we still haven't solved the problem of messages sent before the START is received that arrive after the STACK is received.

Device Usage (OPEN, IO, RELEASE)

Let's take the case of a program opening the LPT at a DN82, outputing some data to it, and doing a RELEASE.

The flow for the OPEN is UUOCON calls the device search routines. If someone is already using the LPT, its DDB will be found and the error return will be taken. If it can't find the DDB or if the open was for a generic device, NETSER is entered at TSTNET which calls GENNET which calls MAKDDB to make a DDB and then calls NCSCNT to send a CONNECT INIT for the LPT.

When the 82 receives the connect, if some other node is not already using the LPT, it returns a CONNECT CONFIRM. If the LPT is in use, it returns a CONNECT REJECT (which is actually a DISCONNECT message).

NCSCNT calls NETHIB to wait for a reply. If the connect is rejected, the NETRSN byte will have the reason for the reject; if it is confirmed, the IOSCON bit will be set; and, if the 82 goes offline (crashes or becomes unreachable because another node crashes), then RMVNDB will put a special value into the NETRSN byte to indicate that. Assuming that the connect is confirmed, NETSER returns to UUOCON which returns to the user who sets up output buffers and does an output. The 82 has been active itself and has sent a DATA REQ to the -10. If this arrives before the -10 does the output, the output UUO will call LPTOUT in NETLPT which will get to LPTOU2 which will build the DAP header for the message and send it on its way. The process continues until the data request count is exhausted whereupon LPTOUT calls DLYDRQ to wait for activity on the link and goes back to try again.

When the program is done with the LPT and does the RELEASE, NETSER's NETRLS is called to set the IOSREL bit in the DDB. When the RELEASE UUO is done, USRXIT is reached and notices that both ASSCON and ASSPRG are off so it calls ZAPNET which calls NCSDSC which sends a DISCONNECT and waits for it to be confirmed and then calls UNLDDB to remove all knowledge about using that device.

String Output From SCNSER

The SCNSER to NETSER interface suffers from a single problem, namely that SCNSER is designed to let you request it output a character but never to let you request the next character to be output. While this works fine for things like the CTY on KAs and KIs and DC10s, it loses big in any message oriented environment like networks. Let's look at the flow for an OUTSTR UUO and assume that we have two data requests we can fill.

SCNSER starts processing the data at OUTST1 where it gets a character and calls TYO7W who falls into TOPOKE who finds that we are idle and calls XMTINT who wanders all over the place but eventually retrieves the character from the chunks and goes to TYPE who calls D85TYP via LDBISR.

D85TYP tucks the character into the LDB, forces a software requested interrupt to get to interrupt level at NETINT who finds the LDB and calls SCNTTO who calls SCNTT7 who calls TTXDAT who decrements the data request counter, retrieves the character, puts it in a NCL message and calls XMTINT to get another character to the output routine. XMTINT finds nothing in the TTY chunks (remember, we're still on the first character of the OUTSTR) and when it returns TTXDAT sees this because D85TYP wasn't called to set LRRTTO. We get back to SCNTTO who calls NETWRT to send the character and we return to TOPOKE (skipping TYPE and XMTINT) which returns to OUTST1.

We pick up the second character and go through the exact same sequence to output another single character message which will leave the data request counter at zero if the network is slower than the -10. Thank God the network is slower (except possibly of KA10s...).

OUTST1 now picks up the third character and gets it to TTXDAT who sees that the data request counter is 0 and sets the LRRTTW bit meaning it is waiting for some data requests to arrive and returns without calling XMTINT who would have turned off LDLOIP when it would have found no more characters in the buffers. OUTST1 now picks up the fourth character, calls TYO7W, falls into TOPOKE who sees that the line is active and immediately returns to OUTST1. Likewise, several more characters are processed and finally characters are put into the chunks at a respectable rate.

When data requests finally arrive, NETSER gets to TTYDRQ which clears the LRRTTW bit and returns to NCTDSP which goes to INPDIS which goes to NETRDD which goes to SCNTTO which builds one or more long messages which are sent to the TTY. Things continue in this mode until OUTST1 runs out of characters and returns to the user and TTXDAT finally empties the TTY chunks and leaves the TTY idle. Amazing.

A fairly simple change to this which would result in an impressive throughput improvement would be to treat network TTYs exactly like PTYs and not let TOPOKE start output to them. This would let the TTY chunks fill with characters and circumvent the idiocy that happens with the first few. Next, PTSTRT would be modified to do the TOPOKE function on network terminals. Something like this will be done for 6.04.

Terminal Echoing in the TOPS-10 Networks

Most characters are echoed by the network -11s and when they are, things move fairly smoothly. Under several circumstances, the -11s decide that they can't echo a character and all hell breaks loose. Before we can examine the fireworks, we have to understand the protocol functions involved.

Terminal Protocol - As with the unit record devices, DAP is used to interface with terminals. The software interface was described above; this subsection will describe the applicable message types.

1. DATA - This is used to move data from the -11 to the -10. Unlike unit record devices, these do not need data requests to be sent. This is an optimization for terminals in that terminals must always be able to deliver data to the -10, even if the input buffer is full. Otherwise, there would be no way to type CTRL/C to a looping program if typeahead has filled the input buffer. The -11 tries to send data to the -10 as soon as possible which means most characters are delivered one at a time. However, there is code to delay sending another message if one has already been sent but not acknowledged. This mechanism has the characteristic that things run more efficiently as load increases.

2. STATUS - This message controls most of the status bits of the terminal. One bit, deferred echo, is important to the discussion below.

3. CONTROL - This message has several subtypes:

    1. Echo pipeline marker - explained below.

    2. Character gobbler - When the -11 receives this, all characters in the output queue for the terminal are discarded. (This implements the CTRL/O function.)

    3. Characteristics - More status bits and bytes of information.

4. Auto Dial

Abbreviations below - In the following flow diagrams, these abbreviations will be used:

1. types(char) - This means the user types "char" on the keyboard of a network TTY.

2. DATA(list) - This is a list of characters sent in the indicated direction.

3. ACK - The NCL level acknowledgement.

4. EDE - Enter Deferred Echo mode.

5. LDE - Leave Deferred Echo mode.

6. EPL(num) - Echo PipeLine marker. Num is the serial number.

Normal Input - Below is an example of normal input to the -1Ø. The user is typing "ABCD"

```
USER                    -1Ø             notes
types(A)
DATA(A) -->                             1
types(B)                                2
                  <--   ACK
DATA(B) -->                             3
types(C)
types(D)
                  <--   ACK
DATA(C,D)         -->                   4
                  <--   ACK
```

1. The -11 sends the data as soon as it can.

2. The user typed another character before the ACK for the previous one came back. Therefore, the -11 will hold onto it for a while.

3. The ACK finally arrived so the -11 was able to send the data.

4. In this case the user typed two characters before the ACK for the previous message arrived so this DATA message was able to pack another character in.

In the succeeding examples, the "types" and "ACK" events will not be shown.

Typing a Control Character - When a user types something the -11 cannot echo (rubout or control characters other than TAB), it enters deferred echo mode. This means that the -10 will do all following echoing until the -11 says it is doing echoing again. This will not happen until the -11 discovers that there are no more characters it has sent to the -10 that have not yet been echoed. This is accomplished by the echo pipeline marker. Whenever the connection is in a deferred echo state, the -11 sends an EPL after each DATA message. When the -10 sees an EPL, it remembers to send it back when it believes the -11 can continue echoing. EPLs are marked with a serial number that is incremented each time the -11 sends one so it can recognize when the last EPL comes back. In this case, the user types "A^BCD".

user		-10	notes
DATA(A) -->			
EDE -->			1
DATA(^B)-->			
EPL(123)-->			2
	<--	DATA(^,B)	3
	<--	EPL(123)	4
LDE -->			5
DATA(C) -->			
DATA(D) -->			

1. The user just typed CTRL/B and the -11 informs the -10 that it must handle future echoing.

2. The -11 sends an EPL to mark the data in the network.

3. The -10 echoes the CTRL/B.

4. The -10 returns the EPL with the same serial number.

5. The -11 sees that the serial number of the last EPL received matches the last sent and tells the -10 it will echo now.

Again, but assuming that the user types the "C" before the EPL gets back.

```
user                          -1Ø                notes
DATA(A) -->
EDE     -->
DATA(^B)-->
EPL(123)-->
DATA(C) -->                                      1
EPL(124)-->                                      2
                  <--     DATA(^,B)              3
                  <--     EPL(123)
                  <--     DATA(C)                4
                  <--     EPL(124)               5
LDE     -->                                      6
DATA(D) -->
```

1. The user types "C" and it has to be sent without echoing.

2. And another EPL has to be appended.

3. The -1Ø finally echoes the CTRL/B and returns the EPL. However, the serial number doesn't match so the -11 ignores it.

4. The -1Ø echoes the "C".

5. And returns the second EPL.

6. The serial number matches and the -11 says it will do echoing again.

Rubbing Out Data - There are times when the -10 has to prevent the -11 from echoing data for fear of getting the typescript messed up. Rubouts are one case, single character IO is another. Here the user types "AZ<rub>BCD".

user		-10	notes
DATA(A) -->			
DATA(Z) -->			
EDE -->			1
DATA(rub)-->			
EPL(5) -->			
	<--	DATA(\,Z)	2
DATA(B) -->			
EPL(6) -->			
	<--	DATA(\,B)	3
	<--	EPL(6)	
LDE -->			
DATA(C,D)-->			

1. The user types the rubout and the -11 enters deferred echo mode.

2. The -10 echoes the standard rubout echo. It won't type the closing backslash until the first non rubout character is typed so it can't let the -11 echo yet and defers the EPL.

3. It finally echoed the backslash and the next character. Now it can also send an EPL back.

DIFFERENCES BETWEEN NCL AND NSP

1. The major difference is that NCL numbers messages on an end to end basis (i.e., node to node only), whereas NSP does so on only an end to end basis per logical link. This means that if a message is lost, a very rare occurrence based on our present experience, then communication on all NCL links between the two nodes pauses until things get straightened out. On an NSP network, only the one logical link pauses.

2. NCL is a device oriented protocol, i. e. connections may reference devices as well as tasks. NSP only allows reference to task names. However, there is nothing to prevent an NSP implementation from reserving some names for internal use and passing the data to a device driver instead of a task. In fact, in some systems all device drivers are tasks....

3. NSP has no routing at present.

4. The NCL startup sequence contains information as to what devices that node has.

ROUTING

Routing in the TOPS-1Ø networks is based on a node learning who its neighbors are via:

1. the NCL NODEID message which neighbors exchange before starting NCL communication between each other.

2. receiving an NCL NEIGHBORS message from a node it is in communication with.

Consider the sample network where node A is a DEC-1Ø with two DN87 or DN87S front-ends (nodes B and C) each connected to a DN82 (nodes D and E). Each DN82 is connected to the other, providing multiple paths between all nodes. Assuming that only node B is down, when it does come up it will learn that node A is a neighbor from a NODEID message sent over the DL1Ø or DTE2Ø and will learn that node D is also from a NODEID sent on the synchronous line. It will learn about the existance of the other's nodes from NEIGHBORS messages that nodes send after NCL starts. One possibility is that it will learn about node C from node A and node E from node D. It is important to realize that startup can occur in any order. In an extreme case, B could learn that A is in the network before it got a NODEID from A saying it is a neighbor!

Routing is based upon a node calculating the "cost" of sending messages to its destination via each of its neighbors. A node knows the cost of reaching each of its neighbors. These are determined presently at assembly time. The cost of a synchronous link is something like 1Ø, and the cost of a 1Ø-11 link is 1ØØ, high enough to discourage routing through the -1Ø if there is another path available. (In 6.Ø3 there had better not be another path....) The cost of links between nodes other than neighbors is determined from the NEIGHBORS messages received from each other node. The contents of the message is a list of the sender's neighbors and the cost of reaching each one. Everytime the topology of the network changes, nodes will execute their routing algorithm and store the results so that routing each message will take very little time.

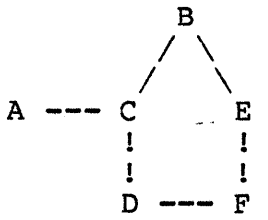
Routing from NETSER will be very simple, it will merely send messages on the least costly path. Therefore, messages destined for nodes B and D will be sent to B and messages meant for C and E will be sent to C. For example, the cost from A to D via B is 11Ø; whereas the cost from A to D via C is 12Ø. Should node B go offline, NETSER will find that messages to D will have to be sent via node C at a cost of 12Ø.

Routing in the -11s is one step more complex. -11s keep two paths (primary and secondary) and send data via the secondary if the output queue to the primary are significantly longer than to the secondary, where "significantly" will be described in the design spec. Therefore, messages from D to A may go via B as the primary path (cost 110) or via E as the secondary path (cost 120).

In general, this scheme is rather simplistic but works reasonably well in small networks. Problems include:

1. NETSER's insistance on only using one front end, which will probably be maintained since front ends really should be willing to offload the -10. Note that node A will never take advantage of the greater throughput available by sending messages to any of the -11s via both front ends. One solution is to link front ends together and let them optimize throughput. Another is to give NETSER a more intelligent routing algorithm at the expense of some efficiency.

2. Congestion problems are poorly handled if the congestion occurs more than one node away. Consider a network with the configuration below and assume each link has the same capacity.



If node A is generating traffic to node D at the line's capacity, everything will work optimally. However, if node B begins doing the same thing, it will send everything to node C since that will be the primary route. Since that line can handle the load, messages will not be routed via node E. However, inside node C, the queue for the link to D will quickly grow and it will start routing messages to its secondary path, B. B will look at the message and promptly send it back to C, which will now exceed that link's capacities and finally some overflow will occur to E.

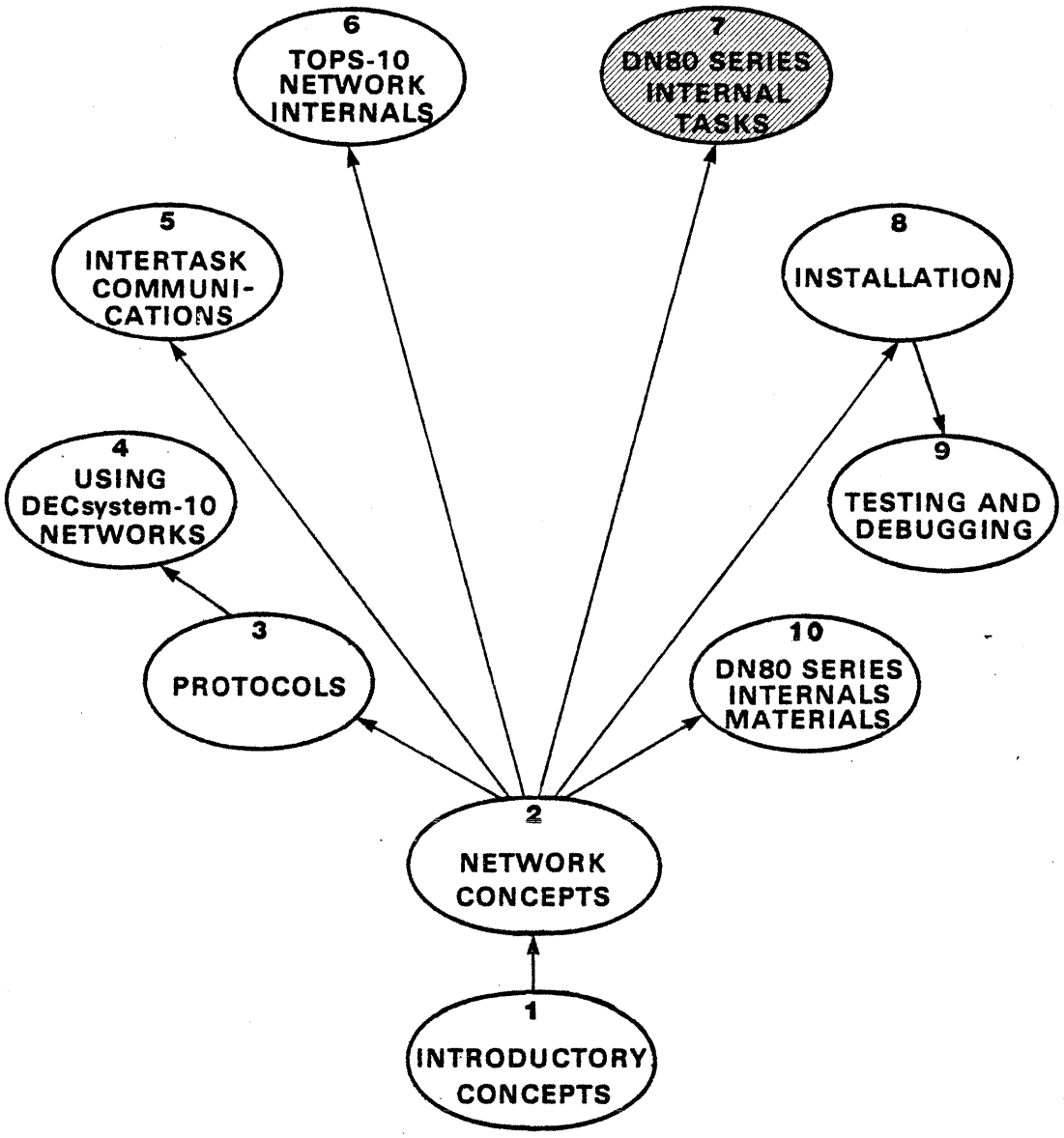
Adaptive routing algorithms like the ARPAnet's do not suffer from this problem but sometimes will try to route a message back to the sender. (Studies suggest that a special check to prevent this does decrease the average residency on messages in a network with adaptive routing).

[End of NCL.RNO]

DN80 Series Internal Tasks
Module 7

<<For Internal Use Only>>

Course Map



M8 0277

Introduction

The DN80, DN81, and DN82 remote stations support user written tasks (NOT TO BE CONFUSED WITH THE PSEUDO DEVICE TSK:) which run in the -80 series. There are several (-80) system calls which make terminal data available to these tasks. There are also a data structure and scheduling code to run these tasks in the -80.

When installations need to perform special functions in the DN8x, such as polled terminal handling, screen formatting, data field checking, or input consolidation, a special option for the DN82 system software provides monitor services to these user-supplied programs and tasks by means of macro calls.

The various user tasks and their installations are outlined in the following pages.

Objectives

Upon completion of this module, the student will be able to

1. List 3 cases where one might use -80 series user tasks.

2. Draw a flow-chart (identifying the user calls) of a screen managing task using an -80 series user task.

What are DN80 Series Internal Tasks

GENERAL CODING RULES

The DN82 system assumes that the user-written code is correct. Hence, programming errors in a user task can corrupt the system software. Furthermore, user tasks must:

1. not change the processor level, and

2. not modify the trap and device interrupt vectors.

Programmers are encouraged to restrict use of system macros to making system calls only.

In general, the macros perform their functions and return immediately. Except where results are returned in registers or memory, the registers, memory, and stack are left unmodified by the execution of a macro. In general, macro arguments may be in registers, in core, or immediate, but they may not be on a task's stack. Exceptions are the words BLOCKING or NONBLOCKING as arguments, and the names of jobs in WAKE, SEND, and TRIGER macros.

How They Work

TASK SCHEDULING

On system initialization, the DN82 flags all tasks as runnable using the starting addresses given in the TSKGEN macro. A task that is started with a TRIGER macro begins at this starting address. Tasks should take this into account and should consider starting with a HIBER macro, if they want to wait for a TRIGER macro.

If a power fail or other system restart occurs, the tasks are restarted using the restart addresses given in the TSKGEN macro. Before restarting the tasks, however, the DN82 reclaims all previously allocated buffers (chunks). Therefore, tasks must not reference chunks they possessed prior to system failure.

The DN82 maintains four queues of tasks, one for each of the four levels of task priority. Tasks in the four priority queues are scheduled on a round-robin basis. All tasks in a given queue must be in a wait state before any tasks in a lower priority queue are executed.

Note that the queues of tasks to run is looked at as the last (lowest priority) queue examined in the main loop.

When scheduled for execution, the task is given a unit of run time. The unit of run time is an assembly parameter applying to all user tasks, whose default value is one-tenth of a second. As

each task is started by the scheduler, it runs to completion or until one of the following conditions occurs:

1. HIBER macro is executed.
2. PUT macro is executed in BLOCKING mode.
3. GET macro is executed in BLOCKING mode.
4. A user timeout occurs.
5. An EXIT macro is executed.
6. A higher priority task becomes runnable.

When a HIBER macro is executed, the task is removed from the Active Task List until the specified I/O activity occurs, or until the specified amount of time has passed. The task is then placed back into the Active Task List.

When a PUT or GET macro is executed in BLOCKING mode, the task is removed from the Active Task List until the requested operation is completed. When the requested operation is completed, the task is rescheduled into the Active Task List. The next runnable task is started when a task goes into BLOCKED I/O wait mode.

When an EXIT macro is executed, the issuing task is permanently removed from the Active Task List. The task can be run again only if the DN82 system is reloaded or restarted, or if another task issues a TRIGER macro for the task.

If a task remains runnable for more than its unit of run time, the task is rescheduled and placed at the end of its priority queue. This prevents loops in a user task from locking out other tasks at the same or lower (specified) priority level.

CORE MANAGEMENT

The DN82 system controls free core through the use of linked lists of system buffers (chunks). Each chunk is the size specified by CNKSIZ, a system parameter defined in the DN82 system software. Chunks are dynamically allocated and deallocated by the DN82 system and user tasks through the use of the CNKGET and CNKFRE macro calls.

The first word of a chunk contains either a pointer to the next chunk in the list, or, if a chunk is last in a list, a zero. In the first chunk, several words following the link word contain the control information for a network (NCL) message. (See the tables in Module 10.)

SYSTEM MACRO CALLS

The system macro calls available to user tasks for controlling task execution or I/O handling are divided into several categories. The format and function of each of these commands is defined and an example given in the following pages.

TASK CONTROL:

TSKGEN	Log a task into the system

INTERTASK COMMUNICATION:

SEND	Send a message to another task
RECEIVE	Receive a message from another task
WAKE	Wake up another task
TRIGER	Restart another task

CORE MANAGEMENT:

CNKGET	Allocate a buffer
CNKFRE	De-allocate a buffer

CLOCK HANDLING:

TIMER	Get system up-time

SUSPENSION OF TASK EXECUTION:

HIBER	Suspend a task
EXIT	Delete a task

I/O CONTROL:

OPEN	Initialize terminal for exclusive use by task
RELEASE	Release terminal from exclusive use by task
GET	Logical read from terminal or network
PUT	Logical write to terminal or network
IMGPUT	Logical write to terminal with no editing

TSKGEN Macro Call

function:

user tasks are defined in the DN82 system through the TSKGEN macro. This macro informs the DN82 system that a specific task is installed in the system and gives the system the information needed to schedule the task for execution. This nonexecutable statement must be the first statement in a task.

FORMAT:

TSKGEN name,stadr,rstadr,priority,pdlsiz

name - The 3-character task name used for the SEND, WAKE, and TRIGER macro calls.

stadr - The starting address of the task, used when the DN82 is first loaded and when TRIGER macro calls are handled.

rstadr - The restart address of the task, used when the DN82 is restarted after a power fail or other system restart.

priority - Task priority in the range 0 to 3, where 3 is the highest priority.

pdlsiz - The number of words (in octal) in the stack the task requires. Space for system interrupts is allocated in addition to the task space.

EXAMPLE:

```
TSKGEN FOO,FOOBEG,FOORST,3,20

        FOO             ;task name is FOO
        FOOBEG          ;starting address is FOOBEG
        FOORST          ;restart address is FOORST
        3               ;priority level is 3
        20              ;stack length is 16 (decimal) words
```

SEND Macro Call

function:

the SEND macro is used to send a 1-word message to a task. Each task has a queue to receive messages sent to it. The size of the queue is an assembly parameter for the DN82 system code.

If the receiving task is hibernating or has already executed a BLOCKING RECEIVE, it is awakened. If the SEND is successful, the N bit is cleared in the Processor Status (PS) word. If the SEND is unsuccessful, the N bit is set in the PS word. A SEND will be unsuccessful if the receiver's queue of messages is already full.

FORMAT:

SEND name,adr

name - The name of the receiving task as given in its TSKGEN macro.

adr - The location of the data word to be passed to the receiving task.

EXAMPLE:

```
SEND    ODT,%0          ;send to task ODT the contents of
                        register 0
BMI     DIE             ;if SEND unsuccessful, go to DIE
SEND    FOO,FOOBAR      ;send to task FOO the contents of
                        ;  memory location FOOBAR
BMI     DIE
```

RECEIVE Macro Call

function:

a task that expects to receive messages can get the next message by executing a RECEIVE macro. If data is returned, the N bit in the PS word is cleared. If a NONBLOCKING RECEIVE is executed when no messages are outstanding, the macro returns immediately with the N bit set in the PS word.

FORMAT:

```
RECEIVE  adr,word
```

adr - The address where the data should be returned

word - Either the word BLOCKING or the word NONBLOCKING

EXAMPLE:

```
RECEIVE  %4,NONBLOCKING      ;see if anyone sent message
                             ;if so the data is in register 4
BLP      WIN                 ;go to WIN if message received
```

HIBER Macro Call

function:

the HIBER macro is used to suspend task execution for a specified period of time - until I/O activity occurs, or until a WAKE macro for this task is executed. The argument is optional. If present, the argument specifies the number of seconds to suspend execution, after which the task is marked runnable regardless of I/O activity.

FORMAT:

```
HIBER  [arg]

arg     - Optional - number of seconds
```

EXAMPLE:

```
HIBER    3    ;suspend for three seconds
HIBER         ;suspend until I/O is done
```

WAKE Macro Call

function:

the WAKE macro is issued by a task to awaken a hibernating task.

FORMAT:

```
WAKE  name

name    - The name of the task to be awakened
```

EXAMPLE:

```
WAKE     FOO ;wake up task FOO
```

TRIGER Macro Call

function:

the TRIGER macro is used by a task to start another task that has completed.

FORMAT:

```
TRIGER  name
```

name - The name of the task to be started

EXAMPLE:

```
TRIGER   WEX ;request system to run task WEX
```

CNKGET Macro Call

function:

a task can request a core buffer (chunk) by using the CNKGET macro. If a chunk is available, CNKGET returns a chunk address in the specified location and clears the Z bit in the PS word. If free-core is unavailable, CNKGET clears the contents of the given address, sets the Z bit in the PS word, and returns.

FORMAT:

```
CNKGET  adr
```

adr - Location where chunk address is to be returned

EXAMPLE:

```
CNKGET   %3  ;get a chunk and return its address
             ;in register 3
```

CNKFRE Macro Call

function:

a task releases buffers to the free-core pool by using the CNKFRE macro.

FORMAT:

```
CNKFRE  adr
```

adr - Location of address of the chunk to be freed

EXAMPLE:

```
CNKFRE   CLAIRE ;release chunk at address CLAIRE
```

TIMER Macro Call

FUNCTION:

The DN82 system uses the line frequency clock to keep track of time. To obtain the up-time for the DN82 in clock ticks, the TIMER macro can be used. The TIMER macro returns the system up-time (in clock ticks) in the specified address. Note: remember 50Hz/60Hz discrepency if task is to be transported.

FORMAT:

```
TIMER  adr
```

adr - The address where the system time is to be returned

EXAMPLE:

```
TIMER    %4  ;put up-time in clock ticks into
             ;register 4
```

EXIT Macro Call

function:

a task that has finished executing can issue an EXIT macro to declare the task non-runnable. Another task, however, may issue a TRIGER macro to restart a task that has EXITED.

FORMAT:

EXIT

Terminal I/O Macros

Four macros are provided for controlling TERMINALS: OPEN, PUT, GET, and RELEASE. The first argument for each of these macros is a channel designator. For the CTY, the channel designator is -1. For other terminals, the channel designator is the physical line number. The second argument for each macro is either the word KEYBOARD or the word PRINTER. KEYBOARD specifies the terminal as an input device; PRINTER specifies the terminal as an output device.

The DN80 series software can be used to intercept data at a point between the terminal and the Network Control Language (NCL) processor. When a task has exclusive use of a keyboard, characters entered on that keyboard are sent to the task. The task can then send the data into the network (NCL) by using I/O macro specifications.

When no task has exclusive use of a keyboard or no task is waiting for input from the keyboard, input characters are passed directly to NCL to go onto the HOST-10. When a task GETs characters from a keyboard, the characters are not given to NCL. With the PUT macro, a task may give characters to NCL as though they came from a keyboard.

If a task has exclusive use of a keyboard, the DN82 maintains a queue of keyboard characters for the task. The size of the queue is an assembly parameter for the DN82.

When a terminal has not been assigned for exclusive use by a task, any task may do PUTs and GETs to the terminal. In common applications, the CTY is usually left free so that any task may issue PUTs to it for error messages.

The PUT and GET macros may be executed as BLOCKING calls, in which case the task is suspended while it waits for completion. If the macro is executed as a NONBLOCKING call, the macro returns immediately and, if successful, the Z bit in the PS word is cleared.

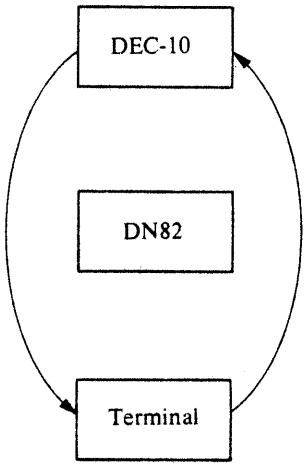


Figure 7-1 Data Paths Without Exclusive Use of Terminal

DN80-SERIES REMOTE STATION USER INTERFACE

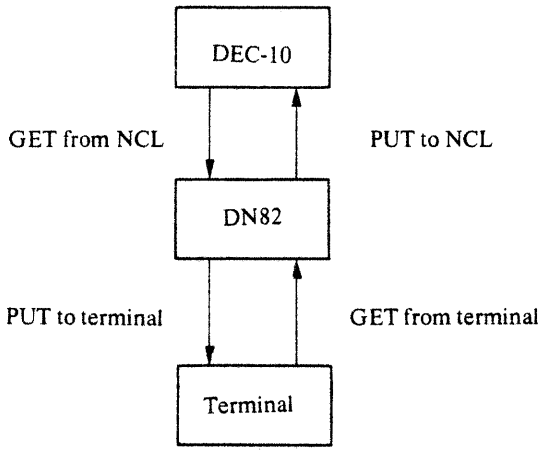


Figure 7-2 Data Paths With Exclusive Use of Terminal

OPEN Macro Call

FORMAT:

```
OPEN  line,word
```

line - Line number or -1 for CTY

word - The word KEYBOARD or the word PRINTER

EXAMPLE:

```
OPEN    #2,PRINTER      ;get exclusive use of TTY's printer
MOV     #0,%5           ;load line designator into reg. 5
OPEN    %5,KEYBOARD     ;get exclusive use of line 0's
                        ;keyboard
```

NOTE

To obtain exclusive use of both the keyboard and the printer, two macro calls must be used: one specifying PRINTER; the other, KEYBOARD.

RELEASE Macro Call

FORMAT:

```
RELEASE  line,word
```

line - Line number or -1 for CTY

word - The word KEYBOARD or the word PRINTER

EXAMPLE:

```
MOV      #-1,%3          ;load CTY designator
RELEASE  %3,PRINTER      ;release CTY printer
RELEASE  %3,KEYBOARD     ;release CTY keyboard
```

NOTE

To relinquish both the printer and the keyboard, two macro calls must be used: one specifying PRINTER; the other, KEYBOARD.

PUT Macro Call to a Terminal

FUNCTION:

To print characters on a terminal, the PUT macro is used with the argument PRINTER specified. Characters printed using this macro are given fillers as though they had come from the DECsystem-10. If the macro is successful, the N bit in the PS word is cleared. If the macro is unsuccessful, the N bit in the PS word is set.

FORMAT:

```
PUT  line,PRINTER,word,adr
```

line - The line number

word - The word BLOCKING or the word NONBLOCKING

adr - The address of a single character to print on the terminal

EXAMPLE:

```
PUT  #-1,PRINTER,BLOCKING,101
                          ;print char in location 101 on CTY
CLR  %2                   ;specify line 0 in reg 2
PUT  %2,PRINTER,NONBLOCKING,%4
                          ;print char in reg 4 on line 0
BNE  LOST                 ;branch if it didn't type
```

PUT Macro Call to NCL

FUNCTION:

The PUT macro, with the argument KEYBOARD specified, may be used to pass characters to NCL as though they had been typed on the keyboard. If the macro is successful, the N bit in the PS word is cleared. If the macro is unsuccessful, the N bit in the PS word is set.

FORMAT:

```
PUT  line,KEYBOARD,BLOCKING,adr
```

line - The line number

adr - The address of a single character to pass to NCL

EXAMPLE:

```
PUT  #3,KEYBOARD,BLOCKING,#101
          ;send upper-case A from line 3
```

GET Macro Call from a Terminal

FUNCTION:

The GET macro, with the argument KEYBOARD specified, may be used to request characters from the terminal. If the macro is successful, the N bit in the PS word is cleared. If the macro is unsuccessful, the N bit in the PS word is set.

FORMAT:

```
GET  line,KEYBOARD,word,adr
```

line - The line number

word - The word BLOCKING or the word NONBLOCKING

adr - The address where the character is to be returned

EXAMPLE:

```
MOV     #11,%0          ;specify line number 9
GET     %0,KEYBOARD,BLOCKING,%1
                        ;get next character from it into
                        ;register
```

GET Macro Call from NCL

FUNCTION:

The GET macro, with the argument PRINTER specified, may be used to get a character from NCL. If the macro is successful, the N bit in the PS word is cleared. If the macro is unsuccessful, the N bit in the PS word is set.

FORMAT:

```
GET  line,PRINTER,word,adr
```

line - The line number

word - The word BLOCKING or the word NONBLOCKING

adr - The address where the character is to be returned

EXAMPLE:

```
MOV      #7,IRVING          ;load memory with line number
GET      IRVING,PRINTER,NONBLOCKING,%3
                            ;get next output for line 7
                            ;into register 3
```

IMGPUT Macro Call

FUNCTION:

The IMGPUT macro is provided to print characters on terminals when no fillers or other editing characters are desired.

FORMAT:

```
IMGPUT  line,PRINTER,word,count,adr
```

line - The line number

word - The word BLOCKING or the word NONBLOCKING

count - The number (in octal) of characters in the string to be printed

adr - The address of the first character of the string

EXAMPLE:

```
FOO:    .ASCII/DECSYSTEMS/
                        ;10.-character string to be printed
IMGPUT  #-1,PRINTER,BLOCKING,12,FOO
                        ;type the string on the CTY
```

Sample

The task shown below is a simple routine to read addresses from the console terminal and print the contents of those addresses. The user types in enough digits to specify the address and then types a slash (/). The task then prints the contents of the address. This simple task demonstrates the use of many DN82 system macro calls.

```
TSKGEN   ODT,20
         CTY= -1
ODT:     OPEN    CTY,KEYBOARD
           BMI   ODT,99                      ;IN CASE ALREADY IN USE
         OPEN    CTY,PRINTER
           BMI ODT,99                        ;IN CASE ALREADY IN USE
                 MOVE #HLPTXT,R1             ;GET TITLE
                 JSR PC,ODTSTR               ;TYPE IT
ODT.10:    CLR          ODTLOC               ;CLEAR ADDRESS WE ARE
                                             ;EXAMINING
ODT.12: GET     CTY,KEYBOARD,BLOCKING,R0     ;GET A CHARACTER FROM
                                             ;THE CTY
           BMI   ODT.99                      ;SHOULD NOT LOSE HERE
         PUT     CTY,PRINTER,NONBLOCKING,R0  ;ECHO THE CHARACTER
           BMI   ODT.99                      ;IN CASE WE LOSE
         BIC     #^C177,R0                   ;STRIP THE PARITY
         CMP     #57,R0                      ;"/" OPENING CURRENT
                                             ;LOCATION
         BEQ     ODTOPN                      ;NO
         CMP     R0,#70                      ;CHECK FOR NOT NUMERIC
         BPL     18$                         ;NOT #
         CMP     R0,#60                      ;CHECK FOR NOT NUMERIC
         BMI     18$                         ;NOT #
         ASL     ODTLOC                      ;CONVERT ASCII TO #
         ASL     ODTLOC
         ASL     ODTLOC
         BIC     #70,R0                      ;MAKE CHARACTER A
                                             ;BINARY NUMBER
         ADD     R0,ODTLOC                   ;ACCUMULATE LOCATION
         BR      ODT.12                      ;GET REST OF NUMBER
18$:     MOV     #QESTXT,R1                  ;I DON'T UNDERSTAND
         JSR     PC,ODTSTR                   ;TYPE "?"
         BR      ODT.10                      ;TRY AGAIN
ODTOPN:  MOV     #TABTXT,R1
         JSR     PC,ODTSTR                   ;TYPE SPACES
         MOV     @ODTLOC,R0                  ;EXAMINE MEMORY
         JSR     PC,ODTOTY                   ;TYPE CONTENTS
         MOV     #CRLTXT,R1                  ;CARRIAGE RETURN/LINE
                                             ;FEED
         JSR     PC,ODTSTR                   ;TYPE CRLF
         BR      ODT.10                      ;BACK TO TOP
```

```
;HERE TO TYPE AN OCTAL NUMBER
ODTOTY: MOV    R0,-(P)                  ;SAVE DATA
        ROR    R0
        ROR    R0
        ROR    R0
        BIC    #160000,R0               ;STRIP PHANTOM BITS
        BEQ    20$
        JSR    PC,ODTOTY                ;KEEP GOING FOR REST
                                        ;OF WORD
20$:    MOV    (P)+,R0                  ;GET NEXT MOST
                                        ;SIGNIFICANT PART
        BIC    #^C7,R0                  ;STRIP EXTRA BITS
        BIS    #60,R0                   ;CONVERT TO ASCII
        PUT    CTY,PRINTER,NONBLOCKING,R0
        RTS    PC
ODTSTR: MOVB   (R1)+,R0                 ;GET NEXT BYTE TO TYPE
        BNE    10$                      ;0?
        RTS    PC                       ;YES, DONE
10$     PUT    CTY,PRINTER, NONBLOCKING,R0
        BR     ODTSTR                   ;GO TO NEXT CHAR
ODT.99: EXIT                            ;FATAL ERROR SO STOP
                                        ;RUNNING
ODTLOC: .BLKW  1                        ;CURRENT LOCATION TO
                                        ;EXAMINE
HLPTXT: .ASCIZ / RUNNING ODT/           ;TITLE

QESTXT: .ASCIZ /?       /
TABTXT: .ASCIZ /    /                   ;SPACES
CRLTXT: .BYTE  15,12,0                  ;CRLF
        .EVEN
```

Module Test

1. Give 3 cases where you might use an -80 series user task.

2. Draw a flowchart of an -80 series user task (expressly including the user calls) to perform one of the following:

   a. echo numerals only to terminal 2 on a DN82, sending these numerals to the host when a non-numeric character is encountered or when 10 numerals have been typed.

   b. copy all information going to the host from the keyboard of terminal 5 onto the printer of terminal 12

   c. write a task which will send 1 of 5 messages to the host from terminal 7. This message is specified by the number obtained by a RECEIVE macro call.

This page intentionally left blank

Evaluation Sheets

1. Give 3 cases where you might use an -80 series user task.

   a. screen-form filling

   b. multi-drop control

   c. monitoring transactions

   d. system measurement

   e. multi-terminal simulation

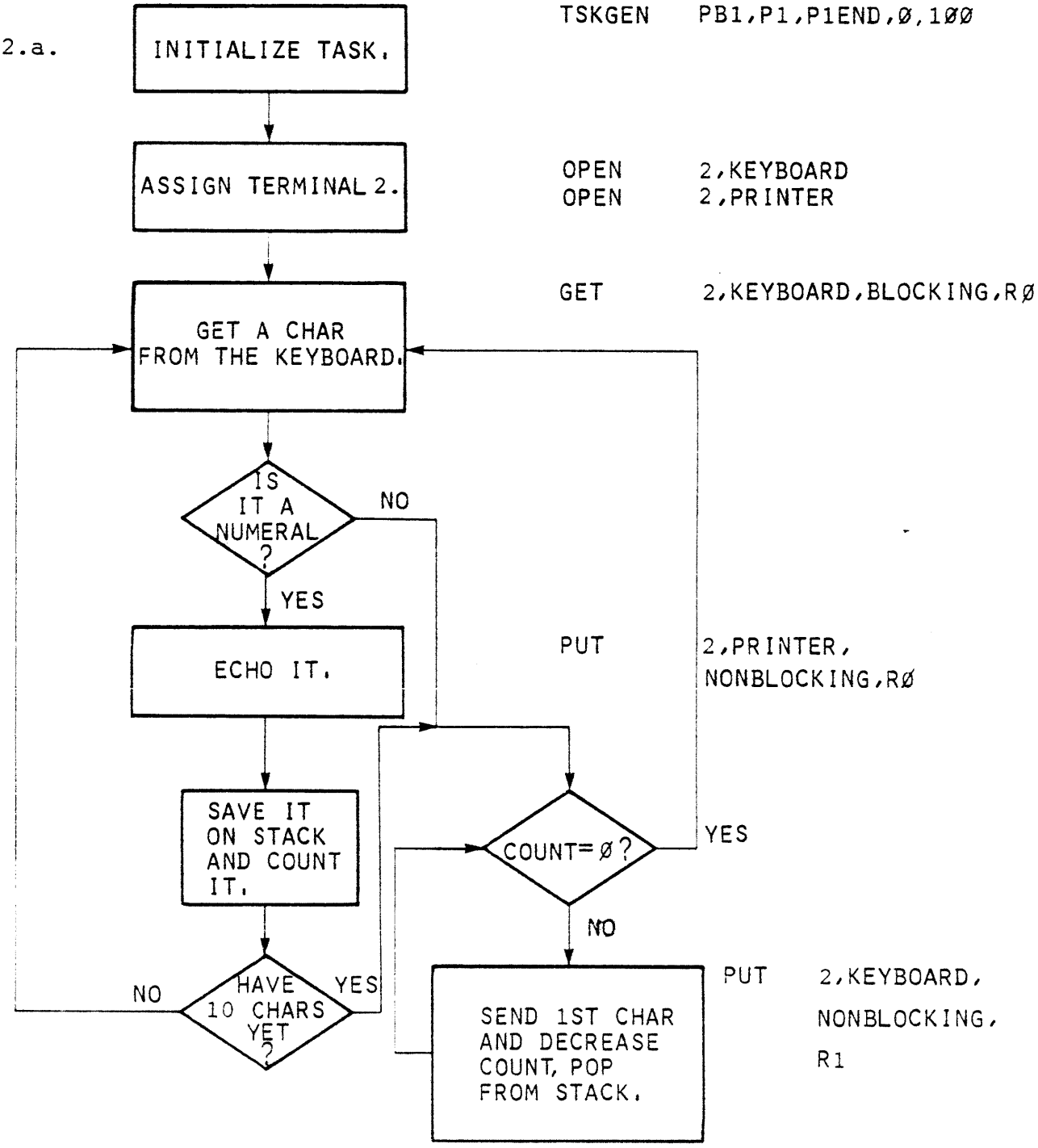
   f. debugging

   g. error logging

2. Draw a flowchart of an -80 series user task (expressly including the user calls) to perform one of the following:

   a. echo numerals only to terminal 2 on a DN82, sending these numerals to the host when a non-numeric character is encountered or when 10 numerals have been typed.

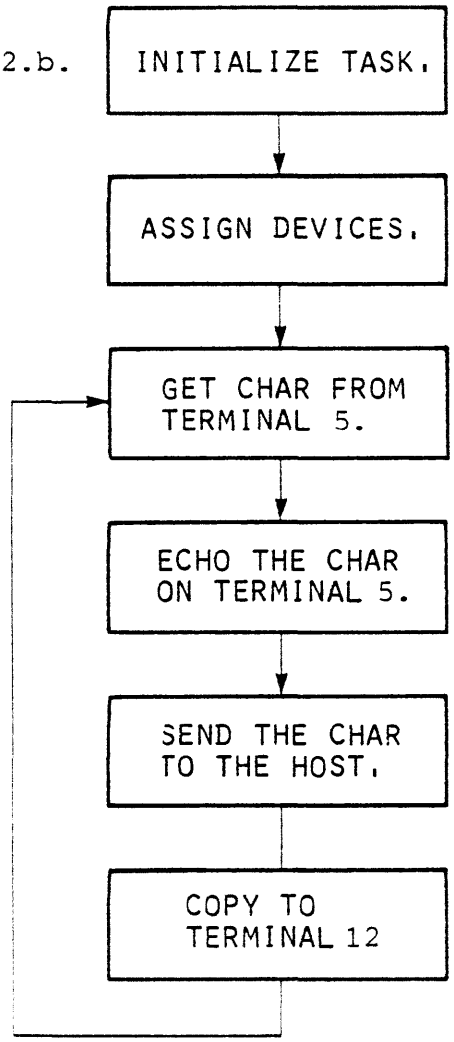
2.a.



2. Draw a flowchart of an -80 series user task (expressly including the user calls) to perform one of the following:

   b. copy all information going to the host from the keyboard of terminal 5 onto the printer of terminal 12

2.b.



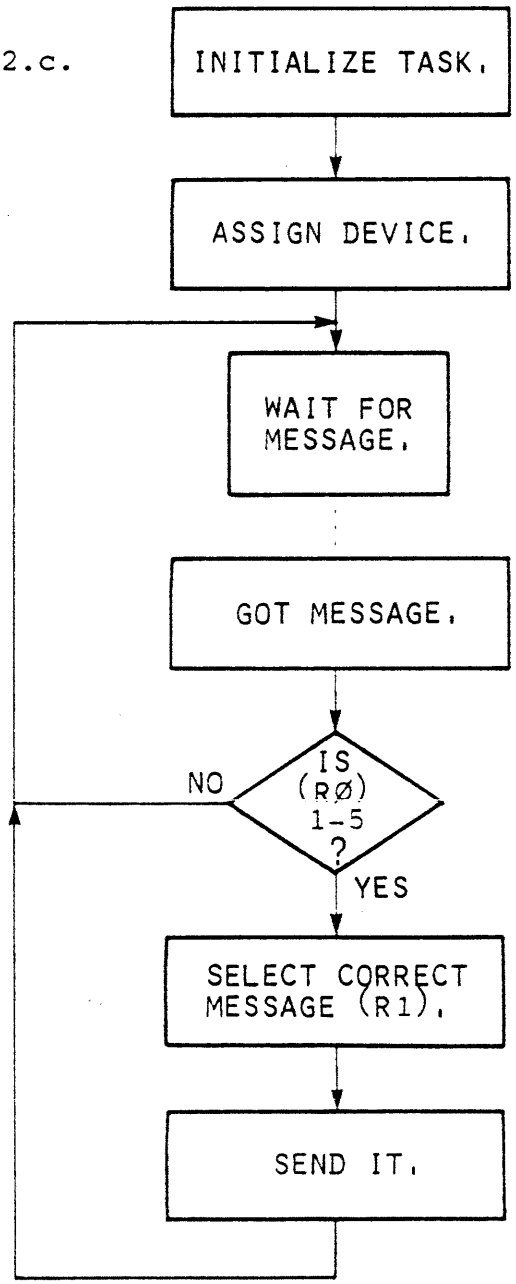

```
TSKGEN   PB2,P2,P2END,Ø,Ø.

OPEN     5,  KEYBOARD
OPEN     12, PRINTER

GET      5,  KEYBOARD,BLOCKING,RØ

PUT      5,  PRINTER,NONBLOCKING,RØ
             NOTE NON-EXCLUSIVE USE
             OF TERM 5'S PRINTER

PUT      5,  KEYBOARD,NONBLOCKING,RØ

PUT      12,PRINTER,NONBLOCKING,RØ
```

M8 0116

2. Draw a flowchart of an -80 series user task (expressly including the user calls) to perform one of the following:

   c. write a task which will send 1 of 5 messages to the host from terminal 7. This message is specified by the number obtained by a RECEIVE macro call.

2.c.

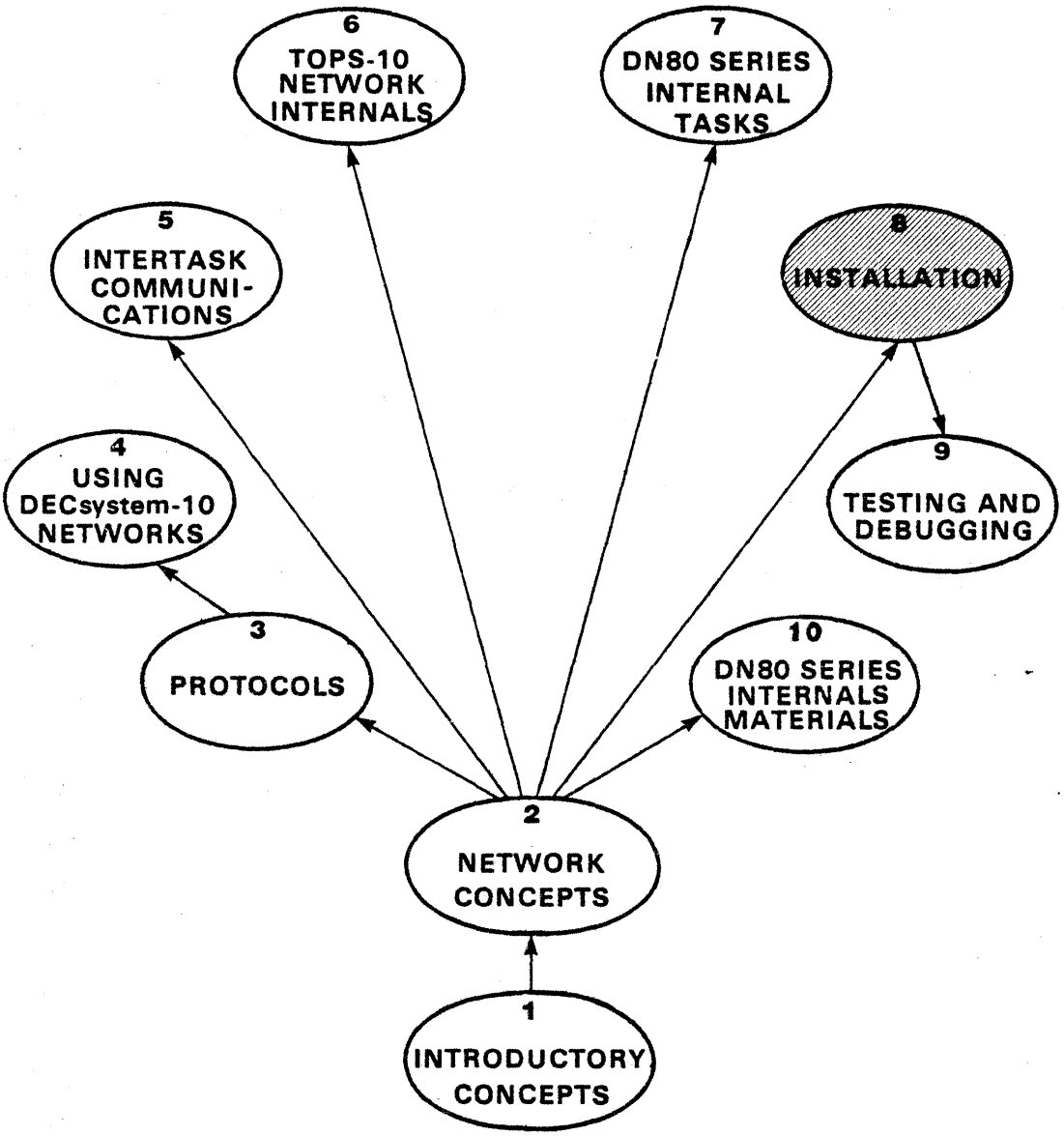



```
TSKGEN    PB3,P3,P3END,0,100
OPEN      7,KEYBOARD
RECEIVE   R0,BLOCKING
PUT       7,KEYBOARD,BLOCKING,R1
```

M8 0115

Installation
Module 8

Course Map



M8 0277

Introduction

This module details the steps necessary to assemble and load the front-end/remote stations on a DECsystem-10 network.

Objectives

Upon completion of this module, the student will be able to

1. Select the correct source modules to generate code from any specific 80 series node.
2. Select and correctly set the appropriate flags, constants, and assembly switches for that node.
3. Assemble the code.
4. Load that code into the appropriate 80 series node.
5. Set the system for automatic reloading nodes.

NOTE

The text for this module will consist of the "Networks Software Installation Guide" from the DEC-1Ø notebooks.

This page intentionally left blank

decsystem10

NETWORKS SOFTWARE INSTALLATION GUIDE

AA-5156B-TB

April 1978

OPERATING SYSTEM AND VERSION: TOPS-10 6.03 Monitor

SOFTWARE AND VERSIONS:

BACKUP	V2(216)
BOOT11	V4A(44)
DDT11	V1E(10)
DDT92	V3(7)-1
DTELDR	V1(21)
MACDLX	V27A(667)
NETLDR	V2A(110)
PAL10	V142A(143)

To order additional copies of this document, contact the Software Distribution Center, Digital Equipment Corporation, Maynard, Massachusetts 01754

digital equipment corporation · maynard. massachusetts

First Printing, July 1977
Second Printing, April 1978

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by DIGITAL or its affiliated companies.

The postage-prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECtape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-11
DECCOMM	DECSYSTEM-20	TMS-11
ASSIST-11	RTS-8	ITPS-10

CONTENTS

CONTENTS (CONT.)

FIGURES

CONTENTS (CONT.)

Page

TABLES

PREFACE

This manual is written for software installers and system operators responsible for installing DECsystem-10 networks software on communications front ends and remote nodes.

This publication does not cover the design of a network. It assumes that the network topology has already been decided upon and that the hardware and DIGITAL-supplied software is available.

This installation guide is organized as follows:

Chapter 1 provides an introduction to DECsystem-10 network configurations and an overview of the installation procedures.

Chapter 2 describes the DIGITAL-supplied software and how to copy it to your system's storage areas.

Chapter 3 describes in detail, one of the more important phases in network installation: the generation of a tailored configuration file for each communication node in the network.

Chapter 4 describes the assembly of the software and the selection of the source modules used as input to the assembly.

Chapter 5 contains the loading procedures for the communications front ends and the remote nodes.

Chapter 6 describes DDT11, a dynamic debugging tool for PDP-11 based network nodes. DDT11, a DECsystem-10 program, allows you to examine dumps of remote station memory and modify software that is currently running in a remote or local PDP-11 based node. DDT92 is used to examine dumps and modify software of a PDP-8 based DN92 remote node.

Appendix A contains a comprehensive list of all available switches; Appendix B briefly describes the notation used in this document.

Documents referenced in this manual or which might prove useful during the installation and check-out procedures are:

DECsystem-10 Monitor Installation Guide	DEC-10-OMIGA-A-D
DECsystem-10 Networks Programmer's Guide and Reference Manual	DEC-10-ONPGA-A-D
DECsystem-10 Operating System Commands	AA-0196C-TB
DECsystem-10 DDT Dynamic Debugging Technique	DEC-10-UDDTA-A-D
DECsystem-10 Software Notebooks	
BACKUP Specification	Notebook 12
BOOT11 Specification[1]	Notebook 10
DTELDR Specification[2]	Notebook 10
INITIA Specification[1]	Notebook 10
NETLDR Specification[2]	Notebook 10

[1] updated 1978

[2] added 1978

CHAPTER 1

THE DECsystem-10 NETWORK

This chapter provides a cursory overview of the DECsystem-10 network hardware and an introductory outline of the software installation procedures.

1.1 NETWORK OVERVIEW

The DECsystem-10 network is a configuration of DECsystem-10 processing systems, communications control systems (front ends), and remote stations interconnected via communications lines. The lines, also referred to as links, can be as short as a few feet when connecting a front end to its central processor, or many miles long when connecting remote stations to the central site via telephone lines or radio relay links.

The computer systems that attach to a link are referred to as nodes. Nodes are identified by a node number and a node name. The node number is a two-digit octal number, limiting the maximum number of nodes in a DECsystem-10 network to 63 (77 octal). The node name is one- to six-alphanumeric characters, of which the first character must be alphabetic.

DECnet Compatible Port software running on a DN80-series communications device allows a program running on a DECsystem-10 node to communicate with a program running on a DECnet operating system such as RSX-11M.

1.1.1 Communications Front Ends

A communications front end is a dedicated communications facility that acts as an interface between a processing system and a network. The software that runs in a communications front end is tailored to the protocol (or set of rules) governing the transfer of information in that particular network. At least one communications front end is required for each DECsystem-10 that is part of a network. A front end with asynchronous line support can relieve the host processor of a significant portion of the processing requirements for terminal support. It also reduces the amount of memory in the host that is devoted to code and storage buffers. DECsystem-10 communications control systems are the DC75NP and DN85 Synchronous Front Ends and the DN87 and DN87S Universal Synchronous/Asynchronous Front Ends (see Table 1-1).

1.1.2 Remote Stations

A remote station is a small computer system that allows access to the network from locations that are distant from a central processing system. Remote stations usually support three classes of input/output devices: terminals (hardcopy or video), line printers, and card readers. A remote station also runs software supporting the communications protocol of the network. Typical remote stations supported on a DECsystem-10 network include the DN80, DN81, DN82 and DN92 (see Table 1-1).

Table 1-1
Characteristics of Communications Nodes

Node Type	Node Usage	Asynch. Lines	Synch. Lines	Line Printer	Card Reader
DC75NP	Front End	-	1-4		
DN85	Front End	-	1-12		
DN87	Front End	0-96(Note)	0-10(Note)		
DN87S	Front End	0-112(Note)	0-12(Note)		
DN80	RJE Station	-	1-4	1	1
DN81	Concentrator	1-32	1-4		
DN82	RJE & Conc.	1-32	1-4	1	1
DN92	RJE & Conc.	1 to 16[1]	1	1	1

Note: The asynch./sync. line combinations for the DN87 and DN87S are as follows:

Maximum Asynch. Lines		Maximum Synch. Lines	
DN87	DN87S	DN87	DN87S
96	112	0	0
64	64	2	4
32	32	6	8
0	0	10	12

[1] Up to 12 if used with a card reader or line printer; up to 8 if used with both a card reader and a line printer.

1.1.3 Network Configurations

Network configurations (topologies) are generally determined by the geographical distribution of the nodes, the volume and scheduling of communications traffic, and the cost of lines and hardware.

Configurations supported by the DECsystem-10 include simple network topologies such as point-to-point, star, and multidrop. They also include more complex multilink configurations incorporating features such as route-through, multiple hosts, and dynamic topologies.

1.1.3.1 Simple Topologies - The simplest topologies contain only point-to-point connections. Figure 1-1 shows a network consisting of one direct link between the two nodes TWO and THREE.

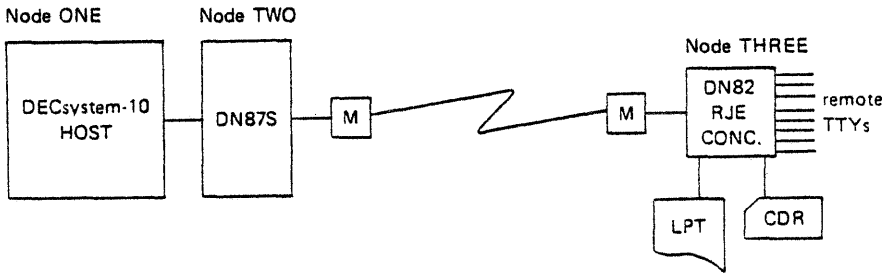


Figure 1-1 Point-to-Point Configuration

Multiple point-to-point connections, all using the same host processor, form a star configuration. Figure 1-2 shows a network consisting of nodes THREE, FOUR, and FIVE all connected to node TWO, the front-end processor for the host computer at node ONE.

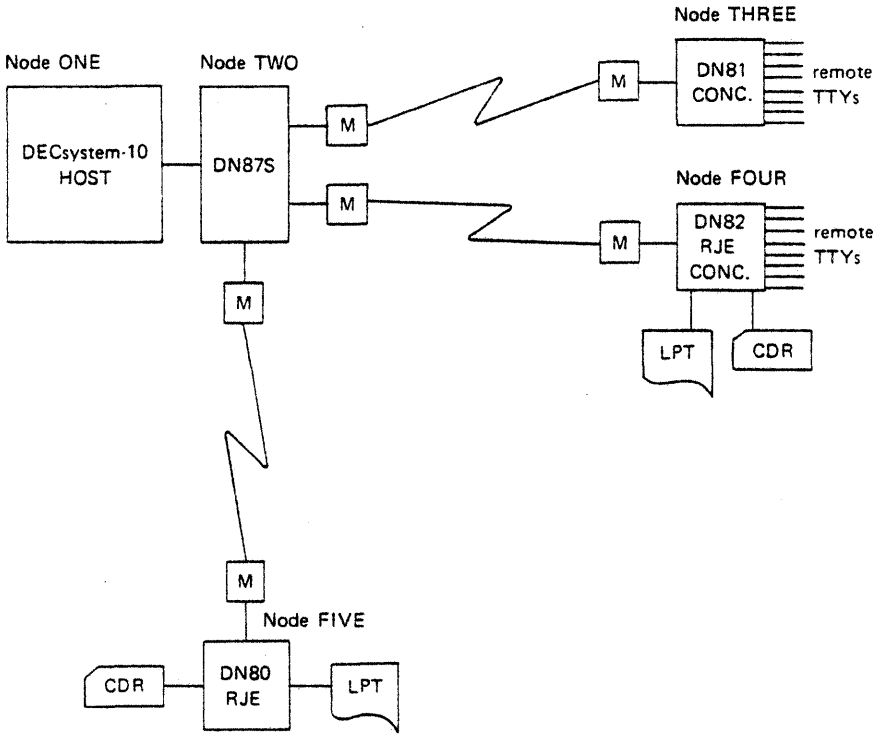


Figure 1-2 Star Configuration

A special form of a single-link network is the multidrop (or multipoint) configuration. In this network, a single link is shared by more than two nodes. Figure 1-3 shows three tributary nodes (THREE, FOUR, and FIVE) all on the same link to a control node, node TWO. In this configuration, tributary nodes can communicate only with the control node, not with each other.

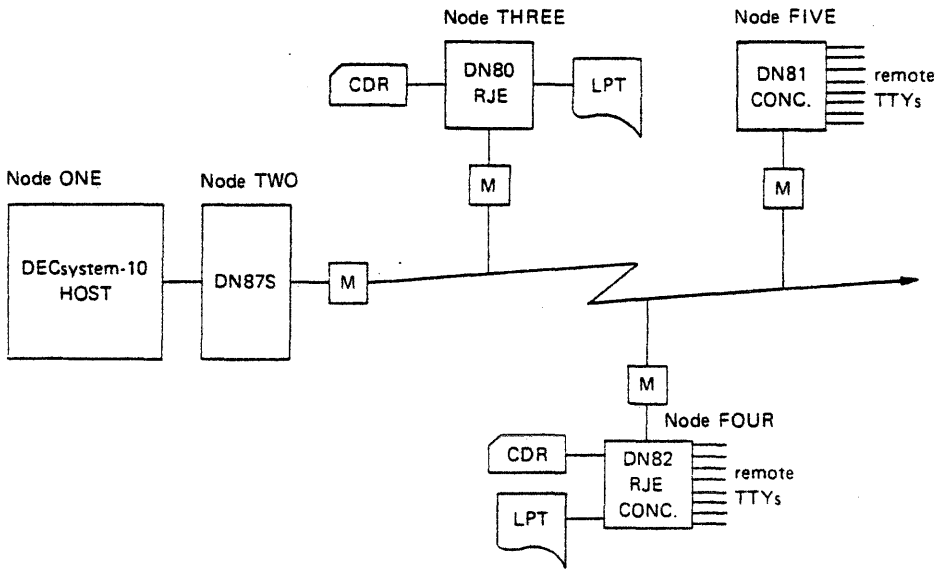


Figure 1-3 Multidrop (Multipoint) Configuration

1.1.3.2 **Complex Topologies** - Complex topologies are composed of multiple links and multiple nodes. A multilink configuration as shown in Figure 1-4 can include multiple DECsystem-10s in the same network. A multihost configuration permits the user at a terminal of a remote station to select his host with the SET HOST command. The route-through capability allows communication between two nodes that are only indirectly connected via one or more intermediate nodes (nodes FOUR and SEVEN in Figure 1-4).

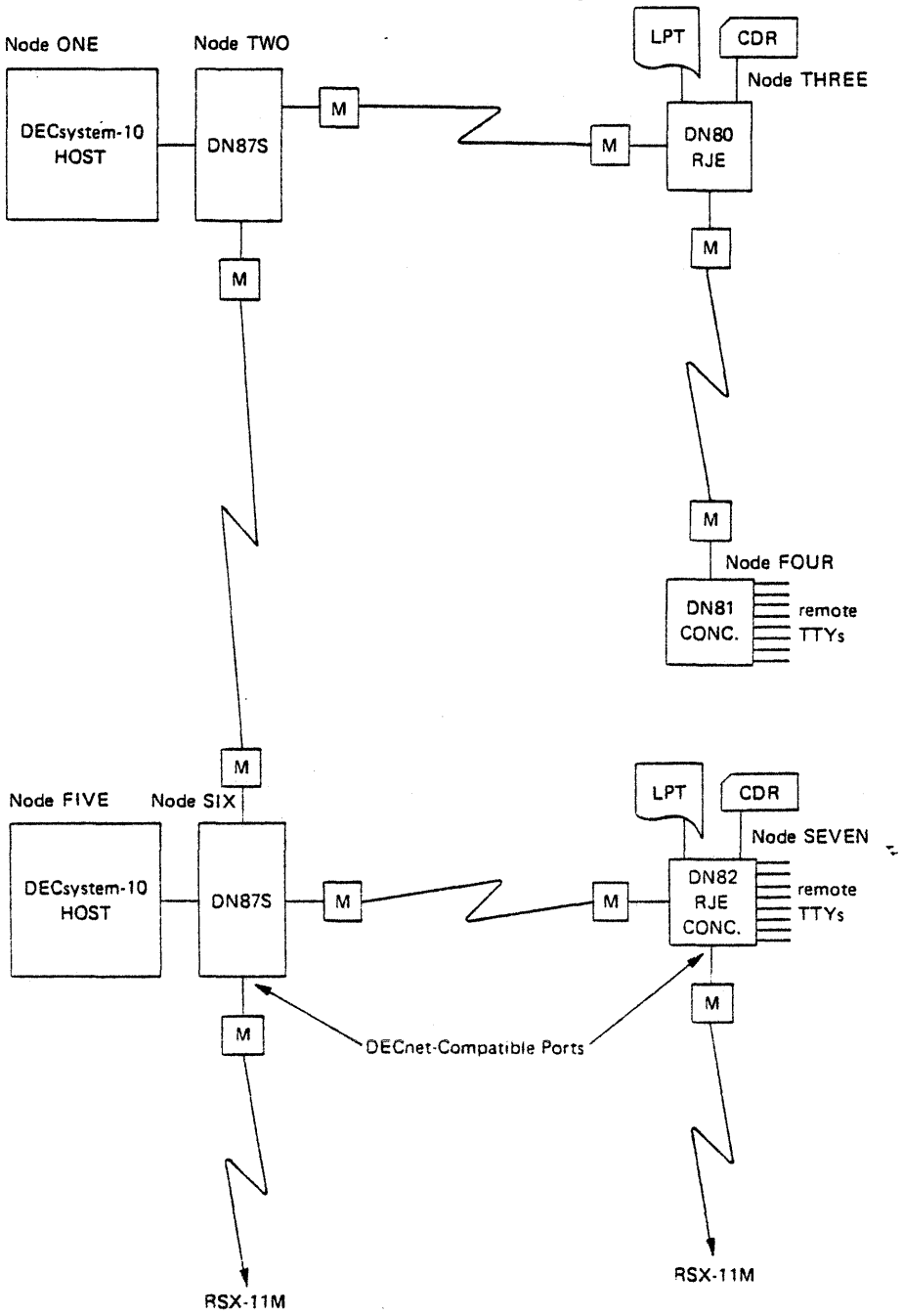


Figure 1-4 Multilink Configuration

1.2 INSTALLATION OVERVIEW

This manual describes the generation and installation of network software, summarizing briefly the installation for a DECsystem-10 host and giving detailed attention to installations on all other nodes. More complete information on software to be installed on the

DECsystem-10 host can be found in the "DECsystem-10 Monitor Installation Guide". A fully operational TOPS-10 monitor and familiarity with its use are prerequisites for using the procedures contained in the following chapters.

1.2.1 Monitor Installation Summary

When you create a new monitor to support TOPS-10 networks, observe the following guidelines:

1. Run MONGEN in its four modes (see below for suggestions):

   HDWGEN - to specify central processor hardware facilities

   TTYGEN - to specify terminal characteristics

   NETGEN - to specify network hardware

   FGEN - to specify necessary Feature Test (FT) switches.

2. Perform the following compilations with MACRO-10:

Input	Output
HDWCNF, TTYCNF, NETCNF, FCNF, S, COMMON	COMMON.REL
HDWCNF, TTYCNF, NETCNF, FCNF, S,DTEPRM, COMDEV	COMDEV.REL

   NOTE

   DTEPRM is used only with KL10 processors.

HDWCNF, TTYCNF, NETCNF, FCNF, S, NETPRM, COMNET	COMNET.REL
HDWCNF, FCNF, S, COMMOD	COMMOD.REL
FCNF, S, NETPRM,TSKSER	TSKSER.REL

   NOTE

   TSKSER is an unbundled software product needed only if task-to-task or DECnet Compatible Port capabilities are required.

3. LINK all .REL files together with TOPL10 to create the monitor core image.

4. Do a SAVE of SYSTEM.EXE to create the monitor file.

When running:	Respond positively to the following prompts:
HDWGEN	ALLOW JOBS TO BE LOCKED IN CORE (must be specified to use network) #HIGH PRIORITY QUEUES (default=0; must be greater than 0 to use HPQ switch in NETLDR.INI)
TTYGEN	LINES which RUN INITIA AT STARTUP- Such lines can also be described in TTY.INI.
NETGEN	All the available parameters.
FGEN	Specify either KAFULL, KIFULL, KLFULL to turn on all necessary switches. Other monitors do not support networks.

1.2.1.2 **INITIA Setup** - If you specify in MONGEN (TTYGEN) that certain lines are to "run INITIA at startup", you should create a file called TTY.INI on SYS:. This file uses INITIA keywords and option switches to describe how each terminal (TTY) is to be controlled (see INITIA Specification in Software Notebook 10). If TTY.INI does not exist on SYS:, INITIA assumes that no special initialization is needed and prints only the startup message.

To give options in TTY.INI for a line at a host DECsystem-10, specify the TTY line number and switches. For example, the following entries in TTY.INI give the options for TTY's on lines 13 to 15, 30, and 35 to 37.

```
TTY13-15: RCVS:150 XMTS:2400 FILL:3
TTY30: RCVS:300 XMTS:9600
TTY35-37: FILL:2 WID:72 PAGE NO TAB NO FORM NO LC
```

To give options in TTY.INI for a line at a remote node, specify the node name, TTY line number and switches. For example, the following entries in TTY.INI give the options for TTY's at remote nodes NOVA and NIM on lines 31 and 45.

```
NOVA_TTY31: SPEED 2400 PAGE
NOVA_TTY45: RCVS:150 XMTS:2400
NIM_TTY31: SPEED 2400
NIM_TTY45: SPEED 2400 PAGE NO TAB
```

> NOTE
>
> The underline (_) or backarrow (←) between the node name and the TTY line number is required.

1.2.2 Network Installation Requirements

To perform the appropriate network generation and installation procedures, you must have access to the system programs listed in Table 1-2 which apply to your system.

Table 1-2
Required System Software

Program Name	Required for Installing	Function
BACKUP	all nodes	Copies network software from the distribution tape to the system disk.
BOOT11	DC75NP,DN85,DN87	Loads network software into DL10-interfaced communications front ends.
CREF	DN92	Prepares a cross-reference listing following assembly of software.
DTELDR	DN87S	Loads network software into DTE20-interfaced communications front ends.
MACDLX	DC75NP,DN80-series	Assembles PDP-11 software on the DECsystem-10.
an editor (TECO or SOS)	all nodes	Creates configuration files, edits program text.
PAL10	DN92	Assembles PDP-8 software on the DECsystem-10.

In addition, the programs listed in Table 1-3 must be obtained from the Network Support Tape.

Table 1-3
Network Support Tape Programs

Program Name	Required for Installing	Function
NETLDR	DN80,81,82,92	Downline loads network software into a remote node.
DDT11	DC75NP,DN80-series	Allows you to examine and deposit code and data in a running PDP-11 node and to read PDP-11 core dumps that have been stored in the DECsystem-10.
DDT92	DN92	Allows you to examine and deposit code in a running PDP-8 node and to read PDP-8 core dumps stored in the DECsystem-10.
source code	all nodes	Used to assemble network software for each network node. Source files for DC75NP and DN80-series nodes are *.P11 files; for DN92 nodes, *.PAL files. See Section 2.2, "Network Source Modules", for descriptions of these source modules.

1.2.3 Installation Summary

The network software installation procedures can be grouped into the following four major functions:

1. Copy the network software files from the network support tape to your disk area. This is discussed in Chapter 2.

2. Create a configuration file for each node, reflecting the environment in which the node will operate. A description of the configuration file entries as well as selection information is contained in Chapter 3.

3. Assemble the network software for each node. A description of the assembly procedure and a list of the source files applicable to each type of node are contained in Chapter 4.

4. Load each node with its tailored software package and run an initial systems check. The load procedures vary according to whether the node is remote or local. In addition, if the node is local, loading procedures are again differentiated by the type of interface to the DECsystem-10. A description of the loading and initial system checking procedures are contained in Chapter 5.

> NOTE
>
> This system check is also done whenever a node is reloaded (but not when it is restarted manually).

During the installation of a remote node, or subsequently when the node is running, you may need some means to examine the node software. Two such facilities are the remote debugging tools, DDT11 or DDT92. These programs are discussed in Chapter 6.

CHAPTER 2

COPY THE DISTRIBUTED SOFTWARE TO DISK

The procedures in this chapter are a guide to copying the DIGITAL-supplied network software from the network support tape to your disk. The network software is used in subsequent chapters to generate customized code for each network node.

2.1 DISTRIBUTION MEDIA

The DECsystem-10 network software for the DC75NP, DN80-series and DN92 nodes is available on the:

Network Support Tape
BACKUP format (interchange mode)
800 bits/in. (31.5 rows/mm)
7 track Order no. DEC-10-OMNTA-C-MC7
or
9 track Order no. DEC-10-OMNTA-C-MC9

This tape contains the following files:

NETWRK.DIR	DNCDDP.P11	DNNCL.P11
DN9223.PAL	DNCDDS.P11	DNNSP.COR
CHK11.P11	DNCNFG.P11	DNNSP.P11
DDT11.EXE	DNCOMM.P11	DNRDE.P11
DDT11.MAC	DNCRD.P11	DNTRCE.P11
DDT11.RNO	DNCTAB.P11	DNTSK.P11
DDT92.EXE	DNDBG.P11	DNTTY.P11
DDT92.MAC	DNDCMP.P11	NETBLD.CTL
DDT92.RNO	DNDEV.P11	NETLDR.CTL
D111E.RND	DNDH11.P11	NETLDR.EXE
DN2741.P11	DNDL10.P11	NETLDR.HLP
DN9223.CTL	DNDM11.P11	NETLDR.MAC
DN92.PAL	DNDN11.P11	NETLDR.RND
DN9210.RND	DNDTE.P11	NET2A.RND
DN92.RNO	DNLBLK.P11	NETWRK.DDT
DN92.SIG	DNLPT.P11	NEWDN8.RND
DNCDDH.P11		S.P11

COPY THE DISTRIBUTED SOFTWARE TO DISK

The code needed for doing task-to-task and DECnet Compatible Port (DCP) operations is contained on a separate unbundled support tape:

```
TSKSER Support Tape
BACKUP format (interchange mode)
800 bits/in. (31.5 rows/mm)
7 track  Order no. DEC-10-CTSKA-A-MC7
           or
9 track  Order no. DEC-10-CTSKA-A-MC9
```

This tape contains the following files:

```
TSKSER.DDT
TSKSER.MAC
```

2.2 NETWORK SOURCE MODULES

When you are ready to assemble the software for each node, you must supply the appropriate assembler with an assembly file list of source modules. This section briefly describes the purpose of each source module that you received on the network support tape. Determining the particular set of modules that apply to each type of node is discussed in Section 4.1.

The following modules apply to all except DN92 nodes; all modules except DNDBG and DNTRCE must be included in each assembly. For DN92 nodes, see below.

filename.P11 is the node configuration file and is not supplied by DIGITAL; you must create this file according to the instructions in Chapter 3. The filename may be C.P11 or representative of the node (e.g.,CN8222.P11). This file must be the first specified at assembly time.

S.P11 contains the STOPCD, symbol, and macro definitions used by the network software. This file must be the second specified at assembly time.

DNCNFG.P11 processes the configuration parameters and feature test switches that you entered in the filename.P11 file. This file must be the third specified at assembly time.

DNCOMM.P11 contains common data and code such as the main loop and clock routines.

DNNCL.P11 contains the network control language routines.

DNDCMP.P11 contains the DIGITAL Data Communications Message Protocol (DDCMP) code.

DNTRCE.P11 contains the code to support the optional tracing facility. This is a field service diagnostic tool.

DNDBG.P11 contains the debugging storage blocks.

DNLBLK.P11 contains line block definitions and the CHK11 interface. This file must be the next-to-last specified at assembly time.

CHK11.P11 contains the code that performs the initial hardware check of each device present on the node. This file must be the last specified at assembly time.

The following modules are device drivers; their inclusion in the assembly file list depends on each node's configuration.

DNDL10.P11 contains interface driver code for the DL10 communications channel.

DNDTE.P11 contains interface driver code for the DTE20 communications channel.

DNCDDQ.P11 contains synchronous line driver code for the DQ11 communications interface on the DN80 series.

DNCDDS.P11 contains synchronous line driver code for the DS11 communications interface on the DC75NP.

DNCDDP.P11 contains synchronous line driver code for the DP11 communications interface on the DC75NP.

DNCDDH.P11 contains line driver code for the DH11 16-line asynchronous serial line multiplexer. This module is used when the asynchronous line uses DDCMP to communicate with RDX-type devices or other nodes.

DNDM11.P11 contains the DM11 modem control routines.

DNDH11.P11 contains the DH11 asynchronous line interface code. This module is used when communicating with TTY's and other single character devices or if using DDCMP over asynchronous lines in a network.

DNLPT.P11 contains the code for the line printer routines.

DNCRD.P11 contains the code for the card reader routines.

DNDN11.P11 contains the code to support the DN11 automatic dialing interface device.

The following modules perform other network functions; their inclusion in the assembly file list is dependent upon the node configuration and any special operating environment that may be required.

DNNSP.P11 contains the code to support the DECnet compatible port.

DNDEV.P11 contains the NCL (Network Control Language) interface code to handle device access for line printers, card readers, and terminals.

DNTTY.P11 contains the terminal routines.

DN2741.P11 contains the BCD translation tables and code to support an IBM 2741 terminal.

DNCTAB.P11 contains special character tables for TTYs and line printers.

DNRDE.P11 contains the code to support remote data entry terminals on multidrop lines.

DNTSK.P11 contains code to allow the scheduling of special purpose tasks in the PDP-11 while the remote station is running.

The following source modules for the DN92 are found on the network support tape:

DN9223.PAL contains a configuration file for a node named SNOWY numbered node 23, which has one LP05 line printer, one card reader, and 12 TTY's.

DN9223.CTL contains a batch control file that builds the DN9223.PAL configuration file and assembles the remote station software using DN9223.PAL and DN92.PAL files.

DN92.PAL contains the DN92 source program.

2.2.1 Supplementary Files

In addition to the source modules described above, a number of other files are distributed on the Network Support Tape. These files contain useful information, examples of control files used for generating networks and executable DDT11 and DDT92 files. These files are described below in alphabetical order.

File	Contents
D111E.RND	A RUNOFF input file documenting changes from DDT11 Version 7 to Version 10. This file must be run through RUNOFF to produce a D111E.DOC file for printing.
DDT11.EXE	The executable DDT11 file.
DDT11.MAC	Source code for DDT11, Version 10.
DDT11.RNO	A RUNOFF input file documenting DDT11. This file must be run through RUNOFF to produce a DDT11.MEM file for printing.
DDT92.EXE	The executable DDT92 file.
DDT92.MAC	Source code for DDT92, to examine a (PDP-8) DN92. The use of this file is described in Section 6.1.1, "DDT92", of this manual.
DDT92.RNO	A RUNOFF input file documenting DDT92. This file must be run through RUNOFF to produce a DDT92.MEM file for printing.
DN92.RNO	A RUNOFF input file documenting DN92 stations. This file must be run through RUNOFF to produce a file for printing.
DN92.SIG	RUNOFF output that can be printed. This file contains a brief description of DN92 remote station installation.
DN9210.RND	A RUNOFF input file which documents the changes between DN92 version 07 and version 10. The file must be run through RUNOFF to produce a .DOC file for printing.
DNNSP.COR	A patch to DNNSP.P11 (see the 6.03 Beware file).
NET2A.RND	A RUNOFF input file that contains changes between Version 2 and Version 2A of NETLDR. This file must be run through RUNOFF to produce a NET2A.DOC file for printing.

NETBLD.CTL The control files used to build DN8x-series software for certain systems. It serves as an example for other network-building control files.

NETLDR.CTL A control file used to create NETLDR.EXE, the executable file, from source code, NETLDR.MAC, and the intermediate relocatable file, NETLDR.REL. This creation also produces a memory map and a cross-reference (CREF) listing.

NETLDR.EXE The stored executable binary file for NETLDR.

NETLDR.HLP The NETLDR help file.

NETLDR.MAC The NETLDR source file.

NETLDR.RND A RUNOFF input file that contains a brief description of NETLDR. This file must be run through RUNOFF to produce a NETLDR.DOC file for printing.

NETWRK.DDT Patches to COMNET, NETSER and COMDEV.

NETWRK.DIR Directory of files on Network Support Tape.

NEWDN8.RND A RUNOFF input file containing new feature information for DN80-series software. This file must be run through RUNOFF to create a NEWDN8.DOC file for printing.

2.3 COPY PROCEDURE

The first step in the installation procedure is to copy the distributed software from the network support tape to your system disk. In the DECsystem-10, the area that is allocated for DIGITAL supplied software is [10,7] corresponding to ersatz device DEC:.

The network support tape is generated using the BACKUP program in interchange mode. Detailed information on BACKUP (including operator, error, and warning messages) can be found in "System Programming Procedures and Techniques" in Software Notebook No. 12. The following BACKUP command sequence can be used to copy the network support tape to DEC:.

```
.R BACKUP               ;load the BACKUP program
/TAPE MTxnnn            ;use drive nnn on magnetic tape
                        ;controller x
/INTERCHANGE            ;use interchange mode
/REWIND                 ;rewind tape to load point
/DENSITY 800            ;specify tape density (800 or 1600 bpi)
/FILES                  ;print each filename being copied
/SSNAME ALL             ;specify all save sets
/RESTORE DEC:=DSK:      ;restore to device DEC: ([10,7])
/EXIT
```

2.4 COPY EXAMPLES

In the following examples the network support tape is mounted on magnetic tape drive MTD141.

Example 1:

Print the directory of the support tape on the line printer:

```
.R BACKUP                    ;load the BACKUP program

/TAPE MTD141                 ;use drive MTD141
/PRINT DSK:                  ;create directory file on disk
!
*Done

/REWIND                      ;rewind tape to load point
/^C                          ;exit BACKUP

.PRINT BACKUP.LOG            ;print out the directory
[LPT31:BACKUP=/Seq:3597/Limit:24, 1 File]

.
```

Example 2:

Copy the entire network support tape to DEC:.

```
.R BACKUP                    ;load the BACKUP program

/TAPE MTD141                 ;use drive MTD141
/INTERCHANGE                 ;restore in interchange mode
/REWIND                      ;rewind tape to load point
/FILES                       ;print each filename copied
/RESTORE DEC:=DSK:           ;restore to device DEC:
!NETWRK DIR
CHK11    P11
DDT11    EXE
DDT11    MAC
DDT11    RNO
DDT92    MAC
DDT92    RNO
DDT92    EXE
D111E    RND
DN2741   P11
DN92     CTL
DN92     PAL
DN92     RND
DN92     SIG
DN9210   RND
DNCDDH   P11
DNCDDP   P11
DNCDDQ   P11
DNCDDS   P11
DNCNFG   P11
DNCOMM   P11
DNCRD    P11
DNCTAB   P11
DNDBG    P11
DNDCMP   P11
DNDEV    P11
DNDH11   P11
DNDL10   P11
DNDM11   P11
```

```
DNDN11   P11
DNDTE    P11
DNLBLK   P11
DNLPT    P11
DNNCL    P11
DNNSP    COR
DNNSP    P11
DNRDE    P11
DNTRCE   P11
DNTSK    P11
DNTTY    P11
NETBLD   CTL
NETLDR   CTL
NETLDR   EXE
NETLDR   HLP
NETLDR   MAC
NETLDR   RND
NET2A    RND
NETWRK   DDT
NEWDN8   RND
S        P11

*Done

/REWIND                    ;rewind tape to load point
/^C                        ;exit BACKUP

.
```

Example 3:

Verify that the RESTORE operation copies the files correctly, as part of the above copy command sequence, by replacing the /^C with the following:

```
/CHECK DEC:=DSK:           ;verify the restore operation
!
*Done

/REWIND                    ;rewind tape to load point
/^C                        ;exit BACKUP

.
```

CHAPTER 3

CREATE A CONFIGURATION FILE

One of the source modules used in the assembly of the node software is the node-specific configuration file. You must create one of these files for each node in the network that is not a host processor. Network software for a host processor is assembled during the monitor installation procedure. (See the COMNET and NETPRM modules in the DECsystem-10 Monitor Installation Guide.)

The rest of this chapter covers the selection of configuration file entries, their allowable values and defaults, and several representative configuration files.

3.1 SELECT THE FILE ENTRIES

Use any convenient editor and create a file with a name reflecting the particular node you are configuring. In this document, the SOS editor is used to create files, and the file naming convention is:

```
filename = DCttnn   (for DC75NP nodes)
         = DNttnn   (for DN8x and DN92 nodes)
```

where

tt denotes the type of node (for example, 75 for a DC75NP, 82 for a DN82 and 92 for a DN92).

nn represents the node number associated with the node.

> **NOTE**
>
> If a configuration file entry can have a value other than 0 or 1 (OFF or ON), you can enter the number in either octal or decimal. A number followed by a decimal point is taken as decimal; without the decimal point it is taken as octal. A number can also be specified as decimal by preceding it with a ^D (up-arrow D).

Not all entries are appropriate for all types of nodes. Applicable entries for each node are shown in Table 3-1. Configuration file entries are also described in Appendix A, "Configuration File Switches".

As an example, to invoke the SOS editor to create a configuration file for a DN87 with a node number of 16, enter:

```
.SOS DN8716.P11          ;invoke SOS to create a file
```

SOS will reply with the following:

```
Input: DN8716.P11        ;SOS confirms the new file
00100                    ;SOS prompts you for the first entry
```

For a DN92 node, you could create a configuration file called DN92nn.PAL.

For example:

```
.SOS DN9224
Input:  DN9224

00100  TTYN = 10        ;8 TTY's excluding the CTY
00200  CDRN =0          ;no card reader
00300  OURNNM = 24      ;node number 24
00400  $
*ES
[DSKC:DN9224]
```

If you use the distributed DN9223.PAL configuration file supplied with the software, the remote station is defined as containing one LP05 line printer, one card reader and 12 TTY's at node number 23. If you do not specify any configuration file, the software assembles a default node called DN9272 with 16 TTY lines, one card reader and one uppercase LP05 line printer.

You are now ready to generate the configuration file. The rest of Section 3.1 describes the available entries. Section 3.2 discusses macros that are either required for specific features or that will facilitate the definition of lines and terminals.

Table 3-1
Configuration File Entries by Node Type

Entries (macros)	Node Type							
	DC75NP	DN80	DN81	DN82	DN85	DN87	DN87S	DN92
OURNNM	x	x	x	x	x	x	x	o
(node name)	x	x	x	x	x	x	x	o
PDP11	x	x	x	x	x	x	x	
FT.D75	x							
FT.D80		x						
FT.D81			x					
FT.D82				x				
FT.D85					x			
FT.D87						x		
FT.87S							x	
SCBMAX	x	x	x	x	x	x	x	
NLINES	o	o	o	o	o	o	o	
TTYN			o	o		o	o	o
(TDEF)			o	o		o	o	
(DHCNFG)			x	x		x	x	
(DHUSE)			x	x		x	x	
TnnDSL*			o	o		o	o	o
TnnWID*			o	o		o	o	o
TnnXS*			o	o		o	o	
TnnRS*			o	o		o	o	
TnnTAB*			o	o		o	o	o
FT.RNN			o	o		o	o	
TnnRNN			o	o		o	o	
FT2741			o	o		o	o	
DEFBCD			o	o		o	o	
FT.RDE			o	o		o	o	
DN11N			o	o		o	o	
FTHOST			o	o		o	o	
FT.DCP**		o	o	o	o	o	o	
(NSPLST)		o	o	o	o	o	o	
FT.MPT**		o	o	o	o	o	o	
FT.CTY			o	o		o	o	
DGUTS	o	o	o	o	o	o	o	o
DEBUG	o	o	o	o	o	o	o	o
FT.TSK			o	o		o	o	
FT2BIT			o	o		o	o	
FTOLDC	o	o	o	o	o	o	o	
FTRACE	o	o	o	o	o	o	o	
CTYWID								o
CTYTAB								o
DEFINE								o
LPTN		o		o				o
LA180								o
FFLPLC								o
LPTWID		o		o				o
CDRN		o		o				o
DELROM								o
FTOLDC		o	o	o	o	o	o	o

x = required entries
o = optional entries
blank = not applicable

**one of a pair of entries: if one is zero, the other must be 1.
*can be defined by TDEF macro.

The defaults set by the option macro are shown in Table 3-2. They are also listed on the first page of the module DNCNFG.P11.

Table 3-2
Option Macro Defaults

Node	PDP-11	DL10	DTE20	SYNC I/O	NLINES	TTY's	CDR	LPT
75	15	1	0	DS11	4	0	0	0
80	40	0	0	DQ11	4	0	1	1
81	40	0	0	DQ11	4	32	0	0
82	40	0	0	DQ11	4	32	1	1
85	40	1	0	DQ11	4	0	0	0
87	40	1	0	DQ11	4	32	0	0
87S	40	0	1	DQ11	4	32	0	0

> NOTE
>
> OURNNM defaults to 72 for the DN92, 77 for DN80-series and DC75NP.

> NOTE
>
> Assembled software cannot exceed memory size of the node into which it is to be loaded (see Section 4.3, "Check Memory Size").

3.1.1 Required Entries

The following five entries are mandatory for all PDP-11 nodes.

OURNNM=nn — This is the node number declaration where nn is a two-digit octal number with a range of 01-77. Each node's number must be unique in the network.

```
.MACRO  NODE  MNAME
        MNAME <nodename>
.ENDM
```

This macro sets up the symbolic name for the node. Nodename must be unique in the network and must consist of one to six uppercase alphanumeric characters. The first character must be alphabetic.

PDP11=mn — This entry specifies the model number of the node's PDP-11 processor (e.g., 40 for a DN8x series node, 15 for a DC75NP).

FT.typ=1 This entry declares the type of node where:

typ = D75 for a DC75NP
= D80 for a DN80
= D81 for a DN81
= D82 for a DN82
= D85 for a DN85
= D87 for a DN87
= 87S for a DN87S

SCBMAX=mx This entry specifies the maximum number of nodes that the network will support. The number can be greater than the actual number of nodes to allow for expansion at the cost of currently unused storage. **It must not be less than the actual number of nodes or the system will fail erratically.** The number, mx, includes the node being configured as well as any DECsystem-10 host processors in the network.

3.1.2 Line/Terminal Entries

The following entries define the lines and terminals on each node. They are optional and default values are assigned whenever the entry applies to the type of node being configured.

NLINES=#sy This entry specifies the number of synchronous lines attached to the node. If omitted, any PDP-11 based node is built for four synchronous lines. The allowable values for NLINES varies from 0 for a DN87 or DN87S front end supporting only asynchronous lines to a maximum of 12 for DN85 or DN87S; and 10 for a DN87. This entry does not apply to the DN92 since it can have only one synchronous line.

TTYN=#as This entry determines the number of DH11-type interfaces (asynchronous lines) that the node will support by specifying the number of allowable terminals (16 terminal lines = one DH11). Code is generated for a maximum of n terminals connected to lines numbered 0 through n-1, excluding the CTY. If this entry is omitted, the default number of lines will be used. For a PDP-11 based node, the default is set in DNCNFG.P11 (see Table 3-2); the default is either 0 or 32. For a DN92 node, the default is 16. The allowable values for TTYN vary from 0 (for a DC75NP, DN80, or DN85) to 96 for a DN87 and 112 for a DN87S.

> **NOTE**
>
> Line numbers, used in the context of this publication, refer to local lines on the network nodes, and can be in the range 0 to 112. Do not confuse these local line numbers with the TTYmmm teletype numbers that appear in the configuration messages sent by a host's front end at startup time. TTYmmm numbers are assigned dynamically.

Each of the terminals declared in the TTYN entry has the following default characteristics:

- hardwired (as opposed to dataset line)
- 72 column width
- autobaud detection
- no hardware tabs
- may be assigned by any network node
- is not a 2741-type terminal

NOTE
To indicate that a terminal is a 2741-type; give its speed as 134. baud.

If you want to override any of the above defaults, use one or more of the following entries.

NOTES
Any of the following entries, of the form Tnxxx, can also be specified using generic terms. When the entry applies to the console terminal, use CTY. When the entry applies to all the lines specified in the TTYN entry, use TTYxxx.
When you specify a line number n, omit leading zeros. Line numbers can be in the range 0 to 112.

TnDSL=1 — This entry specifies that line n is a dataset line and connects to a terminal via a modem rather than being hardwired. One such entry is required for each dataset line.

TnWID=w — This entry specifies that the terminal connected to line n has a column width of w characters rather than the default of 72. One entry is required for each terminal with a non-default column width. Maximum column width is 132 columns.

TnXS=s
TnRS=s — This pair of entries is used for nodes that do not have or do not use autobaud detection. They specify the transmit speed (XS) and receive speed (RS) for an asynchronous line. Transmit speed is the speed from the node processor to the terminal; receive speed is the speed from the terminal to the node processor. The line number, n, must be in octal. The speed of the line in baud, s, is usually entered in decimal.

Acceptable line speeds (in baud) are:

50.	600.
75.	1200.
110.	1800.
*134.	2400.
150.	4800.
200.	9600.
300.	

* Use this line speed only for 2741-type terminals.

If only the transmit speed is defined, the receive speed defaults to the same value. If the terminal is a split-speed terminal, both TnRS and TnXS must be specified for that line.

If the terminal is hardwired and is always set for a particular line speed, you may want to submit these entries even if the node supports autobaud detection. If you implement the entries, you do not have to perform the steps of autobaud detection each time the terminal is to be used. This is particularly useful if it is difficult to change the line speed on a terminal, since it prevents the line speed from reverting to a low speed when the connection is broken (a break character is sent) and sets the line speed to the specified value when the node is reloaded.

TnTAB=1 This entry specifies that the terminal connected to line n has hardware tabs. One entry is needed for each such terminal.

FT.RNN=1 This entry specifies that code to support restricted terminal devices is to be generated. A restricted terminal device is one that can only be connected to the host specified for it. This entry generates only the support code; a TnRNN entry, specifying the appropriate control host, is required for each line so restricted.

TnRNN=nodenum
This entry specifies that the terminal on line n will accept a connection from the host specified by nodenum. One entry is required for each restricted terminal. If any terminal on this node is to be restricted, you must also specify an FT.RNN=1 entry.

FT2741=0 This entry specifies that code to support 2741-type terminals is not to be generated.

DEFBCD=bll. This entry specifies the default typeball for all the 2741-type terminals on this node. Acceptable values for bll. are:

938.	BCD
963.	EBCDIC
987.	APL correspondence
988.	APL (EBCDIC)

Individual 2741-type terminals on this node can use elements other than the default by invoking the SET TTY ELEMENT command.

3.1.3 Other Optional Entries for PDP11-Based Nodes

The following entries to the configuration file are optional and are used to create special operating environments, set rules of protocol, or invoke special network features.

FT.RDE=1 — This entry specifies that code to support remote data entry pseudo devices is to be generated. If this entry is not present, the code is not generated.

DN11N=1 — This entry specifies that code to support the DN11 automatic-dialing interface device is to be generated. If this entry is not present, the code is not generated.

FTHOST=0 — This entry suppresses code needed for the SET HOST command. If it is not used, code to support the SET HOST command is generated.

FT.DCP=1 — This entry specifies that code to support the DECnet-compatible port is to be generated. This feature permits the interconnection of the DECsystem-10 network with a DECnet-type network and allows tasks in one network to communicate with tasks in the other. If this entry is not present, the code is not generated.

NOTES
The DECnet-compatible port support (FT.DCP=1) and multidrop support (FT.MPT=1) are mutually exclusive on the same node.
Whenever the entry FT.DCP=1 is present, the configuration file must also contain the NSPLST macro listing all the NSP nodes accessible to this node. The NSPLST macro is described in Section 3.2.4.

FT.MPT=1 — This entry specifies that code to support multidrop (multipoint) lines is to be generated. If this entry is not present, the code will not be generated.

NOTES
If multidrop support is to be generated, you must include the source module DNCDDH.P11 in the MACDLX assembler input list.
The multidrop support (FT.MPT=1) and DECnet-compatible port support (FT.DCP=1) are mutually exclusive on the same node.

FT.CTY=1 — This entry specifies that code to support the use of the hardcopy terminal on the DL11 as a DECsystem-10 terminal is to be generated.

DGUTS=1 — This entry specifies that error recovery code is to be generated. When this code is active, the node will attempt to recover from "soft" errors. Soft errors include lack of buffer or table space and incorrect

message formats. Hard errors, such as functionally inoperative hardware, are still fatal. Note that when this recovery feature is activated, certain debugging facilities such as the ASSERT and TWIDDL macros are disabled. (See FTASRT and FTTWID in the NEWDN8.DOC file.) If this entry is not present, all errors are fatal, and no error recovery is attempted; however, full debugging facilities are available. It is recommended that this feature test switch be set off (DGUTS=0).

FT.TSK=1 — This entry generates code to support the scheduling of special purpose user tasks in the PDP-11 while the remote station is running. The effect of this support is multiprocessing within the node. If this entry is not present, the code is not generated.

FT2BIT=1 — This entry determines the minimum length of the stop bit on terminal lines operating at 300 baud or faster. The default value of 1 (ON) sets the stop bit to twice the length of a data bit. A value of zero (OFF) sets the stop bit to the length of a data bit. At line speeds under 300 baud, the minimum length of the stop bit is always twice the length of a data bit.

FTOLDC=1 — This entry specifies that this node is to be generated with the version of DDCMP protocol used prior to Release 6.03. The default value is 0 (OFF). If you are running Release 6.03 or later, do not include this entry.

FTRACE=1 — This entry specifies that TRACE macros, embedded in the network software code, are to be expanded. This feature is a field service diagnostic tool.

Other entries for PDP-11 based nodes are described in Appendix A, "Configuration File Switches".

3.1.4 Other Optional Entries for DN92 Nodes

The following entries for a DN92 configuration file are optional but serve to define parameters for a line printer, a card reader and to specify certain special conditions.

Entry	Meaning
OURNNM=nn	Node number declaration; nn is a two-digit octal value 01 to 77. Each node number in the network must be unique. The default for this entry is 72.
DEFINE DN92ID <nodename>	Node name declaration. The nodename can be one to six uppercase alphanumeric characters. Each character must be preceded by a double quote (") and separated from others by a semicolon (;). The first character must be alphabetic. The default for this entry is DN92.

Entry	Meaning
LPTN=0	No line printer.
LPTN=1	LP05 line printer.
LA180=1	LA180 line printer.
FTLPLC=0	Printer is uppercase only.
FTLPLC=1	Printer is both uppercase and lowercase.
LPTWID=204	Printer has 132 column line width.
LPTWID=120	Printer has 80 column line width.

> NOTE
>
> With LPTN=1, defaults are 132 column, uppercase only; with LA180=1, defaults are 132 column, both uppercase and lowercase. If no printer entry is made in the configuration file, defaults are LP05, 132 column, uppercase.

CDRN=0	No card reader.
DELROM=1	System should print an error message and halt if an error occurs in the DN92; if this line is not placed in the configuration file, the DN92 is restarted from its ROM.
FTOLDC=1	This entry specifies that this node is to be generated with the version of DDCMP protocol used prior to Release 6.03. The default value is 0(OFF). If you are running Release 6.03 or later, do not include this entry.

3.1.4.1 DN92 Configuration File Defaults - If no configuration file is specified for a DN92 remote station assembly, the software automatically assembles a remote node with number=72, 16 TTY's, one card reader and one 132 column, uppercase only LP05 line printer. The default switches are summarized below.

Switch	Default
OURNNM=	72
DEFINE DN92ID< >	DN92
TTYN=	20(8) (16.TTY's)
TnnWID=	72
CTYWID=	72
LPTN=	1
LA180=	0
FTLPLC=	0
LPTWID=	204(8) (132.columns)
CDRN=	1

3.2 CONFIGURATION DEFINING MACROS FOR PDP-11 BASED NODES

There are four macros that can be included in the configuration file. The TDEF, DHCNFG, and DHUSE macros are used in the configuration of asynchronous lines and terminals. The NSPLST macro is used whenever a node is to support one or more DECnet-compatible ports.

3.2.1 The TDEF Macro

The TDEF macro is an alternative way of defining the line/terminal entries of the form Tnxxx=value described in Section 3.1.2. It is applicable only if the node includes asynchronous terminals assigned to DH11 lines. It is useful only if you are defining terminals with other than default attributes. The form of the macro is:

```
TDEF index, <list>
```

where

index — is a symbol whose value represents the local line number n (octal) being defined. This symbol should be initially set equal to the first line to be defined. Each TDEF macro increments the index value for the next TDEF macro.

list — is a list of entries of the form xxx or <xxx,value> where

- xxx — is the two- or three-character identifier in the Tnxxx entry.
- value — is the value to be assigned to the Tnxxx entry. If value is omitted, the entry is given the value 1.

For example,

```
TINDX=4
TDEF TINDX,<<RS,2400.>,<WID,80.>,TAB>
TDEF TINDX,<DSL,<RNN,10>>
```

defines the following entries:

```
T4RS=2400.
T4WID=80.
T4TAB=1
T5DSL=1
T5RNN=10
```

3.2.2 The DHCNFG Macro

A DHCNFG macro definition in the configuration file (C.P11) defines the attributes of DH11 lines by using calls to the TDEF macro. Note the following example.

```
.MACRO DHCNFG
TINDX=6                                 ;set index to 6
TDEF TINDX,<<RS,150.>,<XS,2400.>>       ;line 6
TDEF TINDX,<<RS,150.>,<XS,2400.>>       ;line 7
TDEF TINDX,<<RS,150.>,<XS,2400.>>       ;line 10 (octal)
TINDX=12                                ;set index to 12
TDEF TINDX,<<RS,2400.>>                 ;line 12
TDEF TINDX,<<RS,2400.>>                 ;line 13
TINDX=50                                ;set index to 50
TDEF TINDX,<<RS,300.>>                  ;line 50
TDEF TINDX,<<RS,300.>>                  ;line 51
TDEF TINDX,<<RS,300.>>                  ;line 52
TDEF TINDX,<<WID,80.>,TAB,DSL>          ;line 53
DHUSE (NTT,NAL,NMPT,TRIB)               ;see Section 3.2.3
.ENDM
```

The above macro entries define the following lines:

Line No.	Non-Default Attributes
6	split baud, 150 receive/2400 transmit
7	split baud, 150 receive/2400 transmit
10	split baud, 150 receive/2400 transmit
12	2400 baud, receive and transmit
13	2400 baud, receive and transmit
50	300 baud, receive and transmit
51	300 baud, receive and transmit
52	300 baud, receive and transmit
53	dataset line, hardware tabs, and width of 80 columns

The combination of the TDEF and DHCNFG macros allows you to change the assignment of terminals to DH11 lines by merely reordering the macro calls.

3.2.3 The DHUSE Macro

Whenever you have asynchronous lines attached to a node, you must have a DHUSE macro defined within the DHCNFG macro. This is true even if all the lines and terminals are to assume the default characteristics. The format of the DHUSE macro is as follows:

```
DHUSE (NTT,NAL,NMPT,TRIB)
```

where

NTT — is the number of terminals; this argument can be specified as TTYN to use the defaults given in Table 3-2, "Option Macro Defaults".

NAL — is the number of point-to-point DDCMP lines.

NMPT — is the number of multipoint DH11 lines.

TRIB — is the number of multipoint tributary DH11 lines.

All but the first argument apply only to DDCMP asynchronous lines and are usually set to 0.

For example, a node with 64 (decimal) terminals, all taking default characteristics, can be defined in the configuration file with the following entries:

```
.MACRO DHCNFG
DHUSE (100,0,0,0)          ;number of terminals in octal
.ENDM
```

The TTYN entry (Section 3.1.2) is not required; by default, it becomes equal to the first argument in the DHUSE macro.

3.2.4 The NSPLST Macro

Whenever a node is configured to support DECnet-compatible ports, you must include the NSPLST macro in the configuration file. The general form of the macro is as follows:

```
.MACRO NSPLST

NSP arg1,arg2,...arg8     ;define first DECnet-compatible port
NSP arg1,arg2,...arg8     ;define next DECnet-compatible port
.
.
.

.ENDM NSPLST
```

An NSP line definition is required for each DECnet node line with which this node will be communicating. The NSP entry has a rigid format and consists of eight arguments, all of which are required. The following is a detailed description of each of the eight arguments.

arg1 is the number of the synchronous line on this node to which the NSP node is to be connected. While NCL nodes can be connected to any available synchronous line, NSP nodes must only be connected to lines explicitly generated for them. The acceptable values for arg1 are 0 through NLINES-1.

arg2 is the node number of the NSP node to be connected to the line specified in arg1; it is used to identify this NSP node within the NCL network. While it has no correlation to the node number assigned to the NSP node in the DECnet system, it will probably avoid confusion to set arg2 to the NSP node's DECnet number, so long as the number does not conflict with an existing network node in the DECsystem-10 network.

arg3 is the NSP node's node name and is used by the DECsystem-10 as an alternate to the node number. The argument can be a maximum of six characters and, if punctuation is included, must be enclosed in angle brackets. The relation of NCL to NSP node name is the same as for node number in arg2; using the same name avoids confusion.

arg4 is the identification of the software that is running on the NSP node. It is part of the information returned to you when you issue the NODE command. The value of this argument is an ASCII string usually identifying the operating system and its version number. Use angle brackets to enclose the

argument if it includes embedded blanks or punctuation; for example, <RSX11M V1>.

The default length is 14 for a DN8x node in a DECsystem-10 network. To change the length, include the following entry in the configuration file:

```
SIDSIZ=len      ;len = new argument length
```

If arg4 is longer than SIDSIZ, the excess is ignored.

arg5 is the date associated with the entry in arg4. It is part of the information returned to you when you issue the NODE command. The value of this argument is an ASCII string and is enclosed in angle brackets if embedded blanks or punctuation are included; for example, <21 JUN 77>.

The default length is 10 for a DN8x node. To change the length, include the following entry in the configuration file:

```
DATSIZ=len      ;len = new argument length
```

If arg5 is longer than DATSIZ, the excess is ignored.

arg6 is the node number of the NCL node to which the NSP messages will be sent; this is usually a DECsystem-10 node. The rationale behind this is that only task-to-task communication can occur between NCL and NSP nodes. A DECsystem-10 node is the only type of NCL node supporting task-to-task software.

arg7 is the maximum data message length that the NSP node will accept. This must be the same value as that specified for the NSP node when it was generated in DECnet. The value must also be less than or equal to the maximum data message length of any DECsystem-10 node in the network.

arg8 is the device configuration of the NSP node. It is coded as a list of pairs, the first member of each pair being an NCL object type and the second member of the pair being the number of those objects available at the NSP node. Each pair in the list is enclosed in angle brackets with a comma between members. If the list includes more than one pair, each pair is separated by commas and the list is enclosed in angle brackets.

For example, if the NSP node can accept up to four simultaneous logical link connections for tasks, as well as a line printer (another task under NSP, but a device as far as the DECsystem-10 is concerned), the appropriate entry for arg8 would be:

```
<<3,1>,<11,4>>
```

The interpretation of the above entry is one line printer (object type 3) and four tasks (object type 11). The object types supported by NCL are:

1	Terminal
2	Card Reader
3	Line Printer
11	Task

3.3 SAVE THE CONFIGURATION FILE

When you have entered all the applicable entries to the configuration file, save it on your disk area. In Chapter 4, it becomes part of the assembly command string. If the editor is one that provides line numbering, such as SOS, store the file without the line numbers. For example, the file that was opened in Section 3.1 should be closed with the following command:

```
*ES      ;save the file without line numbers.
```

3.4 EXAMPLES OF CONFIGURATION FILES

The following examples are included to provide you with reference material as you generate your specific configuration files.

Example 1

A DN87 front end at node 16 to support one synchronous line and a maximum of 64 asynchronous lines for local terminals. Some of the local terminals are to have other than default characteristics.

```
.SOS DN8716.P11
Input: DN8716.P11
00100    OURNNM=16
00200    .MACRO  NODE    ARG
00300            ARG     <EJWMN>
00400    .ENDM
00500    PDP11=40
00600    FT.D87=1
00700    SCBMAX=25
00800    NLINES=1
00900    FT2741=0
01000    FTHOST=1
01100    FT.CTY=1
01200    FT.RNN=1
01300    TTYRNN=14
01400    TTYN=100
01500    .MACRO DHCNFG
01600    TINDX=5
01700    TDEF TINDX,<<RS,150.>,<XS,2400.>> ;LINE 5
01800    TDEF TINDX,<<RS,150.>,<XS,2400.>> ;LINE 6
01900    TDEF TINDX,<<RS,150.>,<XS,2400.>> ;LINE 7
02000    TINDX=12
02100    TDEF TINDX,<<RS,150.>,<XS,2400.>>
02200    TDEF TINDX,<<RS,150.>,<XS,2400.>>
02300    TINDX=17
02400    TDEF TINDX,<<RS,150.>,<XS,2400.>>
02500    TDEF TINDX,<<RS,150.>,<XS,2400.>>
02600    TINDX=23
02700    TDEF TINDX,<<RS,150.>,<XS,2400.>>
02800    TDEF TINDX,<<RS,150.>,<XS,2400.>>
02900    TDEF TINDX,<<RS,150.>,<XS,2400.>>
03000    TINDX=36
03100    TDEF TINDX,<<RS,150.>,<XS,2400.>>
03200    TINDX=47
03300    TDEF TINDX,<<RS,150.>,<XS,2400.>>
03400    TDEF TINDX,<<RS,2400.>,<XS,2400.>>
03500    TDEF TINDX,<<RS,2400.>,<XS,2400.>>
03600    TINDX=53
03700    TDEF TINDX,<<RS,2400.>,<XS,2400.>>
03800    TINDX=55
```

```
03900   TDEF TINDX,<<RS,2400.>,<XS,2400.>>
04000   TDEF TINDX,<<RS,2400.>,<XS,2400.>>
04100   TDEF TINDX,<<RS,2400.>,<XS,2400.>>
04200   TDEF TINDX,<DSL>
04300   TDEF TINDX,<DSL>
04400   TDEF TINDX,<DSL>
04500   TDEF TINDX,<DSL>
04600   TINDX=74
04700   TDEF TINDX,<<RS,2400.>,<XS,2400.>>
04800   DHUSE(TTYN,0,0,0)
04900   .ENDM
05000   $
*ES
```

.

Example 2

A DN82 remote station at node 22 to support three synchronous lines and 32 asynchronous lines for remote terminals. Some of the terminals are 2741-type and are usually equipped with an EBCDIC type ball.

```
.SOS DN8222.P11
Input: DN8222.P11
00100   OURNNM=22
00200   .MACRO  NODE    MNAME
00300           MNAME   <CTCH22>
00400   .ENDM
00500   PDP11=40
00600   FT.D82=1
00700   TTYN=40
00800   SCBMAX=25
00900   FTHOST=1
01000   FT.RNN=1
01100   TTYRNN=26
01200   FT2741=1
01250   T10XS=134.
01300   DEFBCD=963.
01400   NLINES=3
01500   .MACRO  DHCNFG
01600   DHUSE(TTYN,0,0,0)
01700   .ENDM
01800   $
*ES
```

.

Example 3

A DN87 front end at node 27 to support the default of four synchronous lines. Two of the synchronous lines are DECnet-compatible ports.

```
.SOS DN8727.P11
Input: DN8727.P11
00100   OURNNM=27
00200   .MACRO  NODE    MNAME
00300           MNAME   <NEXT>
00400   .ENDM
00500   PDP11=40
00600   FT.D87=1
00700   SCBMAX=25
00800   FTHOST=1
00900   FT.CTY=1
01000   FT.DCP=1
```

```
01100    TTYN=0
01200    .MACRO  DHCNFG
01300    DHUSE(TTYN,0,0,0)
01400    .ENDM
01500    .MACRO NSPLST
01600    NSP 40,75,ML75,RSX11M,<11/76>,26,192,<<2,1>,<3,1>,<11,4>>
01700    NSP 41,60,PK60,RSX11M,<11/76>,26,192,<<2,1>,<3,2>,<11,3>>
01800    .ENDM
01900    $
*ES
```

Example 4

A DC75NP front end at node 15 to support two synchronous lines.

```
.SOS DC7515.P11
Input: DC7515.P11
00100    OURNNM=15
00200    .MACRO  NODE    MNAME
00300            MNAME   <IT>
00400    .ENDM
00500    FT.D75=1
00600    PDP11=15
00700    NLINES=2
00800    SCBMAX=25
00900    $
*ES
```

Example 5

A DN92 remote station at node 44 to support eight TTY's and one LP05 line printer (the $ used to exit from SOS is the echo of the ESCape (ALTMODE) key).

```
.SOS

FILE: DN9244.PAL
INPUT:  DN9244.PAL
00100    TTYN=10
00200    OURNNM=44
00300    DEFINE DN92ID<'N;'E;'W;'N;'O;'D>
00400    CDRN=0
00500    LPTN=1
00600    $
*ES

[DSKC:DN9244.PAL]
```

CHAPTER 4

ASSEMBLE THE SOFTWARE

The software for the PDP-11 based nodes is assembled using the MACDLX assembler. MACDLX runs on the DECsystem-10 and generates PDP-11 code (see Sections 4.1 through 4.4). Software for DN92 nodes (PDP-8 based) is assembled using PAL10 on the DECsystem-10. The installer can run PAL10 and CREF or use the DN92.CTL file from the DN92 Distribution Tape (see Section 4.5).

4.1 SELECT ASSEMBLY SOURCE MODULES FOR PDP-11 BASED PROCESSORS

All the assembly source modules, except for the configuration file, will be located in the [10,7] area if the procedure described in Section 2.3 has been followed. A short description of each module can be found in Section 2.2.

The selection of specific modules depends upon the devices attached to the node, the type of communication lines, the protocol to be followed, and any special network features. Table 4-1 provides a cross-reference of modules to the type of network nodes. For any particular type of node, some modules are required, some are optional, and others are not applicable.

The inclusion of any module noted as being optional in Table 4-1 is dependent upon one or more feature test switches specified in Table 3-1 or on specific hardware components. The relationships are as follows:

Module	Must be included if:
DNNSP.P11	FT.DCP=1
DNCDDS.P11	the DC75NP has a DS11 synchronous line interface
DNCDDP.P11	the DC75NP has a DP11 synchronous line interface
DNCDDH.P11	FT.RDE not zero, or using DH lines in a point-to-point configuration
DN2741.P11	FT2741=1
DNDN11.P11	DN11N=1
DNRDE.P11	FT.RDE=1
DNTSK.P11	FT.TSK=1
DNTRCE.P11	FTRACE=1
DNDH11.P11	TTYN not zero
DNCTAB.P11	TTYN or LPTN not zero
DNDM11.P11	FT.DM11=1(default)

Table 4-1
Assembly Modules By Node Type

Module	Type Of Node						
	DC75NP	DN80	DN81	DN82	DN85	DN87	DN87S
C.P11	x	x	x	x	x	x	x
S.P11	x	x	x	x	x	x	x
DNCNFG.P11	x	x	x	x	x	x	x
DNCOMM.P11	x	x	x	x	x	x	x
DNNCL.P11	x	x	x	x	x	x	x
DNNSP.P11		o	o	o	o	o	o
DNDEV.P11		x	x	x		x	x
DNDCMP.P11	x	x	x	x	x	x	x
DNDL10.P11	x				x	x	
DNDTE.P11							x
DNCDDQ.P11		x	x	x	x	x	x
DNCDDS.P11	o						
DNCDDP.P11	o						
DNCDDH.P11			o	o		o	o
DNDM11.P11			o	o		o	o
DNDH11.P11			x	x		x	x
DNTTY.P11			x	x		x	x
DN2741.P11			o	o		o	o
DNCTAB.P11		x	x	x		x	x
DNDN11.P11			o	o		o	o
DNRDE.P11			o	o		o	o
DNLPT.P11		x		x			
DNCRD.P11		x		x			
DNTSK.P11			o	o		o	o
DNTRCE.P11	o	o	o	o	o	o	o
DNDBG.P11	o	o	o	o	o	o	o
DNLBLK.P11	x	x	x	x	x	x	x
CHK11.P11	x	x	x	x	x	x	x

x These modules are required for this type of node.
o These modules are optional for this type of node.
blank These modules do not apply to this type of node.

4.2 SPECIFY ASSEMBLY OUTPUT FOR PDP-11 BASED PROCESSORS

The general MACDLX command string for assembling the node software is:

```
*dev:binfile.ext,dev:lstfile.ext/CRF=dev:srcfile.ext/switches,...
```

where

dev: is the physical or logical device on which the input or output files are or will be located. If dev: is omitted, DSK: is assumed.

binfile.ext is the output binary program file. If .ext is omitted, .BIN is assumed.

lstfile.ext is the output program listing file. If .ext is omitted, .LST is assumed.

/CRF — is the switch that includes cross-reference information in the listing file. The CREF program is then used to format the listing before printing.

srcfile.ext... — are the input source module files. The first three and last two modules shown in Table 4-1 must be in the order listed. Other modules are entered between these two groups and may be in any order.

The output binary program file is used when loading the node from the host processor. (See Chapter 5.)

The output program listing file with cross-reference information is used to generate a file-specific DDT11 file for on-line checkout and testing of node software. (See Chapter 6.)

4.3 CHECK PROGRAM SIZE FOR PDP-11 BASED PROCESSORS

At this point, it is advisable to ensure that the program you have just assembled will fit in the memory space available on the node into which it is to be loaded.

If it does not fit into the available node memory space, you must reassemble your modules, omitting some. You can check this either by examining the terminal output of your assembly at your terminal, or by looking at your output listing. For example, in Example 1, Section 4.4, the following terminal output occurs when assembly is complete:

```
12453   057512  000014              508.2  .PRINT Q          ;
size of program in octal K words
15142   073260  000017                4.8  .PRINT  CHKSIZ    ;
Size of program in octal K words.

ERRORS DETECTED:  0
```

The third value on the third line (000017) indicates the amount of memory the assembled program modules will occupy. This value must not exceed the memory in the node into which the software is to be loaded. Standard node memory sizes are listed in Table 4-2, Node Memory Size.

Table 4-2
Node Memory Size

Node	Memory (words)	
	Decimal	Octal
75	8K	10K
80,81,82, 85,87	16K	20K
87S	28K	34K

For the example cited above, the assembled software will fit in an 80-series node, but not in a DC75NP.

On the last page of the CHK11.P11 assembler output listing, immediately prior to the cross-reference table, the size of the assembled program is given in octal K words (1K=4096 words). You can also find the size in the cross reference table under CHKSIZ=. For example:

```
CHKSIZ=   000025
```

means that the program is 21K (decimal) words in length. This value also types out on your terminal as part of the assembly statistics.

4.4 EXAMPLES FOR PDP-11 BASED PROCESSORS

Example 1

Assemble the software for the DN87 front end at node 16. The configuration file, DN8716.P11, is from Example 1 of Section 3.4.

```
.R MACDLX

*DSK:DN8716,DN8716.CRF/CRF=DN8716.P11,S.P11,DNCNFG.P11,DNCOMM.P11
,DNNCL.P11,DNDEV.P11,DNDCMP.P11,DNDL10.P11,DNCDDQ.P11,DNDM11.P11,
DNDH11.P11,DNTTY.P11,DNCTAB.P11,DNTRCE.P11,DNDBG.P11,DNLBLK.P11,C
HK11.P11

 12453  057512  000014                    508.2  .PRINT  Q        ;
size of program in octal K words
 15142  073260  000017                      4.8  .PRINT CHKSIZ    ;
Size of program in octal K words.

 ERRORS DETECTED:  0

 *DSK:DN8716,DN8716.CRF/CRF=DN8716.P11,S.P11,DNCNFG.P11,DNCOMM.P1
1,DNNCL.P11,DNDEV.P11,DNDCMP.P11,DNDL10.P11,DNCDDQ.P11,DNDM11.P11
,DNDH11.P11,DNTTY.P11,DNCTAB.P11,DNTRCE.P11,DNDBG.P11,DNLBLK.P11,
CHK11.P11
 RUN-TIME: 54 48 14 SECONDS
 CORE USED:  42K


*^C

.
```

Example 2

Assemble the software for the DN82 remote station at node 22. The configuration file, DN8222.P11, is from Example 2 of Section 3.4.

```
.R MACDLX

*DN8222,DN8222.CRF/CRF=DN8222.P11,DSK:S.P11,DNCNFG.P11.DNCOMM.P11
,DNNCL.P11,DNDEV.P11,DNDCMP.P11,DNCDDQ.P11,DNDM11.P11,DNDH11.P11,
DNTTY.P11,DN2741.P11,DNCTAB.P11,DNLPT.P11,DNCRD.P11,DNLBLK.P11.CH
K11.P11

 13033  060670  000015                    263.7  .PRINT  Q       ;
size of program in octal K words
 15722  074410  000020                      4.8  .PRINT CHKSIZ   ;
Size of program in octal K words.

 ERRORS DETECTED:  0

 *DN8222,DN8222.CRF/CRF=DN8222.P11,DSK:S.P11,DNCNFG.P11,DNCOMM.P1
1,DNNCL.P11,DNDEV.P11,DNDCMP.P11,DNCDDQ.P11,DNDM11.P11,DNDH11.P11
,DNTTY.P11,DN2741.P11,DNCTAB.P11,DNLPT.P11,DNCRD.P11,DNLBLK.P11,C
HK11.P11
 RUN-TIME:  44 46 12 SECONDS
 CORE USED:  40K


*^C
.
.
```

Example 3

Assemble the software for the DN87 front end at node 27. The configuration file, DN8727.P11, was generated in Example 3 of Section 3.4.

```
.R MACDLX

*DN8727,DN8727.CRF/CRF=DN8727.P11,DSK:S.P11,DNCNFG.P11,DNCOMM.P11
,DNNCL.P11,DNNSP.P11,DNDEV.P11,DNDCMP.P11,DNDL10.P11,DNCDDQ.P11,D
NTTY.P11,DNCTAB.P11,DNTRCE.P11,DNDBG.P11,DNLBLK.P11,CHK11.P11

 13690  050052  000013                    456.6  .PRINT  Q       ;
size of program in octal K words
 16399  062446  000015                      4.8  .print chksiz   ;
Size of program in octal K words.

 ERRORS DETECTED: 0

 *DN8727,DN8727.CRF/CRF=DN8727.P11,DSK:S.P11,DNCNFG.P11,DNCOMM.P1
1,DNNCL.P11,DNNSP.P11,DNDEV.P11,DNDCMP.P11,DNDL10.P11,DNCDDQ.P11,
DNTTY.P11,DNCTAB.P11,DNTRCE.P11,DNDBG.P11,DNLBLK.P11,CHK11.P11
 RUN-TIME:  34 33 10 SECONDS
 CORE USED:  36K


*^C

.
```

Example 4

Assemble the software for the DC75NP front end at node 15. The configuration file DC7515.P11 was generated in Example 4 of Section 3.4.

```
.R MACDLX

*DC7515,DC7515.CRF/CRF=DC7515.P11,S.P11,DNCNFG.P11,DNCOMM.P11,DNN
CL.P11,DNDCMP.P11,DNDL10.P11,DNCDDS.P11,DNTRCE.P11,DNDBG.P11,DNLB
LK,P11,CHK11.P11

  9543  024704  000006                        67.5  .PRINT  Q       ;
size of program in octal K words
 12232  036372  000010                         4.8  .PRINT CHKSIZ   ;
Size of program in octal K words.

 ERRORS DETECTED:  0

 *DC7515,DC7515.CRF/CRF=DC7515.P11,S.P11,DNCNFG.P11,DNCOMM.P11,DN
NCL.P11,DNDCMP.P11,DNDL10.P11,DNCDDS.P11,DNTRCE.P11,DNDBG.P11,DNL
BLK.P11,CHK11.P11
 RUN-TIME:  24 24 7 SECONDS
 CORE USED:  29K


*^C

.
```

4.5 ASSEMBLING DN92 SOFTWARE

The PDP-8 assembler PAL10 can be found on the TOPS-10 CUSP tape if it is not already on your system. It must be run to assemble DN92 software.

To run the assembler, you must have:

DN92.PAL	DN92 source modules.
PAL10	The PAL10 assembler.

You can also have:

a configuration file	You can use DN9223.PAL from the distribution tape if it serves your purposes; otherwise, create your own or alter DN9223.PAL to reflect your installation. Without a configuration file, a default node is assembled (see Section 3.1.4.1, "DN92 Configuration File Defaults").
CREF	The TOPS-10 cross-reference program should be on your system. Without the CREF program, you cannot obtain a cross-reference listing.

ASSEMBLE THE SOFTWARE

To assemble the software, specify an input string to PAL10 in the form:

```
output-file.BIN,CREF-file/C=configuration-file.PAL,DN92.PAL
```

File extensions should be as indicated. You cannot run the cross-reference program unless you specify a CREF-filename.

> **NOTE**
>
> A maximum configuration .BIN file occupies about 150 disk sectors; the .CRF file, about 900.

The following example illustrates a successful assembly of the DN92 software. It illustrates use of the C.PAL configuration file, creation of a binary file called DN9224.BIN and an output file that can be used to run CREF. It is useful to print the CREF file (as shown in the example) to have a cross-referenced listing for your records. This file is also needed if you wish to run DDT92. Underlined entries are user input; all other entries are output by the system.

Step 1: Run the PAL10 assembler.

```
.R PAL10
*DN9224.BIN,DN9224/C=C.PAL,DN92.PAL
.ERRORS DETECTED:  0
 LINKS GENERATED:  629
 RUN-TIME:  21 SECONDS
 6K CORE USED
*^C
```

To exit from the PAL10 assembler, use CTRL/C.

Step 2: Run the cross-reference program:

```
.R CREF

*DSK:=DN9224.CRF
[CREFXKC 22K core]
*^C
```

To exit from CREF, use CTRL/C.

When CREF is run, it creates a file called DN9224.LST which can be printed.

Step 3: (optional) Print the CREF output file:

```
.PRINT DN9224.LST
[LPT31:DN9224= /Seq:4562/Limit:433, 1 File]
```

Once this sequence has been completed, you are ready to downlineload the DN92.

4.5.1 Submitting a DN92 Control File

The DN92 distributed software contains a control file called DN92.CTL which can be used to perform the above assembly automatically.

To use DN92.CTL, use the SUBMIT command:

```
.SUBMIT DN92.CTL
[INP31:DN92= /Seq:114/Time:00:05:00
```

This control file automatically generates DN9223.BIN and DN9223.LST. It uses the DN9223.PAL configuration file which specifies:

```
node number 23;
one card reader;
one LP05 line printer;
12 TTY lines.
```

When the control file has been executed, it creates a DN9223.LOG file which can be examined to verify the assembly.

Once the control file has executed correctly, you are ready to downline load the DN92.

CHAPTER 5

LOAD THE SOFTWARE

This chapter covers the loading of network software into the following types of network nodes:

- local DL10-interfaced nodes
- local DTE20-interfaced nodes
- remote nodes

Installation of software on hosts is covered in the DECsystem-10 Monitor Installation Guide. The operation of and output from the hardware test programs CHK11 (for PDP-11 based nodes) and SYSCHK (for PDP-8 based remote nodes) are also described. CHK11 is invoked whenever a PDP-11 based node is loaded and started; SYSCHK, whenever a PDP-8 based node is loaded and started. These programs perform initial hardware surveys of the node to ensure that node devices and node memory are functioning correctly.

All network nodes are loaded from a DECsystem-10 host. To load a remote node, all intervening nodes between it and the host-node must be running their respective software. Local nodes (communications front ends) are loaded by either BOOT11, if a DL10 interface is used, or DTELDR, if a DTE is used. Remote nodes are loaded by NETLDR. NETLDR loads assembled software through a communications front end (for example, a DN87S) down a specified synchronous line to a remote station (see Figure 5-1). This is called downline loading.

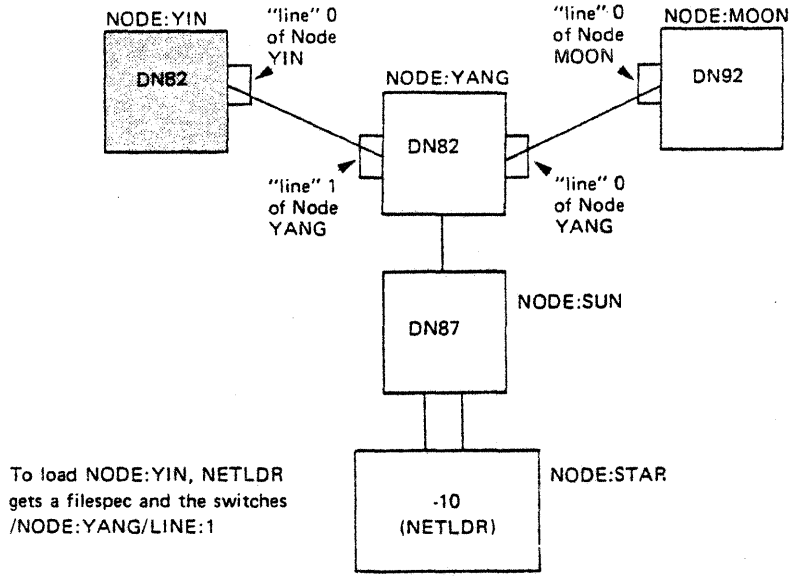


Figure 5-1 A Typical Network Configuration

5.1 LOAD CODE FOR A LOCAL NODE

Local nodes in a DECsystem-10 network are any of the communications front ends attached to the host DECsystem-10's. The loading procedure for these nodes varies according to the node's interface to the host. DC75NP's, DN85's, and DN87's are attached via the DL10 communication channel; the program used to load these nodes is BOOT11. The DN87S is attached to the host via the DTE20 interface; the program used to load a DN87S is DTELDR.

5.1.1 Load Over a DL10 (BOOT11)

To load a communications front end over the DL10, use the BOOT11 program found in the system SYS: area.

Start BOOT11 by typing:

```
.R BOOT11
```

TOPS-10 will load and start BOOT11 which will then prompt you with:

```
FILE:
```

As a response, BOOT11 expects a standard DECsystem-10 file descriptor followed by one or more of the following switches:

/CLEAR:addr — Zero PDP-11 memory from 0 to addr-1. If :addr is not present, it defaults to all of core.

/START:addr — Load the PDP-11 from the specified DECsystem-10 file and start the -11 at the octal address addr. If :addr is not present, the default is the starting address of the file that was loaded. If :addr is present, it must be an even octal value.

/PORTNO:p — Load the PDP-11 that is attached to port number p on the DL10. (DL10 0 has its ports numbered 0 through 3; DL10 1 has its ports numbered 4 through 7.) BOOT11 requires that the port number be specified if the total number of ports on this DECsystem-10 is greater than one.

The following response to BOOT11's prompt will zero all memory of the PDP-11 on port number 2 and will then load and start the PDP-11 with the program located in file DSK:DN8527.BIN.

```
FILE:DN8527.BIN/CLEAR/START/PORTNO:2
```

If you reply to the BOOT11 prompt with a carriage return, the following default file descriptor and switches are assumed:

```
FILE:DSK:PDPXI0.BIN/START/PORTNO:0
```

BOOT11 keeps you informed of its progress with the following messages:

```
"CLEARING PDP-11     (PDP-11 being zeroed)

"PDP-11 LOADING      (Loading in process)

"PDP-11 LOADED       (Loading completed)

"PDP-11 STARTED      (PDP-11 now executing)
```

Additional information on BOOT11 (including progress, warning, and error messages) can be found in the BOOT11 specification in the Software Notebooks 10.

5.1.2 Load Via a DTE20 Interface (DTELDR)

To load a communications front end via the DTE20 interface, use the DTELDR program found in the system SYS: area.

Start DTELDR by typing:

```
.R DTELDR
```

TOPS-10 will load and start DTELDR which will then prompt you with:

```
*
```

DTELDR expects you to reply with a file specification (or assume a default), an action switch, and optional modification switches. The default file specification is:

```
DSK:DTELxy.BIN
```

where

x is the CPU number (only 0 is allowed in the current implementation). If x is omitted, 0 is assumed.

y is the DTE number (0 through 3) of the PDP-11 interface to the DECsystem-10. Note that DTE0 is reserved for the Console/Diagnostic Processor.

The action switch to load a communications front end is:

```
/RELOAD:xy
```

where x and y have the same definitions as above. RELOAD dumps the appropriate front end and then loads it with the specified file and starts the primary protocol. The front end is then enabled for communications.

Modification switches that may be applicable are:

/IMAGE Specifies that the .BIN file is in image format; that is, the formatted binary output is unpacked (one 8-bit frame per DECsystem-10 word.) The default is packed format or four 8-bit frames per DECsystem-10 word. The output of MACDLX is packed format.

/NODUMP Cancels the dump of PDP-11 memory that is automatically taken when /RELOAD loads a file. The file used for a dump is DSK:DTEDxy.Bzz where zz is the first unused file in the sequence IN, 00, 01, ..., 99.

/NOLOG Cancels the automatic error logging that is in effect whenever you load a front end.

To load a DN87S on DTE number 2, with a program located in DSK:DN8721.BIN, and skip the automatic dump, answer the prompt with:

```
*DN8721.BIN/RELOAD:2/NODUMP
```

If the program has been located in the default location DSK:DTEL02.BIN, you need only type:

```
*/RELOAD:2/NODUMP
```

Another action switch available with DTELDR is /AUTO.

/AUTO Automatically reloads a crashed front end with its unique default file (DTEaxy.Bzz). With the /AUTO switch, the default load device is SYS: and the default dump device is XPN:;a is L (load) or D (dump) and x, y and zz are all as defined above.

To take advantage of the /AUTO feature, you must ensure that the default files contain the appropriate programs.

For example:

DSK:DTEL01.BIN must contain the program for the number 1 front end.

DSK:DTEL02.BIN must contain the program for the number 2 front end.

By including the following sequence in the OPR.ATO file, you can provide for both the automatic loading of a front end at DECsystem-10 initialization time, as well as the subsequent automatic reloading of any front end that might crash.

```
:SLOG                       ;log on a subjob
:DEF DTE=                   ;name it DTE
DTE-R DTELDR                ;load and start DTELDR
DTE-/INIT:1                 ;start communication with front end #1.
DTE-/RELOAD:2/NODUMP        ;LOAD DTE number 2
DTE-/AUTO                   ;set automatic reload
```

For additional information on DTELDR, refer to the DTELDR specification in Software Notebook 10 or the DTELDR.HLP file.

5.2 LOAD A REMOTE NODE (NETLDR)

NETLDR is the DECsystem-10 utility which is used to downline load remote nodes (DC75NP, DN80, DN81, DN82 and DN92). If the network support tape (distribution tape) was copied according to the instructions of Chapter 2, NETLDR will be available in the area [10,7] of the system disk. For NETLDR to run properly, there must be a bootstrap program running in the remote node. The detailed discussion of downline loading is divided into three sections: activating the remote node, automatically executing NETLDR in response to a load request from the remote station, and manually running NETLDR from the host system. The DN92 remote station requires NETLDR Version 2A(100) or later.

5.2.1 Activate the Remote Node

Activating the remote node consists of powering up the unit and starting the bootstrap program that is stored in the bootstrap ROM. While these are two distinct operations in a PDP-11 node, the bootstrap ROM in a DN92 node starts automatically when the power is turned on. The remote PDP-11 based node may be equipped with either a BM873 or an M9301 ROM. Consult your field service engineer to determine which is present on any particular node.

5.2.1.1 **Activate the Remote PDP-11 Node** - If the remote system contains a BM873 bootstrap ROM, enter 173000 into the address switches. Press the LOAD ADDRESS and START switches and the bootstrap program will begin sending load requests to the node connected to line 0. The load requests contain the node's type and serial number.

If node THREE in Figure 5-2 contains a BM873 ROM, a load request for a PDP-11 node, serial number 0 will be sent out on line 0. If line 0 of node THREE is inoperable, the node cannot be loaded.

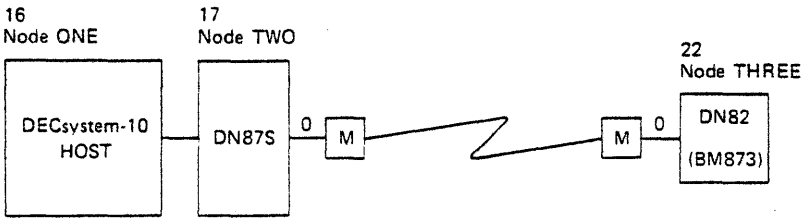


Figure 5-2 Loading a BM873-Equipped Remote Node

If the remote system contains an M9301 ROM, enter 173002 into the address switches, and press the LOAD ADDRESS and START switches. The bootstrap program will perform a carriage return and a line feed, and then will wait for operator input. If there is no TTY action, the system probably has a BM873 ROM. Enter 173000 into the address switches, press LOAD ADDRESS and START, and refer to Section 5.2.2 or Section 5.2.3.

For PDP-11 based nodes, operator input consists of two parts: M9301 switches to the ROM program and a command string to be forwarded to NETLDR. Since the NETLDR command string can also contain switches, all M9301 switches must be entered first, before any part of the NETLDR input.

In the case where M9301 switches are immediately followed by NETLDR switches, the two types must be separated by some character other than carriage return, control character, or rubout. A carriage return is echoed as a carriage return line feed and then ignored. A control character or rubout deletes the current switch (but previous correctly entered switch values remain in effect). The first carriage return after any NETLDR input terminates the console operation; the ROM then begins to send load requests to an adjacent node.

There are three M9301 switches:

/Lsyn# tells the ROM to transmit the load requests on DQ11 line syn#. If either syn# or the entire switch is not present, the load requests are sent out on line 0.

/Nnn tells the ROM to designate node nn as the system (usually a DECsystem-10) to receive the load requests. In a multihost network, this switch will determine which host is to perform the down-line load. If either nn or the entire switch is not present, the node accepts loading from any host.

/Sser tells the ROM to override the default serial number in the load request with the number ser. If either ser or the entire switch is omitted, the serial number defaults to zero.

If you issue a carriage return in place of entering M9301 switches or a NETLDR command string, the load request will be the same as for a BM873 node. It will specify a PDP-11 node, serial number 0, and will be sent on line 0. If line 0 is inoperable, the node will not be loaded.

However, the M9301 switches give you substantial flexibility regarding alternate load paths. (See Figure 5-3.) If line 0 on node THREE is inoperable, you can route the load request to node FOUR by specifying the M9301 switch:

/L1

Node FOUR converts the load request to a station control message, adds its own node name, adds the number of the synchronous line leading back to the node requesting the load, and then forwards the message to the host at node SIX.

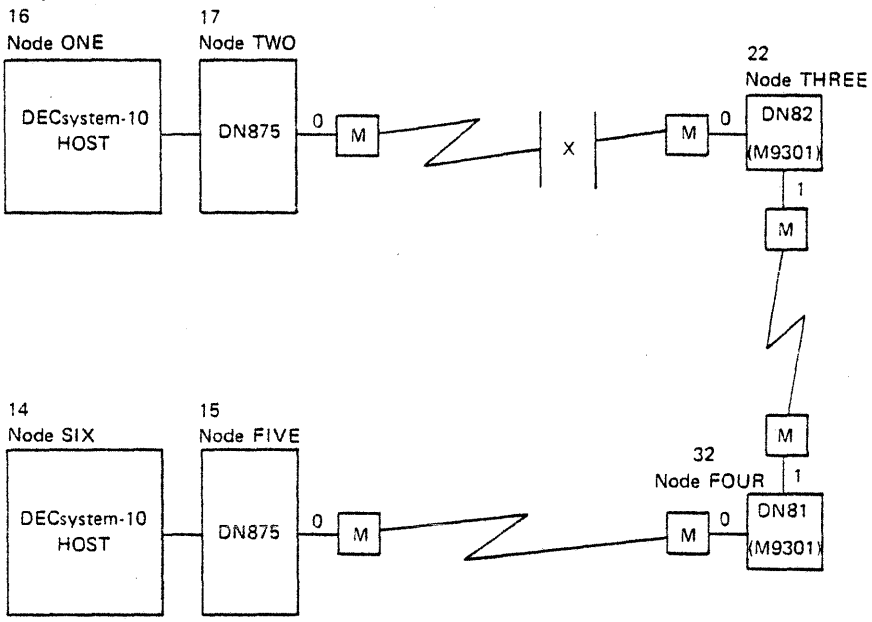


Figure 5-3 Loading an M9301-Equipped Remote Node

If the network consists of more than one host, the /N switch allows you to specify the host that is to do the loading.

If the network contains multidrop nodes, you can use the M9301 switch /S to identify which node on the synchronous line is requesting the load.

For example, the following operator input generates a load request; this request specifies that node 10 is to load the remote node with the program DIAG1.BIN:

```
/N10DIAG1.BIN/LOAD
```

Blanks between switches or commands are ignored.

If the NETLDR command string is included, it has the following format:

```
filespec/optswitch/optswitch...
```

where

filespec — is the dev:filename.ext[p,pn] designation of the file to be loaded into the remote node. If a filespec is specified, filename must be present; the rest of the filespec defaults to a .BIN file on your user disk area.

The optional NETLDR switches that are applicable in bootstrap messages are as follows:

/IMAGE — specifies that the file to be read is in image (unpacked) mode. This is the default mode for a PDP-8 node.

/LOAD — specifies that NETLDR is to load a file into the remote node but not start the program.

/PACKED — specifies that the file to be read is in packed mode (four 8-bit frames per DECsystem-10 word). This is the default mode for a PDP-11 node.

/START:addr — specifies that the program that was just loaded is to be started at address addr. If the command string did not include a filespec, NETLDR starts the program that is currently in memory. This switch is not necessary if filespec is present and if the program to be loaded is to start at its default start address.

5.2.1.2 **Activate the Remote PDP-8 Node** - If the remote node is a DN92, its ROM bootstrap program automatically begins execution when the station is turned on. As a network node that needs to be loaded, the DN92 sends a 'load request' over its synchronous line. The node which receives the request then transmits the request to the DECsystem-10. Once the DECsystem-10 has received the request, it automatically runs NETLDR (unless the SCHED bit has been set to 1000 - see below, Section 5.2.2). NETLDR then loads the DN92 station.

As the request is sent and downline loading occurs, the RUN and BUS lamps on the DN92 operator's console are lit and the ADDRS and DISP indicators change. If these indicators do not change after a minute or two, downline loading has not occurred and the station will not be operable. When the "%" appears, the ROM has begun execution and is waiting for input from the operator's console. If no operator input occurs within one minute, the ROM issues the default load request.

Messages that appear as successful loading occurs represent initialization and the host system prompt. For example:

```
[INITIALIZING NETWORK DN92 V0.7 NODE "DN92"]
%%TTY NO XMT FLAG
      .
      .
      .
RS300 KL10 SYS#1242 14:37:36
```

Any messages about TTY's with "NO XMT FLAG" indicate that the specified device is not connected to the DN92 processor. Once the system message and period (.) have appeared, the terminals are connected and it is time to log in to the DECsystem-10.

> NOTE
>
> You can also start the DN92 using the DN92 operator's console by pressing the keys:
>
> <INIT>
>
> 70<LXA>
>
> 0<LA>
>
> <INIT>
>
> <RUN>
>
> (See the DN92 User's Guide for more information about these keys.)

You can use three switches with the DN92 ROM that are similar to those available for the M9301 ROM.

These switches are:

Switch	Meaning
/Nnn	Specifies node nn as the host to receive the load requests.
/Rd	Specifies that program execution is to begin at location d in remote station memory (d is an octal value of 1 to 5 digits).
/Sser	Specifies the serial number to be sent in the load request as q. This is the serial number of the node to be loaded.

5.2.2 Invoke NETLDR Automatically

When a remote node is activated, the bootstrap program in the ROM sends load requests over one of its synchronous lines to an adjacent node (the adjacent node must be running). A load request contains the type of node to be loaded and, optionally, the number of the host node to do the loading as well as a NETLDR command string.

The adjacent node converts the load request into an NCL station control message and adds its node name and the number of the line that leads back to the node to be loaded. The station control message is then sent to the host processor. If SCHEDule bit 1000 is not set, the DECsystem-10 then automatically loads and runs the NETLDR program. The DECsystem-10 operating system command, SET SCHED 1000, inhibits the automatic running of NETLDR.

If the original load request included a file specification, NETLDR will load that file into the specified node directly. If no file is present, NETLDR returns an error message:

```
?NETFSM-File-spec missing
```

If NETLDR decides that the request does not contain sufficient information, it searches the SYS:NETLDR.INI file for an entry matching the load request information supplied.

Each entry in the NETLDR.INI file consists of two parts separated by an equal sign. The left half of each entry is of the form:

```
/NODE:nodeid/LINE:syn#/TYPE:type/SERIAL:ser
```

where

/NODE:nodeid	specifies the node name or node number of the node adjacent to the node to be loaded.
/LINE:syn#	specifies the number of the synchronous line on the adjacent node that leads to the node to be loaded.
/TYPE:type	specifies the type of node to be loaded. Use DN82 for DN80-series nodes. Use DN92 for DN92 stations.
/SERIAL:ser	specifies the serial number of the node to be loaded. This switch can be used if the remote node has an M9301 ROM or is a DN92 remote station. It applies only if the /S switch has been used with the ROM (see Section 5.2.1.1 "Activate the Remote PDP-11 Node" for an M9301 ROM, or Section 5.2.1.2, "Activate the Remote PDP-8 Node" for a DN92).

If the contents of the left half contain less information than the load request, the system matches according to the information provided. The right half of each entry is of the form:

```
=command1,command2...
```

where

command	is a file specification and/or optional switches as described in Section 5.2.3.

Assume, for example, that the NETLDR.INI file at the host Node SIX in Figure 5-3 contains the following entry:

```
/NODE:FOUR/LINE:1/TYPE:DN82=DN8222
```

A load request from node THREE, routed through node FOUR, would match on the above entry. NETLDR would then load the node connected to line 1 of node FOUR with the file SYS:DN8222.BIN. (For a load file, SYS: is the default device and .BIN is the default file extension.)

If there is no match in NETLDR.INI, the following message appears:

```
?NETNMI - Cannot find match in SYS:NETLDR.INI
```

In this case, the NETLDR.INI file should be corrected.

In a multihost network, any host may receive the load request sent by a BM873 ROM (or an M9301 ROM started at 173000). If the host receiving the request does not have access to the appropriate load file the node will not get loaded. You can provide for host discrimination by controlling its receptiveness to load requests, either globally or specifically.

For the automatic downline loading of any DN92 node, SYS:NETLDR.INI must contain a line of the following form:

```
/TYPE:DN92=DN9224.BIN
```

If the software has a name other than DN9224.BIN, that name should be used; if the software is not on SYS:, the [ppn] of the appropriate area must be added to the file specification. For example, the following line in NETLDR.INI specifies that any DN92 node that sends a load request to NETLDR will be loaded with the file DN9224.BIN:

```
/TYPE:DN92=DN9224.BIN
```

To get an automatic upline dump of the remote station before downline loading, insert a /DUMP switch before the command file specification. For example, the following line in NETLDR.INI will upline dump memory of any DN92 remote note onto the user's area on the DECsystem-10:

```
/TYPE:DN92=/DUMP,DN9224.BIN
```

The dumped file is called nodename.LSD.

If a host is to disregard all load requests, the system operator can inhibit the running of NETLDR by typing the operator-privileged command:

```
.SET SCHED 1000
```

If, however, only requests from certain nodes are to be disregarded, there is a NETLDR switch, /IGNORE, that applies only to NETLDR.INI file entries. For example:

```
/NODE:NODED/LINE:1=/IGNORE
```

will disregard all requests to load a node connected to line 1 of node NODED.

```
/NODE:NODED=/IGNORE
```

will disregard all requests to load any node connected to node NODED.

5.2.3 Operator Use of NETLDR

The NETLDR program can be run from the host node to dump, load, and execute programs in a remote node. A dump can be taken of a running node, but a node can only be loaded if it has sent an upline load request to the host. Load the NETLDR program:

```
.R NETLDR
```

The program prompts you with:

```
FILE:
```

The program expects you to reply to the prompt with a command string of the form:

```
FILE:filespec/CPUtype/NODE:nodeid/LINE:syn#/optswitch/optswitch...
```

where

filespec	is of the form dev:filename.ext[p,pn]. If this is a load operation, filename is required. The other arguments default to a .BIN file on your user disk area. If this is a dump operation, the entire file specification can be omitted and will default to nodename.LSD on your disk area. If the node to be dumped is running (/SELF is included), nodename is the name of the node itself. If the node is not running (/LINE:linenum is included), nodename is the name of the adjacent node.
/CPUtype	is a required switch entry specifying the type of processor at the remote node. For the DN80 series enter /PDP11 or /11; for a DN92 remote station, enter /PDP8 or /8.
/NODE:nodeid	is a required switch entry used to locate the node to be operated on. The switch value, nodeid, can be either a node name or a node number. If the remote node is running, nodeid identifies the node itself. If the remote node is not running, the usual case in a load operation, nodeid identifies an adjacent node that is running.
/LINE:syn# /SELF	is a required switch entry, used with the /NODE switch to identify the node being operated on by NETLDR. If the remote node is not running, the usual case in a load operation, syn# specifies the synchronous line on the adjacent node that leads to the node being loaded. If the remote node is running, the case when requesting a dump, replace the /LINE switch with /SELF to denote the operation is for the node itself.

The following NETLDR switches are optional; some are applicable to load operations, others to dump operations.

/DUMP /DUMP:c-d /DUMP:c-*	creates an octal dump of a specific portion of the remote node's memory. The three forms of the switch imply a full memory dump, a dump between the limits c and d, and a dump from some specified address c to 17777 for a PDP-8 or 77776 for a PDP-11.

If the command string does not specify a filespec, the default filespec for the dump file is nodename.LSD on DSK:. (See filespec above for a definition of nodename.)

/IMAGE — specifies that the file to be read is in image (unpacked) mode. This is the default mode for a node specified as CPUtype /PDP8 or /8.

/LOAD — specifies that NETLDR is to load a file into the remote node but not start the program.

/PACKED — specifies that the file to be read is in packed mode (four 8-bit frames per DECsystem-10 word). This is the default mode for a PDP-11 node.

/START:addr — specifies that the program that was just loaded is to be started at address addr. If the command string did not include a filespec, NETLDR starts the program that is currently in memory. This switch is not necessary if filespec is present and if the program to be loaded is to start at its default start address.

5.2.4 Examples

To illustrate the features of NETLDR, the following examples for loading a node are provided.

To load the file NYC.BIN from DSKN area [14,16] on the host node into the node connected to line 0 of node BOSTON and start the program at address 1000, issue the following string to NETLDR in response to the "File:" prompt:

```
DSKN:NYC[14,16]/NO:BOSTON/LI:0/11/START:1000
```

To dump locations 0 to 77776 of node BOSTON onto DSK: as a file named BOSTON.LSD, issue the following string to NETLDR in response to the "File:" prompt:

```
File:  /DUMP/NODE:BOSTON/SELF/PDP11
```

5.3 INITIAL HARDWARE CHECK (CHK11) FOR PDP-11 BASED NODES

The last source module assembled into the node's software in Chapter 4 was CHK11.P11. CHK11 is a hardware test module for PDP-11 based nodes and is brought in and started whenever a node is loaded. CHK11 runs hardware diagnostics on the node's memory and devices to see that they are functioning correctly. The names and quantities of each device type are printed on the system's console terminal if one is present. Representative CHK11 output for a successful load of a DN87 front end at node 16 could be:

```
INITIALIZING DN87 V11(23) APR 1977 - DN8716 (16)
   100000 BYTES OF MEMORY
     MF11-UP
     KW11-L
     KG11-A
     DL10
   4 DM11-BB'S
   4 DH11'S
   1 DQ11   SKIPPING SPEC CHAR TEST
```

```
RESTARTING DN87 V11(23) APR 1977 - DN8716 (16)

   RX220F SYS #514/546 5:23:18
```

Similar output for a DN82 remote station at node 22 could be:

```
Initializing DN82 V11(23) APR 1977 - CTCH22 (22)

   100000 BYTES OF MEMORY
     MF11-UP
     KW11-L
     KG11-A
   NOT CHECKING DL10
   1 CR11
CR11 not Rdy
   1 LP11
LP11 #0 Not Rdy
   2 DM11-BB's
   2 DH11's
   2 DQ11's   SKIPPING SPEC CHAR TEST
Restarting DN82 V11(23) APR 1977 - CTCH22 (22)

   RX220F SYS #514/546 12:31:03
```

If a problem is detected by CHK11, the error information is printed in the following format:

```
?  devnam#devnum(ADR=devaddr)
  ERROR AT srcaddr
    (message describing problem)
    REG/ADR=addr GD=expval BD=actval XOR=varbits
    [FATAL ERROR}
```

where

devnam	is the device name.
devnum	is the device number. Device numbers are sequential, starting with 0.
devaddr	is the first address of the device.
srcaddr	is the address in the source listing where you should look for this error.
addr	is the real address in memory where the error occurred.
expval	is the expected value in addr.
actval	is the actual value in addr.
varbits	are the bits resulting from the EXCLUSIVE OR operation on the expected and actual values.

The message FATAL ERROR is printed if the error is such that CHK11 is unable to continue (memory errors or KW11 clock errors). When CHK11 stops on a fatal error, a stop code is displayed in the console data lights. Table 5-1 lists the error stop codes and their meanings.

Table 5-1
CHK11 Error Stop Codes

Stop Code	Meaning
1	Time-out or bus error (trap to address 4)
2	DL10 error
5	No console terminal
6	Memory error
7	KW11 clock error

For additional information about CHK11 and a complete list of CHK11 error messages, refer to DECsystem-10 Networks Programmer's Guide and Reference Manual.

5.4 INITIAL HARDWARE CHECK FOR THE DN92

When the DN92 is downline loaded by NETLDR, SYSCHK is executed. SYSCHK performs a cursory check of the DN92 hardware and issues messages at the DN92 operator's console.

> NOTE
>
> SYSCHK is not executed when the DN92 is restarted manually.

SYSCHK execution takes about a minute. A SYSCHK message may be preceded by either %% or ??; these characters do not have special meaning.

All messages which can be output by SYSCHK or the DN92 code are listed below in Table 5-2. With the exception of the first which is always seen when the station has been successfully loaded, all messages are in alphabetical order IGNORING the % or ? characters. References to 'DP8E' are to the synchronous line interface modem controller. Generally, if these DP8E messages appear, you must contact field service.

Table 5-2
DN92 Software Messages

Message	Meaning
[INITIALIZING NETWORK DN92 V#.#NODE "name"]	SYSCHK has begun. #.# contains the version number of the DN92 software, and "name" is the nodename specified in the configuration file.

Table 5-2 (Cont.)
DN92 Software Messages

Message	Meaning
BAD MESSAGE TYPE	The DN92 has received unrequested data from the DECsystem-10.
CARRIER BACK	Carrier from the synchronous modem has returned.
CARRIER LOST	Carrier from the synchronous modem has been lost.
DKC8 CLOCK ERROR	Fatal error. SYSCHK has measured the speed of the DKC8 line frequency clock and found it either too fast or too slow. The clock may be malfunctioning or not present; or the processor speed may be incorrect.
DN92 CRASH PC=#####	Fatal error detected by software. The ROM is automatically restarted and issues a default load request after one minute. The DN92 operator can dump core by starting the dump program at location 201.
DN92 INT ERR	A hardware problem in the DN92 or an incorrect definition of the configuration at assembly time.
DN92 WON'T RUN ON A PDP8-I	DN92 software cannot run on a PDP8-I (SYSCHK looks for a BSW instruction).
DP8E BUS ERROR	A receive or transmit bus request was not serviced within one baud; either receive or transmit modem clock is faulty.
DP8E CHAR DETECT SKIP FAILED	Hardware problem in DP8E detected by SYSCHK (SYSCHK received a sync character from the DP8E, but the special character flag did not come up).
DP8E FIELD SELECT FAILED	Either DP8E not installed or serious hardware failure; contact Field Service.
DP8E FLAG WON'T CLEAR	SYSCHK could not clear a flag in the DP8E.
DP8E GT 9600 BAUD	SYSCHK determined that the synchronous line speed was greater than 9600 baud. Throughput may be degraded with synchronous links faster than 9600 baud.

Table 5-2 (Cont.)
DN92 Software Messages

Message	Meaning
DP8E IS IN LOOPBACK MODE	SYSCHK sent a pattern out on the synchronous line but it came back. A modem on the line is probably in loopback mode; reset it and either reload or restart the program at PC 200.
DP8E MODEM NOT READY	Be sure the modem on the synchronous line is powered on and ready (SYSCHK READ STATUS 1 instruction).
DP8E NO CARRIER	SYSCHK attempted to initialize the DP8E but the READ STATUS 2 instruction indicates that CARRIER/AGC is not present. Verify that modems are ready and sending and that the link between the modems is working.
DP8E NOT CLEAR TO SEND	Modem may not be on or not be ready.
DP8E NOT RECEIVING	Synchronous line interface and modem are ready but no characters are being received.
DP8E RCVD #	A non-SYNC character was received from the synchronous line; SYSCHK loops until it receives a SYNC character.
DP8E READ CHAR DETECTED # RIGHT 3	Problem in DP8E hardware. (SYSCHK expects a 3, but received an unexpected character (#) after the SYNC character.)
DP8E TERM NOT READY	Problem in DP8E hardware (SYSCHK set terminal ready, executed a DP8E READ STATUS 2 instruction and found terminal ready not set).
DP8E WC OR CA WRONG	Problem in DP8E hardware (SYSCHK finds incorrect values in databreak registers).
DP8E XMT NOT READY	Problem at modem or between modem and DP8E; clock may not be received from modem correctly (SYSCHK's attempt to send several SYNC's failed to complete quickly).
LPT FLAG WON'T CLEAR	Hardware problem in line printer interface (SYSCHK could not clear line printer flag).
LPTTRBL	Line printer has timed out and FTTRBL flag has been set in configuration file.

Table 5-2 (Cont.)
DN92 Software Messages

Message	Meaning
RESTARTING DN92 NODE"####"	This message appears on all DN92 TTY's whenever the station is restarted manually; #### indicate the node name defined in the configuration file (default = DN92).
RDCHK-REFEED CARD	Card-reader error in reading a card. Read the card in again.
TTY NOT CONNECTED	This message can appear on any TTY; it indicates that the terminal is not connected to a host. It generally indicates a problem at an intermediate node or a crash of the DECsystem-10. This message also appears when a viable connection is broken. If the DN92 receives input from an unconnected TTY, it sends a connect request to the host node.
TTY# IS RUNNING OPEN	The interface to TTY# (# is the line number) is receiving a continuous stream of null characters; this condition is not fatal but will overload the station if the stream is running faster than 110 baud.
TTY# NO XMT FLAG	The indicated line # does not exist. If the line is part of the installation, contact Field Service.
TTY# RCV FLAG WON'T CLEAR	Fatal hardware error; contact Field Service.
TTY# XMT FLAG WON'T CLEAR	Hardware problem; contact Field Service.

CHAPTER 6

DDT11 AND DDT92 FOR REMOTE DEBUGGING

DDT11 is a symbolic debugging program that runs on the DECsystem-10 and communicates with a running PDP-11 node. (Although DDT11 is an unsupported (category C) program, information about it is included here in the interests of network users that have both DECsystem-10 and PDP-11 nodes.) When DDT11 is used to examine a remote node processor, that processor and any intermediate node processors must be running.

DDT92 is a debugging program that communicates with a running PDP-8 node (see Section 6.1.1, DDT92).

> NOTE
>
> The facilities of DDT11 are not available through a DECnet-compatible port. Nodes to be interrogated and debugged via DDT11 must be DECsystem-10 network nodes.

DDT11 uses CAL11. and NODE. UUOs to examine and deposit information in a remote node; it therefore requires POKE and LOCK privileges for some of its operations. DDT11 can also read .BIN files produced by MACDLX and dump files produced by one of the node loaders: BOOT11, DTELDR, or NETLDR. POKE privileges are not required for analyzing dump files.

> NOTE
>
> To read DTELDR dumps, you must use DDT11 version 1E (10) or later and the DTELDR dump file must have a file extension of .BIN.

Output from DDT11 is printed on the user's terminal unless it is redirected to the line printer.

6.1 CREATE A FILE-SPECIFIC DDT11

The standard PDP-11 instructions are always defined as symbols in any DDT11. Additional symbols can also be defined for specific files using the cross-reference listings generated in Section 4.2. The following command sequence will generate a DDT11 file containing symbols that are specific to the software generated for a DN82 node with a node number of 22. (The convention for naming files follows that described in Section 3.1.)

```
.R DDT11              ;load a copy of a standard DDT11

DDT11 1E(10)          ;DDT11 identifies itself
Input: DN8222/S       ;add symbols from a specific program
 loaded nn symbols    ;DDT11 tells you how many symbols were loaded

Input: ^Z             ;exit DDT11

EXIT                  ;DDT11 signs off

.SSAVE DN8222         ;name and save the node-specific DDT11
DN8222 saved          ;DN8222.EXE is saved

.                     ;TOPS-10 prompt
```

6.1.1 DDT92

DDT92 can be used to examine remote running PDP-8 nodes such as the DN92. Unlike DDT11, DDT92 does not recognize symbols, so for useful work a CREF listing of the DN92 code must be available. With that restriction, the DDT92 user utilizes the same commands as the DDT11 user.

To compile and save DDT92, follow these steps:

```
.EXECUTE DDT92.MAC/COMP       ;force compilation and linking
MACRO: DDT11
LINK: loading
[LNKXCT DDT11 Execution]
DDT11 3(7)-1                  ;Version number of DDT92
Input: ^Z                     ;CTRL Z to exit MACRO
EXIT
.SSAVE DDT92                  ;store DDT92
DDT92 saved
.RUN DDT92                    ;run DDT92
DDT11 3(7)-1 = DDT92
Input: /NODE:24/8/L:0         ;respond to the Input prompt from
                              DDT92 with the number of the node
                              FROM which you wish to examine the
                              DN92, the PDP-8/8 switch and the
                              "Line" number over which
                              communication will occur (see
                              Figure 6-1).
```

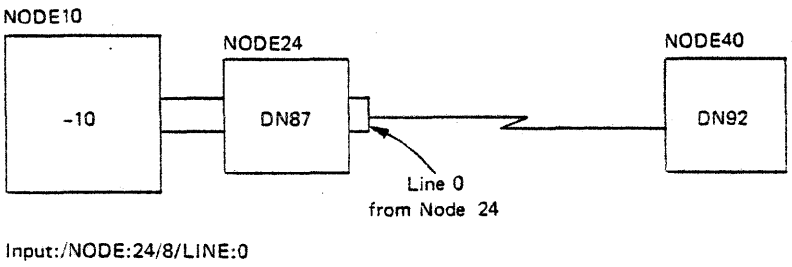


Figure 6-1 Specifying a Remote Node to DDT92

6.2 INITIAL DDT11 DIALOG

To symbolically examine code that is running in a remote node, you must load the node-specific copy of DDT11 that corresponds to the software running in the remote node. For example, if the file-naming conventions of Section 3.1 were used, a DN82 node with a node number of 22 would have the following files associated with it:

```
DN8222.BIN      PDP-11 executable code
DN8222.LST      Node software program listing
DN8222.EXE      File-specific copy of DDT11 for this node.  (-10 code)
```

With DN8222.BIN loaded into the remote node via NETLDR (Section 5.2), you must now run the corresponding file-specific DDT11 on the DECsystem-10. Type the following:

```
.RUN DN8222
```

TOPS-10 will load and start the DDT11 file, DN8222.EXE. DDT11 will print one line identifying itself by version number and specifying the cross-reference file it was generated from. The response to the above load could be:

```
DDT11 1E(10) = DN8222/ SYMBOLS = DSK:DN8222.CRF
```

DDT11 then asks you what you want to look at with the prompt:

```
Input:
```

The input specification is either a node identification or a core dump file. Use one or more of the following:

`/PORT:portnum` — where portnum is the DL10 port connection to a DC75NP, DN85, or DN87 front-end node.

`/NODE:nodenum` / `/NODE:nodename` — where nodenum or nodename identify a DN87S front-end node or a DN80-series remote node.

`filespec` — where filespec is of the form dev:filename.ext [p,pn] and identifies a core dump file to be examined.

If the remote node to be examined is not running network software but only the down-line load ROM, use /NODE to identify an adjacent node that is running and include a /LINE switch entry as follows:

`/NODE:nodenum/LINE:syn#`
`/NODE:nodename/LINE:syn#`
— where nodenum or nodename identify the adjacent running node and syn# specifies which line on the adjacent node is connected to the remote node to be examined.

If you intend to deposit into a running node you must enable your input for patching by including the /PATCH switch in your input specification. In addition, to patch a running node, you must have POKE privileges. For example, to examine and deposit into a running node numbered 22, enter:

```
Input: /PATCH/NODE:22
```

After accepting your input specification, DDT11 waits for you to request one of the DDT11 functions.

6.3 DDT11 TYPE OUT MODES

DDT11 output can take the form of instructions, numbers, bytes, ASCII text, or addresses. Numeric information can be displayed in any numeric radix from 2 (binary) through 16 (hexadecimal). There is both a permanent and a temporary type out mode and numeric radix. The initial permanent modes are instruction format and octal radix.

To set a temporary type out mode, type one of the following where n is an optional entry and determines the amount of output per line. If n is present, it must be entered as an octal number:

(ESC) nA — sets type out to address format. Each type out will consist of n addresses. (n defaults to 1)

(ESC) nB — sets type out to byte mode. Each type out will consist of n bytes. (n defaults to 2)

(ESC) nC — sets type out to numeric word format in the current radix. Each type out will consist of n words. (n defaults to 1)

(ESC) nI — sets type out to EBCDIC byte mode. Each type out will consist of n EBCDIC bytes. (n defaults to 2)

(ESC) S — sets type out to instruction format. Each type out will represent 1, 2, or 3 words, depending on the instruction type.

(ESC) nT — sets type out to ASCII text format. Each type out will consist of n ASCII characters. (n defaults to 2)

To set a temporary numeric radix, enter the following where r is an octal number:

(ESC) rR — sets the current numeric radix to r. The permissible range for r is 2 (binary) through octal 20 (hexadecimal).

To reset any temporary type out mode and/or numeric radix back to the permanent settings, press the RETURN key.

To set a new permanent type out mode and/or numeric radix, use the format for a temporary setting replacing the single ESC character with two ESC characters. For example:

(ESC) (ESC) B — sets the permanent type out mode to byte format, two bytes at a time by default.

(ESC) (ESC) rR — sets the permanent numeric radix to r. Again, the permissible range for r is 2 (binary) through octal 20 (hexadecimal).

6.4 DDT11 FUNCTIONS

The following DDT11 functions are some of the more commonly used facilities of DDT11. For additional information on DDT11, see the DECsystem-10 Networks Programmer's Guide and Reference Manual.

6.4.1 Examine Memory

When you first start DDT11, the location pointer is set to location 0. To position yourself at some other location, enter:

```
addr/      ;an octal address

or

label/     ;a symbolic label in your program
```

DDT11 will display the contents of that location according to the current type-out mode. (See Section 6.3.)

To display the contents of the next address, enter:

```
(LF)       ;LINE FEED key
```

To display the contents of the previous address, enter:

```
↑ or ^     ;up-arrow or circumflex character
```

If you are examining a program sequentially and arrive at a branching instruction, a line feed will step to the next sequential instruction. To take the branch and continue examining in the branch routine, enter

```
(TAB)      ;TAB key
```

If the TAB key is used on other than a branch instruction, DDT11 displays the contents of the location specified by the address of the last quantity typed and sets the location pointer to this address.

If the last display was in symbolic notation and you wish to see it as a number in the current radix, enter:

```
=          ;equal sign character
```

6.4.2 Insert Additional Labels

If you are using a nonspecific copy of DDT11 to examine a dump file, addressing will be all numeric. To insert labels, position the location pointer to the address to be labeled and enter the label followed by a colon. Consider the following display:

```
2450/ XOR %5,67040(%4)   ZIP:
```

When you entered 2450/, DDT11 displayed the contents of and set the location pointer to address 2450. By then entering ZIP:, you set that label equal to 2450. To examine the same location symbolically, enter ZIP/.

```
ZIP/ XOR %5,67040(%4)
```

6.4.3 Dump Memory

Occasionally, it is desirable to dump portions of PDP-11 memory to a disk file on the DECsystem-10. Use the $D command to initiate a dump and set the dump limits as follows:

loaddr<hiaddr> (ESC) D

where

loaddr	is the lowest address to be dumped
hiaddr	is the highest address to be dumped
(ESC) D	is the dump command

DDT11 will then prompt you for a file identification as follows:

FILE:

Enter a file name and extension. If you omit the extension, the default is .LSD.

When the file identification is accepted by DDT11, the file is opened, the dump is recorded, and the file is then closed. This means that each range of memory locations that is dumped is recorded in separate dump files.

Dump limits can be defaulted as follows:

hiaddr> (ESC) D	dumps 0 through hiaddr
loaddr (ESC) D	dumps loaddr through to the end of the file
(ESC) D	dumps the entire file

6.4.4 Search for a Word

DDT11 has a search facility that will scan a specified area in PDP-11 memory for either a match or a no-match condition on a 16-bit search argument. The formats of the two commands are as follows:

loaddr<hiaddr>srchwd (ESC) W (match)
loaddr<hiaddr>srchwd (ESC) N (no-match)

where

loaddr	is the lower limit of the search area
hiaddr	is the upper limit of the search area
srchwd	is the search argument: the word to be searched for
(ESC) W	is the DDT11 command to search for all words that match the search argument
(ESC) N	is the DDT11 command to search for all words that do not match the search argument

If search limits are not specified, the previous limits are used. Initial limits are both zero when examining a running PDP-11 node. When reading a dump file, the initial limits are the entire file.

Searching does not necessarily mean that you have to match on the full 16-bit word. DDT11 has a mask facility to provide for the matching of specific bits in a word. The initial value of the mask is 177777: match on all bits. To change the mask, enter the following:

ESC M/ 177777 newmask<RET>

where

ESC M/	is the DDT11 command to display the current mask and set the location pointer.
177777	is DDT11's reply; the current mask is displayed in octal.
newmask	is the new mask to be in effect.

For example, if you wish to perform a search that matches only on the right half of a word, bits 7 through 0, set the mask equal to 377. A match on only the left half of a word, bits 15 through 8, would require a mask of 177400.

6.4.5 Deposit into Memory

To patch a program that is running in a node or make changes to a binary file residing in DECsystem-10 auxiliary storage, load DDT11 as described in Section 6.2. Note that patching a running node requires you to have POKE privileges. Use the examine functions to locate and open the location to be patched. For example, the following examine entry displays location 3314. (DDT11 is in numeric word type-out mode.)

```
3314/ 12737
```

The location pointer is now set to 3314. If you now enter an octal value followed by RETURN, the contents of 3314 will be replaced by this new value.

If you enter an octal number consisting of more than 16 bits, only the rightmost 16 bits will be deposited.

6.4.6 Monitor a Location

When you are examining a running node, it may be useful to monitor a particular location for changes. The $V command can be used to display the contents of an opened location when the contents change. For example, to open location FRECNT and monitor its contents, enter:

```
$C FRECNT/ 141 $V
```

DDT11 will then display each change in contents as:

```
FRECNT/   143
FRECNT/   141
FRECNT/   142
FRECNT/   145
FRECNT/   141
      .
      .
      .
```

The masking feature of the search function is also active during the monitor function. If the initial mask value is in effect (177777), the monitored location is displayed when any of the 16 bits changes. If, for example, you want to monitor a change in bit 12 of some status word, set the mask as follows:

```
$M/ oldmask 10000i (RET)
```

where

$M/	is the command to display the current mask and set the location pointer
oldmask	is the current mask displayed by DDT11
10000	sets a new mask with only bit 12 set on

APPENDIX A

CONFIGURATION FILE SWITCHES

Entries which can be made in the configuration file are listed below in Table A-1 in alphabetical order. Each entry in the configuration file must be in the form:

ENTRY=value

No spaces are allowed. Switches listed in this appendix that are not included in Table 3-1 "Configuration File Entries by Node Type" are used only for debugging. Some switches apply only to certain node types. The definitive list of switches for the node for which you are assembling software can be found in the .LST listing created at assembly time (for example, when assembling software for a DN92 remote station, all switches with their set values are listed at the end of the .LST listing.) The value given for each switch in Table A-1 is a typical value. Most values are either 0 or 1; others are octal.

Table A-1
Configuration File Switches

Switch	Description
ABRGM=0	If nonzero, text storage area for DGUTS error messages is optimized (default = 0).
CDRN=1	Number of card readers at the remote station (0 or 1 only.
CNKFLD=1	Memory field which contains chunks (0 for first, 1 for second, 2 for third field).
CNKSIZ=10	Size of memory chunk; range is 4 to 200. This value must be a power of 2 (default is eight words = 10 octal).
CP.SIZ=xx	Defines buffer size (in bytes) for the POKE debugging aid. The POKE aid is disabled if this buffer is not defined.
CTRLN=2	Number of controllers using DDCMP.
CTYWID=110	Width of hard copy terminal or CTY at the remote station (default = 72 characters = 110 octal).
DEBUG=1	Enables debugging features - (if 0, none or little consistency checking is done; if 1, some checking occurs, reloads are done on serious errors; if -1, reloads occur on all errors).

Table A-1 (Cont.)
Configuration File Switches

Switch	Description
DEFBCD=bll.	Default typeball for 2741 terminals: 938.=BCD 963.=EBCDIC 987.=APL correspondence 988.=APL (EBCDIC)
DELROM=1	If 1, overrides ROM restart at the remote station when the host processor crashes (default = 0).
DEVN=list	Available devices; list is TTYN+CDRN+LPTN+...
DGUTS=0	If 1, causes the remote station to attempt error recovery (default = 0). If defined, FT.CHK=0, FTASRT=0 and DEBUG=0; i.e., these features are disabled. If FT.CTY is non-zero, a message is output at the CTY for each error encountered.
DIDLLS=1	If defined, code to permit setting DH11 line parameters via DDT11 is generated. Three symbols are defined: Location — Contents DHHLC — DH11 hardware address. DHHLN — Line number from DH11. DHHLO — Parameter word (value for DH11-line parameter register). When the hardware address is assigned to DHHLC, the parameters are set and DHHLC is reset to zero.
DHC=syn#	Line has a DM11.
DHL=syn#	Line type.
DHS=s	Line speed (in baud).
DMPHDR=1	If 1, DDCMP message headers can be selectively dumped into a circular buffer. The buffer is the size of a DMPHDR header (8 times the number of DMPHDR bytes).
DMPHIN=value	If DMPHDR is defined, DMPHIN can specify the number of incoming DDCMP headers to record (default = DMPHDR).
DMPHLN=addr	If DMPHDR is defined, DMPHLN gives the address of the 'line block' for the DDCMP line.
DMPHOU=0	The number of outgoing DDCMP headers to record if DMPHDR is defined (default = 0).

Table A-1 (Cont.)
Configuration File Switches

Switch	Description
DMPMSG=0	Enables selective dumping of data messages on a DDCMP line. (This feature requires the FT.SNK sink debugging aid to be nonzero.)
DN11N=0	Number of DN11 dial-out units at a remote station.
FT.ANF=0	If 1, causes inclusion of network features (default = 0; 0 only for DC75(11/20); otherwise 1).
FT.BIG=1	If 1, causes additional code to improve error reporting and line control to be used (set by another switch, 0 for DC75NP, 1 for others).
FT.CHK=0	If 1, the code for consistency checking is used and TWIDDL macros are enabled; always 0 if DGUTS=1.
FT.CTY=1	If 1, allows hardcopy terminal on DL11 to be used as a TTY (default = 1).
FT.DCP=1	If 1, causes inclusion of DECnet Compatible Port software. This switch is mutually exclusive with FT.MPT. If FT.DCP=1, the NSPLST macro must be in the configuration file.
FT.DDT=0	If 0, leaves room for PDP-11 ODT.
FT.DTE=0	If 1, DTE20 support is needed (e.g., for DN87S).
FT.HLP=1	If 1, debugging messages can be sent to the remote station CTY.
FT.MPT=0	If 1, multidrop lines are supported (in this case, DNCDDM.P11 must be in the program list for MACDLX and the value for FT.DCP must be 0).
FT.PFL.=0	If 1, 'profiler' feature to sample PC register and create an in-core histogram of program activity is enabled (default = 0).
FT.QSB=1	If 1, QSYNC bit in DDCMP protocol is implemented.
FT.RDE=1	If 1, remote data entry devices are supported on the network.
FT.RNN=1	If 1, restricted devices are supported. This switch must be 1 if TnRNN=nodenum is used.
FT.ROM=1	If 1, certain remote stations with bootstrap ROM's are supported. (This switch must be 1 for remote stations not attached to the host processor by a DL10 or a DTE20.)

Table A-1 (Cont.)
Configuration File Switches

Switch	Description
FT.SLB=1	If 0, all debugging information is omitted from line blocks. This switch should always be 1 (default = 1).
FT.SNK=0	If 1, SINK debugging macros are enabled (default = 0).
FT.SOU=0	If 1, 'message source' feature is enabled (permits insertion of predefined messages once or periodically into operating code). Default = 0.
FT.STC=1	If 1, DDT11 can be used to do remote examines/deposits of PDP-11 memory.
FT.TSK=0	If 1, special purpose tasks (e.g., multiprocessing) can be scheduled within the node.
FT.typ	The value of this switch declares the node type and includes the code for that type of node at assembly time. Allowable node types are: D75, D80, D81, D82, D85, D87 and 87S.
FT2BIT=1	If 1, sets minimum length of stop bit for certain terminal lines at 2. This switch should be 1 for terminal lines operating at 300 baud or faster.
FT3.02	If 1, code is compatible with DDCMP Version 3.02.
FT873=1	If 1, BM873 bootstrap ROM is present.
FT2741=0	If 1, 2741's are supported.
FTAPTH=1	Enables alternate paths for NCL logic.
FTASRT=1	If 1, the ASSERT macro is enabled.
FTBIGL=0	If 0, unused code is not listed.
FTBSTC=0	If 1, code for automatic boostrapping of remote stations with ROM's is included.
FTCHNK=0	Always 0.
FTCLEA=0	If 1, core is cleared before a start or restart is run. Always 0.
FTDL10=0	If 1, DL10 interfaces can be used. This switch should be 1 for a DN75, DN85 or DN87.
FTDH11=0	If 1, DH11 asynchronous line controllers can be used. This switch should be 1 if any TTY's are present.

Table A-1 (Cont.)
Configuration File Switches

Switch	Description
FTDM11=0	If 1, DM11 modem controllers can be used. If any TTY lines are dataset lines, this switch must be 1.
FTDP11=0	If 1, code for DP11 synchronous line controller is used.
FTDQ11=0	If 1, code for DQ11 NPR synchronous line interface is used. This switch must be for a DN82, DN85 or DN87.
FTDS11=0	If 1, code for DS11 is used. This switch must be 1 for a DC75NP.
FTDU11=0	Always 0.
FTEVEN=0	If 1, messages are padded to end on an even word boundary. This switch must be 1 when DQ11 interfaces are pesent.
FTExxx=1	If 1, the specified 2741 typeball is supported. The variable xxx can be: 938, 963, 987 or 988.
FTHOST=0	If 0, the SET HOST command is unsupported.
FTKG11=0	If 1, KG11 devices that compute CRC's and LRC's are supported.
FTLBAC=0	If 1, loopback test can be used.
FTLPLC=0	If 1, the LA180 line printer can output in lowercase.
FTNSYN=nn	Number of SYN characters sent before a DDCMP control message (must be even and greater than or equal to 2; default = 8).
FTOLDC=0	Must be 0 for TOPS-10 6.03 and later.
FTQSYN=nn	Number of SYN characters sent after a message (must be even and greater than or equal to 2; default = 2).
FTRACE=0	Used as diagnostic tool; causes TRACE macros to be expanded.
FTTRBL=0	If 1, a message is sent to the CTY when a device times out.
FTROLL=0	If 1, a 'TROLL' debugging feature is implemented.
FTSILO=32	Sets DH11 silo alarm level; must be less than 64 and a power of 2.
FTSLCT=0	Determines selection algorithm for multidrop lines; always 0.

Table A-1 (Cont.).
Configuration File Switches

Switch	Description
FTSTCD=1	Must be 1 for any remote station debugging. If 1, a user with POKE privileges can deposit in remote memory with DDT11.
FTTWID=0	If 1, TWDDL macros are expanded. These macros are used for debugging.
JIFSEC=60	The number of jiffies in one second; (always 60 for a 60 Hz PDP-11 remote station, 50 for a 50 Hz station).
LA180=1	If 1, LA180's are enabled.
LPTFAK=0	If 1, LPT output is discarded.
LPTN=0	If 1, a line printer is supported at the remote station.
LPTWID=204	Width of line printer at remote station (80=120 octal; 120=170 octal; 132=204 octal).
MAXOLN=244	Maximum output message size, in bytes.
NCL.LG=0	If nonzero, NCL trace is enabled, as follows: If — Action 0 Default-no action 1 Traces all messages routed by the node. 2 Traces only device-control messages destined for the node. 3 Equivalent to both 1 and 2. If this switch is nonzero, symbols BUGBF and BUGQ are defined. BUGBF points to the 64-times-10-word circular buffer for the trace; BUGQ is one word containing the address of the next entry in the trace. (Each 10-word entry in the buffer contains the top three words of the stack, the message length and the first 12 bytes of the message. Generally, the top three words of the stack indicate the origin of the message.)
NLINES=#sy	Number of synchronous lines attached to the node. If omitted, default = 4; for DN87, DN87S with only asynchronous lines, this must always be 0; can be up to 12 for DN85, DN87, DN87S.
NUMSYN=10	Number of SYN characters at start of message.
OURNNM=nn	Our node number. Can be 01 to 77 (octal); must be unique within the network.

Table A-1 (Cont.)
Configuration File Switches

Switch	Description
PDP11=mn	Model number of PDP11 processor at node. For example, must be 40 for DN8x, 15 for DC75NP.
PEEKDM=0	If 1, enables a facility to monitor modem-control state transitions on a DH11/DM11 line (default = 0).
PRFL.L=bound	Defines low-boundary value for profiler histogram (default = 1000 (octal)).
PRFL.H=bound	Defines high-boundary value for profiler histogram (default = 60000 (octal)).
PRFL.S=power	Defines resolution of the profler histogram. Each bar has a width of 2 to the PRFL.S power (default = 8; i.e., 2 to the 8th power).
REPDWN=36	Number of DDCMP <REP> messages that must pass before the line is considered down.
REPSEC=2	Number of seconds between DDCMP <REP> transmissions on an idle line.
REPTIM=170	RET timeout in jiffies (120 jiffies = 170 octal = 2 sec).
RPLPOL=1	If non-zero, remotely operated line printers are enabled to send messages to the CTY when they need assitance (default = 1).
SCBMAX=mx	Number of nodes the network can support. Must be greater than or equal to 2 since it includes the host processor (DECsystem-10) and the node being installed. This defaults to NLINES*2 for 80-series nodes; to NLINES for DC75NP.
SDELAY=1	Always 1; delay in seconds.
SLSDMP=0	If 1, enables a facility to monitor synchronous line driver activity. If used, two buffers of SLSDMP words are defined; one is for the receiver, one for the transmitter.
TnDSL=1	Specifies that line n is a "dataset line" (connects to a TTY via a modem). One such entry is required for each dataset line.
TnRNN=nodenum	Defines a restricted node number. This specifies which node is the default host for the specified TTY. This cannot be used unless FT.RNN=1.
TnRS=s	Receive speed for an asynchronous line with no autobaud detect. This switch should be used in conjunction with the TnnXS switch. Receive speed is the speed from the terminal to the node processor.

Table A-1 (Cont.)
Configuration File Switches

Switch	Description
TnTAB=0	If 1, the terminal on line n has hardware tabs.
TnWID=w	Sets width of terminal connected to line n. Terminals specified with this switch must be attached via DH11 interfaces. The default width is 72 characters (110 octal); allowable widths are 80 (120 octal) and 132 (204 octal).
TnXS=s	Transmit speed for a line with no autobaud detect. This switch should be used in conjunction with the TnRS switch. Transmit speed is the speed from the node processor to the terminal.
TTYMIC=120	Maximum number of characters that can be in an input message from a TTY.
TTYN=#as	Number of terminals that can be connected, excluding the CTY. The number of terminals can be 16 times the number of DH11 asynchronous line interfaces that are supported. The allowable range is 0 to 112 (0 to 160 octal).

APPENDIX B

NOTATION

Abbreviation	Meaning
addr	memory address
bll.	typeball for 2741-type terminals
c	memory location (lower bound)
CPU type	type of processor (PDP-11 OR PDP-8)
d	memory location (upper bound)
len	argument length (NSPLST macro)
mn	PDP-11 model number (15, 40)
mmm	TTY terminal number
mx	maximum
n	asynchronous line number (0 to 112)
nn	node number (0 to 77 octal)
nodeid	either nodename or node number
nodename	node name, up to 6 alphanumeric characters, starting with an alphabetic character
nodenum	node number (0 to 77 octal)
portnum	port number (0 to 7)
s	speed of line (in baud)
ser	serial number of remote CPU
syn#	synchronous line number (0 to 12)
typ	type of node (e.g. D75, D80, 87S)
type	node type (e.g., DN80, DN92)
w	width of line (in columns)
x	CPU number (DTELDR)
xxx	feature identifier (e.g., RNN,TAB)
y	DTE number (DTELDR)
#as	number of available asynchronous lines (0 to 112; TTYN equals this value)
#sy	number of synchronous lines (0 to 12; NLINES equals this value)

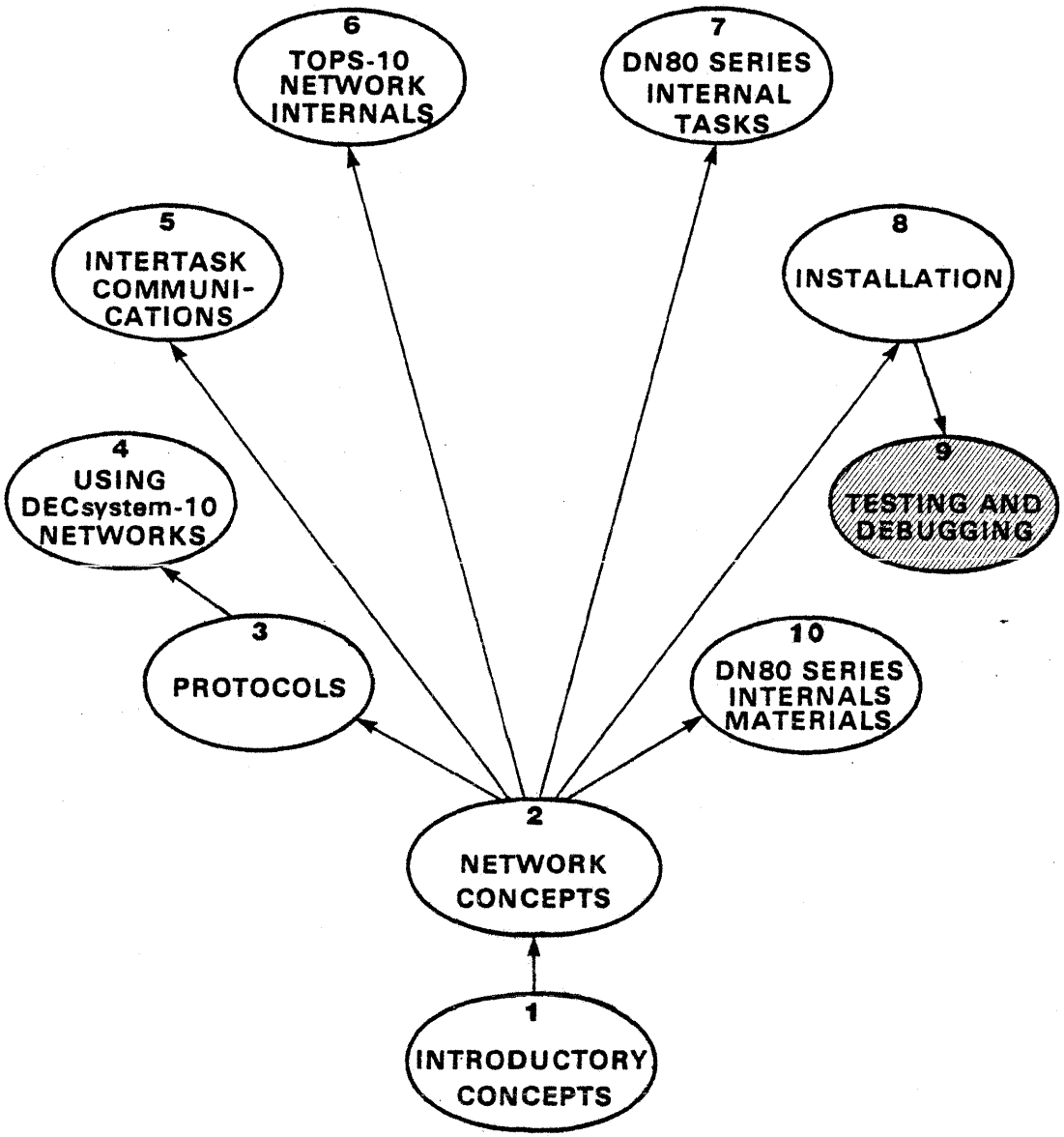
INDEX

Module Test

1. Perform the Installation Lab, please see your instructor for the particulars.

Testing and Debugging
Module 9

Course Map



M8 0277

Introduction

There are two major tools for testing and debugging remote nodes on DECsystem-10 networks. The first is a once-only diagnostic called CHK11, a diagnostic type program, which is loaded with the remote node code and is executed at system startup (not at restarts, since this code is overlayed). The other tool is DDT11 (called DDT60 for DN60 series code). DDT11 runs on a host -10 and uses -10 UUO'S to perform functions (e.g., EXAMINE, DEPOSIT) on the specified -10 network node. The command language is quite similar to -10 DDT, and is symbolic.

Objectives

Upon completion of this module, the student will be able to

1. Examine specified locations in a running remote station front end, and make the same examination from dump files.

2. Patch a running 80 series remote station.

Resources

The DECsystem-10 Networks Installation Guide

INITIAL HARDWARE TESTER - CHK11

CHK11 is a hardware test module for the DN8Ø, DN81, DN82, DN6Ø, DN61, DN87, and DN87S. When the system program for one of these machines is started, CHK11 is brought in to perform an initial hardware survey. This module makes sure the devices and memory of the system are functioning correctly. CHK11 does not perform a total system diagnostic check, but rather checks for the most common problems affecting the DN systems.

NOTE

There is an analogous program for the (PDP-8 based) DN92 called SYSCHK. For full details, see the DECsystem-1Ø Networks Software Installation Guide, AD-5156A-TI.

During execution, CHK11 prints the names and quantities of each device type present in the system. All CHK11 test information is printed on the DN system's console terminal (CTY).

```
INITIALIZING DN87 V11(23) APR 1977 - DN8716 (16)

     100000 BYTES OF MEMORY
       MF11-UP
       KW11-L
       KG11-A
       DL10
     4 DM11-BB'S
     4 DH11'S
     1 DQ11   SKIPPING SPEC CHAR TEST
RESTARTING DN87 V11923) APR 1977 - DN8716 (16)

     RSX220F SYS #514/546 5:23:18
```

Similar output for a DN82 remote station at node 22 could be:

```
Initializing DN82 V11(23) APR 1977 - CTCH22 (22)

     100000 BYTES OF MEMORY
       MF11-UP
       KW11-L
       KG11-A
     NOT CHECKING DL10
     1 CR11
CR11 not Rdy
     1 LP11
LP11 #0 Not Rdy
    2 DM11-BB'S
    2 DQ11'S SKIPPING SPEC CHAR TEST
Restarting DN82 v11(23) APR 1977 - CTCH22 (22)

   RSX220F SYS #514/546 12:31:03
```

If a problem is detected by CHK11, the error information is printed in the following format:

```
?  device-name #number (ADR=address)
    ERROR AT listing-address
      message-describing-problem
      REG/ADR=address GD=expected BD=received XOR=bits in
           in question    value        value       question
      [FATAL ERROR]
```

If the problem is a device error, CHK11 prints the device name, number and address. Device numbers are sequential, starting with zero. The address printed is the first address of the device. The listing-address informs the user where in the CHK11 source listing to look for the error.

Next, CHK11 prints a brief message describing the error. The line following the message gives the actual address in question, along with the value CHK11 expected to find in that location, the value CHK11 actually did find, and the exclusive OR (XOR) between the expected and actual values. The XOR reveals which bits in the location differ between expected and actual values.

If CHK11 is unable to continue, it prints the FATAL ERROR message and halts. The operator of the system must evaluate the problem, correct it, and reboot the system. Examples of fatal errors are memory errors or problems with the KW11.

When CHK11 halts on a fatal error, a stop-code is displayed in the data lights on the console. (Obviously, a problem with the CTY cannot be printed on the CTY.) Table 8-1 lists the error stop-codes and their meanings.

Table 9-1
CHK11 Stop Codes

STOP CODE	MEANING
1	time out or bus error (trap to address 4)
2	DL10 error
5	no console terminal
6	memory error
7	KW11 error

To display the address of the error in the data lights, the operator presses CONTINUE after the machine halts.

To restart CHK11, the operator presses CONTINUE a second time.

If CHK11 completes the tests with no discovered errors, it transfers control to the DN system code and the system is then running.

If any errors are detected, CHK11 prints:

? ERRORS DETECTED - DO YOU WANT TO PROCEED (Y OR N)?

Any response other than "Y" causes CHK11 to halt.

To continue the system for purposes of testing, the operator types a "Y". CHK11 then prompts to verify that response with the question:

ARE YOU SURE?

A "Y" response to this question causes control to pass to the DN system program. Any response to this question other than "Y" causes CHK11 to halt.

NOTE

Under certain conditions, although an error is reported by CHK11, the DN system may still be partially functional. If the limited functionality is sufficient for user needs, the operator may use the above mechanism to proceed.

Bit Clear and Set Test

CHK11 uses the Bit Clear and Set Test to test those bits in a device register that can be both written and read.

CHK11 first attempts to clear all the READ/WRITE bits that are to be tested in a register. It then checks that all those bits actually did clear. If any tested bit is not clear, CHK11 prints an error message. For example:

```
? LP11 #0 (ADR = 177514)
   ERROR AT 15352
     BIT DID NOT CLEAR
     REG/ADR = 177514 GD = 0 BD = 100 XOR = 100
```

In the example above, the problem occurred with the register of device LP11 number 0. The address of the first device register is 177514. The listing address of the error is 15352. The error message indicates that a bit did not clear. The register under test was 177514. The expected content of that register was zero, the value found in that register was 100 (octal). Therefore, CHK11 could not clear bit 6 in the LP11 status register. This bit is shown by the XOR value 100.

After the register is known to be cleared, CHK11 sets and checks each bit in the register individually. Each bit is set and tested to make sure it is set. If set, the bit is then cleared, and checked to make sure that it cleared. If any bit is not set when tested, CHK11 prints an error message. For example:

```
? DQ11 #0 (ADR= 177264)
   ERROR AT 15376
     BIT DID NOT SET
     REG/ADR = 177264 GD = 100 BD = 0 XOR 0 100
```

A bit in the DQ11 failed to set. The DQ11 first device address is 177264. The listing address of the error is 15376. The error message indicates that a bit did not set. The line following it indicates that the status register in question is at location 177264. The GD value is the expected value. The BD value is the actual value. The XOR value indicates that the bit that could not be set was bit 6.

If any bit could not be cleared after being set, CHK11 prints an error message. The XOR value in this message indicates the bit that was set, but not cleared. For example:

```
? CR11 #0 (ADR= 177160)
   ERROR AT 15422
     BIT DID NOT CLEAR
     REG/ADR = 177160 GD = 0 BD = 2 XOR = 2
```

Bit 1 in the CR11 status register could not be cleared. The location of that register is 177160.

Interrupt Test

CHK11 performs the Interrupt Test to verify that the various devices interrupt correctly. The test initially checks to make sure that the device interrupts by establishing a legitimate interrupt condition in the device register(s) and initializing timer. CHK11 then lowers the Processor Interrupt (PI) level to zero. If the device does not interrupt within the allotted test time, a time-out error occurs. For example:

```
? DN11 #0 (ADR = 175200)
   ERROR AT 16244
     DID NOT INTERRUPT
```

The DN11 status register is 175200. The listing address of this error is shown to be 16244. The error message indicates that the device did not interrupt.

If the device does interrupt, CHK11 saves the device interrupt level and verifies that the interrupt was to the correct vector address. If the interrupt was to an incorrect vector address, an error message is printed. For example:

```
? KW11 #0 (ADR = 171200)
   ERROR AT 16274
     INTERRUPTED TO WRONG VECTOR
     ADR/REG = 171200 GD = 240 BD = 300 XOR = 140
```

The listing address is 16274. The first hardware address of the KW11 is 171200. CHK11 then prints an error message. The device address in which the interrupt bits were set is the ADR/REG value 171200. The GD value indicates the vector address to which the interrupt should have trapped. The BD value is the actual address trapped to. The XOR provides the bits that differ in the vector addresses.

CHK11 then tests the interrupt conditions for the device by individually setting interrupt enable bits and verifying that an interrupt occurs when each related interrupt condition is set. If the interrupt traps to the wrong vector address at any point in this test, the above error message is displayed. After each interrupt occurs, CHK11 verifies that the device trapped to the same PI level as it did on the first interrupt. If the PI level differs, an error message is printed. For example:

```
? DH11 #0 (ADR = 175240)
   ERROR AT 16354
     INTERRUPTED TO DIFFERENT PI LEVEL
     ADR/REG = 175240 GD = 240 BD = 200 XOR = 40
```

The first hardware register for the device DH11 #0 is 175240. The listing address of the error is 16354. CHK11 then prints an appropriate error message. The value 175240 on the next line specifies the device address in which the interrupt bits were set. The GD value indicates the device's first interrupt level (bit locations 5-7). The BD value is the device's current priority level. The XOR displays the difference in priority levels.

After verifying that the proper interrupts occur with the interrupt enable bit set, CHK11 makes sure that an interrupt does not happen if that bit is cleared. This is done by setting the interrupt condition bit, lowering the PI level to zero and waiting a specified amount of time for an interrupt to occur. If an interrupt occurs, an error message is printed. For example:

```
? DN11 #3 (ADR = 175220)
   ERROR AT 16442
     INTERRUPTED WITHOUT INTERRUPT ENABLED
```

If no error was discovered by these interrupt tests, CHK11 assumes that the interrupts are working correctly.

Memory Test

This test checks to verify that all bits located in core above this routine can be read and written by setting a bit, reading it, and then rotating the pattern. (This is called a sliding bit pattern.) If any bit cannot be written or read, an error occurs. An error in memory is fatal and the machine halts with a stop-code of 6 displayed in the console data lights. For example:

```
? MEM ERR
   ERROR AT 12242
     REG/ADR = 20000 GD = 1 BD =0 XOR = 1
     FATAL ERROR
```

The listing address of this error is 12242. The address 20000 on the next line is the word in memory under test. The GD value is what CHK11 expected to find in that location. The BD value is what was in that location. The bit in question is shown by the XOR value.

Determining the DL10 Base Address

In addition to testing the hardware, CHK11 also determines the DL10 Base Address. The DL10 Base Address must be a location that is a multiple of 2000. The following algorithm is used:

CHK11 evaluates the first location following the CHK11 code that is a multiple of 2000. A BIS instruction is performed on the Clear NXM, Clear Parity Error, Clear Word Count Overflow, and the Clear 11-Interrupt bits in that location. CHK11 then clears that location and tests the contents. If the location does not contain a zero, an error message of the following form is printed:

```
? CAN'T CLEAR DL10 OR MEM
   ERROR AT 12244
     ADR/REG = 20000 GD = 0 BD = 1 XOR = 1
```

This error message indicates that the reference in the program listing is address 12244. ADR/REG shows the location that could not be cleared. The GD value is the expected result. The BD value is the actual contents of that location. The XOR indicates the bit(s) in error.

If the location is clear, CHK11 then sets the NXM, Parity Error, Word Count Overflow, and 11-interrupt bits, and tests those bits. If any of these bits is not set, an error message of the following form is printed:

```
? ERROR AT ADR 12326
   DL10 OR MEM ERR
     ADR/REG = 100000 GD = 105200 BD = 5200 XOR = 100000
```

This message indicates that the error occurred at listing address 12326. ADR/REG indicated the location under test. The GD value is the expected value. The BD value is the actual contents of the location. The XOR reveals the bit(s) in error.

If the bits are set properly, CHK11 issues the BIS instruction to clear those bits. The location is then tested and, if it contains a zero value, it is the DL10 Base Address. If the location does not contain zero, CHK11 repeats the procedure using an address which is 2000 words larger than the one just tested. This is repeated using multiples of 2000 until a DL10 Base Address is found or until the memory space is exhausted.

The following sections describe the error messages applicable to the various devices.

Console Terminal

If there is no console terminal interface, or there is an error in the interface that does exist, a fatal error occurs and the machine halts with a stop-code value of 5 in the console data lights.

Memory Error Message

MEM ERR

During the memory test, the contents read from ADR differ from the expected value. This is a fatal error with a stop-code value of 6.

DL10 or Memory Error Messages

CAN'T CLEAR DL10 OR MEMORY

To locate the DL10 base address, CHK11 places 425000 into a location and if the location is the DL10 base address, it is cleared. The following instruction is a CLR which should definitely leave the location zeroed. If the location is not clear, the fatal error message is printed and the machine halts with a stop-code of 2 displayed in the data lights.

DL10 OR MEM ERR

After a location is cleared, the value 105200 is deposited into that address. That location is then tested and if the value was not retained, an error has occurred. This is a fatal error and the machine halts with a stop-code of 2 displayed in the data lights.

CR11 Error Messages

CR11 NOT READY

The Busy Bit in the status register is set. This may indicate a power-off condition. The device cannot be checked.

CR11 TIMED OUT

A read was performed with the interrupt enable set. The error bit or the done bit was not set within the predetermined time limit.

INTERRUPT DID NOT OCCUR

With the interrupt condition on and the interrupt enable set in the device, lowering the priority level to zero did not cause an interrupt to occur.

INTERRUPTED TO WRONG VECTOR

The device interrupted to the wrong vector address.

DH11 Error Messages

BIT DID NOT CLEAR

A bit did not clear in the DH11 register being tested.

BIT DID NOT SET

A bit failed to set in the DH11 register being tested.

DATA ERROR

While CHK11 was testing the DH11 in loopback, the data received differed from the data transmitted. The ADR/REG value is the line under test. The GD value is the transmitted data. The BD value is the received data. The XOR shows the bits that differed.

DATA ERROR BIT SET

While CHK11 was testing the DH11 in loopback, the Next Received Character Register had the DATA OVERRUN bit (bit 14) set, the FRAMING ERROR bit (bit 13) set, or the PARITY ERROR bit (bit 12) set.

ILL XMIT INT

An interrupt occurred to the XMIT vector when the XMIT interrupt bit was not set. The output following this message indicates the line number under test.

ILLEGAL INT

An undetermined interrupt occurred while CHK11 was waiting for a receiver or transmit interrupt in the DH11 loopback test.

ILLEGAL LINE NUMBER

An incorrect line number was set in the Next Character Received register. The output following this message indicates the number of the line under test.

INTERRUPT DID NOT OCCUR

With the interrupt condition on and the interrupt enable set in the device, lowering the priority level to zero did not cause an interrupt to occur.

INTERRUPTED TO DIFFERENT PI LEVEL

The current Processor Interrupt level of the device differs from the first interrupt level.

INTERRUPTED TO WRONG VECTOR

The device interrupted to the wrong vector address.

INTERRUPTED WITHOUT INTERRUPT ENABLED

An interrupt occurred when the interrupted bit was set and the interrupt enable bit was not set.

NO INTERRUPT

With interrupts enabled and the PI level at zero, an interrupt did not occur within the alloted time. The output following this message indicates the line under test.

NOT VALID DATA

The VALID DATA PRESENT bit (bit 15) was not set in the Next Received Character register. The output following this message indicates the line under test.

SILO FULL

With the silo alarm level set to 4, the receiver interrupt bit became set, indicating that the number of characters stored in the silo exceeds the alarm level. The output following this message indicates the number of the line under test.

XMIT NXM

While CHK11 was testing the DH11 in loopback, the NXM bit (bit 10) was set in the System Control Register. The output following this message indicates the line that was under test.

DL1Ø Error Messages

BIT DID NOT CLEAR

A bit did not clear in the DL1Ø status register being tested.

BIT DID NOT SET

A bit failed to set in the DL1Ø status register being tested.

INTERRUPT DID NOT OCCUR

With the interrupt condition on and the interrupt enable set in the device, lowering the priority level to zero did not cause an interrupt to occur.

INTERRUPTED TO DIFFERENT PI LEVEL

The current Processor Interrupt level of the device differs from the first interrupt level.

INTERRUPTED TO WRONG VECTOR

The device interrupted to the wrong vector address.

INTERRUPTED WITHOUT INTERRUPT ENABLED

An interrupt occurred when the interrupt bit was set and the Interrupt Enable bit was not set.

DM11BB Error Messages

BIT DID NOT CLEAR

A bit did not clear in the DM11BB register being tested.

BIT DID NOT SET

A bit failed to set in the DM11BB register being tested.

BUSY DID NOT SET

After CHK11 set the Scan Enable bit, the Busy Bit was tested and found to be cleared.

BUSY DID NOT CLEAR

After CHK11 set the Clear Scan bit and waited an appropriate amount of time, the Busy Bit was found not to be cleared.

CLEAR SCAN ERROR

After CHK11 set the Clear Scan bit in the status register, a bit that should have been cleared was set. The bits that should be cleared are: DONE, Maintenance Mode, Interrupt Enable, Scan Enable, and Line Number Field.

INTERRUPT DID NOT OCCUR

With the interrupt condition on and the Interrupt Enable set in the device, lowering the priority level to zero did not cause an interrupt to occur.

INTERRUPTED TO DIFFERENT PI LEVEL

The current Processor Interrupt level of the device differs from the first interrupted level.

INTERRUPTED WITHOUT INTERRUPT ENABLED

An interrupt occurred when the interrupt bit was set and the Interrupt Enable bit was not set.

DN11 Error Messages

BIT DID NOT CLEAR

A bit did not clear in the DN11 register being tested.

BIT DID NOT SET

A bit failed to set in the DN11 register being tested.

INTERRUPT DID NOT OCCUR

With the interrupt condition on and the Interrupt Enable set in the device, lowering the priority level to zero did not cause an interrupt to occur.

INTERRUPTED TO DIFFERENT PI LEVEL

The current Processor Interrupt level of the device differs from the first interrupt level.

INTERRUPTED TO WRONG VECTOR

The device interrupted to the wrong vector address.

INTERRUPTED WITHOUT INTERRUPT ENABLED

An interrupt occurred when the interrupt bit was set and the Interrupt Enable bit was not set.

DP11 Error Messages

BIT DID NOT CLEAR

A bit did not clear in the DP11 register being tested.

BIT DID NOT SET

A bit failed to set in the DP11 register being tested.

INTERRUPT DID NOT OCCUR

With the interrupt condition on and the Interrupt Enable set in the device, lowering the priority level to zero did not cause an interrupt to occur.

INTERRUPTED TO DIFFERENT PI LEVEL

The current Processor Interrupt level of the device differs from the first interrupt level.

INTERRUPTED TO WRONG VECTOR

The device interrupted to the wrong vector address.

INTERRUPTED WITHOUT INTERRUPT ENABLED

An interrupt occurred when the interrupt bit was set and the Interrupt Enable bit was not set.

DQ11 Error Messages

BIT DID NOT CLEAR

A bit did not clear in the DQ11 register being tested.

BIT DID NOT SET

A bit failed to set in the DQ11 register being tested.

CAN'T CLR DATASET FLG

The test program was not able to clear the DQ11 data set flag.

ERR INT

While CHK11 was determining the DQ11 special characters, an interrupt occurred on Vector B.

ILLEGAL INTERRUPT

While CHK11 was determining the DQ11 special characters, an interrupt occurred that was not a special character interrupt.

INTERRUPT DID NOT OCCUR

With the interrupt condition on and the Interrupt Enable set in the device, lowering the priority level to zero did not cause an interrupt to occur.

INTERRUPTED TO DIFFERENT PI LEVEL

The current Processor Interrupt level of the device differs from the first interrupt level.

INTERRUPTED TO WRONG VECTOR

The device interrupted to the wrong vector address.

INTERRUPTED WITHOUT INTERRUPT ENABLED

An interrupt occurred when the interrupt bit was set and the Interrupt Enable bit was not set.

MORE THAN 4 SPEC CHAR DET

More than four special characters were detected in the DQ11.

SECONDARY REG ERROR

The data loaded into the secondary register does not agree with the data read. ADR/REG indicates the secondary register under test.

SPEC CHAR CODE FIELD Ø

A special character interrupt occurred, but the special character code field contains a zero.

DS11 Error Messages

BIT DID NOT CLEAR

A bit did not clear in the DS11 register being tested.

BIT DID NOT SET

A bit failed to set in the DS11 register being tested.

KG11 Error Messages

CAN'T LOAD BCC CORRECTLY

The data loaded into the Data Register is not the same as the data read from the BCC register.

INCORRECT CRC

While CHK11 was calculating the Cyclic Redundancy Check (CRC) polynomials, the simulated value differed from the actual value calculated by the KG11.

NOT PRESENT

The KG11 is not connected or it cannot be accessed.

KW11 Error Messages

fast

A clock tick occurred too soon. This is a fatal error with a stop-code of 7.

INTERRUPT DID NOT OCCUR

With the interrupt condition on and the Interrupt Enable set in the device, lowering the priority level to zero did not cause an interrupt to occur.

INTERRUPTED TO WRONG VECTOR

The device interrupted to the wrong vector address.

NOT PRESENT

The KW11 is not present or it is defective. This is a fatal error with a stop-code of 7.

SLOW

There was no response from the KW11 in the allocated time, which is significantly greater than one tick time. This is a fatal error with a stop-code of 7.

LP11 Error Messages

BIT DID NOT CLEAR

A bit did not clear in the LP11 status register.

BIT DID NOT SET

A bit failed to set in the LP11 status register.

INTERRUPT DID NOT OCCUR

With the interrupt condition on and the Interrupt Enable set in the device, lowering the priority level to zero did not cause an interrupt to occur.

INTERRUPTED TO WRONG VECTOR

The device interrupted to the wrong vector address.

INTERRUPTED WITHOUT INTERRUPT ENABLED

An interrupt occurred when the interrupt bit was set and the Interrupt Enable bit was not set.

LP11 #N NOT READY

The error bit is set in the LP11 status register.

PA611 Error Messages

BIT DID NOT CLEAR

A bit did not clear in the PA611 status register.

BIT DID NOT SET

A bit failed to set in the PA611 status register.

PP11 Error Messages

BIT DID NOT CLEAR

A bit did not clear in the PP11 status register.

BIT DID NOT SET

A bit failed to set in the PP11 status register.

INTERRUPT DID NOT OCCUR

With the interrupt condition on and the Interrupt Enable set in the device, lowering the priority level to zero did not cause an interrupt to occur.

INTERRUPTED TO WRONG VECTOR

The device interrupted to the wrong vector address.

INTERRUPTED WITHOUT INTERRUPT ENABLED

An interrupt occurred when the interrupt bit was set and the Interrupt Enable bit was not set.

PR11 Error Messages

BIT DID NOT CLEAR

A bit did not clear in the PR11 status register.

BIT DID NOT SET

A bit failed to set in the PR11 status register.

INTERRUPT DID NOT OCCUR

With the interrupt condition on and the Interrupt Enable set in the device, lowering the priority level to zero did not cause an interrupt to occur.

INTERRUPTED TO WRONG VECTOR

The device interrupted to the wrong vector address.

INTERRUPTED WITHOUT INTERRUPT ENABLED

An interrupt occurred when the interrupt bit was set and the Interrupt Enable bit was not set.

TC11 Error Messages

BIT DID NOT CLEAR

A bit did not clear in the TC11 register being tested.

BIT DID NOT SET

A bit failed to set in the TC11 register being tested.

INTERRUPT DID NOT OCCUR

With the interrupt condition on and the Interrupt Enable set in the device, lowering the priority level to zero did not cause an interrupt to occur.

INTERRUPTED TO DIFFERENT PI LEVEL

The current Processor Interrupt level of the device differs from the first interrupt level.

INTERRUPTED TO WRONG VECTOR

The device interrupted to the wrong vector address.

INTERRUPTED WITHOUT INTERRUPT ENABLED

An interrupt occurred when the interrupt bit was set and the Interrupt Enable bit was not set.

Debugging

REMOTE DEBUGGER - DDT11

DDT11 is a tool that can be used as a remote debugger for privileged users in a network. DDT11 is distributed as a Class C unsupported software product. Its description is included in this document because it may be of interest to network users who have both DECsystem-10s and PDP-11s in the network.

DDT11 can be used to examine and deposit information in a remote PDP-11 memory. Since this program uses the CAL11. UUO, the user must have system POKE privileges. DDT11 can also be used to read PDP-11 core dumps that are stored in a file produced by BOOT-11. (See BOOT-11 Programming Specification in the DECsystem-10 Software Notebooks.)

Output from DDT11 is printed on the user's terminal unless it is directed to the line printer. DDT11 uses the standard PDP-11 instruction set defined as symbols. As in DDT-10, additional symbols can be defined by opening a location and entering a symbol name terminated by a colon. Symbols can also be read from a MACDLX cross reference listing file.

Symbols defined in the source by means of colons are also available for printing from DDT11. Symbols defined in the source using equal signs are available for input, but are suppressed on DDT11 output. Refer to Chapter 4 of the DECsystem-10 Networks Installation Guide.

Initial Dialogue

When DDT11 is started, it first asks the user to enter the input specification with the prompt

INPUT:

The user should enter one of the following input specifications:

1. /PORT:portnum

   where portnum is the number of the DL10 port connection to the DN87, DC75NP, or DN61.

2. /NODE:node-num

   where node-num is the number of the node.

3. /NODE:node-name

   where node-name is the name of the node.

4. dev:filnam.ext[proj,prog]

   to identify a core dump file to be examined.

If /NODE is specified, the user can also enter the /LINE switch of the form

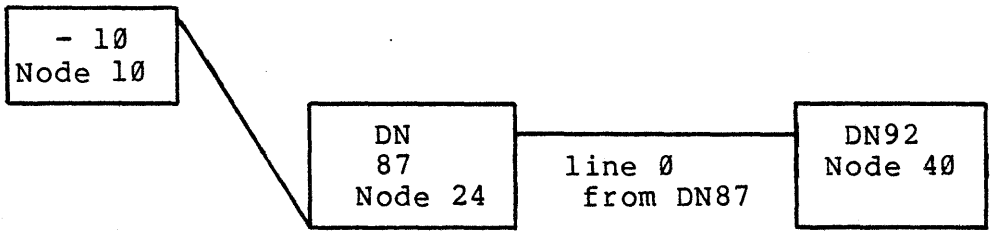
/LINE:linenum

where linenum indicates that the node to be debugged is connected to line linenum off of the node specified by /NODE.

DDT92 - DDT92 is a DDT11 that has been modified so that it can be used to examine remote running PDP-8 nodes such as the DN92. The DDT92, included on the network support tape, is a MACRO file which must be compiled and linked before it can be used. Once this has been done, DDT92 can be used to examine a remote DN92 node. Unlike DDT11, DDT92 does not recognize symbols; so, for useful work, a CREF listing of the DN92 code must be available. With that restriction, DDT92 operates with the same commands as DDT11.

To compile and save DDT92, the following procedure will do the job:

```
.EXECUTE DDT92.MAC/COMP          ;force compilation and
                                 ;linking
MACRO: DDT11
LINK: loading
[LNKXCT DDT11 Execution
DDT11 3(7)-1                     ;Version number of
                                 ;DDT92
Input: ^Z                        ;CTRL Z to exit MACRO
EXIT
.SSAVE DDT92                     ;store DDT92
DDT92 saved
.RUN DDT92                       ;run DDT92
DDT11 3(7)-1 = DDT92
Input: /NODE:24/8                ;respond to the Input
                                 prompt from DDT92 with
                                 the number of the node FROM
                                 which you wish to examine
                                 the DN92, and the PDP-8/8
                                 switch (see Figure 6-1).
```



Input:/NODE:24/8/LINE:0

DDT11 Functions

Type-out Modes - Type-out from DDT11 may contain instructions, numbers, bytes, ASCII text, EBCDIC text, or addresses.

As in DDT-10, there is a temporary as well as permanent type-out mode. The initial output mode is instruction format.

Note: The "$" character indicates that the ESC key is pressed; all numeric arguments are assumed octal.

The following are used to set the print-out modes:

Format	Sample Output	Function
$ns	ADD 4,TAG+1 ADD 4,2	Symbolic instructions. Each instruction is represented by 16, 32, or 48 bits depending on the instruction type.

Format	Sample Output	Function
$nC	69. 105	Numeric, in current radix. If n is omitted, 1 is assumed.
$nA	FOO	Set address format print-out
$nT	PQ	7-bit ASCII text. Each print-out is n characters long. If n is omitted, 2 is assumed.

$nB	201,202	Byte mode print-out. If n is not given, 2 is assumed. Each print-out is n bytes long. (8-bit bytes).
$nI	ABCD	EBCDIC print-out. Each print-out is n characters long. If n is omitted, 2 is assumed.

Radix Change - The following changes the radix of numeric print-outs:

to n (for 1<n<20): 1101011000000100	$nR	$2R could yield
hexadecimal	$16R	D604

Permanent Vs. Temporary Modes - The following commands set and terminate prevailing and temporary modes:

Format	Sample Type-in	Function
$	$C $10R	To set a temporary print-out or address mode or a temporary radix as shown in the commands above.
$$	$$C $$10R	To set a prevailing print-out or address mode on a prevailing radix in the commands above.
<CR>		To terminate temporary modes and revert to prevailing modes.

Storage Words - The following command examines storage words:

Format	Sample Output	Function
adr/	LOC/012134	To open and examine the contents of any address or symbol in current print-out mode.

Related Storage Word - The following commands examine related storage words - closing the current word (making any modification typed in), opening the following related words, and examining them in the current print-out mode

Command Format	Function
(line feed)	To examine ADR+1 (or next instruction if typeout is instruction format)
^ (up arrow)	To examine ADR-1
(TAB)	To examine the contents of the location specified by the address of the last quantity typed, and to set the location pointer to this address.
(backslash)	To examine the contents of address of last quantity typed, but not change the location pointer.
(carriage return)	To close the currently open word, without opening a new word, and revert to permanent print-out modes.

Retyping in Modes Other Than Prevailing or Temporary - Each of the following commands specifies the mode in which DDT11 should immediately reprint the last expression typed by DDT11 or the user. Neither the temporary nor the prevailing mode is altered.

=	To repeat the last print-out as a number in the current radix e.g.

1000/ JSR PC,FOO=47372546

/	To print-out, in the current print-out mode, the contents of the location specified by the address in the open instruction word, and to open that location, but not move the location pointer.

Typing In - Current print-out modes do not affect typing in; instead, the following are performed:

1234	To type in octal values

Symbols - The following is used to define DDT11 symbols:

symbol:	SYM:	To insert or redefine a symbol in the symbol table, and give it a value equal to the location pointer (.)

Special DDT11 Symbols - The following are special DDT11 symbols:

.(point)	To represent the address of the location pointer
$M	The search mask register

Arithmetic Operators - Expressions are evaluated from left to right without regard to the type of operator.

The following arithmetic operators are permitted in forming expressions:

+or BLANK>	Two's complement addition
-	two's complement subtraction
*	Integer multiplication
' (apostrophe)	Integer division (remainder discarded)
!	Inclusive OR
^X (CTRL X)	Exclusive OR
&	Logical AND

Field Delimiters in Symbolic Type-ins - The following is a field delimiter:

one or more spaces	JMP SUBRTE	To delimit op-code name

Searching - The following commands are used for searching:

a<b>c$W	200<250>0	To set a lower limit (a), an upper limit (b), a word to be searched for (c), and search for that word
a<b>c$N	351<731>0$N	To set limits and search for a not-word (i.e., locations which do not contain specified amount)
$M/	$M/ 177777	To examine the mask used in searches (initially contains all ones)

ADVANCED FEATURES

Advanced features of DDT11 include the capabilities of searching for values, dumping output to a line printer, and monitoring PDP-11 memory locations.

Masking and Searching - Searches and watching for changes in value are performed using a mask. The mask register is specified by typing $M. To open and examine the mask register, a "/" is entered following the $M. When the mask register has been opened, it can be changed by depositing a new value into it.

Two types of searches are used in DDT11. The format of a mask search is:

a<b>c$W

where a is the lower limit for the search, b is the upper limit for the search, and c is the value to be matched. The format for a not-match search is:

a<b>c$N

where a is the lower limit for the search, b is the upper limit for the search, and c is the value not to match.

If search limits are not specified, the previous limits are used. If examining a running PDP-11, the initial limits are both zero. If reading a dump file, the initial limits are the entire file.

Dumping - Occasionally, it is desirable to make a line printer dump of the DDT11 output. The $D command is used for this purpose with logical device LPT specified. Limits for the command are given as for word searches and the current output format is used. For example:

```
1000<2000>$D
```

means that locations 1000 through 2000 are to be dumped.

Monitoring a Location - When DDT11 is being used to watch a running PDP-11, it may be useful to monitor a particular location for changes. The $V command is used to monitor an opened location and displays the value in the location if it differs from the last displayed value. For example, to monitor location 160050, open the location, and then type $V.

```
160050/   63   $V
160050/   67
160050/   63
.
.
.
```

The typing of any character on the keyboard stops the $V command.

Module Test

1. Perform the debugging lab. The instructor will provide the particulars for this test.

This page intentionally left blank

Evaluation Sheet

1. Perform the debugging lab. The instructor will provide the particulars for this test.

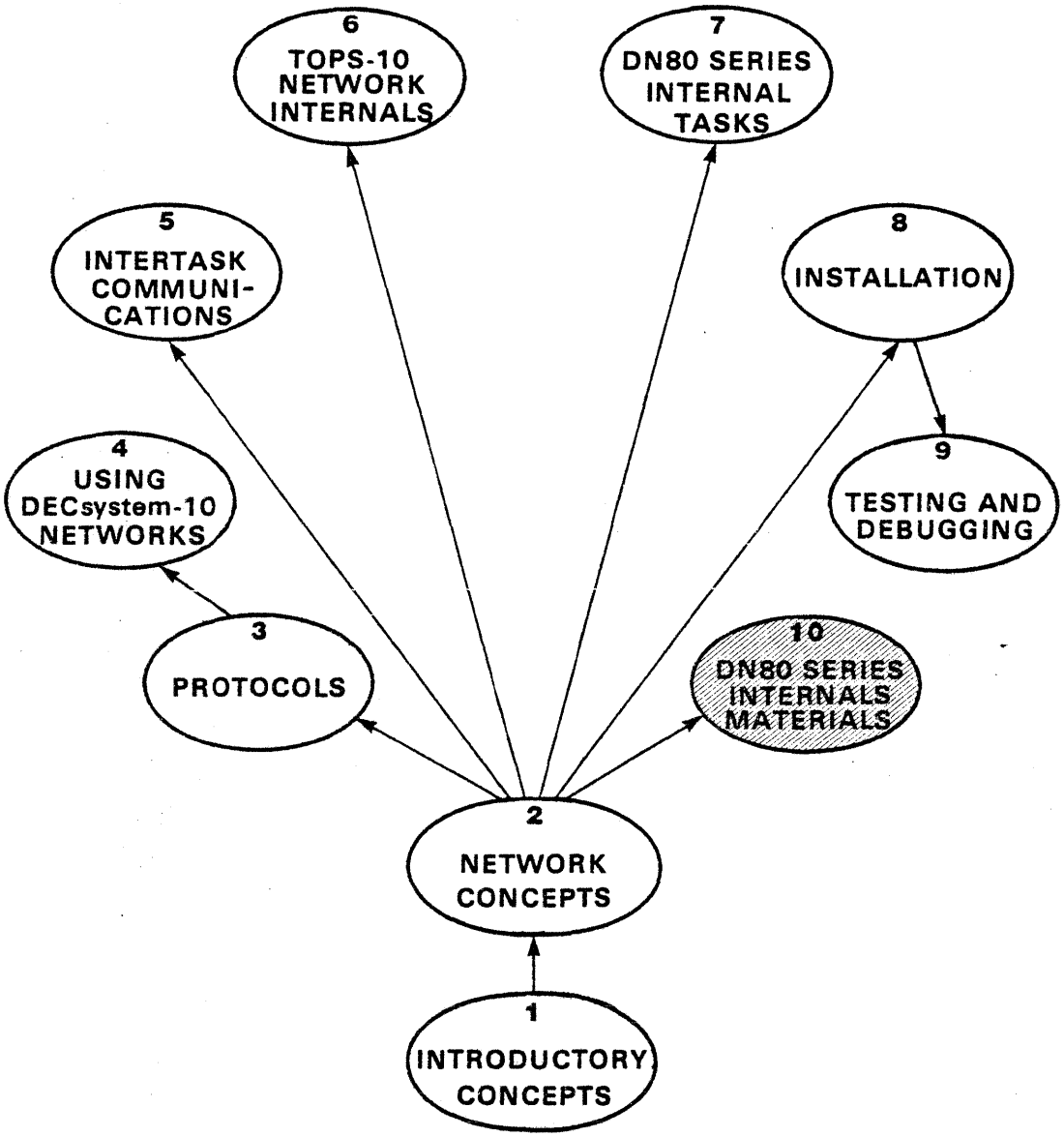
Performance will be evaluated on an individual basis.

This page intentionally left blank

DN80 Series Internals Materials
Module 10

<<For Internal Use Only>>

Course Map



M8 0277

Introduction

This module presents the data structures and flow charts of the 80-series internal organization. These materials are designed to augment reading and understanding of assembly listings (or microfiche).

Objectives

Upon the completion of this module, and class discussion the student will be able to:

1. Draw a flow chart of the PDP-11 routines in the DN8x code.

2. Delineate the major flow of control within any of the DN8x nodes, i.e., what happens before what, and when.

CHUNK WORDS
First Block of Chunk List

	Link to Next Chunk in Current Message
CN.MLK	Link to Next Message
CN.LEN	Length of Msg (Incl NCL header but not BCC)
CN.TIM	Timer Used by DDCMP
CN.DDB	Address of Device Data Block (DDB) Which Sent Message
CN.SCB	Pointer to Station Control Block (SCB) Window for Msg
CN.DDC	DDCMP Header Starts Here
CN.ADR	Address of Next Byte to Use
CN.CNT	Count of Bytes Left in Message
CN.NCN	Save Location for NCN (NCL Message Number)
	DDCMP Header BCC
CN.NCT	Offset in Message of NCL Section (referred to as NCT)
	DATA . . . .

All Succeeding Chunks as Below

Link to Next Chunk in Current Message
DATA . . .

Data Pointer Definitions:

CNKLN1=CNKSIZ-CN.NCT	Amount of Data in First Chunk
CN.DT2=2	Starting Offset to Data in Other Chunks
CNKLN2=CNKSIZ-CN.DT2	Data Length in Those Chunks

DEVICE DATA BLOCK (DDB)

DB.STS	Status Word (Note: CLRDDB clears some bits, see *)	(1)
DB.LNK	Link to Next DDB	
DB.HDW	UNIBUS Address for Device	
DB.RPC	Default Start Addr. - CLRDDB moves this to DB.OPC	
DB.TPC	PC if Timer goes off	
DB.DVT	Device Attributes	
DB.WID and DB.RLN	Width or Record Length	
DB.UNI,,DB.ACR	Unit Number | Auto crlf point	
DB.ROT,,DB.OBJ	NCL Remote Object Type | NCL Object Type	(2)
DB.CHK,,DB.MDR	Max Chunks before Data RQ | Max Data Req's if OP Dev	
DB.RCN,,DB.RNN	Node to Reconnect To | Restricted Node for Dev	(opt)
DB.OLA	Link Address for This Device	
DB.RDT	Remote Data Type	opt(3)
DB.TSK	Task Address for This Device 4 Words	(opt)
	1. Printer Get Task	
	2. Keyboard Get Task	
	3. Printer Put Task	
	4. Keyboard Get Task	
DB.ZER and DB.DCS	Start Clearing Here on Restart Device Control Status	
DB.MML	Maximum Message Length for Device	
DB.DCM	| Data Code and Mode	(4)
DB.RLA	Remote Link Address for This Device	

DEVICE DATA BLOCK (cont)

DB.SCB	Station Control Block (SCB) Address for This Device	
DB.OBF	Output (from-ten) Buffer Pointer	
DB.OLN	Length of Current Message	
DB.OCN	Count for Current Submessage	
DB.OAD	Current Byte Pointer	
DB.OPC	Addr To Run at When Run Request in Queue	
DB.COL,,DB.ODR	Column Count	Count of Output Data Req's
DB.CCN,,DB.IDR	Compressed Char Count	Count of Input Data Req's
DB.TIM	Type of Timer	Eight Bit Timer
DB.HLD	Character we are holding	
DB.VFU	Pointer To LPTVFU for LPT	
DB.CHR	Character we are uncompressing	
DB.IBF	Pointer to Input (from ten) Buffers	
DB.ICC		Input Character Count
DB.ICN	Input Message Counters: Total	
	Incremental	
	Address of Count	
DB.IAD	Input Character Address	

DB.SIZ is Size of DDB
(opt) = Optional

(DDB)

(1) Bits in DB.STS

```
DS.CAC=000001   SEND CONNECT ACCEPT
DS.DSC=000002   SEND DISCONNECT CONFIRM
DS.QUE=000004   DEVICE HAS REQUEST TO RUN IN QUEUE
DS.OUT=000010   DEVICE DOES OUTPUT(E.G. LPT)
DS.ACT=000020   DEVICE ACTIVE
DS.DIE=000040   ABORT - OTHER NODE DISAPPEARED
DS.COR=001000   DEVICE WANTS TO RUN WHEN CORE IS FREE
DS.XCH=002000   NEED TO SEND CHARACTERISTICS
DS.EPL=004000   NEED TO SEND AN ECHO PIPELINE MARKER
DS.IQU=010000   INPUT QUEUED TO NCL - USED SO TTY'S DON'T HAVE
                 TWO MESSAGES IN PIPE AT ONCE
DS.TTY=020000   DEVICE IS A TTY
DS.XDS=040000   NEED TO SEND DB.DCS TO OTHER GUY
DS.CON=100000   DEVICE IS CONNECTED
DS.CLR=C<DS.TTY!DS.OUT!DS.QUE!DS.ACT>   BITS CLRDDB WILL CLEAR
```

(2) OBJECT TYPES

```
OBJMCR=0
OBJTTY=1
OBJCTY=OBJTTY
OBJCDR=2
OBJLPT=3
OBJPTR=4
OBJPTP=5
OBJPLT=6
OBJMTA=7
OBJDTA=10
OBJTSK=11
OBJRDE=12
OBJRDM=OBJRDE
OBJRDP=OBJRDE
OBJRDA=OBJRDE
OBJCDP=13
```

(DDB)

(3) Remote data types

RDEMPT=1	Multipoint RDE type
RDEPTP=2	P-P RDE
RDEASC=4	ASCII RDE
RDEBRK=100000	ASCII type break has been seen

(4) Data Code and Mode

B0=ASCII
B1=EBCDIC
B2=IMAGE
B3=HOLLERITH (CDR only)
B4=DEC IMAGE (CDR only)
B5=Reserved
B6=Compressed Format

DH11 Block and the Associated Line Control Blocks (LCB)'s

DHnBLK	UNIBUS Address of This DH11 (0 if DH does not exist)
DHBBAR	Active Lines Bit-Mask (Line Active if Bit Set)
DHB.BN	Number of First Line in This Group
DHB.DM	UNIBUS Address of DM11BB (Modem Control) (0 if none)
DHB.VC	DH11 Unibus Address
	NOTE: There is no space left here, the LCB's follow immediately and sequentially at this point
********	********** * LINE CONTROL BLOCKS (LCB)'S * **********
LC.CAR	DM11 Modem Control Data for This Line *
LC.SPD	Encoded Speed for DH11 Line
LC.BLK	Link to Device or Line Block
LC.INS	Address of Input Service Routine
LC.OUS	Address of Output Service Routine
	DM11 Modem Control Data for NEXT Line
	Encoded Speed for Next Line
	. . .

DHB.SZ is size of DH11 block
LC..SZ is size of LCB
Bits in LC.CAR

```
        AUTBIT=100000   Sign Bit = Look for CR to start message
        ALNBIT=040000   Line is eligible for auto-baud detect
        CARBIT=020000   Carrier is present
        RNGBIT=010000   Ring is up
        DSLBIT=004000   Line is Dataset line
        LOSBIT=000077   Lost carrier on last tick
```

LINE BLOCKS

LB.STS	Status Bits	*
LB.LNK	Bit in Line Number Position	
LB.DVS,,LB.LNU	Device Service Select | Line Number	**
LB.LVL	Link Level	
LB.DHB and LB.SLA	Device Control Block for DH11 (if DH line) or... Synchronous Line UNIBUS Address (if synch line)	
LB..LN and LB.SLV	4 Bit DH11 Line Number (if DH line) or... Synchronous Line Vector (if synch line)	
LB.LCB	Line Control Block Address (DH11)	
	Optional Multipoint Locations	(opt)
LB.OCN	Count of Messages Sent and ACK'd by other end	
	Count of all NAK's (including next 3 counts)	
	Count of NAK's for REP Response	
	Count of NAK's for BCC	
	Count of NAK's for No Room	
LB.ICN	Count of Messages Received ok	
	Count of Bad Messages	
	Count of REP's which elicited NAK's	
LB.BNN	Node for Booting | Timer for Booting	
LB.ZER and LB.SCB	Zero from Here On Reset Pointer to Station Control Block (SCB)	
LB.LAR,,LB.ROK	Last Msg ACK Rcvd | Last Msg Rcvd ok	
LB.HSN,,LB.LAP	Highest Msg Sent | Last Msg ACK Processed	
LB.REP	REP's With No Response | REP Timer (1/sec)	
LB.TRY,,LB.NCD	Count of BCC NAK's Rcvd for 1st Msg in OP Q | Last NAK Code Sent	
LB.TRY	Count of BCC NAK's Rcvd for 1st Message in Output Queue	(opt)
LB.XDN	Synch Line XMT-DONE Interrupt Vector	

LINE BLOCK (cont)

LB.CTL	Next Control Message to Transmit (8. bytes)	
LB.COB	Current Output Buffer 1. Pointer to 1st Chunk of Current Msg	
	2. Pointer to Current Chunk	
	3. Number of Bytes Left	
LB.BOO	Pointer to Bootstrap Msg to Send	
LB.OBF	Output Buffers 1. Pointer to First Buffer	
	2. Pointer to Last Buffer (or zero)	
	3. Number of Messages in Queue	
LB.RDN	Dispatch on Receive Done	
LB.CIB	Current Input Buffer (used by interrupt level) 1. Current Chunk Address	
	2. Characters Left in Message	
LB.SRX	NON-0 if Async DDCMP used or if No DQ11's 4 Words -- Pair of Dummy Address, Word Count for Xmt.	(opt)
LB.SRR	4 Words -- Pair of Dummy Address, Word Count for Rcv.	(opt)
LB.SLB	1. Count of Synchronous Line Error Interruptions	(opt)
	2. Last Error Interrupt HDW Status	(opt)
	3. Count of Synchronous Line Xmt. Timeouts	(opt)
LB.CTY	Addr of String for Line	(opt)
LB.CRS	Synchronous Interface Status at Crash	(opt)
LB.IPT	Input Putter (Relative to Line Block)	
LB.ITK	Input Taker (Relative to Line Block)	
LB.IBF	Input Buffer 1st 8 Chars are Control Msg or Data Msg Header For Data Msgs, 5th Word is Link to Chunks	

LB.SIZ is size of Line Block

(Line Block)

* Bits in LB.STS

LS..ST=000001	Need to send a START
LS.STK=000002	Need to send a STACK
LS.XNK=000004	Need to send a NAK
LS.XAK=000010	Need to send an ACK
LS.XRP=000020	Need to send a REP
LS.NRP=000040	Need to send REP response
LS..RQ=000100	Receiver interrupt queued
LS..XQ=000200	XMT DONE interrupt queued
LS..RG=000400	Synch line receiver active
LS..XG=001000	Synch XMT active, (may be idling)
LS..RN=002000	Received NAK
LS.XCT=004000	Line is transmitting control msg
LS.XDT=010000	Line is transmitting data
LS.SSY=020000	Must strip SYNC before next message
LS.MPT=040000	Line is multi-point line
LS..SS=100000	Stripping SYNC's now

** Bits in LB.DVS

LS..DP=0000	DP11 Device line
LS..DS=0002	DS11 Device line
LS..DU=0004	DU11 Device line
LS..DV=0006	DV11 Device line
LS..DQ=0010	DQ11 Device line
LS..AL=0012	Lowest Async Dev Type Code
LS..DH=0012	DH11 Device line

STATION CONTROL BLOCK (SCB)

SB.FLG	Flags	*
SB.LAN,,SB.HXN	Last ACK'd NCL Msg Number | Highest NCL Msg Xmitted	
SB.TIM,,SB.RMN	Timer - for REP and START | Receive Message Number	
SB.IMQ	Input Message Queue	
SB.OMQ	Output Message Queue	
SB.SQS	Length of Sequential Node Ctl Info (below) (variable length)	
SB.LBA	Addr of Line Data Block (LDB) for Station (Best Choice)	
	Level for Best Choice	
	Address of LCB for 2nd Best Choice	
	Level for 2nd Best Choice	
SB.NGH	Routing Information: For NGHMAX (Maximum of Neighbors) A Pair of Entries for Each Node consisting of: SCB Address Link Level	
SB.SNM	NOTE: The Rest of the Table is NOT cleared by CLRSCB Station Name (Text) 46 octal Bytes	
SB.SID	Software ID (Text) 54 octal Bytes	
SB.DAT	Software Date (Text) 72 octal Bytes	
SB.NNM	Node Name (Binary Number)	
SB.DFA	Drop Went Down at This Uptime	(opt)
SB.WHA	Address of Caller of L.DOWN	(opt)
LB.WHN,,LB.WHS	Seq No. of Last ACK'd Msg | Copy of LB.OBF+4	(opt)

SB.SIZ is size of SCB

(SCB)

* Bits in SB.FLG

SBF.IU=000001	SCB in use (For TENSCB,means Port Enabled)
SBF.IC=000002	In Contact (i.e. Have START/STACK exchanged)
SF.HID=000004	Have Node ID for This Station
SBF.NB=000010	Need to Send NEIGHBORS to Node... Set on Contact and when NGH Changes
SBF.RP=000020	REP is Outstanding
SBF.RR=000040	Owe REP RESPONSE
SF.XAK=000100	Need to Send NCL-ACK to Station
SBF.NK=000200	Need to Send NCL-NAK
SBF.SK=000400	Need to Send NCL-STACK
SBF.SQ=001000	Node is Sequential
SBF.NQ=002000	Has Request in NCL Queue
SF.XCN=004000	Need to Send CONFIFURATION
SF.XRC=010000	Need to Send REQUEST CONFIGURATION
SF.MCR=020000	Node has Command Decoder
SF.FAK=040000	Future ACK - ACK Rcvd but Msg not back from DDCMP

```
                STOPCD'S    (Stop Codes)
S..AMC= 0       ASSERT MACRO CALL - DEFAULT CODE
S..NXM= 1       BUS TRAPS'S, ADDRESS ERROR'S, ETC.
S..DL10=2       DL10 ERRORS
S..DTE= 2       DTE20 ERRORS (NOTE - SAME AS DL10)
S..CNK= 3       CHUNKS ARE MESSED UP
S..ILS= 4       ILLEGAL INSTRUCTION
S..CTY= 5       NO CTY
S..MEM= 6       MEMORY ERROR (E.G. PARITY, OR CAN'T READ WRITE BITS)
S..KW11=7       KW11 ERROR
S..NCN=10       NO CONNECTION FOR RECEIVED DATA
                 OR CONNECTION NUMBER USED BY SOME OTHER NODE
S..BDT=11       BAD DATA TYPE REQUESTED BY 10
S..CHK=12       CHECK 11 ERROR
STOP CODE MACRO
        FIRST ARGUMENT IS CODE FOR STOP
        SECOND ARGUMENT IS SEVERITY
.MACRO  STOPCD  CD,TYPE
.IIF NB <CD>,   S.....=S..'CD
.IIF B <CD>,    S.....=0
        Z=1
.IIF IDN <.'TYPE>,<.DEBUG>,Z=DEBUG
.IIF NE Z,      TRAP S.....
        S.....=0
.ENDM   STOPCD
```

Task Blocks

TK.STS	Status Word	*
TK.LNK	Link to Next Task Block	
TK.RQL	Run Queue Link	
TK.PRI	Pointer to Priority Queue	
TK.JSA	Address to Continue Task	
TK.RSA	Address to Use on Restart	
TK.PDL	1. Address of Push Down List (for this task)	
	2. Current Push Down List Pointer (this task)	
TK.ZER and TK.TIM	Start Clearing Here on Restart Seconds Timer	
,,TK.QTM	Quantum Time - Counts Ticks	
TK.TPC	Where to Go When Clock Goes Off	
TK.DTK	Address of This DB.TSK	
TK.SPT	Send Queue Putter	
TK.STK	Send Taker	
TK.SQU	Send Queue Size	
TK.ARG	Save Value to Return to Caller	

TK.SIZ is size of Task Block

* Bits in TK.STS

TK.RUN=100000	Task is Runnable
TK.LGI=040000	Job is Logged in - i.e. Hasn't Existed Yet
TK.TRG=020000	Task Has Been Triggered by Another Task
TK.WAK=010000	Something Has Occured to Wake Task
TK.NOP=004000	Device on This Call was not Opened
TK.SLP=002000	Sleep Bit - Will Not Wake Until Specified No. of Counts (Jiffies) or Host Dies

To Ten Blocks

Note: Order is Significant

TT.FLK	Forward Link
TT.RLK	Reverse Link
TT.ALC	*obs* Space Allocated for This Block
TT.HDL	Length of Header
TT.QHD	1st QPR Word, Length of Header
TT.QFN	Function
TT.QDV	Device (NCL)
TT.QSP	Spare
TT.QFW	First Word - Line, Ind Msg Length
TT.ADR	Address of Real Data
TT.USR	User Supplied Data
TT.EFN	*obs* - Event Flag No.

To Eleven Blocks

Note: Order is Significant

TE.LNK	Address of Next Chunk in Message
TE.QPR	No. of Bytes Left to Xfr in QRP Message
TE.LEN	Total Length of Message
TE.FFW	Copy of 1st Word for Current QPR Message
TE.LIN,,TE.LNK	Line, NCL flags | Space Left in Current Chunk
TE.QHD	QPR Message Header - Length of 1st Msg
TE.QFN	Function
TE.QDV	Device (Better be NCL)
TE.QSP and TE.CMP	Spare Word - Used fo Start of User Data for Comp
TE.ADR	Addr to Put Data of Next Fragment

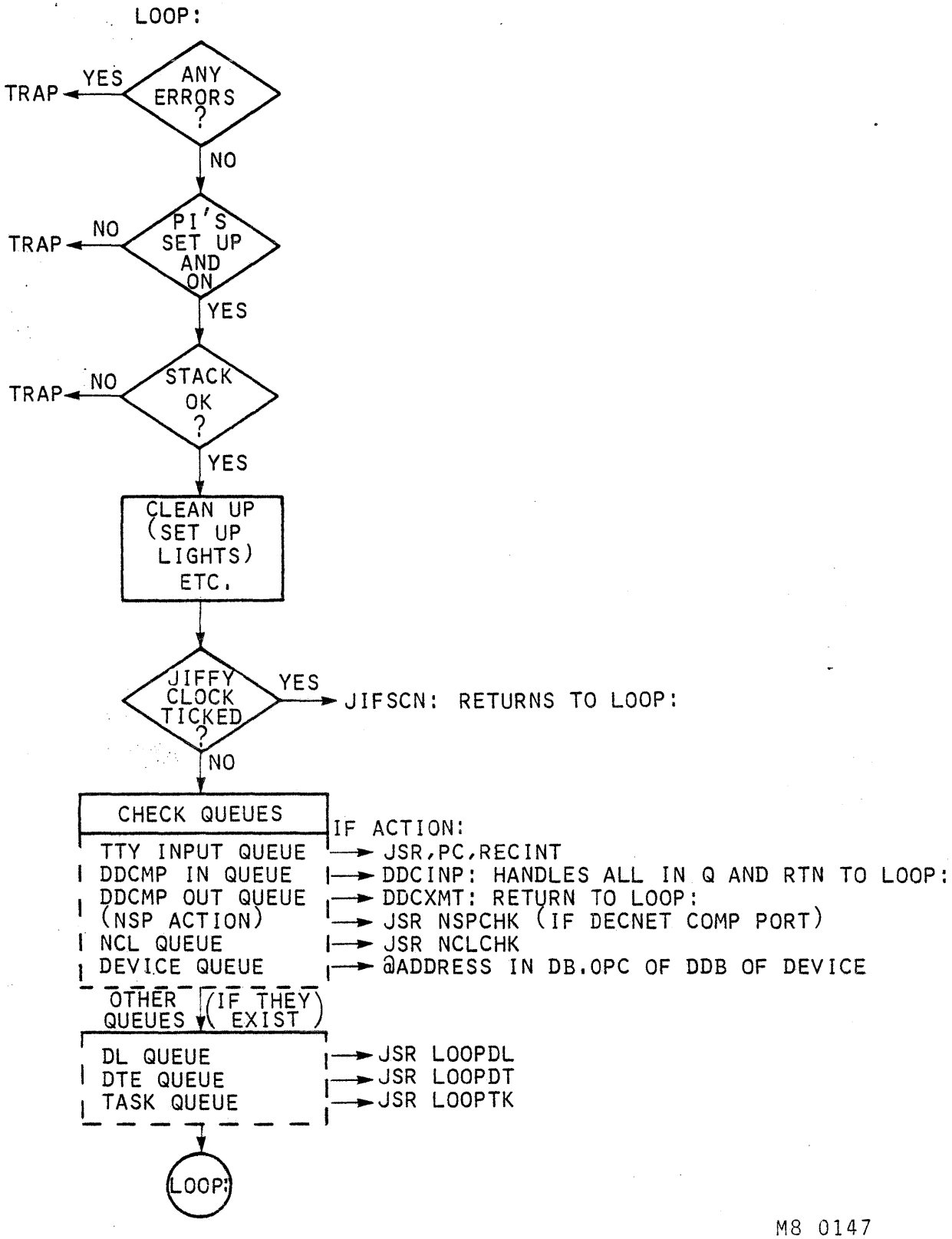
Device Dependent Section of DDB (follows other part directly)

TTY DDB

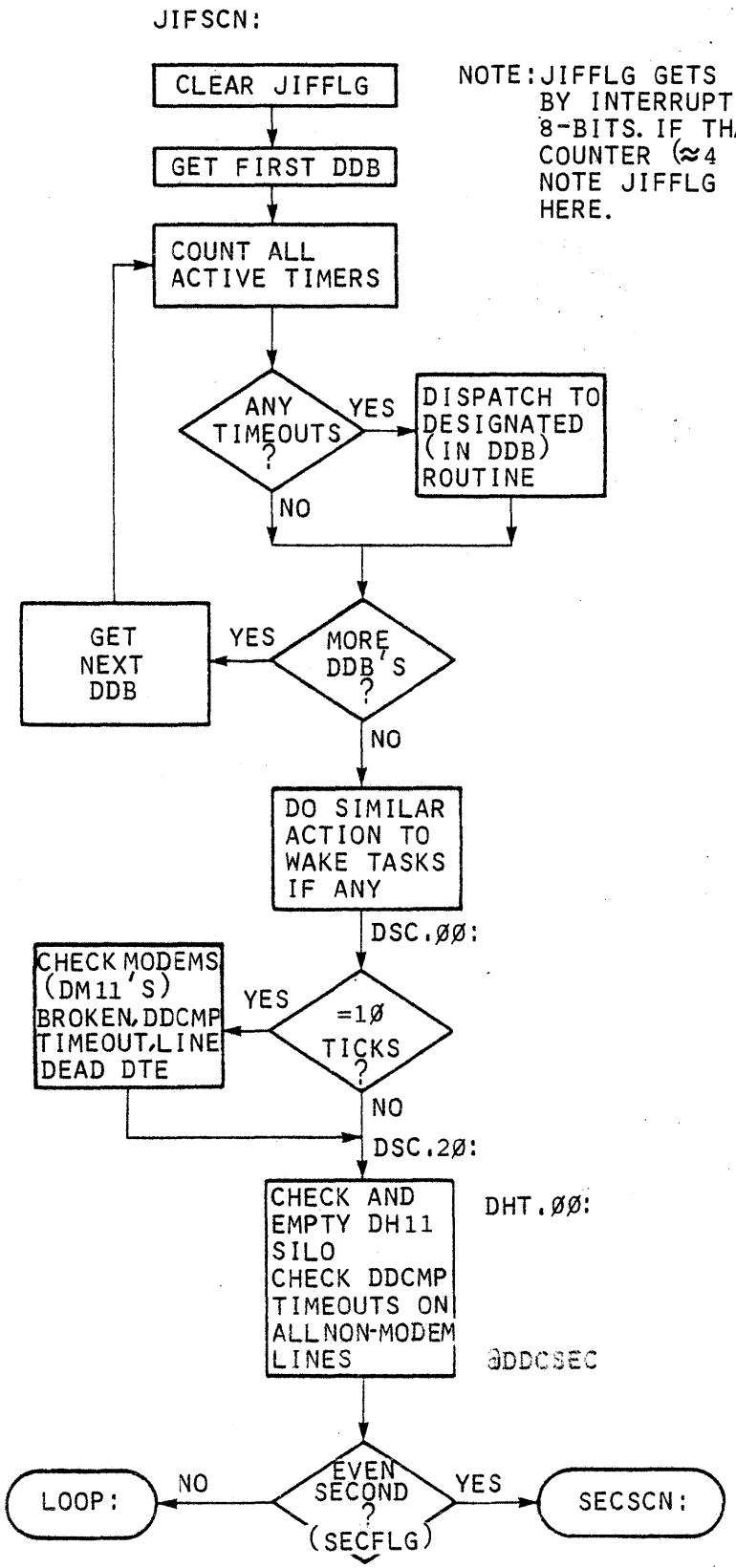
DB.BIT	Bit per Line (DH11 line = Bit)	
DB.FIL,,DB..LN	Filler for 10 thru 15 | 4 Bit Line Number	
	More Filler	
	More Filler	
DB.EPL,,	Echo Pipeline Serial | More Filler	
DB.LCB	Line Control Block (LCB) Address	
DB.DNT,,DB.DNS	DN11 Timer (Secs) DN11 Table Disp, Sign=Dial	(opt)
DB.TZR and DB.DNR	Clear From Here on Restart DN11 Request Word	(opt)
DB.BCD	If Non-0, Line is 2741	*
DB.STR	Pointer to String to Type	
DB.TOC	Count of Output Characters in Chunk	
DB.TOB	TTY Output Pointers: -- This is Ptr to 1st Char	
	This is Pointer to Last Character	
DB.BUF,,DB.ASP	Char for DH11 to Type | ASAP Char (priority char)	
DB.FTM	Fill Time on Character	
DB.PCN	Printer Count (Count of Chars from NCL)	(opt)
DB.PPT	Printer Putter	(opt)
DB.PTK	Printer Taker	(opt)
DB.KPT	Keyboard Putter	(opt)
DB.KTK	Keyboard Taker	(opt)
DB.KQU	Queue for Keyboard	(opt)
	Second Word for Fill Time	

DB.TSZ is size of TTY DDB

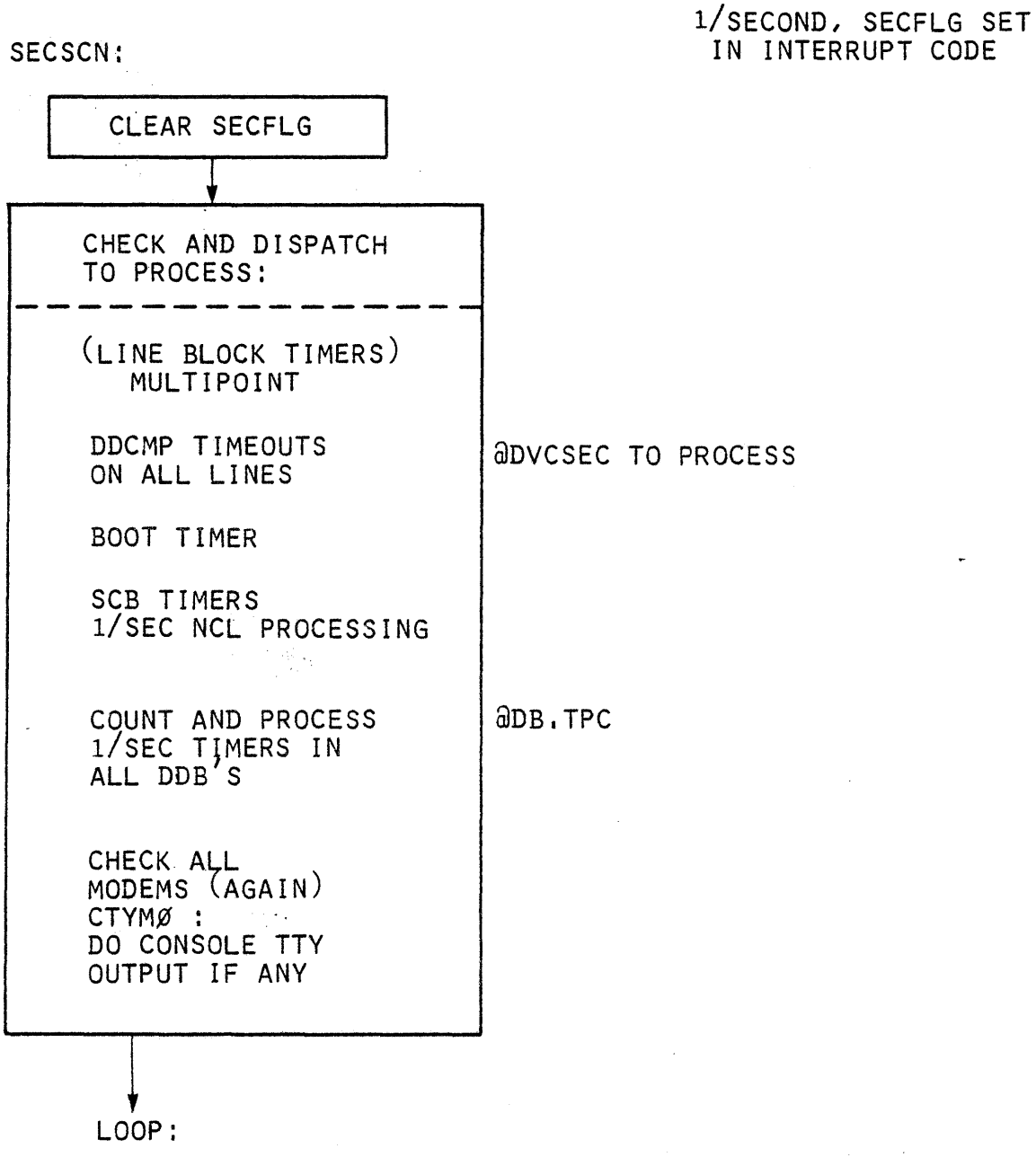
```
                                        (TTYDDB)
*  Bits in DB.BCD
        BCD274=100000    ;I AM A 2741
        BCDXRB=040000    ;SENDING REVERSE BREAK
        BCDKBL=020000    ;IF KEYBOARD IS LOCKED
        BCDPRL=010000    ;IF PRINTER IS LOCKED
        BCDCDB=004000    ;LAST TIME WE REVERSED LINE IT WAS BECAUSE
                         ; WE NEEDED INPUT
        BCDCOD=003400    ;MASK FOR CODE
        BCDBRK=000200    ;PROCESSING A RECEIVED BREAK
        BCDUPS=000100    ;IF IN UPPERSHIFT MODE
        BCDOCR=000040    ;LAST CHARACTER OUT WAS A CR
        BCDRCR=000020    ;LAST CHARACTER RECEIVED WAS A CR
        BCDCON=000010    ;LAST CHARACTER IN WAS "CONTROL" FAN
        BCDTDY=000004    ;TTY TIDY MODE
        BCDAPL=000002    ;SPECIAL "APL-MODE"
                         ; NO SPECIAL HANDLING FOR UPARROW
                         ; BREAK IS TO BE TREATED AS CC
        BCDHDB=000001    ;TERMINAL HAS DEBREAK FEATURE
        BCDB27=C<BCDAPL> ;ALL 2741 BITS
                         ; AS SHIFT HAS BEEN SENT
```


LOOP:
ANY ERRORS ?
YES
TRAP
NO
PI'S SET UP AND ON
NO
TRAP
YES
STACK OK ?
NO
TRAP
YES
CLEAN UP (SET UP LIGHTS) ETC.
JIFFY CLOCK TICKED ?
YES
JIFSCN: RETURNS TO LOOP:
NO
CHECK QUEUES
IF ACTION:
TTY INPUT QUEUE
JSR,PC,RECINT
DDCMP IN QUEUE
DDCINP: HANDLES ALL IN Q AND RTN TO LOOP:
DDCMP OUT QUEUE
DDCXMT: RETURN TO LOOP:
(NSP ACTION)
JSR NSPCHK (IF DECNET COMP PORT)
NCL QUEUE
JSR NCLCHK
DEVICE QUEUE
@ADDRESS IN DB.OPC OF DDB OF DEVICE
OTHER QUEUES (IF THEY EXIST)
DL QUEUE
JSR LOOPDL
DTE QUEUE
JSR LOOPDT
TASK QUEUE
JSR LOOPTK
LOOP:

M8 0147

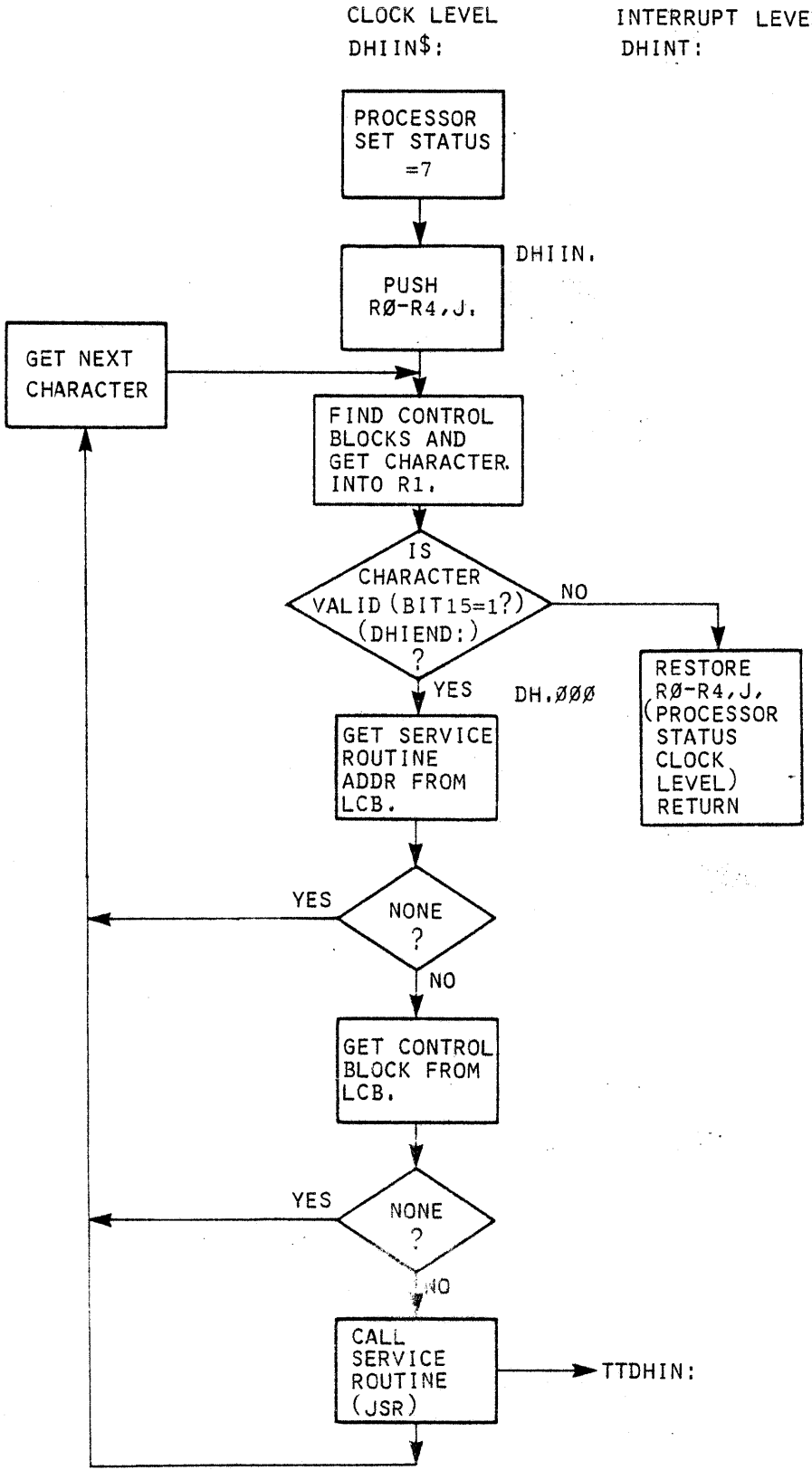

JIFSCN:
CLEAR JIFFLG
GET FIRST DDB
COUNT ALL ACTIVE TIMERS
ANY TIMEOUTS ?
YES
DISPATCH TO DESIGNATED (IN DDB) ROUTINE
NO
GET NEXT DDB
YES
MORE DDB'S ?
NO
DO SIMILAR ACTION TO WAKE TASKS IF ANY
DSC.ØØ:
CHECK MODEMS (DM11'S) BROKEN, DDCMP TIMEOUT, LINE DEAD DTE
YES
=1Ø TICKS ?
NO
DSC.2Ø:
CHECK AND EMPTY DH11 SILO
CHECK DDCMP TIMEOUTS ON ALL NON-MODEM LINES
DHT.ØØ:
@DDCSEC
LOOP:
NO
EVEN SECOND ? (SECFLG)
YES
SECSCN:
NOTE: JIFFLG GETS INCREMENTED BY INTERRUPT CODE (KW11) 8-BITS. IF THAT CODE WRAPS COUNTER (≈4 SEC) TRAP OCCURS, NOTE JIFFLG IS CLEARED HERE.

M8 0146

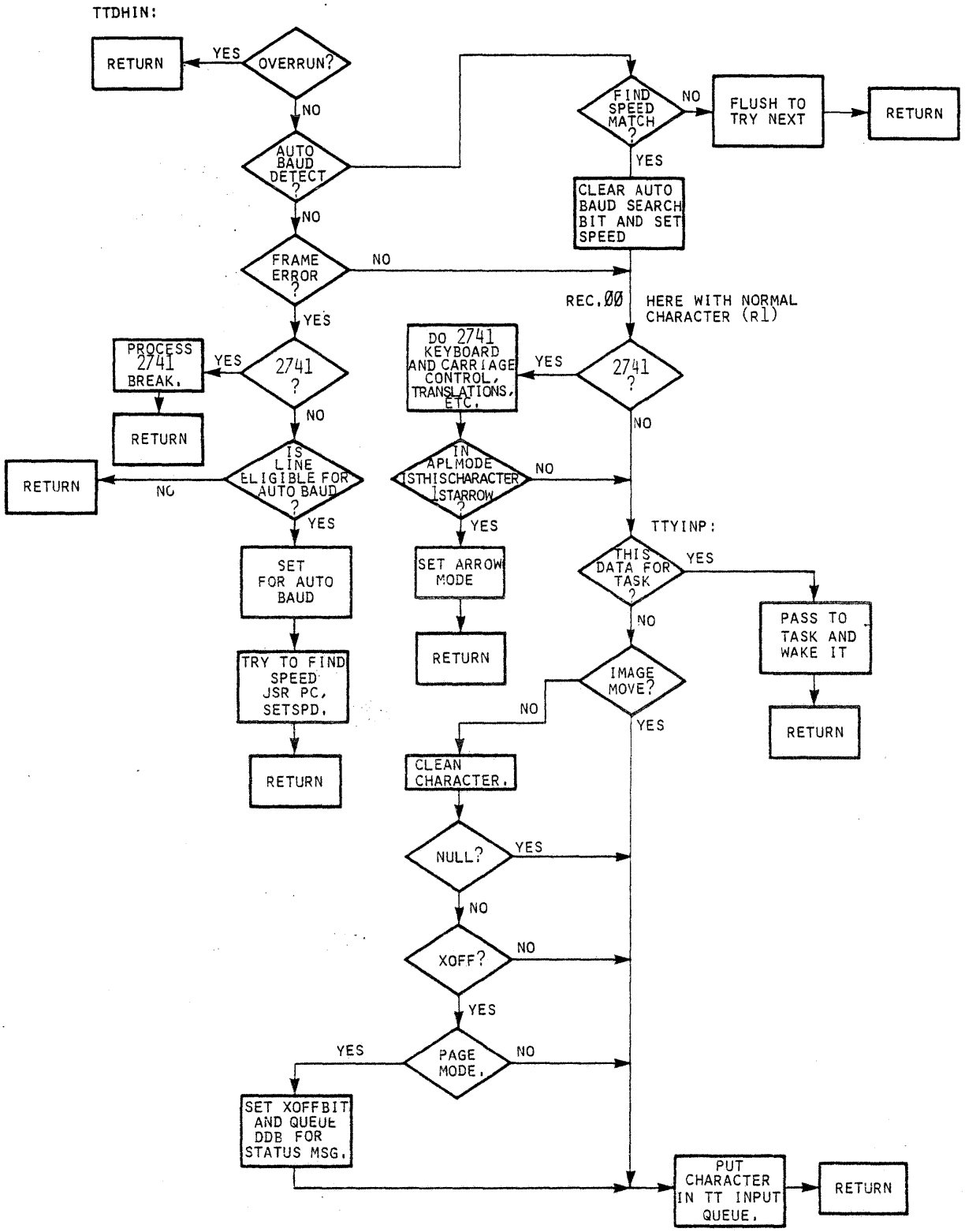

1/SECOND, SECFLG SET
IN INTERRUPT CODE
SECSCN:
CLEAR SECFLG
CHECK AND DISPATCH
TO PROCESS:
(LINE BLOCK TIMERS)
MULTIPOINT
DDCMP TIMEOUTS
ON ALL LINES
@DVCSEC TO PROCESS
BOOT TIMER
SCB TIMERS
1/SEC NCL PROCESSING
COUNT AND PROCESS
1/SEC TIMERS IN
ALL DDB'S
@DB.TPC
CHECK ALL
MODEMS (AGAIN)
CTYMØ :
DO CONSOLE TTY
OUTPUT IF ANY
LOOP:

M8 0130

DH
RECEIVE
SERVICE
ROUTINE



M8 0133


TTDHIN:
OVERRUN?
YES
RETURN
NO
AUTO BAUD DETECT ?
NO
FRAME ERROR ?
NO
YES
2741 ?
YES
PROCESS 2741 BREAK.
RETURN
NO
IS LINE ELIGIBLE FOR AUTO BAUD ?
NO
RETURN
YES
SET FOR AUTO BAUD
TRY TO FIND SPEED JSR PC, SETSPD,
RETURN
FIND SPEED MATCH ?
NO
FLUSH TO TRY NEXT
RETURN
YES
CLEAR AUTO BAUD SEARCH BIT AND SET SPEED
REC.ØØ
HERE WITH NORMAL CHARACTER (R1)
2741 ?
YES
DO 2741 KEYBOARD AND CARRIAGE CONTROL, TRANSLATIONS, ETC.
IN APLMODE ISTHISCHARACTER ISTARROW ?
NO
YES
SET ARROW MODE
RETURN
NO
TTYINP:
THIS DATA FOR TASK ?
YES
PASS TO TASK AND WAKE IT
RETURN
NO
IMAGE MOVE?
NO
YES
CLEAN CHARACTER.
NULL?
YES
NO
XOFF?
NO
YES
PAGE MODE.
YES
NO
SET XOFFBIT AND QUEUE DDB FOR STATUS MSG.
PUT CHARACTER IN TT INPUT QUEUE.
RETURN

M8 0132

Module Test

1. Select a routine for which no flow chart was given and create a flowchart for it. Your instructor may suggest a routine.

2. From memory, and in order, list the queues which are processed in LOOP:, Note which processing routines return in place and which return to the top of LOOP:.

This page intentionally left blank

Evaluation Sheet

1. Select a routine for which no flow chart was given and create a flowchart for it. Your instructor may suggest a routine.

   Answer to be evaluated by the instructor.

2. From memory, and in order, list the queues which are processed in LOOP:, Note which processing routines return in place and which return to the top of LOOP:.

   Check your answer with the flowcharts in the module text.

This page intentionally left blank

Appendix A

This page intentionally left blank

Data Structures

Name: CHUNK

Description: The first chunk in a message contains no data, only control information. The remainder of the chunks in a message contain the first two control words and then the content of the message.

Defined in: DEFINE

CHNXT	Ptr. to Next CHUNK
CHLEN	Bytes of Data in CHUNK
MSGNXT	Ptr. to Next MSG in Queue
MSGPRV	Ptr. to Previous MSG in Queue
MSGLCH	Ptr. to Last CHUNK in MSG
MSGCLP	Ptr. to "CHUNK length" Field of Last CHUNK
MSGPTR	Ptr. to 1st Empty Char. Position in MSG (in last CHUNK)
MSGCNT	Count of Bytes Left in Last CHUNK
MSGBCC	Accumulated BCC
MSGSNL	No. of Remaining Bytes Before SYN Insertion
MSGLEN	Overall Length of MSG
MSGID	MSG ID
MSGFGS	Flags
MSGNLR	No. of Logical Records in MSG
	Unused Bytes in 1st CHUNK of MSG
	/ /

MSGFGS: MSGTSP = B0 MSG received in transparent mode

Name: KCB

Description: KMC11 Control Block. One KMC11 exists for each set of 4 synchronous lines on the DN20; there is a KMC11 Control Block for each KMC11 in the system.

Defined in: DEFINE

MDFGE	PDP-11 Flags
MDFGK	KMC11 Flags
MDALE	PDP-11 Alive Counter
MDALK	KMC11 Alive Counter
MDLCB	LCB Ptr. for Line 0
	LCB Ptr. for Line 1
	LCB Ptr. for Line 2
	LCB Ptr. for Line 3
MDTIC	Counter for One Second Code
MDALKS	Previous KMC11 Alive Counter

MDFGE:
```
MDFER = B0   PDP-11 is running
MDFEA = B4   Active toggle
```

MDFGK:
```
MDFKR = B0   KMC11 is running
MDFKA = B4   KMC11 Active Response
```

Name: LCB

Description: Line Control Block. One LCB exists for each enabled communications line. It contains status information and pointers to other control blocks pertinent to the line.

Defined in: DEFINE

LB.STS	Status Bits
LB.LNU	Line Number
LB.SLA	Address of DUP11 CSR
LB.SLV	Address of DUP11 Interrupt Vector
LB.MSG	Ptr. to MSG Being Sent
LB.CMA	Ptr. to Control MSG Being Sent
LB.CMC	Length of Control MSG Being Sent
LB.SLO	Ptr. to Input Silo Control Block
LB.SO1	1st KMC11 Silo
LB.SO2	2nd KMC11 Silo
LB.MD	Ptr. to KMC11 Control Block
LB.SE1	Count of Line Error Interrupts
LB.SE2	Status Reg. 1 at Last Error
LB.SE3	Status Reg. 2 at Last Error
LB.SE4	Count of Receiver "Not Fast Enough"
LB.SE5	Count of Transmitter "Not Fast Enough"
LB.SE6	Count of Clear-to-Send Failures
LB.OC1	Count of MSG's Sent and ACKed

LB.OC2	Count of NAK's Received (plus wrong ACK's)
LB.OC3	Count of Invalid Responses to TTD
LB.OC4	Count of Invalid Responses to MSG's
LB.OC5	Count of TTD's Sent
LB.OC6	Count of WACK's Received in Response to MSG's
LB.OC7	Count of EOT's in Response to MSG's
LB.OC8	Count of Invalid Bids or Responses to Bids
LB.OC9	Count of RVI's Received While Transmitting
LB.IC1	Count of MSG's Received O.K.
LB.IC2	Count of Bad BCC's
LB.IC3	Count of NAK's Sent in Response to Data MSG's
LB.IC4	Count of WACK's Sent
LB.IC5	Count of TTD's Received
LB.IC6	Count of EOT's Sent or Received Which Abort the Stream
LB.IC7	Count of MSG's Ignored
LB.IC8	Count of MSG's with Invalid Char. Following DLE
LB.IC9	Count of Attempts to Change Modes in a MSG
LB.TTO	Count of Transmitter Timeouts
LB.CSD	Clear-to-Send Delays in JIFFY's
LB.SE7	Count of Silo Overflows
LB.MDS	Length of Silo Warning Area
LB.MDU	Max. Warning Used Since Depth Last Set
LB.TYP	Line Driver Type
LB.EQW	Time to Wait for ENQ

LB.EQN	Number of ENQ's to Send
LB.DEW	Ticks to Wait Before Enabling Modem Interrupts
LB.RTY	Times to Retry Current Operation
LB.CH1	Ptr. to Primary CHUNK (input)
LB.CH2	Ptr. to Secondary CHUNK (input)
LB.CH3	Ptr. to Primary CHUNK (output)
LB.CH4	Ptr. to Secondary CHUNK (output)
LB.MSC	Count of MSG's Waiting to be Sent
LB.CHD	Count of Times CHUNK's Depleted
LB.TC1	Ptr. to TCB for BSC Driver
LB.TC2	Ptr. to TCB for Translater
LB.DIC	Count of Dataset Interrupts
LB.DIS	Status of Dataset Interrupt
LB.DIP	Status of Dataset Interrupt
LB.ERS	Error Bits Stored Here
LB.XCL	Count of Times Transmitter Clock Lost
LB.RCL	Count of Times Receiver Clock Lost
LB.XST	Last Transmitter Status Register
LB.XCT	Count of Transmit and Status Interrupts
LB.RST	Last Receiver Status Register
LB.RCT	Count of Receiver Interrupts
LB.CCH	CHUNK Currently Being Received
LB.CCR	Ptr. to Length Field of CHUNK Being Filled
LB.CCD	Ptr. to Current Data Location in CHUNK Being Filled
LB.DVT	Device Type

LB.FGS	Flags	*
LB.CRD	Number of Chars. Read So Far in This MSG	

LB.STS:

LS.KIL = B0	Kill current function
LS.CMP = B1	Function complete
LS.CIE = B2	Post BSC task on next char.
LS.LWR = B3	Last write - no more read
LS.CTL = B4	Writing a control message
LS.CAR = B5	Carrier since starting receiver
LS.XND = B6	Transmission is ending
LS.MDE = B7	KMC11 has detected an error
LS.XRN = B8	Transmitter running
LS.RRN = B9	Receiver running
LS.XGO = B10	Waiting for delayed clear-to-send to start transmitter
LS.RGO = B11	Waiting for write initiation to start receiver
LS.KLC = B12	"KILL" has completed
LS.RSE = B13	Receive silo has been emptied
LS.ERR = B14	Error on the line
LS.ENB = B15	Line has been enabled

LB.TYP:

1 = DQ11
2 = KMC11/DUP11
3 = DUP11

LB.FGS:

LF.SIM = B0	Simulate (emulate) mode
LF.PRI = B1	Primary timeout

Name: SCB

Description: Input Silo Control Block. Each synchronous line has two character "silos" associated with it. Status information for these silos is maintained in separate control blocks which are pointed to by the Line Control Block.

Defined in: DEFINE

MDSLFG	Flags	*
MDSLPT	Silo Ptr. - Next Char. Goes Here	
MDSLPW	Silo Warning Level	
MDSLPL	Silo Limit	
MDSLCC	Ptr. to Current CHUNK	

MDSLFG:

MDSLFE = B0	KMC11 should interrupt the PDP-11 when storing data in the silo
MDSLFF = B1	Silo is full
MDSLFO = B2	Silo has overflowed
MDSLFW = B4	Silo has reached warning level
MDSLPG = B7	KMC11 has received pointers

Name: TCBSC

Description: Task Control Block for BSC driver task.

Defined in: DEFINE

TCEW	Event word	
TCHAIN	Ptr. to Next TCB	
TCPC	Task's PC When Inactive	
TCPS	Task's PS When Inactive	
TCSP	Task's Stack Ptr. When Inactive	
TCSPT	Ptr. to Base of Stack Storage	
TCMSG1	Ptr. to Oldest MSG Queued to This Task	
TCMSG2	Ptr. to Newest MSG Queued to This Task	
TCCHK1	Ptr. to Newest CHUNK Queued to This Task	
TCGMC	Counter for Waiting for Storage	
TCLCB	Ptr. to LCB for This Task	
TCTIM	Wait Time (in Jiffies)	
TCST2	Secondary Status Bits	*
TCFG1	Flag Bits	*
TCFG2	Flag Bits	*
TCCMA	Ptr. to Control MSG to Prompt for Next Data MSG	
TCCMC	Length of Control Message	

TCEW:	EBFCHK=100	Waiting for free CHUNK
	EBQCHK=40000	Waiting for queued CHUNK
	EBINTR=20000	Waiting for device interrupt routines
	EBCOMP=10000	Waiting for $COMPQ to drain
	EBTENI=4000	Waiting for -10 interrupt
	EBQMSG=2000	Waiting for a MSG
	EBPERM=1000	Waiting for permission from HASP
	EBDATA=400	Waiting for queue to need data
	EBREQP=200	Waiting for HASP to request permission
	EBQPER=40	Waiting for operator response
	EBTIME=20	Waiting for timer
	EBTXDN=10	Waiting for "to -10 done" interrupt
	EBTEDN=4	Waiting for "to -11 done" interrupt
	EBWAIT=1	Waiting

TCST2:	TCLBK=B0	Current output line has been broken
	TCDTA=B1	There is data in this record
	TCEOT=B2	An EOT has been sent or received
	TCESC=B3	Last char. from PTR was ESC
	TCIGS=B4	Last char. was IGS
	TCXET=B5	Block ended in ETX, expect EOT next
	TCAK1=B6	Sending: next ACK expected is ACK-1 Receiving: last ACK response was ACK-1
	TCNRD=B7	No response to data message. After sending ENQ, accept wrong ACK as NAK

TCFG1:	TCPRI=B4	Interpret PTR carriage control on input
	TCPRO=B5	Interpret PTR carriage control on output
	TCTSP=B6	Do output in transparent BISYNC
	TCCMP=B7	Do component selection
	TCCPS=B8	Use compress/expand functions
	TCPCO=B9	Page counter has overflowed
	TCPCE=B10	Page counter interrupts enabled
	TCOBS=B11	Use "old" (2780) BISYNC
	TCDMP=B12	Output being dumped

TCFG2:	TCOPR=B0	Output permission requested
	TCOPG=B1	Output permission granted
	TCORN=B2	Output running
	TCOEF=B3	Output EOF started
	TCOEC=B4	Output EOF completed
	TCOAB=B5	Output abort started
	TCOAC=B6	Output abort completed
	TCIPR=B7	Input permission requested
	TCIPG=B8	Input permission granted
	TCIRN=B9	Input running

```
TCIAB=B10  Input abort started
TCIAC=B11  Input abort completed
TCIEC=B12  Input EOF completed
TCIWR=B13  Input permission was requested
```

Name: TCDTE

Description: Task Control Block for DTE driver task.

Defined in: DEFINE

TCEW	Event word
TCHAIN	Ptr. to Next TCB
TCPC	Task's PC When Inactive
TCPS	Task's PS When Inactive
TCSP	Task's Stack Ptr. When Inactive
TCSPT	Ptr. to Base of Stack Storage
TCMSG1	Ptr. to Oldest MSG Queued to This Task
TCMSG2	Ptr. to Newest MSG Queued to This Task
TCCHK1	Ptr. to Newest CHUNK Queued to This Task
TCGMC	Counter for Waiting for Storage
TCLCB	Ptr. to LCB for This Task
TCTIM	Wait Time (in Jiffies)
TCST2	Secondary Status Bits
TCFG1	Flag Bits
TCFG2	Flag Bits
TCXLT	Ptr. to XLATE Task Being Fed
TCXFR	Count of Bytes Transferred Across DTE
TCVPP	In E3780, Vertical Position on PTR
TCCMSG	Ptr. to MSG Being Sent to -10

DT11HD	Count of Bytes in This Queue	
DT11FN	To -11 Function Code	
DT11DV	To -11 Device Number	
DT11UC	FE	Byte Count
DT11DF	DN60 Function Code	
DT11AD	ADR for EX/DEP (or line ,,device)	
DT11DT	Data for Deposit (or length of indirect MSG)	
DT11GW	Guard Word for DTE	
DT11QP		
DT11AS	ADR Save	
DT11BS	Byte Count Save	
DTXTSZ	Byte Count of Transfer	
DTXTAS		
DTEQSZ	-11 Queue Size	
DLCMSG	CHUNK ADR of MSG Being Built	
DLMBCT	MSG Byte Count	
DLMCTL	MSG Count Left Over for Next Transfer	
DTXADR	Save Indirect Data ADR Here	
DT11Q	To -11 Queue	
DLPMST	Ptr. to Current to -11 MSG	
TCSTFX	For EX/DEP of Status to -10	

TCEW:	EBFCHK=100	Waiting for free CHUNK
	EBQCHK=40000	Waiting for queued CHUNK
	EBINTR=20000	Waiting for device interrupt routines
	EBCOMP=10000	Waiting for $COMPQ to drain
	EBTENI=4000	Waiting for -10 interrupt
	EBQMSG=2000	Waiting for a MSG
	EBPERM=1000	Waiting for permission from HASP
	EBDATA=400	Waiting for queue to need data
	EBREQP=200	Waiting for HASP to request permission
	EBQPER=40	Waiting for operator response
	EBTIME=20	Waiting for timer
	EBTXDN=10	Waiting for "to -10 done" interrupt
	EBTEDN=4	Waiting for "to -11 done" interrupt
	EBWAIT=1	Waiting

TCST2:	TCLBK=B0	Current output line has been broken
	TCDTA=B1	There is data in this record
	TCEOT=B2	An EOT has been sent or received
	TCESC=B3	Last char. from PTR was ESC
	TCIGS=B4	Last char. was IGS
	TCXET=B5	Block ended in ETX, expect EOT next
	TCAK1=B6	Sending: next ACK expected is ACK-1 Receiving: last ACK response was ACK-1
	TCNRD=B7	No response to data message. After sending ENQ, accept wrong ACK as NAK

TCFG1:	TCPRI=B4	Interpret PTR carriage control on input
	TCPRO=B5	Interpret PTR carriage control on output
	TCTSP=B6	Do output in transparent BISYNC
	TCCMP=B7	Do component selection
	TCCPS=B8	Use compress/expand functions
	TCPCO=B9	Page counter has overflowed
	TCPCE=B10	Page counter interrupts enabled
	TCOBS=B11	Use "old" BISYNC
	TCDMP=B12	Output being dumped

TCFG2:	TCOPR=B0	Output permission requested
	TCOPG=B1	Output permission granted
	TCORN=B2	Output running
	TCOEF=B3	Output EOF started
	TCOEC=B4	Output EOF completed
	TCOAB=B5	Output abort started
	TCOAC=B6	Output abort completed
	TCIPR=B7	Input permission requested
	TCIPG=B8	Input permission granted
	TCIRN=B9	Input running

```
TCIAB=B10  Input abort started
TCIAC=B11  Input abort completed
TCIEC=B12  Input EOF completed
TCIWR=B13  Input permission was requested
```

Name: TCXLT

Description: Task Control Block for XLATE task.

Defined in: DEFINE

TCEW	Event word	
TCHAIN	Ptr. to Next TCB	
TCPC	Task's PC When Inactive	
TCPS	Task's PS When Inactive	
TCSP	Task's Stack Ptr. When Inactive	
TCSPT	Ptr. to Base of Stack Storage	
TCMSG1	Ptr. to Oldest MSG Queued to This Task	
TCMSG2	Ptr. to Newest MSG Queued to This Task	
TCCHK1	Ptr. to Newest CHUNK Queued to This Task	
TCGMC	Counter for Waiting for Storage	
TCLCB	Ptr. to LCB for This Task	
TCTIM	Wait Time (in Jiffies)	
TCST2	Secondary Status Bits	*
TCFG1	Flag Bits	*
TCFG2	Flag Bits	*
TCBFP	Ptr. to Line Buffer	
TCBFC	Length of Line Buffer	
TCMSG	Ptr. to Current MSG	
TCHPS	Current Horizontal Position	