60474600

CONTROL DATA CORPORATION

NETWORK PRODUCTS

COMMUNICATIONS CONTROL PROGRAM VERSION 3 TERMINAL INTERFACE PROGRAM (TIP) WRITER'S GUIDE REFERENCE MANUAL

CDC® COMPUTER SYSTEMS
255X SERIES
NETWORK PROCESSOR UNIT
HOST OPERATING SYSTEM
NOS 1

REVISION RECORD

REVISION	DESCRIPTION
A	Initial Release
(12/21/79)	
B	PSR level 528. Replaces SCF procedures.
(1/1/81)	

Publication No.
60474600

REVISION LETTERS I, O, Q AND X ARE NOT USED.

Printed in the United States of America

Address comments concerning this manual to:

CONTROL DATA CORPORATION
Publications and Graphics Division
P. O. Box 4380-P
Anaheim, California 92803

or use Comment Sheet in the back of this manual.

LIST OF EFFECTIVE PAGES

New features, as well as changes, deletions, and additions to information in this manual, are indicated by bars in the margins or by a dot near the page number if the entire page is affected. A bar by the page number indicates pagination rather than content has changed.

PAGE	REV
Cover	-
Title Page	-
ii thru v	B
vi thru viii	A
ix	B
1-1 thru 1-6	A
2-1/2-2	A
3-1/3-2	A
4-1 thru 4-7	A
4-8/4-9	B
5-1 thru 5-29	A
6-1 thru 6-4	A
7-1 thru 7-4	A
8-1 thru 8-5	A
9-1 thru 9-4	A
10-1 thru 10-3	A
11-1 thru 11-8	A
12-1 thru 12-9	A
12-10	B
12-11/12-12	A
13-1/13-2	A
14-1 thru 14-9	A
A-1/A-2	A
Index-1 thru Index-5	B
Index-6	A
Comment Sheet	-
Mailer	-
Back Cover	-

PAGE	REV

PAGE	REV

PAGE	REV

PREFACE

This manual defines the interface between the standard, Communications Control Program (CCP), version 3.1, and a terminal interface program (TIP). The TIP Writer's Guide is intended for systems analysts and designers who plan to write programs supporting:

- A terminal which uses a protocol not currently handled by standard CCP
- A terminal which uses a variation of one of the standard protocols supported by CCP

It is assumed that the reader is familiar with CCP functions as described in the CCP Reference Manual, and with TIP functions as described in the CCP System Programmer's Reference Manual. It is further assumed that the TIP writer is familiar with the CCP CROSS version of the PASCAL programming language, and with the state programming language as described in the manuals listed below.

The TIP writer should also be familiar with host products that directly affect the operation of a TIP: the Network Access Method (NAM) and the Network Definition Language (NDL). Knowledge of the host's Interactive Facility (IAF), Remote Batch Facility (RBF), and Transaction Facility (TAF) may also be helpful, depending on the type of terminal to be added to the network.

When interfacing a new TIP to the system, the TIP writer must use only the structures and functions described in this document. Control Data cannot be responsible for the functioning of any system with TIPs which use any interfaces not described in this manual. Further, when adding new CCP features or corrective code, Control Data can change without notice any TIP-related structure or function not described in this manual.

CONVENTIONS USED

Throughout this manual, the following conventions are used in the presentation of statement formats, operator type-ins, and diagnostic messages:

ALN	Uppercase letters indicate words, acronyms, or mnemonics either required by the network software as input to it, or produced as output.
aln	Lowercase letters identify variables for which values are supplied by the NAM or terminal user, or by the network software as output.
. . .	Ellipses indicate that the omitted entities repeat the form and function of the entity last given.
[ ]	Square brackets enclose entities that are optional; if omission of any entity causes the use of a default entity, the default is underlined.
{ }	Braces enclose entities from which one must be chosen.

Unless otherwise specified, all references to numbers are to decimal values; all references to bytes are to 8-bit bytes; all references to characters are to 8-bit, ASCII-coded characters.

RELATED MANUALS

The manuals listed below contain additional information on software elements of the Control Data Computer Systems and software used in network operations. These manuals can be obtained from CDC Literature and Distribution Services, 308 North Dale Street, St. Paul, Minnesota 55103.

Publication Title	Publication Number
Network Products NOS 1 Operator's Guide	60435600
NOS 1 Reference Manual Volumes 1 and 2	60445300 60445400
Network Products Interactive Facility, Version 1 Reference Manual	60455250
Network Products Transaction Facility, Version 1 Reference Manual	60455340
CYBER Cross System, Version 1 Build Utilities Reference Manual	60471200

Network Products Communications Control Program (CCP) Version 3 Reference Manual	60471400
State Programming Language Reference Manual	60472200
Network Products Communications Control Program, Version 3 System Programmer's Reference Manual	60474500
Network Products Network Definition Language (NDL) Reference Manual	60480000
Network Products Network Access Method (NAM), Version 1 Reference Manual	60499500
Network Products Remote Batch Facility, Version 1 Reference Manual	60499600
CCP Support Software 1 Reference Manual	96836000
CYBER Cross System, Version 1 PASCAL Computer Reference Manual	96836100
CYBER Cross System, Version 1 Macro Assembler Reference Manual	96836500

This product is intended for use only as described in this document. Control Data cannot be responsible for the proper functioning of underscribed features or undefined parameters.

CONTENTS

APPENDIXES

INDEX

FIGURES

TABLES

TIP WRITING OVERVIEW 1

A Terminal Interface Package (TIP) is responsible for converting messages between the terminal format and the virtual terminal format used by application programs in the host. When a terminal logs onto the network, it becomes connected to an application program in the host, and a message transfer session begins. The multiplex subsystem of CCP is responsible for making the physical characteristics of the line transparent to the TIP; that is, the TIP need only concern itself with the characteristics of the terminal. The TIP then makes the physical characteristics of the terminal transparent to the application program in the host, so that the host programs need only concern themselves with the characteristic of a virtual terminal. Two types of virtual terminals are provided: an interactive type for console devices, and a batch type for all non-console devices. Note that a TIP can also provide a transparent mode (no conversion to or from virtual terminal characteristics).

REQUIRED INFORMATION

This manual concerns itself only with the characteristics of a TIP. It is assumed that the reader is familiar with PASCAL language, with the state programming language, and with the functional characteristics of a network processing unit (NPU), as set forth in the CCP 3 Reference Manual. Section 2 of this manual lists the materials which are desirable to have on hand when writing a TIP.

COMMON SUPPORT FUNCTIONS PROVIDED FOR ALL TIPS

CCP provides five support functions that are needed by every TIP: a Service Module (SVM), a Block Protocol Interface Package (BIP), a set of base system subroutines, a text processor, and a multiplex subsystem. The functions provided by each of these interfaces are as follows:

- SERVICE MODULE: This handles all the terminal operational characteristics and status information that is to be exchanged with the host. SVM also provides table construction and initialization, and handles any special messages which the TIP may need to send to the host.

- BLOCK PROTOCOL INTERFACE PACKAGE (BIP): This provides for transportation of data from a host application to the TIP. BIP also provides block flow control for the TIP.

- BASE SYSTEM: The base consists of a monitor which controls task allocation within the NPU and a set of standard subroutines which may be used by a TIP writer. TIPs use standard base subroutines for buffer management and for timing message-related events.

- MULTIPLEX SUBSYSTEM: The multiplex subsystem provides various functions to transfer data between the NPU and the terminal. It isolates the TIP from line characteristics and performs the common functions for controlling the line hardware. The multiplex subsystem consists of both software and hardware. Multiplex hardware consists of a Multiplex Line Interface Adapter (MLIA) and the Communication Line Adapters (CLAs). The software consists of a PASCAL language portion which interfaces with the TIP (this is called the command driver), and a microcode portion which controls the input and output of characters (collectively this is called the multiplex firmware; individually it is called the input processor and the output processor). The multiplex firmware interfaces to the TIP's own input state programs (these are written in state programming language) which aid in translating incoming characters and formatting the incoming message.

- TEXT PROCESSING: Two interfaces are necessary: the first is between the PASCAL coded portions of the TIP and the common text processing programs (called the text processor); the second is between the text processor and the TIP's own text processing programs (written in state programming language).

In general, text processing programs are written to convert text and format (on a character-by-character basis) for downline messages, while input state programs are written to do the same thing for upline messages. Input state programs also assist in demultiplexing the upline messages which the multiplex subsystem multiplexed as they entered the NPU.

The five interfaces for the TIP are shown in figure 1-1.

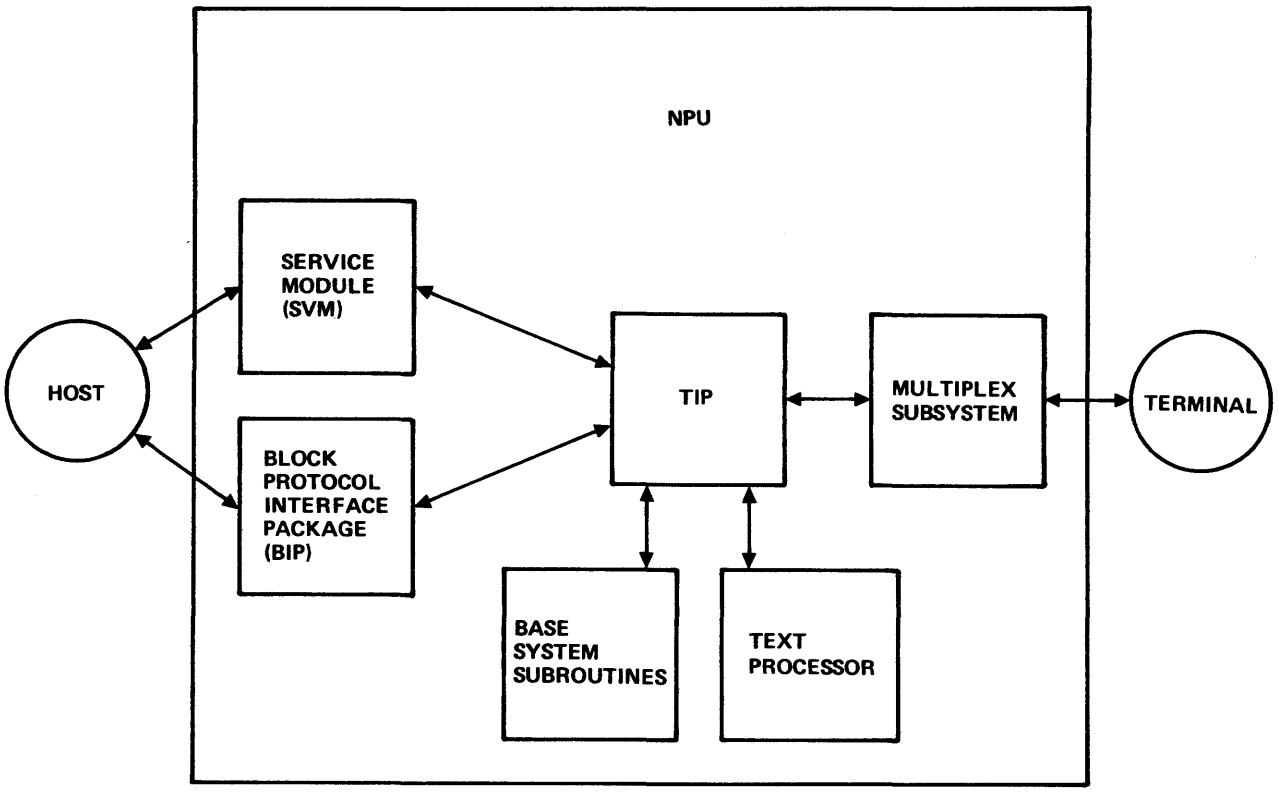


Figure 1-1. Principal Interfaces for a TIP M-588

TIP WRITER'S TASKS

Since a standard Host Interface Package (HIP) is supplied in every local NPU, the TIP writer need not concern himself with actually transmitting or receiving data across the coupler interface to the host. Similarly, since a standard Link Interface Package (LIP) is supplied in every remote NPU and in every NPU connected to a remote NPU, the TIP writer need not concern himself whether his TIP executes in a local or in a remote NPU. The TIP writer must, however, provide the following:

- Modification to system table structures for TIP-defined fields within common tables.
- Link editor code to initialize line and terminal data structures which are unique to the TIP.
- PASCAL code to handle requests from the host to change the status of lines or terminals, and to inform the host that the requested changes were or were not made.
- PASCAL code to support the IVT interface for console devices and the BVT interface for non-console devices. Code to support transparent transmissions for consoles is optional.
- PASCAL code to support all the protocol requirements of the terminal. Error processing must be supplied to handle protocol and other message processing failures.
- State programming language programs to handle the protocol requirements and data translation of upline (incoming) messages.
- State programming language programs to handle the protocol requirements and data translation of downline (outgoing) messages.

The TIP consists of five component modules as shown in figure 1-2. These modules execute on four priority levels, as will be discussed later. The component modules are as follows:

- The first module initializes the Terminal Control Block (TCB) for each terminal serviced (OPS-level initialization). This module is called by the service module to complete the initialization of the dynamic data structure which controls message transfers with a terminal. This module must initialize any TCB fields which the TIP uses as it starts processing a message. The OPS-level TIP can change TCB fields as needed during processing.
- The second module handles special priority calls from the multiplexer subsystem (Mux 2 level TIP).
- The third module contains the bulk of the higher level decision logic (OPS-level TIP). The OPS-level TIP provides overall control of every terminal which uses this protocol. It also provides downline code translation and protocol control functions (text processing is called from the OPS level).
- The fourth module handles input message processing, character-by-character (input state programs). A single set of input state programs is used for all terminals under the TIP's control.
- The fifth module handles output message processing, character-by-character (text processing state programs). The OPS-level TIP calls these programs when the TIP has a full block of data ready to be converted. Text processing state programs are processed on the OPS-level, but are interrupted by the multiplexer subsystem firmware. These programs are not reentrant.

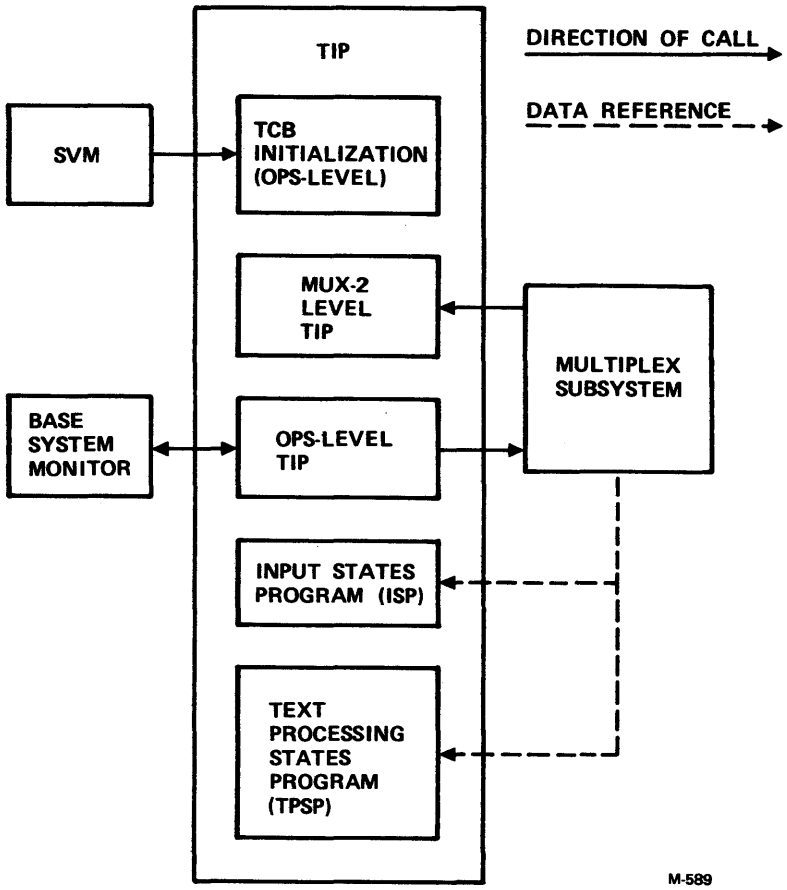


Figure 1-2. TIP Modules

These five modules execute at four priority levels. Priority levels are system defined; the TIP writer can exercise no control over them. The levels are listed in precedence order, with the highest given first:

- Interrupt level 1 for processing output characters to the terminals
- Interrupt level 2 for processing input characters from the terminals
- Mux-2 level for priority processing of certain terminal events. This level might be used to maintain high line utilization, or to provide a high priority response to a terminal input.
- OPS level. This is the lowest NPU processing level. The base system monitor gives control to the OPS-level TIP to perform a single task. The TIP must return control to the monitor when this task is completed or suspended. The monitor assigns control of the NPU by calling all OPS level modules in rotational order from a list. For example, the service module can request the TIP to perform a task by placing the request (in the form of a work list entry or WLE) in the TIP's queue of tasks to be performed. Eventually, the monitor will give control to the TIP to perform the required task.

NOTE

Any mixing of routines between the Mux-2 level and the OPS-level requires reentrant code at both levels.

In summary, the TIP writer has four main tasks, listed as follows:

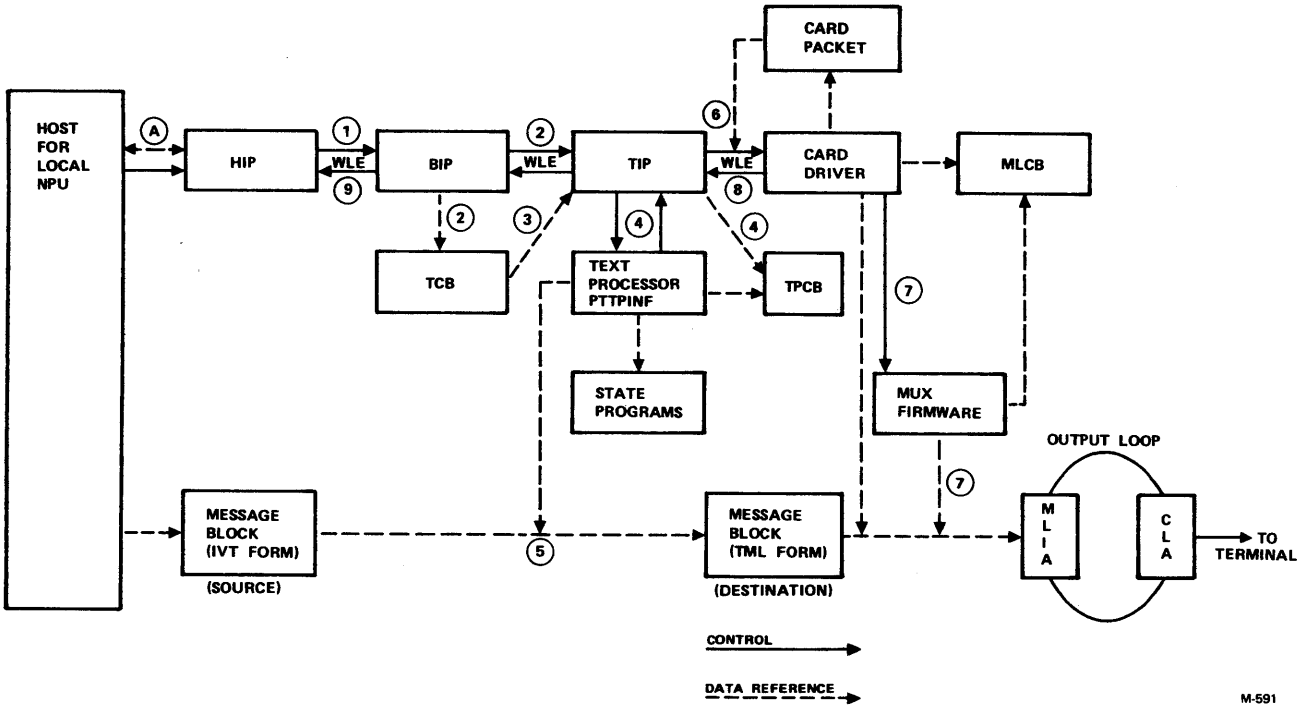
- Writing modifications to the system data structures.
- Writing TIP programs and subprograms in PASCAL language to perform all of the non-character processing functions of the TIP, including error processing.
- Writing state language programs to convert the code, and to format input and output messages, character-by-character.
- Writing link edit directives to initialize the system data structures during the CCP build process.

SUMMARY OF DOWNLINE MESSAGE PROCESSING

After the connection to a terminal has been established, a downline message can be processed. Figure 1-3 summarizes the process.

The HIP or LIP handles the initial step of assembling the message block (MSG or BLK type of block) into buffers in the NPU. The TIP is not involved in this stage of processing.

Once the HIP or LIP acknowledges that the block has been successfully received, that module sends a worklist to the BIP to queue the block to the appropriate TCB. The BIP adds the data block to the TCB's queue and generates a worklist for the TIP. When the TIP gains control with this



NOTES TO FIGURE 1-3:

1. A block of downline message is transmitted to the HIP (local NPU) or LIP (remote NPU). That module assigns buffers to receive the block of data.

2. The HIP or LIP generates a WLE to the BIP for routing block to the TIP.

3. The BIP queues the block to the TCB, and generates a WLE to the TIP.

4. The TIP calls the test processor interface (PTTPINF), after setting up the TPCB with information to control the conversion. TPCB fields are the only means of communication between the TIP's OPS level and the text processor.

5. The text processor uses the TIP's text processing state programs to govern transfer and conversion of the data. Characters are taken from the source buffers, processed, and placed in the destination buffers. Both code and format conversion take place at this time. At the end of text processing, control returns to the OPS level TIP. The block is ready to be sent to the multiplex subsystem. The TIP releases unneeded source buffers.

6. A TIP calls the command driver (PBCOIN) directly to deliver the message. Transfer parameters are in a command packet (NKINCOM). The Multiplex Line Control Block (MLCB) fields are used to control the command driver and the multiplex firmware while the message is being transmitted.

7. The multiplex subsystem transmits the message one character at a time, using firmware and hardware. The OPS level TIP is not involved.

8. After the message is transmitted, the command driver calls the Mux-2 level TIP. The TIP releases buffers, and informs the BIP of a successful message transfer.

9. The BIP creates an acknowledgment message for the host application program.

Figure 1-3. Simplified Downline Message Flow

worklist, the TIP checks the data block's validity and generates a Text Processing Control Block (TPCB). Then the TIP makes a direct call to the text processor interface (PTTPINF). This is the interface to the text processor and the TIP's own text processing state programs. State programs (input, output, and modem) are described in detail in the State Programming Language Reference Manual. If the user is not familiar with such state programming concepts as state instructions, state processes, state programs, and state program pointer tables, he is urged to review these concepts.

User-written text processing state programs implement the character-by-character transfer from the source buffer to the newly assigned destination buffer. During this transfer, code conversion (if any) occurs, and the message is reformatted from IVT/BVT format to terminal format. All parameters needed to perform the text processing (buffer addresses, flags, counters, control fields) are contained in the TPCB. As it converts the message, the text processor changes TPCB fields. This information will be used by the TIP, when it regains control, to find the status of the text processing operation.

The TIP does not regain control until the entire block is text processed, or until the text processor can do no more. The TIP releases any source buffers which are no longer needed. In many cases, the source data is not completely translated. The TIP determines where processing may resume in the source buffer and releases those buffers which have been fully processed.

When the block is fully translated, the message in the destination output buffer is ready to be transmitted to the terminal via the multiplex subsystem. The TIP makes a direct call to the command driver (PBCOIN) and places the parameters for the message transmission in a command packet (NKINCOM). This ends the TIP's concern with the output operation until the command driver returns a worklist informing the TIP either that the message has been transmitted or that the transmission failed.

The multiplex subsystem sends the block, character by character, to the terminal. Output processor firmware frames the data character prior to placing it on the multiplex output loop. A character is placed on the loop in response to an output data demand (ODD) interrupt generated by the communication line adapter for that line.

In the event that the multiplex subsystem cannot send the message, the subsystem informs the TIP of the failure, and the reason for the failure. Depending on the way the TIP is written, the TIP may react as follows:

- It may choose to try sending the message again without notifying the host that a problem exists.
- It may mark the terminal or line inoperative, release the message, and notify the host that (1) the device status has changed to inoperative, and (2) the message was not delivered.

SUMMARY OF UPLINE MESSAGE PROCESSING

Several steps are required to prepare the multiplex subsystem for receiving a message (see figure 1-4). As in the downline case, the HIP or LIP performs the transfer to the host or to the local NPU. The TIP determines if input is permitted at this time (input regulation) by calling a base subroutine, PTREGL, directly. If reception is permitted, the TIP calls the command driver to set up the input transfer. For asynchronous protocols, PTREGL is used to discard input since input is not solicited from those terminals.

Data from the line is placed on the input multiplex loop by the CLA. As the multiplex subsystem picks the character from the line, it adds framing information consisting of a line address. The multiplex subsystem also picks up a line status word (if any is available from the CLA), and places the entire packet (frame and data) into the circular input buffer (CIB). This ends the multiplexing operation.

The multiplex subsystem attempts to demultiplex the data as fast as it is written into the circular input buffer. Demultiplexing is activated by a hardware-generated interrupt that occurs each time a new frame is placed in the circular input buffer.

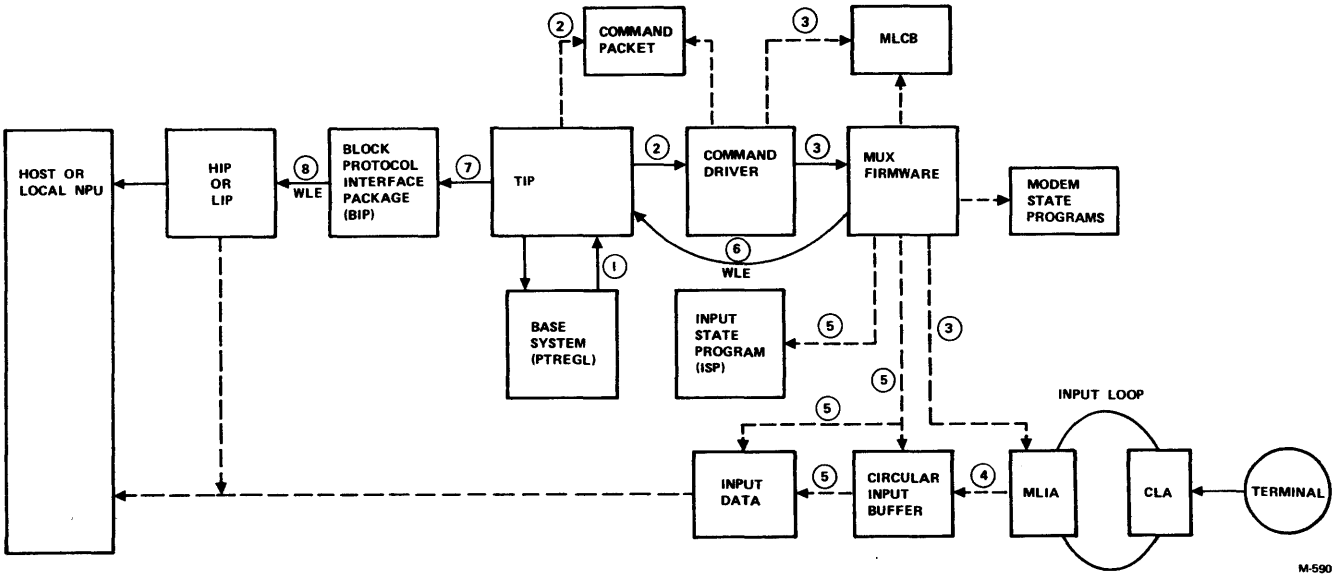
Demultiplexing control information is contained in the modem state programs and the TIP's own input state programs. Note that the modem state programs are a set of common state programs on the firmware level; the TIP writer needs only write the input state programs.

If status information is detected in the frame, a modem state program is called to process it. After the current modem state program finishes processing the status, control passes to the current input state program. This state program controls the conversion of the character and its movement into the line-oriented input buffer. Note that the firmware is responsible for supplying chained input buffers for the data.

Demultiplexing continues until the end of the block is reached. At that time the input state program notifies the TIP of this condition with a worklist. The TIP then routes the processed block to the host through the BIP.

UPLINE TEXT PROCESSING

In some cases it is not possible to convert incoming data in a single real time pass as the characters are being received from the terminal. In such a case, the input state program merely demultiplexes the data into the line-oriented input buffers. Then the block is sent to the OPS level TIP which has a separate set of text processing programs for input data. The OPS level TIP calls the text processor in the same way that the text processor is called to convert downline data. If at all possible, this two-pass text processing should be avoided. The processing cost of translating characters, both upline and downline, normally uses at least 50% of the available NPU resources.



NOTES TO FIGURE 1-4:

1. The TIP calls PTREGL directly to determine if input is allowed at this time.

2. The TIP calls the command driver (PBCOIN) to initiate input processing. Parameters are passed in the command packet (NKINCOM).

3. The Multiplex subsystem prepares the hardware for input processing.

4. When the next input character arrives, it is framed and placed in the circular input buffer. The frame consists of an address. If a line status word is available, it is added to the data packet.

5. The multiplex subsystem uses the number of the line to locate the input state program for that line and the buffer assigned to receive the data. Then the input processor converts the character as directed by the input state program.

6. The multiplex subsystem reports an end of input condition to the OPS level TIP (optionally to the mux level TIP) by means of a worklist entry.

7. The TIP performs whatever operations are needed to finish the input processing. Then it passes the block to the BIP.

8. The BIP passes the block to the HIP or LIP, to be sent upline to the host.

Figure 1-4. Simplified Upline Message Flow

MATERIALS NEEDED FOR WRITING A TIP 2

This section lists the documents that are useful to have available when writing a TIP. The value of each of the items to a TIP writer is also defined.

CCP REFERENCE MANUAL

This manual gives a general description of CCP's external characteristics. Appendices to the reference manual list all the code sets supported by the standard CCP TIPs:

- ASCII
- Teletype paired ASCII
- Bit paired ASCII
- Binary coded decimal (BCD)
- External BCD
- External BCD - APL
- Correspondence
- Correspondence - APL
- Display
- EBCDIC

The reference manual also contains a description of error messages and halt codes.

LANGUAGE AND LANGUAGE SUPPORT MANUALS

Compilation of CCP programs and building of the CCP load file is done in the host computer. Four source languages are used to write the CCP programs. A set of directives is used to compile and integrate a TIP into CCP. Three of these source languages are used by the TIP writer. These are as follows:

- PASCAL language. This is used to write the OPS-level programs, and any Mux-2 level programs.
- State programming language. This is used to write the input state programs and the text processing state programs.
- Macroassembly language. The TIP writer can use this occasionally in the OPS-level programs.

The fourth language (microassembler) is used only in CCP firmware level programs, such as the input processor or the text processor. The TIP writer never uses this language.

Link edit and library maintenance directives provide the commands used to compile the TIP modules and to enter the TIP into the CCP build (load file).

CYBER CROSS REFERENCE MANUALS

The CYBER CROSS compilers and support programs allow the user to write programs for the NPU in languages defined in the last paragraph, and to use the large and efficient resources of a CYBER 70 or 170 to compile the programs, to debug them, and to build the load file which is used by the host to load the NPU.

Five manuals are included in the CYBER CROSS set:

- CYBER CROSS System PASCAL Compiler Reference Manual. This manual defines the PASCAL language and describes how to obtain cross reference listings.
- CYBER CROSS System Macroassembler Reference Manual. This manual defines the macroassembler rules. The macroassembler is used as the basis of the specialized state programming language.
- CYBER CROSS System Microassembler Reference Manual. This manual defines the microassembler rules.
- CYBER CROSS System Link Editor and Library Maintenance Reference Manual. This manual explains the use of the link editor which is used to initialize the CCP data structures. It also defines the format of the CCP library which holds the modules from which the NPU load file is constructed.

PASCAL LANGUAGE REFERENCE

The analyst may wish to read some background information in the theory and use of PASCAL coding. This can be obtained in a highly usable text:

- PASCAL User Manual and Report, Second Edition
  Kathleen Jensen and Niklaus Wirth
  Springer Verlag, 1974

The text can be obtained through any technical book dealer.

CYBER UPDATE REFERENCE MANUAL

The UPDATE programs provide the user with an easy method of modifying the existing CCP libraries.

STATE PROGRAMMING LANGUAGE REFERENCE MANUAL

This manual contains the rules for writing programs in state programming language. The user must write all input state programs and text processing state programs in this language. The manual defines each state instruction. It also contains a description of the modem state programs. These are a set of common support modules for processing status information from the communication line adapters.

NOS 1 INSTALLATION HANDBOOK

This manual describes the method of installing the NOS installation tapes which include the CCP files. The manual also describes the standard build procedure for CCP.

HARDWARE REFERENCE MANUALS

Several hardware reference manuals may be of use in helping the TIP writer to understand the hardware constraints of the NPU. These are as follows:

- 255X Host Communications Processor Reference Manual. This describes the hardware environment of the NPU.
- 2560-1/-2/-3 Synchronous Communications Line Adapter Hardware Maintenance Reference Manual. This describes the hardware environment of the low and medium speed line controllers.
- 2561-1 Asynchronous Communications Line Adapter Hardware Maintenance Reference Manual. This describes the hardware environment of the asynchronous line controllers.
- 2563-1 Synchronous Communications Line Adapter Hardware Maintenance Reference Manual. This describes the hardware environment of the bit insertion line controllers (HDLC, X.25).

TERMINAL PROTOCOL RULES

Each terminal manufacturer should supply an adequate description of the terminal's protocol. This must contain the code definition (including any variations particular to the terminal) of all symbols used by the terminal. It should also include the set of default characters used for non-recognized codes.

LISTINGS

It is often useful to have listings, both of the base/multiplex subsystem/BIP and of other TIPs, to determine how other programmers have solved message handling or interface problems.

CAUTION

The TIP writer should note that TIPs in the standard CCP system use techniques that are in conflict with the guidelines set forth in this manual. CDC will support only those TIPs that are written to conform to the rules set forth in this manual.

MASTER AUDIT LISTING

This is a complete source listing for CCP. It is obtained by using the method described in the UPDATE Reference Manual.

PASCAL X-REF LISTING

PASCAL cross-reference listings are available for all PASCAL programs. The method for producing such listings is given in an appendix of the PASCAL Compiler Reference Manual. If the user has the maximum CCP system (HIP, LIP, and at least three standard TIPs), all the CCP modules cannot be run in a single cross-reference listing. The user should split his cross-reference job into two or more X-REF listings.

PASCAL LISTING

The standard build procedure preserves (as files) the listings containing PASCAL code. These files can be copied to the printer for use in reviewing the base system, the interfaces to the TIP, and TIP coding.

ASSEMBLY LISTINGS

Two assembly listings are also supplied by the standard build procedure:

- Base modules and modem state programs: These listings are useful as examples of input parameters and field initialization requirements for base programs called by the TIPs. The modem state program listings show how control passes to the TIP input state programs.
- TIP state programs: This listings provide coded examples of methods used by TIP writers to solve the character by character protocol processing problems for their equipment. The listings also show how to pass control from the state level to the OPS level TIP.

Assembly listings contain code conversion tables, and state instruction definitions in macroassembly format.

MPLINK LISTING

A set of MPLINK listings are provided for each CCP system variant. These listings indicate the program module addresses.

MPEDIT LISTING

The MPEDIT listing shows the global tables and variables which are initialized in the standard CCP system. This includes all the fields and variables initialized by the standard TIPs.

OBJECT CODE LISTINGS

The PASCAL compiler produces a set of listings of the NPU instructions which are the result of the compilation. This listing can be helpful in debugging code.

The internal structure of a TIP is determined by the characteristics of the terminal's protocol, by the interfaces to the rest of CCP, and indirectly to the application programs which are used in the host. Usually a TIP supports more than one terminal. A TIP can support a variety of related terminal types.

Because of the difference in the rates at which line-related events and NPU-related events occur, the TIP must handle conditions where a terminal event is to occur after a period of time, rather than immediately. The TIP must therefore have programs dependent on waiting for a terminal event to occur. Since the TIP may be required to handle several similar terminals on individual lines, each at its own stage of processing a message, the TIP must have reentrant code.

In planning the structure of a TIP, the writer must consider the following:

- The methods which the system uses to access the TIP
- The structure of the interfaces the TIP must use to access the system
- The use of local and global variables

ENTRY CONDITIONS

CCP uses three types of entry into the TIP:

- The normal processing entry is by means of a worklist to the OPS level TIP programs. The TIP is called to continue processing a message or an I/O status change. This call is made by the OPS-level monitor.
- A high priority entry point can be provided for the TIP to process interrupt-driven tasks on the Mux-2 level. This call is made by the multiplex subsystem's worklist handler.
- A direct entry is made to handle initialization of those variables in the Terminal Control Block (TCB) which are unique to this TIP. This is a direct (return jump) call from the service module. (This is theoretically optional, but is in fact required on all but the simplest TIPs.)

OPS LEVEL ENTRY

The OPS-level TIP is an event-driven module which must surrender control of the NPU for all wait conditions. System modules external to the TIP (or the TIP itself) request the TIP to execute a task. The tasks are queued in a list of unexecuted tasks. The Worklist Control Block (WLCB) is the data structure which handles this task list. Each of the tasks are called a worklist entry (WLE). Unsolicited worklists are generated outside of the TIP. Solicited worklists are generated by the TIP itself.

When the OPS monitor passes control of the NPU to the TIP, the monitor also passes the worklist parameters to the TIP. The TIP must have an entry structure that is able to determine the exact task to be performed. This is often done by switching to the appropriate TIP program based on the workcode in the worklist.

MUX-2 LEVEL ENTRY

The multiplex subsystem enters the TIP's Mux-2 level entry (if it is defined) to process certain types of priority tasks. This entry appears to be a worklist-driven call, but it is in fact a direct call using a return jump. OPS-level processing may have been interrupted by a Mux-2 level call. Therefore any shared OPS-level and Mux-2 level procedures must be written using reentrant code.

TCB INITIALIZATION

The call to initialize those TCB fields that are unique to a terminal must take place before using the terminal for the first time.

LOCAL AND GLOBAL VARIABLES

CCP's global variables are defined for all modules in the system. Since these globals are reserved for system use, they may change as the system changes. Therefore the TIP writer should avoid accessing such fields, and should avoid defining system globals unless it is absolutely necessary.

The necessary uses of system globals are as follows:

- Variables to be initialized by the build process must be globally defined. The TIP writer must provide MPEDIT declarations to specify the values to be preset. The MPEDIT program locates a variable to be initialized by its associated entry point name. The PASCAL compiler allows only global variables to have entry point names.
- In some cases, global variables are also needed to pass parameters between modules of a TIP. This is needed when the size of the TIP exceeds the capacity of the compiler to place all TIP code in a single module.

Local variables are used by the TIP writer as needed. Unfortunately, reentrancy requirements can conflict with such use. Local variables within a TIP should never be assumed to hold valid information after the TIP has given up control of the NPU to wait for an external event to occur.

All variables should be defined uniquely. Multiple definition of variables may compromise the maintainability of the system.

TIP REENTRANCY

Once the OPS level TIP is given control of the NPU, it is allowed to execute until the TIP voluntarily returns control to the system. (Of course, this priority level may be interrupted one or more time for higher priority tasks, but such operations are invisible to the OPS level TIP.) The TIP defines the points where reentrancy is permitted. This matter is discussed in detail in section 8. The system and the compiler do not provide sufficient capabilities to code a TIP without considering reentrancy.

BLOCK PROTOCOL

Data is transported within the network using the block protocol. The TIP has the responsibility of placing the data in block format before it passes it to any other module, and the system provides interface modules to move the blocks (and in some cases, to provide additional, specialized formatting). The TIP must use these interfacing transport modules. The TIP is not allowed to develop its own transport mechanisms in avoidance of these system-provided interfaces.

UNDOCUMENTED SYSTEM INTERFACES

The system modules and the TIPs are compiled together in a single CCP compilation. As the TIP writer designs his TIP, he may become aware of many system modules and globial variables that are reserved for system use. The TIP writer should not use these modules and variables, since CDC reserves the right to alter or delete any of them at any time. CDC warrants only to maintain the interfaces that are defined in this manual.

Implementation of a new TIP requires the following:

- Coding the PASCAL OPS-level and Mux-2 level programs
- Coding the state programs (input and text processing)
- Coding the system global changes
- Coding the initialization directives for MPLINK and MPEDIT
- Coding the system build directives

Each of these coded programs and directive sets are placed in various system and user supplied COMDECKS and are added to the system library on the CYBER host.

CHANGES TO THE BASE SYSTEM GLOBALS

Adding a TIP to the system requires that several data structures be altered to make room for tables that the TIP needs, and to provide standard mnemonics. Extensions are required to the worklist control block (WLCB), the TIP type table, and the terminal control block (TCB).

WORKLIST CONTROL BLOCK (WLCB)

A WLCB is defined for each TIP. It indicates the address of the TIP's OPS-level entry point. The TIP writer must add the following indices to COMDECK CONSTR4:

- BOUWL1
- BOUWL2
- BOUWL3
- BOUWL4

The values selected must be in increasing numerical order and begin with the former value of BODUMMY. BODUMMY is always the last entry, its value is one greater than BOUWL4. The TIP writer must initialize the entries used in the control array, as described in section 12. The format of the worklist entry table is shown in table 4-1.

TABLE 4-1. WORKLIST CONTROL BLOCK

OPS-Level Programs:

These worklists are serviced by the OPS monitor program. They are also part of the worklist array. New entries must be added at the end. The last entry must be BODUMMY which is equal to the last TIP worklist value.

Type/Value	Meaning
BOCHWL = 8;	CONSOLE PROGRAM
BOINWL = 9;	INTERNAL PROCESS
BOMLWL = 10;	MLIA INTERRUPT HANDLER
BOSMWL = 11;	SERVICE MODULE
BOTIWL = 12;	TIMING SERVICES
BOTYWD = 13;	TIP DEBUG
BOLIWL = 14;	LINE INITIALIZER
BODGWL = 15;	ONLINE DIAGNOSTICS
BOCOWL = 16;	HOST INTERFACE PACKAGE (HIP)
BOHDLC = 17;	HDLC LIP
BOM4WL = 18;	MODE 4 TIP - BVT
BOTTYWL = 19;	MODE 3 TIP - TTY - IVT
BOHASP = 20;	HASP TIP
BO27WL = 21;	2780/3780 TIP
BOHHWL = 22;	HASP/360 HIP
BODUMMY = 23;	DUMMY FOR CONSOLE
BOASYNC = BOTTYWL;	ASYNC TIP
Delete this card and add:	
BOVWL1 = 23;	
BOVWL2 = 26;	
BODUMMY = 27;	

TIP TYPE TABLE

The TIP type table defines the characteristics of the TIP and provides linkage to the configuration control parameters. The table is expanded for the four possible user TIPs, and standard mnemonics are defined. The TIP writer must expand the SYTIPTYPT array in COMDECK TYPE and add the following indices to the COMDECK CONSTR4:

- NOUTT1 = 12; USER TYPE 1
- NOUTT2 = 13; USER TYPE 2
- NOUTT3 = 14; USER TYPE 3
- NOUTT4 = 15; USER TYPE 4

These values for the TIP type are permitted in the Network Definition Language (NDL). The content of the TIP type tables must be initialized by the TIP writer, as explained in section 12. Table 4-2 shows the required changes.

USER GLOBAL CHANGES

The user must define the arrays which are used by the service module to process the line and terminal configuration options. Four arrays are added to the user-supplied VARtip COMDECK for line and terminal configuration. These tables define the field descriptor tables (line and terminal) and the action tables (line and terminal).

NOTE

In the following COMDECK descriptions, the COMDECKs have a name AAAtip where AAA is a system defined deck type and TIP is a three character mnemonic supplied by the TIP writer.

TABLE 4-2. SIT, TIP TYPE AND SUBTYPE, AND TERMINAL CLASS

System Interface Table Types:

Type	Value	Meaning
	SYLCBP = ARRAY HLRANGE OF BZLCBF;	
	SYLINO = ARRAY HLRANGE OF BOLINO;	
	SYENTY = ARRAY BCPRILEVEL OF BWWORKLIST;	
	SYLTYT = ARRAY INCLTYP,1..NKRC3OUT OF NBLTYE;	
	SYPRTT = ARRAY 0..1 OF NAPCRY;	
	SYCTCT = ARRAY NOLNSPDS OF NICTCY;	
	SYTMTB = ARRAY COTDPGMS OF CBSYTMT;	
	SYTECT = ARRAY NOTCLASS OF NJTECY;	
	SYTIMTBL = ARRAY 0..1 OF BZLTIME;	
	SYTIPTYPT = ARRAY NOHDLC..NOCLCIAC OF TIPTYPE;	
	SYOVLCB = RECORD	OVERLAY CONTROL BLOCK
	DCOVID : INTEGER;	OVERLAY ID
	DCOVLST : DOOVLSTATE;	OVERLAY STATE
	DCLBN : INTEGER;	LAST BLOCK NUMBER
	DCBN : JSASCIISET;	BLOCK NUMBERS LOADED
	Change this name to:	
	NOUTT4	

System Interface Table:

Type	Value	Meaning
	SITTBL = PACKED RECORD	
	SIENTY : SYENTY;	POINTER TO BWWLENTRY
	SITMTB : SYTMTB;	POINTER TO CBTIMTBL
	SIWLCB : BYWLTY;	POINTER TO BYWLCB
	SIDBSIZE : BECTPTR;	POINTER TO BEDBSIZE
	SITPSIZE : BECTPTR;	POINTER TO BETPSIZE
	SINJTEC : SYTECT;	POINTER TO NJTECT
	SITIMTBL : SYTIMTBL;	POINTER TO BLTIMTBL
	SITIPTYP : SYTIPTYPT;	POINTER TO BJTIPTYPT
	SIOVLBLK : SYOVLCE;	POINTER TO SYOVLCB
	SILCBP : SYLCBP;	POINTER TO HALCBP
	Add:	
	NOVTC1 = 28;	
	NOVTC4 = 31;	

Field Descriptor Tables

The host software must be modified to add new parameters for the configure or reconfigure terminal service messages. Messages are sent downline from the host to initialize fields in these tables. These configuration messages state the values to be placed in the fields by means of a field number/field value (fn/fv) pair. The tables are required, even though the entries may be empty. The following arrays are added to the VARtip COMDECK:

- VALnFDT: ARRAY 0..x OF DDFDTENTRY: LINE FIELD DESCRIPTOR
- VATnFDT: ARRAY 0..x OF DDFDTENTRY: TMNL FIELD DESCRIPTOR

where:

n has the range of 1 through 4, corresponding to user TIP 1 through 4.

x is the number of fn/fv pair entries.

The TIP writer must initialize these tables as described in section 14.

Action Table

The host uses line and terminal configuration service messages to send to downline user-defined line or terminal options. The variables are sent in the form of a field name/field value pair (fn/fv). Action tables define the appropriate type of initialization that is to be performed while setting up each of these fn/fv pairs. The user defines the following arrays in the VARtip COMDECK:

- VALnAT :ARRAY 1..x OF DFATENTRY LINE TABLE ACTIONS
- VATnAT :ARRAY 1..x OF DFATENTRY TERMINAL TABLE ACTIONS

where:

n has the range of 1 through 4 corresponding to user TIP 1 through 4.

x is the number of entries.

The TIP writer also must initialize these action tables as described in section 12.

STATE PROGRAMS

State programs include the input state programs, the text processing state programs, the code translation tables, and common definitions. The state programs use state processing instructions almost exclusively, although any valid macroassembler instruction can be used. The frequently used macroassembler instructions are as follows:

- EQU to equate a protocol name with a value, or to define usage for a file 1 register
- EXT to define entry points, such as a translation table address
- ENT to define the entry point to a state program
- MAC to define the pointers table for a state program
- EMC to end the state program pointers table macro definition
- SPC to add blank lines to a listed output
- EJT to insert a top-of-page command in a listed output
- ADC for address constants

The state programming language is described in detail in the State Programming Language Reference Manual. The deck structure for compiling state programs using UPDATE is described in the UPDATE Reference Manual. There are two common definition decks:

- MUXMACROS defines the input state instructions (macros) for the macroassembler. It also contains EQU statements for multiplex subsystem constants and for workcodes (A0WKn).
- REL4CONST contains EQU statements for the IVT and BVT constants that are common to all CCP programs.

Translation tables are written using macroassembler statements exclusively. The macroassembler language is described in detail in the CYBER CROSS System Macroassembler Reference Manual.

BASE SYSTEM DATA STRUCTURE OVERLAYS

The TIP writer probably will need to define the names of fields in the user portions of the following tables:

- Line control block (LCB)
- Terminal control block (TCB)
- Text processing control block (TPCB, a subset of the multiplex line control block NCLCB)
- CE error file entry (CNCE)

LCB OVERLAY

The entries are made in the tipLCB user COMDECK, and are PASCAL statements of the form:

```
1:(
  BZname : type; description
    .
    .
    .
  );
```

The prefix BZ is used for all LCB fields, and the name must be unique.

TCB OVERLAY

The entries are made in the tipTCB user COMDECK, and are PASCAL statements of the form:

```
1:(
  BSname : Array; IVT fields
           0..6
           OF INTEGER
    .
    .
    .
  );
```

The prefix BS is used for all LCB fields, and the name must be unique.

TEXT PROCESSING CONTROL BLOCK OVERLAY

The entries are made in the tipMLCB user COMDECK, and are PASCAL statements of the form:

```
1:(CNname : type; field description
      .
      .
      .
  );
```

The prefix NC is used for all TPCB fields, and the name must be unique.

CE ERROR FILE ENTRIES

The entries are made in the tipCEFILE user COMDECK, and are PASCAL statements of the form:

```
1:(CNname : type; field description
      .
      .
      .
  );
```

The prefix CN is used for all CE file fields, and the name must be unique.

TERMINAL CHARACTERISTICS TABLE

The TIP writer must create new tables to define the terminal classes to be used with his TIP. These tables define default values for IVT parameters to be used during terminal configuration. The TIP writer adds the indices for the four user terminal classes by adding constants to the CONSTR4 COMDECK. Entries are of the form:

- NOUTC1 = 28; USER TERMINAL CLASS 1
- NOUTC2 = 29; USER TERMINAL CLASS 2
- NOUTC3 = 30; USER TERMINAL CLASS 3
- NOUTC4 = 31; USER TERMINAL CLASS 4

These values are permitted to be defined in the Network Definition Language.

The terminal class arrays must also be extended to include the tables. This is done by adding the following statements to the TYPE COMDECK:

```
NOTCLASS = 0..31; TERMINAL CLASS
NOTTYP = NOTMLIA..NOUTC4; TERMINAL CLASS
```

The content of the new entries in the arrays are supplied by the TIP writer using the guidelines of section 12.

ADDING A TIP TO THE PROGRAM LIBRARY

The TIP writer must add to the program library all the COMDECKs containing code for global modifications, PASCAL TIP code, microassembly language TIP code (this includes the state language programs), and the global initialization statements.

GLOBAL MODIFICATIONS

Modifications are made for adding TIP-defined global variables, for expanding the new TIP's system globals, for defining the service module tables, and for defining new CE error file messages.

TIP-Defined Global Variables

User globals are placed in their own UPDATE COMDECKs. The names are defined to be easily recognized. Calls to these COMDECKs are placed in the following standard system COMDECKs:

COMDECK Name	Content of COMDECK
CONSTUSR	Call to the TIP's constants definition COMDECK
TYPEUSER	Call to the TIP's types definition COMDECK
VARUSER	Call to the TIP's variables definition COMDECK
VALUSER	Call to the TIP's values definition COMDECK
FWDUSER	Call to the TIP's forward definition COMDECK

Examples of each of these COMDECKs are given in figures 4-1 through 4-5.

```
*COMDECK CONSTUSR
*$J+ PAGE EJECT
*********************************
*                               *
*   USER CONSTANTS CALL DECK    *
*                               *
*********************************
*CALL CONSTR4
*CALL CONBLK
*CALL IPCON
*IF DEF,SVMODULE,1
*CALL SVMCONST
*IF DEF,OLDIAG,4
*IF DEF,CLAIA,1
*CALL CONODC
*IF -DEF,CLAIA,1
*CALL CONOLD
*IF DEF,HIP,1
*CALL CONHIP
*IF DEF,MODE4,1
*CALL CONMD4
*IF DEF,TESTGEN,1
*CALL CONTST
*IF DEF,LDDMPO,1
*CALL CONLDP
*IF DEF,IVT,1
*CALL CONIVTC
*IF DEF,ASYNC,1
*CALL CONASYNC
*IF DEF,HLIP,1
*CALL CONLRZQSS
*ENDIF

If a new TIP is written with global constants
defined, the user must supply a call:

   *CALL tipCON

A tipCON COMDECK defines the TIP's constants.
```

Figure 4-1. TIP Constants

```
*COMDECK TYPEUSER
*$J+ PAGE EJECT
**********************************
*                                *
*   USER TYPE DECLARATIONS DECK  *
*                                *
**********************************
*IF DEF,SVMODULE,1
*CALL SVMTYPE3
*IF DEF,ASYNC,1
*CALL TYPASYNC
*IF DEF,OLDIAG,4
*IF DEF,CLAIA,1
*CALL TYPODC
*IF -DEF,CLAIA,1
*CALL TYPOLD
*IF DEF,LDDMPO,1
*CALL TYPLDP
*IF DEF,LRZQSS,1
*CALL TYPLRZQSS
*IF DEF,IVT,1
*CALL TYPIVTC
*IF DEF,HASPTIP,1
*CALL HASPTYPE

If a new TIP is written with global types defined,
the user must supply a call:

   *CALL tipTYPE

A tipTYPE COMDECK defines the TIP's TYPE statements.
```

Figure 4-2. TIP Types

```
*COMDECK VARUSER
**********************************
*                                *
*   USER VARIABLES DEFINITIONS   *
*                                *
**********************************
*IF -DEF,OVLYBUILD
*CALL IPVAR
*CALL VARBLK
*IF DEF,SVMODULE,1
*CALL SVMVAR
*IF DEF,HIP,1
*CALL VARHIP
*IF DEF,MODE4,1
*CALL VARMD4
*IF DEF,TESTGEN,1
*CALL VARTST
*IF DEF,HASPTIP,1
*CALL HASPVAR
*IF DEF,ASYNC,1
*CALL VARASYNC
*IF DEF,HLIP,1
*CALL VARHLIP
*ENDIF
*IF DEF,OLDIAG,4
*IF DEF,CLAIA,1
*CALL VARODC
*IF -DEF,CLAIA,1
*CALL VAROLD
*IF DEF,LDDMPO,1
*CALL VARLDP

If a new TIP is written with new global arrays
defined, the user must supply a call:

   *CALL tipVAR
```

Figure 4-3. TIP Variable Arrays

```
*COMDECK VALUSER
*****************************
*                           *
*   USERVALUES DECLARATION  *
*                           *
*****************************
*IF -DEF,OVLYBUILD
*CALL IPVAL
*IF DEF,SVMODULE,1
*CALL SVMVALUE
*IF DEF,ASYNC,1
*CALL VALASYNC
*IF DEF,HASPTIP,1
*CALL HASPVAL
*ENDIF <-------- If a new TIP is written
*IF DEF,OVLYBUILD   with new global values
VALUE               defined, the user must
********************supply a call:
**                **
**   V A L U E S  **      *CALL tipVAL
**                **
********************
*ENDIF              A tipVAL COMDECK defines
*IF DEF,OLDIAG,4    values for the TIP
*IF DEF,CLAIA,1     defined variables
*CALL VALODC
*IF -DEF,CLAIA,1
*CALL VALOLD
```

Figure 4-4. TIP Value Declarations

```
*COMDECK FORWDUSER
**********************************
*                                *
*   USER FORWARDS DECLARATION    *
*                                *
**********************************
*IF DEF,MODE4,1
PROCEDURE PT4CSTATE(V3IN:BOOLEAN;V3ON=BOOLEAN);
FORWARD;

*CALL SVMFWD
*IF DEF,ASYNC,1
*CALL ASYNCFWD
*IF DEF,IVT,1
*CALL IVTFWD

   <-------- Forward declarations may be needed
             for a new TIP.  In this case, the
             user must supply a call:

                 *CALL tipFWD

             These declarations solve the
             forward calling requirements of the
             PASCAL compilation.
```

Figure 4-5. TIP Forward Declarations

The TIP writer should define a three character mnemonic to identify the TIP. In all the examples which follow, the mnemonic "tip" is used. The following list specifies the COMDECK for TIP-defined globals:

COMDECK Name	Contents
tipCON	TIP-defined constants
tipTYPE	TIP-defined types
tipVAR	TIP-defined variables
tipVAL	TIP-defined value initializations
tipFWD	TIP-defined forward declarations

The TIP-defined variables may be initialized during the system build procedures by means of MPEDIT directives. The directives are contained in unique COMDECKs as shown:

COMDECK Name	Contents
ZCNtip	TIP-defined constants
ZARtip	Array sizes for TIP-defined arrays
ZEXtip	Variable initialization statements

These uniquely defined COMDECKs are called from the standard system COMDECKs:

COMDECK Name	Contents
ZCONUSER	Call to user edit constants
ZARRUSER	Call to user array size definitions
ZEXUSER	Call to user edit directives

An example of this COMDECK structure is shown in figure 4-6.

```
*COMDECK ZCONUSER          User adds a call to his
       *                   constant defining deck,
       *                   ZCNtip. User defines
       *                   ZCNtip as a COMDECK.
*IF DEF, tipnam,1
*CALL ZCNtip
       *
       *
       *


*COMDECK ZARRUSER          User adds a call to his
       *                   array defining deck,
       *                   ZARtip. User defines
       *                   ZARtip as a COMDECK.
*IF DEF, tipnam,1
*CALL ZARtip
       *
       *
       *


*COMDECK ZEXUSER           User adds a call to his
       *                   field initialization
       *                   definition deck, ZEXtip.
       *                   User defines ZEXtip as a
*IF DEF, tipnam,1          COMDECK.
*CALL ZEXtip
       *
       *
       *
```

Figure 4-6. Global Initialization

GLOBAL EXPANSION FOR A NEW TIP

Standard system globals must be expanded to provide the necessary control linkage to base system routines. The following standard COMDECKs are referenced:

COMDECK Name	Necessary Additions
CONSTR4	• Worklist entry indices for the new TIP • New user terminal type names • New user TIP type names
TYPE	• Expanded array for new TIP types • Expanded array for new terminal classes

MPEDIT statements in the ZEXtip COMDECK are used to initialize the values in the expanded arrays created by these tables. These MPEDIT statements initialize the worklist control block, the terminal type table, and the terminal characteristic table (terminal class). Guidelines for using these statements are given in section 12.

Data structures are also customized for each new TIP by adding overlays to existing structures. In each case a unique name should be assigned for a COMDECK containing the type declarations:

COMDECK Name	Contents
tipLCB	TIP-defined extention to LCB
tipTCB	TIP-defined extention to TCB
tipMLCB	TIP-defined extention to TPCB

The TIP writer should insert a call to these TIP defined overlays in the corresponding system COMDECKs:

COMDECK Name	Contents
TYPELCB	Contains calls to TIP-defined overlay
TYPETCB	Contains calls to TIP-defined overlay
TYPENCLCB	Type declarations for the TPCB

Examples of these COMDECKs are given in figures 4-7.

SERVICE MODULE TABLES

The service module requires four tables to be defined for each TIP in the system. These tables define the TIP specifications for processing line and terminal table construction. In general, these tables define the valid ranges of values provided by a user when the system is configured. The table definition may be placed either in the TIP's variable definition COMDECK, or in a unique COMDECK.

COMDECK Name	Contents
unique name	LCB field descriptor table
unique name	TCB field descriptor table
unique name	LCB action table
unique name	TCB action table

If the user determines that the TIP requires one or more new CE error message types, an overlay should be added to that data structure.

```
*COMDECK TYPELCB
*****************************
*                           *
*   LCB DEFN FOR RELEASE 4  *
*                           *
*****************************
*IF DEF,MODE4,1
*CALL TMD4D2
*IF DEF,HASPTIP,1
*CALL HASPLCB
*IF DEF,ASYNC,1
*CALL ASYNCLCB
```

The user must put his call for the new TIP here:

```
    *CALL tipLCB
```

The tipLCB deck is defined as a numbered overlay.

```
*COMDECK TYPETCB
*****************************
*                           *
*   TCB DEFN FOR RELEASE 4  *
*                           *
*****************************
*CALL TYPEIVT
*IF DEF,MODE4,1
*CALL TMD4D1
*IF DEF,HIP,1
*CALL THIPD1
*IF DEF,ASYNC,1
*CALL TASYNC
*IF DEF,HASPTIP,1
*CALL HASPTCB
*IF DEF,HLIP,11
*CALL TYP1HLIP
```

The user inserts his call for the new TIP here:

```
    *CALL tipTCB
```

The tipTCB deck is defined as one or more overlays.

```
*COMDECK TYPENCLCB
*********************************
*                               *
*   USER APPS FOR NCLCB -- R4   *
*                               *
*********************************
 IF DEF,HASPTIP,1
 CALL HASPMLCB
 IF DEF,MODE4,1
 CALL TMD403
*IF DEF,ASYNC,1
*CALL ASYNCMLCB
```

The user inserts his call for the new TIP here:

```
    *CALL tipMLCB
```

The tipMLCB deck is defined as one or more numbered overlays.

Figure 4-7. COMDECKS for LCB, TCB, and TPCB

NOTE

If these new types of error messages are used, changes to the host's engineering file analyzer will also be required. These changes are not discussed in this document.

The following COMDECKS must be added:

COMDECK Name	Contents
tipCEFILE	TIP-defined overlay for CE error messages

A call to this COMDECK is added to the standard system COMDECK:

COMDECK Name	Contents
TYPECNCE	Defines the CE error message overlay.

Figure 4-8 shows the CE error message COMDECK.

```
*COMDECK TYPECNCE
*********************************
*                               *
*   RELEASE 4 CNCEFILE DEFNS    *
*                               *
*********************************
*****************************************************
                                        MODE4TIP
         3:(VM4LINO    : INTEGER      LINE NUMBER
            VM4CA,                    CLUSTER ADDRESS
            VM4TA      : B08BITS;     TERMINAL ADDRESS
            VM4DT      : NODEVTYP;    DEVICE TYPE
            VM4TC      : NOTCLASS;    TERMINAL CLASS
            VM4SPARE   : B02BITS;     SPARE
            VM4ERRCNT  : B06BITS);    ERROR COUNT

                                        HASP TIP
         4:(HSLINO     : INTEGER);    LINE NUMBER

                                        HDLC LIP
         5:(HLINO      : INTEGER;     LINE NUMBER
            HLNID      : B08BITS);    REMOTE NODE ID

                                        ASYNCTIP
         6:(ASYLINO    : INTEGER;     LINE NUMBER
            ASYCA,                    CLUSTER ADDRESS
            ASYTA      : B08BITS;     TERMINAL ADDRESS
            ASYDT      : NODEVTYP;    DEVICE TYPE
            ASYTC      : NOTCLASS);   TERMINAL CLASS
         *CALL tipCEFILE
```

Figure 4-8. CNCEFILE COMDECK

PASCAL MODIFICATIONS

The PASCAL portions of the TIP are added as a single COMDECK with the following name:

COMDECK Name	Contents
tipTIP	All PASCAL code for the user TIP

This COMDECK is called from the standard system COMDECK as indicated:

COMDECK Name	Contents
USERAPPS	Call to PASCAL code for TIPs

Figure 4-9 shows the PASCAL code COMDECK.

```
*COMDECK USERAPPS
*$J+
*****************************************
*                                       *
*   USER APPLICATIONS PROGRAM CALLS     *
*                                       *
*****************************************
*CALL IP
*IF DEF,HLIP,1
*CALL HLIP
*IF DEF,IVT
*CALL FBFMAD
*CALL PARSER
*CALL PTIVTCMD
*ENDIF
*IF DEF,HIP,1
*CALL HIP30
*IF DEF,MODE4,1
*CALL MODE4TIP
*IF DEF,HASPTIP,1
*CALL HASPTIP
*IF DEF,ASYNC,2
*CALL ASY100MS
*CALL ASYNCTIP
*IF DEF,OLDIAG,1
*CALL OLDGOVL
*IF DEF,LDDMPO,1
*CALL LDMPOVL
*IF DEF,SVMODULE,1
*CALL SVM
*IF DEF,LRZQSS
*CALL LRZQSS
*ENDIF
```

If a new TIP is written, the call must be included here:

*CALL tipTIP

This calls all OPS-level TIP COMDECKS.

Figure 4-9. PASCAL Language COMDECK

MACROASSEMBLY LANGUAGE MODIFICATIONS

The macroassembler portions of the TIP (state programming language modules) are added as a single COMDECK with the following name:

COMDECK Name	Contents
tipASSEM	All assembly language decks in the TIP

This COMDECK is called from the standard system COMDECK as indicated:

COMDECK Name	Contents
ASMUSER	Call to TIP's assembly language modules

Figure 4-10 shows the PASCAL code COMDECK.

Four separate assembly decks (idents) should be defined for the TIP. These are the input states programs, the input translation tables, the output text processing state programs, and the output translation tables. If a second pass is defined for input processing, a fifth deck is also needed: the input text processing state programs with their associated input text processing translation tables. Any of these decks can be omitted if not needed. More decks can be defined to separate subfunctions, but in this case more build directives are required. If any macroassembler programs are to be called from the PASCAL code, these can be specified as one or more idents.

To simplify the build directives, each program should have one entry point with the same name as the deck (ident). If COMDECKs are defined for each module, the COMDECK name should be the same as the deck name (ident).

```
*DECK ASMUSER
         NAM     ASMUSER
*************************************
*                                   *
*   DEVELOPMENTAL FILE ASSEMBLIES   *
*                                   *
*************************************
DUMFIL   NUM   0          DUMMY CARD FOR MPLINK
         END   ASMUSER
*IF DEF,LRZQSS,1
*CALL ASMLRZQSS
*IF DEF,MODE4,1
*CALL REL4MD4
*IF DEF,ASYNC,1
*CALL REL4ASY
*IF DEF,HASPTIP,1
*CALL HASPASSEM
*IF DEF,HLIP,1
*CALL LIPIST
*IF DEF,OLDIAG,1
*CALL ISPOLD
```

New TIP must add conditional definition here:

*CALL namTIP

Figure 4-10. ASMUSER Deck

BUILD PROCEDURE

The completed source program is placed on the user input file UCCP, and the standard build procedures are followed. These procedures are described in the NOS 1 Installation Handbook.

The user TIP may be defined as a build option. However, if all systems are to contain the user TIP, the options may be omitted. The examples assume user TIP1 is not a build option (UTIP1) and user TIP2 is a build option (UTIP2).

Build options are involved in two different build steps: the step which generates the CCP object code library (CCPBLB), and the step which generates a CCP variant load module (CCPVAR). The CCP standard build procedures allow a TIP to be included in object library and excluded from a variant build by using the following procedures:

Build step CCPBLB gets its TIP definition from a common deck called TIPDEFS. Build step CCPVAR gets it TIP definition through the TS parameter in the specified variant definition which is an entry in the USERBPS file (see the Expand section of the CYBER Cross Build Utilities Reference Manual). The TS parameter provides options for up to four user-written TIPs. When these options are selected, the following define statements are generated:

```
*DEFINE UTIP1
*DEFINE UTIP2
*DEFINE UTIP3
*DEFINE UTIP4
```

Figure 4-11 shows COMDECK TIPDEFS. User TIP definition should be inserted as indicated if the source code is conditional for compilation.

```
*COMDECK TIPDEFS
*TEXT  DEFAULT TIP DEFINITIONS FOR FULL COMPILE/ASSEMBLY
*DEFINE MODE4
*DEFINE ASYNC
*DEFINE ASYNCEXT
*DEFINE HASPTIP
*DEFINE X25
*DEFINE UTIP1   <------------ Define one to four
*DEFINE UTIP2                 user TIPs
*ENDTEXT
```

Figure 4-11. Compile Definition

The rest of this section is concerned only with the CCPVAR build step, and assumes the user always wants to have the TIPs compiled.

The TIP writer must add the following information to the Autolink input directives decks (see the Autolink section of the CYBER Cross Build Utilities Reference Manual):

- Application (APPL) directives for the new TIPs (a TIP is considered an application) (see figure 4-12)
- Base definition directive if the user TIP is not a build option (see figure 4-13)
- Optional definition directive if the user TIP is a build option (see figure 4-13)
- COMDECK to describe the linking characteristics of the individual modules within the TIP (see figure 4-14)
- Calls to COMDECK (see figure 4-15)

```
*DECK ALIN DIR
      .
      .
      .
APPL = ASYNC
APPL = MODE4
APPL = HASPIP
APPL = UTIP1   ]<------- New TIP directives
APPL = UTIP2   ]         defining the applications
APPL = HIP
      .
      .
      .
```

Figure 4-12. Autolink Application Directives

```
*COMDECK ALDEFS
      .
      .
      .
DEFBASE = SVMODULE
DEFBASE = IVT
DEFBASE = UTIP1   <------ New TIP base directive
DEFBASE = CONSOLE         (TIP is to be included in
DEFBASE = BASESYS         all builds)
      .
      .
      .
*IF  DEF, MODE4
DEF  =  MODE4
*IF  DEF, HASPTIP
DEF  =  HASPTIP
*IF  DEF, UTIP2   <------ Optional TIP definition
DEF  =  UTIP2
*IF  DEF, HLIP
DEF  =  HLIP
      .
      .
      .
```

Figure 4-13. Autolink Definition Directives

```
*COMDECK ALUTIP1
MOD = UT1OPS  (P = F, APPL = UTIP1)  ]
MOD = UT1M1   (P = P, APPL = UTIP1)  ]  Used to
MOD = UT1M2   (P = P, APPL = UTIP1)  ]  generate
      .                              ]  the linking
      .                              ]  directives
      .                              ]  for the
MOD = UT1MN (P = P, APPL = UTIP1)    ]  user TIP
*COMDECK ALUTIP2                     ]  modules
MOD = UT2OPS  (P = F, APPL = UTIP2)  ]
MOD = UT2M1   (P = P, APPL = UTIP2)  ]
MOD = UT2M2   (P = P, APPL = UTIP2)  ]
      .                              ]
      .                              ]
      .                              ]
MOD = UT2MN   (P = P, APPL = UTIP2)  ]
```

Figure 4-14. Autolink Module Directives

```
*DECK ALINPDIR
      .
      .
      .
*CALL ALASYNC
*CALL ALMODE4
*CALL ALHIP
*CALL ALUTIP1   ]<------- Calls to the User TIP
*CALL ALUTIP2   ]         COMDECKS
*CALL ALHASP
      .
      .
      .
```

Figure 4-15. Autolink Application COMDECK Calls

The principal data structures used by the TIP are as follows:

- Worklists, worklist control blocks, and worklist entries
- Line control blocks
- Terminal control blocks
- Command driver packets
- Mux LCBs (MLCBs) and text processing control blocks (TPCBs)
- Port tables
- TIP type tables
- Terminal characteristics tables
- Code translation tables
- Dynamic buffers

This section discusses those aspects of the data structures that are used by the TIP.

WORKLISTS

The worklist control block (WLCB) is the system data structure that controls entry to the user TIP. The TIP writer must initialize some worklist control block fields at build time. The run time TIP, however, cannot access the table directly. The TIP can extract fields from the global worklist variable, BWWLENTRY. Figure 5-1 shows the worklist control block and the worklist entries (WLE). The TIP must be able to decode entries from the Mux-2 level, from the base timer, from the service module, and from the BIP. The TIP can generate worklists to the service module. It is often useful to have the TIP generate worklists to itself following the occurance of an event. The TIP writer must define a worklist for his TIP, and be able to interpret all uses of this worklist by other modules which request the TIP to perform a task.

LINE CONTROL BLOCK (LCB)

One line control block is assigned for each communication line adapter (CLA) which is attached to the NPU. The LCBs are contained in an array of fixed length tables which are built during system initialization. The TIP can access a few of the fields in the system-defined standard portion of the LCB. The TIP can access all the fields in the TIP-defined variable portion of the LCB. In some systems the variable portion may be longer than two words.

To access LCB fields, the TIP uses a variable of type BZLCBP, as shown in the example below:

```
VAR
    (lcbptr) : BZLCBP;
BEGIN
    WITH (lcbptr)↑DO
    BEGIN
        .
        .
        .
```

The LCB is defined in figure 5-2. The figure defines (1) all the fields and (2) the subset of fields that is available to the TIP.

TERMINAL CONTROL BLOCK

One terminal control block (TCB) is created for each terminal device that can support an independent data stream to a host application program. The TCB has two parts:

- A base part. Only two boolean variables may be set by the TIP in this portion.
- A variable part used by the TIP. If the device is interactive, the variable part must include all the interactive virtual terminal (IVT) definitions. The TIP can read the IVT fields, but it cannot change them. The unused space in the variable portion is available for user TIP-defined fields.

When the TIP accesses the TCB, it must use a variable pointer of the BOBUFPTR type. This is necessary since TCBs are contained in buffers. The TCB overlay type is BSTCB. A TCB is accessed as shown in the example below:

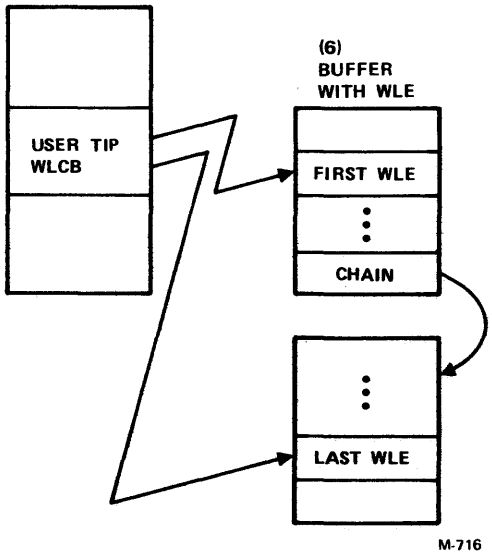
```
VAR
    (tcbptr) : BOBUFPTR;
BEGIN
    WITH (tcbptr)↑.BSTCB DO
    BEGIN
        .
        .
        .
```

The TCB is defined in figure 5-3. The figure defines (1) all the fields (including the IVT parameters) and (2) the subset of fields that is available to the TIP.

COMMAND PACKET

The command packet is an area holding all the parameters that the multiplex subsystem requires when a request for I/O is made. The request takes the form of a call to PBCOIN, with the command packet attached. The TIP fills the command packet fields; it is the responsibility of the multiplex subsystem to alter the MLCB using these field values.

WORKLIST CONTROL BLOCK (ARRAY OF WLCB)



WLCB

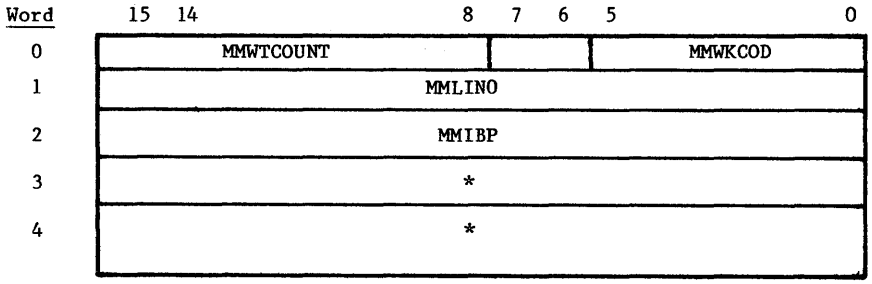
	15	14	9	8	7	0
0	F1		BYCNT			
1			BYPUT			
2			BYGET			
3	BYFEINC			BYINC		
4	BYWLINDEX			*		
5	*					
6	F2	BYMAXCNT		BYPAGE		
7	BYPRADDR					

F1 = BYCOUNTEND
F2 = BYWLREQ
* = reserved for CDC

Field Name	Definition
BYCONTEND	multiprocessor continue flag
BYCNT	number of WLEs
BYPUT	first WLE in chain
BYGET	last WLE in chain
BYFEINC	first entry index
BYINC	WLE size in words
BYWLINDEX	worklist index to WLCB
BYWLREQ	WLE sent to program if set true
BYMAXCNT	number of WLEs to process before next pass
BYPAGE	program page
BYPADDR	program address

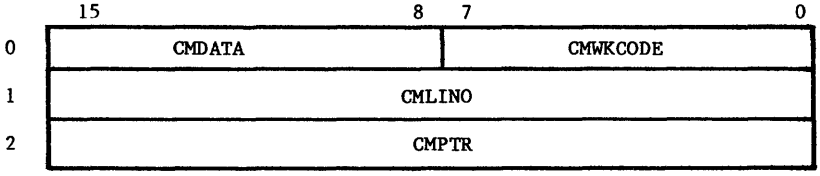
Figure 5-1. Worklist Control Block and Worklist Entries (1 of 2)

Data structure for WLE from MUX Subsystem or Base Timer:

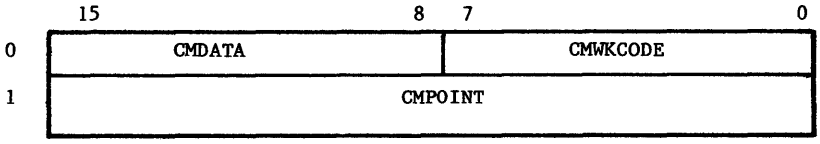


Field Name	Type	Description
MMWTCOUNT	subrange	wait count for timer WC
MMWCCOD	subrange	work code
MMLINO	BOLINO	line number
MMIBP	BOBUFPTR	buffer or TCB address

Data structure for WLE to or from SVM or BIP:

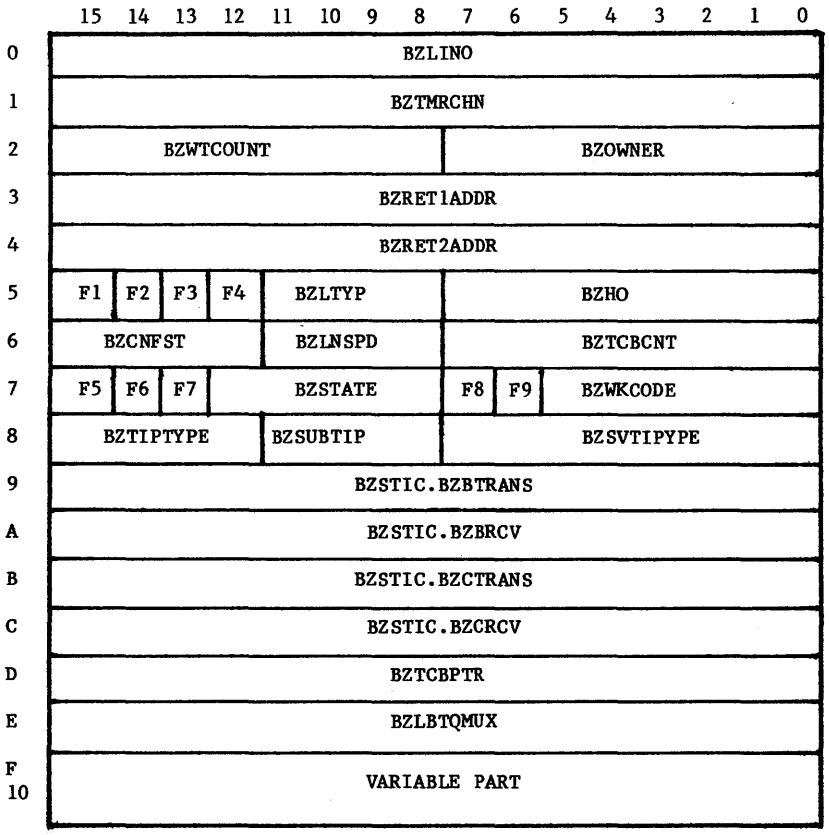


or



Field Name	Type	Description
CMDATA	subrange	variable use data byte
CMWKCODE	subrange	work code
CMLINO	BOLINO	line number
CMPOINT	BOBUFPTR	pointer to buffer or TCB
CMPTR	BOBUFPTR	pointer to buffer or TCB

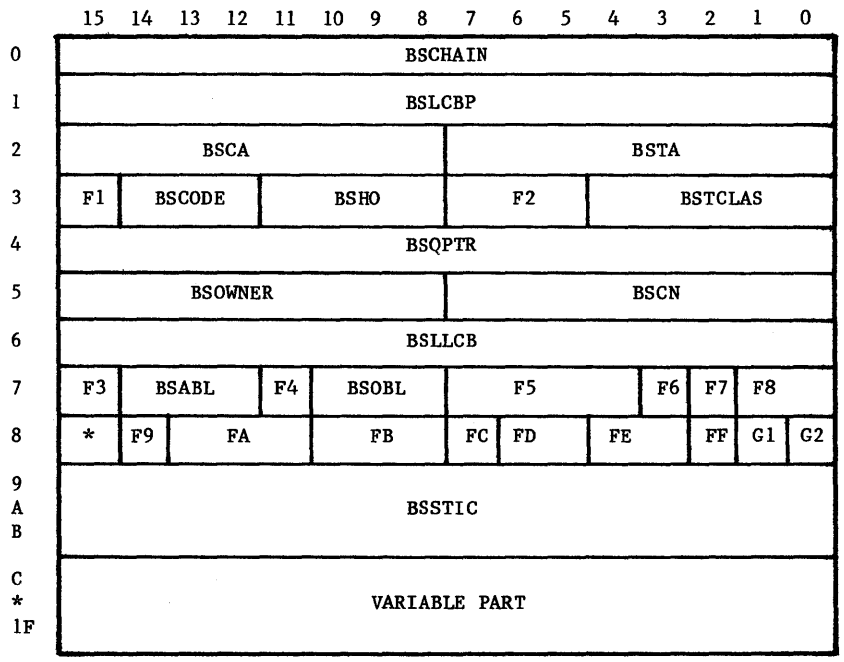
Figure 5-1. Worklist Control Block and Worklist Entries (2 of 2)



ID	Name	Description
	BZLINO	LINE NUMBER
	BZTMRCHN	ACTIVE LCB TIME CHAIN
	BZWTCOUNT	WAIT COUNT
	BZOWNER	NODE ID OF CS
	BZRET1ADDR	INPUT RETURN ADDRESS
	BZRET2ADDR	OUTPUT RETURN ADDRESS
F1	BZTAPEX	TIMAL APPENDAGE EXISTS
F2	BZCHECKQS	OUTPUT QUEUED
F3	BZSMRESP	SM RESPONSE RECEIVED
F4	BZSMTO	SM BEING TIMED OUT
	BZLTYP	LINE TYPE
	BZHO	HOST ORDINAL
	BZCNFST	CURRENT CONFIGURATION STATE
	BZLNSPD	LINE SPEED
	BZTCBCNT	NUMBER OF TCBS ATTACHED
F5	BZTOUTPUT	TERMINAL OUTPUT TIMED OUT
F6	BZTINPUT	TERMINAL INPUT TIMED OUT
F7	BZDIS	LINE DISABLED
	BZSTATE	LINE STATE
F8	BZAUTO	AUTO RECOGNITION
F9	BZDIAG	DIAGNOSTIC IN PROGRESS
	BZWKCODE	LAST WORK CODE
	BZTIPTYPE	TIP TYPE
	BZSUBTIP	SUB TIP TYPE
	BZSVTIPTYPE	SAVE TIP TYPE
	BZBTRANS	BLOCKS TRANSMITTED
	BZBRCV	BLOCKS RECEIVED
	BZCTRANS	CHARACTERS TRANSMITTED
	BZCRCV	CHARACTERS RECEIVED
	BZTCBPTR	TCB CHAIN
	BZLBTOMUX	LAST BUFFER TO MUX
BZSUBTIP	SUBRANGE	SET BY TIP ONLY IF AUTORECOGNIZED BY TIP
BZTCBPTR	BOBUFFPTR	POINTER TO FIRST TCB CHAINED ON THIS LINE
BZLBTQMUX	BOBUFFPTR	LAST OUTPUT BUFFER SENT TO MUX (DEBUG USE)

Figure 5-2. Line Control Block

Base part:



ID	Name	Description
	BSCHAIN	POINTER TO NEXT TCB ON LINE
	BSLCBP	LINE CONTROL BLOCK POINTER
	BSCA	CLUSTER ADDRESS
	BSTA	TERMINAL ADDRESS
F1	BSSTOP	UPLINE STOP OUTSTANDING
	BSCODE	CODE SET IN USE
	BSHO	HOST ORDINAL
F2	BSDEVTYPE	DEVICE TYPE
	BSTCLASS	TERMINAL CLASS
	BSQPTR	QUEUE POINTER
	BSOWNER	
	BSCN	CONNECTION NUMBER
	BSLLCB	LOGICAL LINK CONTROL BLOCK POINTER
F3	BSINOP	TERMINAL INOPERATIVE
	BSABL	AVAILABLE BLOCK LIMIT
F4	BSTBRCCNF	TCB TO BE RECONFIGURED FLAG
	BSOBL	OUTSTANDING BLOCK LIMIT
F5	BSLBTPROC	LAST BLOCK TYPE PROCESSED
F6	BSACPIN	ACCEPT INPUT
F7	BSACPOUT	ACCEPT OUTPUT
F8	BSIPRI	INPUT PRIORITY
F9	BSWAIT	WAITING FOR INIT FLAG
FA	BSBSNLAST	BSN OF LAST BACKED BLOCK
FB	BSBSNCRNT	BSN OF CRNT OUTPUT BLOCK
FC	BSTOTERM	TCB TO BE TERMINATED
FD	BSPARITY	CHARACTER PARITY
FE	BSCHLEN	CHARACTER LENGTH
FF	BSPGWAIT	PAGE WAIT SELECTED
G1	BSXPARENT	TRANSPARENT INPUT SELECTED
G2	BSHOTOGL	HOST ORDINAL TOGGLE
	BSSTIC	STATISTICS

Figure 5-3. Terminal Control Block (1 of 3)

Printout cancelled by operator

Fields accessible to the TIP:

Field Name	Type	Description
BSCHAIN	BOBUFPTR	next TCB chained on the line
BSCA	subrange	cluster address
BSTA	subrange	terminal address
BSSTOP	BOOLEAN	true if STP block outstanding to host
BSCODE	subrange	code set in use
BSDEVTYPE	subrange	device type
BSTCLASS	subrange	terminal class
BSINOP	BOOLEAN	true if terminal inoperative SM sent to host by TIP, set and cleared by TIP
*BSPARITY	subrange	parity in use
*BSCHLEN	subrange	character length
*BSPGWAIT	BOOLEAN	set true if page wait in effect
*BSXPARENT	BOOLEAN	set true if next input is to be transparent, reset by TIP if processed

*IVT variable fields in base

IVT variable part:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
C	BSPGWIDTH								BSPGLENGTH							
D	BSCANCHR								BSCNTRLCHAR							
E	BSUSR1								BSUSR2							
F	F1	F2	*						BSXCNT							
10	F3	F4	F5	F6	F7			F8	BSXCHAR							
11	BSBSCHAR								BSABTLINE							
12	BSCRIDLES								BSLFIDLES							

ID	Name	Type	Description
	BSPGWIDTH	SR	PAGE WIDTH
	BSPGLENGTH	SR	PAGE LENGTH
	BSCANCHAR	C	CANCEL CHARACTER
	BSCNTRLCHAR	C	CONTROL CHARACTER
	BSUSR1	C	USER BREAK 1 CHARACTER
	BSUSR2	C	USER BREAK 2 CHARACTER
F1	BSXTO	B	TRANSPARENT TIMEOUT
F2	BSXCHRON	B	TRANSPARENT DELIMITED BY CHAR
	BSXCNT	SR	TRANSPARENT INPUT CHARACTER COUNT
F3	BSCRCALC	B	CARRIAGE RETURN
F4	BSLFCALC	SR	LINE FEED
F5	BSECHOPLX	B	ECHOPLEX
F6	BSINDEV	B	INPUT DEVICE

Figure 5-3. Terminal Control Block (2 of 3)

```
F7      BSOUTDEV        SR      OUTPUT DEVICE
F8      BSAPL           SR      APL MODE
        BSXCHAR         C       TRANSPARENT TERMINATOR CHARACTER
        BSBSCHAR        C       BACKSPACE CHARACTER
        BSABTLINE       C       ABORT LINE CHARACTER
        BSCRIDLES       C       NUMBER OF IDLES AFTER CR
        BSLFIDLES       C       NUMBER OF IDLES AFTER LF

SR = subrange
C  = CHAR
B  = BOOLEAN
```

Figure 5-3. Terminal Control Block (3 of 3)

The TIP can use seven of the possible types of command packets:

- input
- input after output
- output
- terminate input
- terminate output
- disable line
- turn line around

A sample format of one of these command packets is shown in figure 5-4. Each of the command packets are described in detail in the multiplex subsystem section.

MUX LINE CONTROL BLOCK (MLCB)

The MLCB is used to control the input and ouput of characters over a line, and to provide variable storage for input state processing. This table is never directly accessed by the TIP. This description is for use in TIP debugging. Figure 5-5 shows the MLCB.

TEXT PROCESSING CONTROL BLOCK

The TIP uses the text processing control block (TPCB) to pass information to the microassembly language text processor, and to the TIP's own text processing state programs. These programs also pass data back to the OPS-level TIP through fields in the TPCB. Note that the TPCB is the only way data can be passed back and forth between the OPS-level TIP and the text processing modules of the TIP. The TIP sets up the TPCB prior to calling the text processor via PTTPINF. Figure 5-6 shows the TPCB.

PORT TABLES

The port tables (NAPORT) are multiword entries in an array which is indexed by the port index. The table is not accessible to the run time TIP, nor are its fields initialized by the TIP writer. However, during the debugging phase of a new TIP, it is often useful to observe the contents of the port table and to locate (through the port table) the multiplex LCBs that are being used for a particular line.

The port table is shown in figure 5-7.

TIP TYPE TABLE

The TIP type table contains those system parameters that are necessary to control a TIP. These fields are never accessed by a TIP. The TIP writer must, however, initialize specified fields at build time. The fields to be initialized are defined in section 12.

Figure 5-8 shows the TIP type table.

TERMINAL CHARACTERISTICS TABLE

The terminal characteristics table is used for two purposes:

- To initialize the TCB with default values for IVT parameters
- To initialize the TCB with default values for user calls to the multiplex subsystem.

In the second case, the default values are overridden when the TIP specifies a value for that parameter in the command packet. The run time TIP does not use the terminal characteristic table. However, the TIP writer must supply all the necessary initialization values at build time. These fields are defined in section 12.

Figure 5-9 shows the terminal characteristics table.

CODE TRANSLATION TABLES

The multiplex subsystem defines the format of a code translation table: it is a packed array of characters. The untranslated character is used as an index to find the translated character. Parity should be stripped from the untranslated character: this reduces the size of the table by fifty percent.

Figure 5-10 shows the format of a code translation table.

DYNAMIC BUFFERS

The TIP accesses and manipulates buffers of various sizes. CCP offers four standard buffer sizes: 8, 16, 32, and 64 words long. The TIP accesses a buffer using a pointer word of B0BUFPTR type, as shown:

```
VAR
    (bufptr) : B0BUFPTR;
BEGIN
    WITH (bufptr)↑ DO
    BEGIN
        .
        .
        .
```

Data buffers may contain part or all of a message. In any system there is only one size of data buffer, though different systems may each have a different size data buffer. The standard system uses a 64 word buffer. TCBs also use data buffers, but the size and the content is controlled by the system (the TIP writer selects a size at build time, if his TIP is to be the only TIP in the system).

The TIP writer also uses a B0BUFPTR type pointer to access user-defined variables fields in the TCB. The overlay type for naming TCB fields is BSTCB. A TCB is accessed as shown:

```
VAR
    (tcbptr) : B0BUFPTR;
BEGIN
    WITH (tcbptr)↑.BSTCB DO
    BEGIN
        .
        .
        .
```

A chain of data buffers is shown in figure 5-11. The data buffers are defined in figure 5-12.

SYSTEM CONSTANT DEFINITIONS

Figure 5-13 defines the system constants that are available for TIP use.

SYSTEM TYPE DEFINITIONS

Figure 5-14 defines the system types that are available for TIP use.

SYSTEM VARIABLE DEFINITIONS

The TIP can use a limited set of system variables as shown in figure 5-15. Most of these variables are used to interface to system procedures.

SYSTEM ENGINEERING FILE WORK AREA

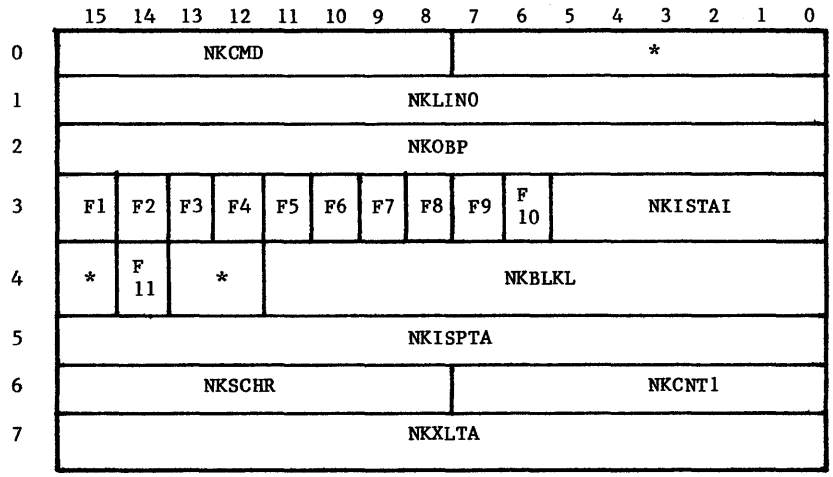
The TIP makes entries to the system engineering work file when component failures are detected. The prototype of the engineering work file entry is built in the system global CNCEOVLY. This global is an array indexed by interrupt level. The TIP can issue a system engineering file entry at either the OPS level or at the Mux-2 level. Each entry in the array is structured for use by the TIP and by other already defined entries. The TIP writer defines field names for constructing each entry. After initializing the work area, the TIP calls PNCEFILE to issue the message.

An example of TIP access to the engineering work file area is as follows:

```
WITH (tcbpointer)↑.BSTCB DO
   WITH CNCEOVLY [(LEVEL)] DO
      BEGIN
      CNCECODE          :=  (error file entry code);
      (line number)     :=  (lcbpointer).BZLINO;
      (cluster address) :=  BSCA;
      (terminal address) :=  BSTA;
      (device type)     :=  BSDEVTYPE;
      (terminal class)  :=  BSTCLASS;
      (error count)     :=  (occurrence count);
   END;
```

The engineering work file area is shown in figure 5-16.

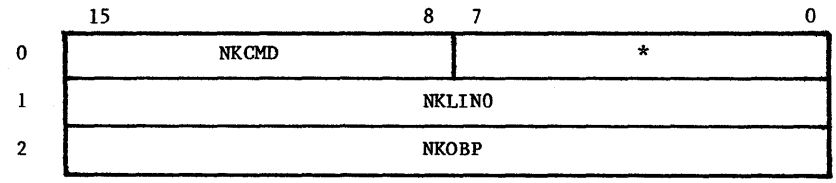
INPUT or INPUT after OUTPUT:

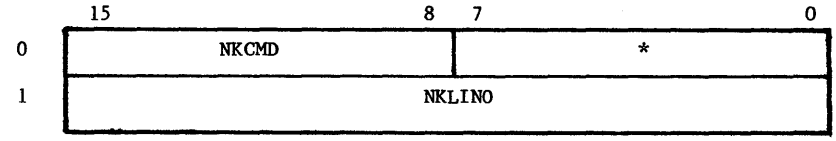


F_1	NKUOP1
F_2	NKUOP2
F_3	NKUOP3
F_4	NKUOP4
F_5	NKUOP5
F_6	NKUOP6
F_7	NKUOP7
F_8	NKUOP8
F_9	NKMVB
F_{10}	NKRPRT

Figure 5-4. Command Packets (1 of 2)

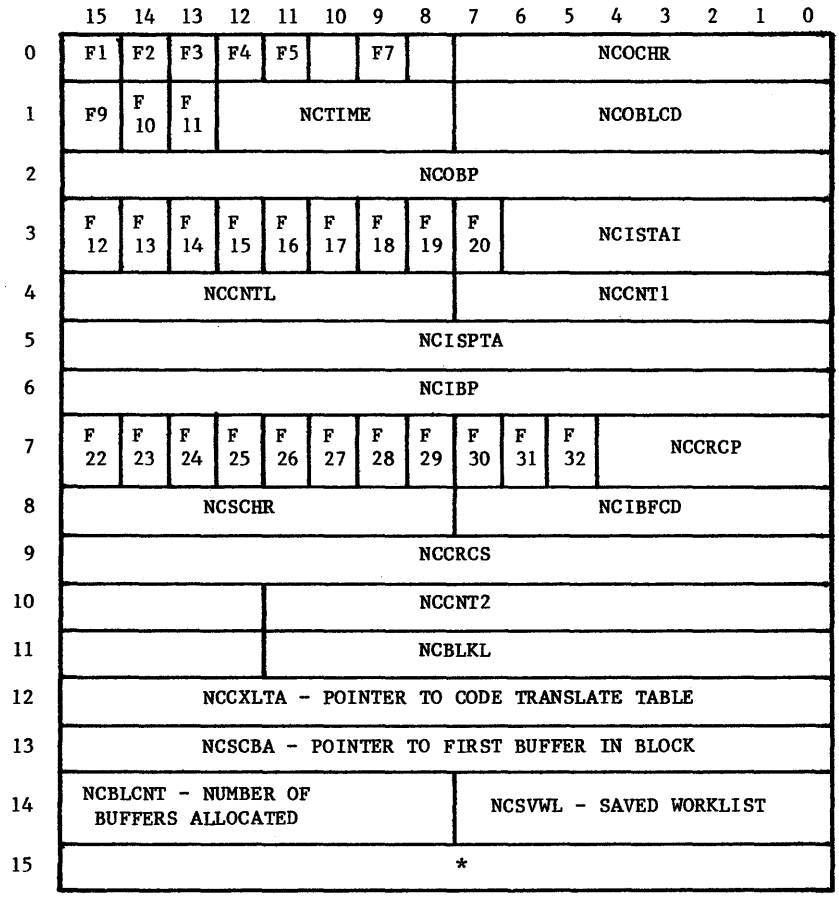
OUTPUT:





Flags	Field Name	Type	Definition
	NKCMD	B08BITS	COMMAND
	NKLINO	INTEGER	LINE NUMBER
	NKOBP	BOBUFPTR	OUTPUT BUFFER POINTER
F1	NKUOP1	BOOLEAN	USER OPTION FLAG 1 (MUX BIT 15)
F2	NKUOP2	BOOLEAN	USER OPTION FLAG 2 (MUX BIT 14)
F3	NKUOP3	BOOLEAN	USER OPTION FLAG 3 (MUX BIT 13)
F4	NKUOP4	BOOLEAN	USER OPTION FLAG 4 (MUX BIT 12)
F5	NKUOP5	BOOLEAN	USER OPTION FLAG 5 (MUX BIT 11)
F6	NKUOP6	BOOLEAN	USER OPTION FLAG 6 (MUX BIT 10)
F7	NKUOP7	BOOLEAN	USER OPTION FLAG 7 (MUX BIT 9)
F8	NKUOP8	BOOLEAN	USER OPTION FLAG 8 (MUX BIT 8)
F9	NKMVB	BOOLEAN	MOVE USER BITS TO LCB FLAG
F10	NKRPRT	BOOLEAN	STRIP PARITY FLAG
	NKISTAI	B06BITS	INPUT STATE INDEX
F11	NKSCENBL	BOOLEAN	SPECIAL CHARACTER TO BE CHANGED
	NKBLKL	B012BITS	BLOCK LENGTH COUNT
	NKISPTA	INTEGER	INPUT STATE POINTER TABLE ADDRESS
	NKSCHR	CHAR	SPECIAL CHARACTER
	NKCNT1	B08BITS	CHARACTER COUNT 1 VALUE (IF NOT = 0)
	NKLXTA	INTEGER	TRANSLATE TABLE ADDRESS

Figure 5-4. Command Packets (2 of 2)



* reserved for CDC

Flags	Field	Type	Description
F1	NCEOBL	BOOLEAN;	END OF BLOCK FLAG
F2	NCNXCCA	BOOLEAN;	NEXT OUTPUT CHAR AVAILABLE
F3	NCLCT	BOOLEAN;	LAST CHARACTER TRANSMITTED
F4	NCBCREQ	BOOLEAN;	BUFFER CHAINING REQUIRED
F5	NCOMFRO	BOOLEAN;	OUTPUT MESSAGE IN PROGRESS
F7	NCODDIN	BOOLEAN;	ODD RECEIVE
	NCOCHR	CHAR;	NEXT OUTPUT CHARACTER
F9	NCSUPCHAIN	BOOLEAN;	SUPPRESS BUFFER CHAINING
F10	NCOBT	BOOLEAN;	GENERATE OBT
F11	NCBZL	BOOLEAN;	RESET TIMER
	NCTIME	B05BITS;	MLX TIMER
	NCOBLCD	B08BITS;	LCD OF OUTPUT BUFFER
	NCOBP	BOBUFPTR;	OUTPUT BUFFER POINTER
F12	NCRINCH	BOOLEAN;	INPUT CHAR IN RIGHT BYTE
F13	NCCAREC	BOOLEAN;	CHARACTER RECEIVED FLAG
F14	NCRIGHTC	BOOLEAN;	LEFT/RIGHT SOURCE FLAG
F15	NCINFRO	BOOLEAN	INPUT MESSAGE IN PROGRESS
F16	NCNOXL	BOOLEAN;	CODE TRANSLATE ACTIVE
F17	NCRPRT	BOOLEAN;	STRIP PARITY BIT
F18	NCSCF	BOOLEAN;	TEXT PROCESS SUPPRESS CHAIN FLAG
F19	NCLASTCH	BOOLEAN;	TEXT PROCESS SOURCE LCD REACHED
F20	NCEOSR	BOOLEAN;	TEXT PROCESS END OF SOURCE REACHED
	NCISTAI	B06BITS;	INPUT STATE PROGRAM INDEX
	NCCNTL	B08BITS;	CHARACTER COUNT ONE LIMIT
	NCCNT1	B08BITS;	CHARACTER COUNTER ONE
	NCISPTA	INTEGER;	INPUT STATE POINTER TABLE ADDRESS
	NCIBP	BOBUFPTR;	INPUT BUFFER POINTER
F22	NCUOP1	BOOLEAN;	USER OPTIONAL FLAG 1
F23	NCUOP2	BOOLEAN;	USER OPTIONAL FLAG 2

Figure 5-5. Multiplex LCB (MLCB) (1 of 2)

```
F24     NCUOP3          BOOLEAN;      USER OPTIONAL FLAG 3
F25     NCUOP4          BOOLEAN;      USER OPTIONAL FLAG 4
F26     NCUOP5          BOOLEAN;      USER OPTIONAL FLAG 5
F27     NCUOP6          BOOLEAN;      USER OPTIONAL FLAG 6
F28     NCUOP7          BOOLEAN;      USER OPTIONAL FLAG 7
F29     NCUOP8          BOOLEAN;      USER OPTIONAL FLAG 8
F30     NCETX           BOOLEAN;      DELAYED ETX WLE GENERATION FLAG
F31     NCMRTO          BOOLEAN;      MODEM RESPONSE TIMEOUT FLAG
F32     NCCARR          BOOLEAN;      LINE CARRIER TYPE (1=CONTROLLED)
        NCCRCP          BO5BITS;      CRC POLYNOMIAL
        NCSCHR          CHAR;         SPECIAL CHARACTER
        NCIBFCD         BO8BITS;      FCD OF INPUT BUFFER
        NCCRCS          INTEGER;      CRC ACCUMULATION
        NCCNT2          BO12BITS;     CHARACTER COUNTER TWO
        NCBLKL          BO12BITS;     CHARACTER COUNTER TWO LIMIT
        NCCXLTA         INTEGER;      CODE TRANSLATE TABLE ADDRESS
        NCSCEA          BOBUFPTR;     FIRST BUFFER OF BLOCK ADDRESS
        NCBLCNT         BO8BITS;      BUFFERS ALLOCATED COUNTER
        NCSVWL          BO8BITS);     SAVED WORKLIST VALUE
```

Figure 5-5. Multiplex LCB (MLCB) (2 of 2)

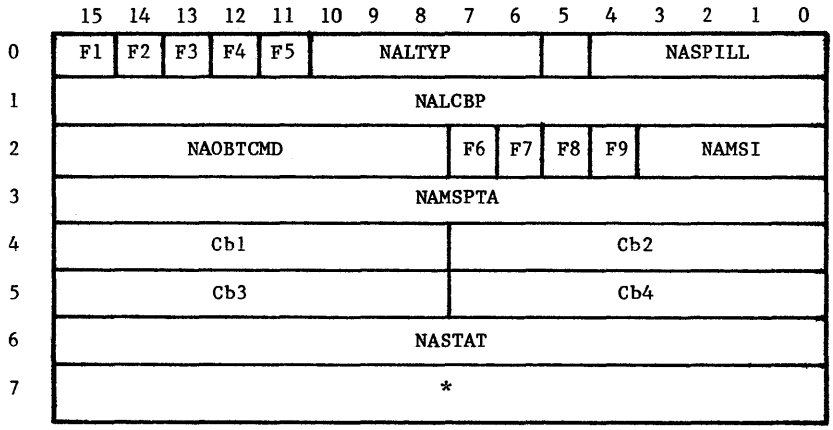
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	NCLCDFCD															
1	F1	*	*	*					NCOBLCD							
2	NCSBP															
3	F2	*	F3	*	F4	F5	F6	F7	F8	*	NCSTAI					
4	NCCNTL - CHARACTER COUNT LIMIT								NCCNT1 - CHARACTER COUNTER 1							
5	NCSPTA - POINTER TO STATE PROGRAMS POINTER TABLE															
6	NCDBP - POINTER TO STATE PROGRAMS TABLE															
7	F9	F 10	F 11	F 12	F 13	F 14	F 15	F 16	F 17	F 18	F 19	NCCRCP				
8	NCSCHR								NOBFCD							
9	NCCRCS															
10	*		NCCNT2													
11	*		NCBLKL													
12	NCCXLTA - POINTER TO CODE TRANSLATE TABLE															
13	NCSCBA - POINTER TO FIRST BUFFER IN BLOCK															
14	NCBLCNT - NUMBER OF BUFFERS ALLOCATED															
15	*															
16	*								F 20							
17	*								F 21							
18	NCFSBA - FIRST STORAGE BUFFER ADDRESS															
19	RESERVED FOR TIP USAGE															

* reserved for CDC

Figure 5-6. Text Processing Control Block (TPCB) (1 of 2)

Flags	Field		Type	Description
	NCLCDFCD	:	INTEGER	FCD AND LCD OF SOURCE BUFFER
F1	NCSUPCHAIN	:	BOOLEAN;	SUPPRESS BUFFER CHAINING
	NCCBLCD	:	BLBBITS;	LCD OF OUTPUT BUFFER
	NCSBP	:	BOBUFPTR;	SOURCE BUFFER POINTER.
F2	NCCCRB	:	BOOLEAN;	DEST. CHAR IN RT. BYTE
F3	NCRIGHTC	:	BOOLEAN;	LEFT/RIGHT SOURCE FLAG
F4	NCNOXL	:	BOOLEAN;	CODE TRANSLATE ACTIVE
F5	NCRPRT	:	BOOLEAN;	STRIP PARITY BIT FLAG
F6	NCSCF	:	BOOLEAN;	TEXT PROCESSOR SUPPRESS CHAIN FLAG
F7	NCLASTCH	:	BOOLEAN;	TEXT PROCESSOR SOURCE LCD REACHED
F8	NCEOSR	:	BOOLEAN;	TEXT PROCESSOR END OF SOURCE REACHED
	NCSTAI	:	BO6BITS;	STATE PROGRAM INDEX.
	NCCNTL	:	BO8BITS;	CHARACTER COUNT LIMIT
	NCCNT1	:	BO8BITS;	CHARACTER COUNTER ONE
	NCSPTA	:	INTEGER;	STATE POINTER TABLE ADRS.
	NCDBP	:	BOBUFPTR;	DESTINATION BUFFER PNTR.
F9	NCUOP1	:	BOOLEAN;	USER OPTIONAL FLAG 1
F10	NCUOP2	:	BOOLEAN;	USER OPTIONAL FLAG 2
F11	NCUOP3	:	BOOLEAN;	USER OPTIONAL FLAG 3
F12	NCUOP4	:	BOOLEAN;	USER OPTIONAL FLAG 4
F13	NCUOP5	:	BOOLEAN;	USER OPTIONAL FLAG 5
F14	NCUOP6	:	BOOLEAN;	USER OPTIONAL FLAG 6
F15	NCUOP7	:	BOOLEAN;	USER OPTIONAL FLAG 7
F16	NCUOP8	:	BOOLEAN;	USER OPTIONAL FLAG 8
F17	NCETX	:	BOOLEAN;	USER OPTIONAL FLAG 9
F18	NCMRTO	:	BOOLEAN;	MODEM RESPONSE TIMEOUT
F19	NCCARR	:	BOOLEAN;	LINE CARRIER TYPE (1=CONTROLLED)
	NCCRCP	:	BO5BITS;	CRC POLYNOMIAL
	NCSCHR	:	CHAR;	SPECIAL CHARACTER
	NCBFCD	:	BO8BITS;	BUFFER FCD.
	NCCRCS	:	INTEGER;	CRC ACCUMULATION
	NCCNT2	:	BO12BITS;	CHARACTER COUNTER TWO
	NCBLKL	:	BO12BITS;	CHARACTER COUNTER TWO LIMIT
	NCCXLTA	:	INTEGER;	CODE TRANSLATE TABLE ADDRESS
	NCSCBL	:	BOBUFPTR;	FIRST BUFFER OF BLOCK ADDRESS
	NCSLCNT	:	BO8BITS;	BUFFERS ALLOCATED COUNTER
F20	--		CHAR	CURRENT SOURCE CHARACTER
F21	--		CHAR	RIGHT SOURCE CHARACTER

Figure 5-6. Text Processing Control Block (TPCB) (2 of 2)



F1 NAION
F2 NAOON
F3 NAISON
F4 NALCBUP
F5 NAISR
F6 NANDCD
F7 NAMTD
F8 NAWAIT
F9 NAOVFE

Cb1 CLA command byte 1
Cb2 CLA command byte 2
Cb3 CLA command byte 3
Cb4 CLA command byte 4

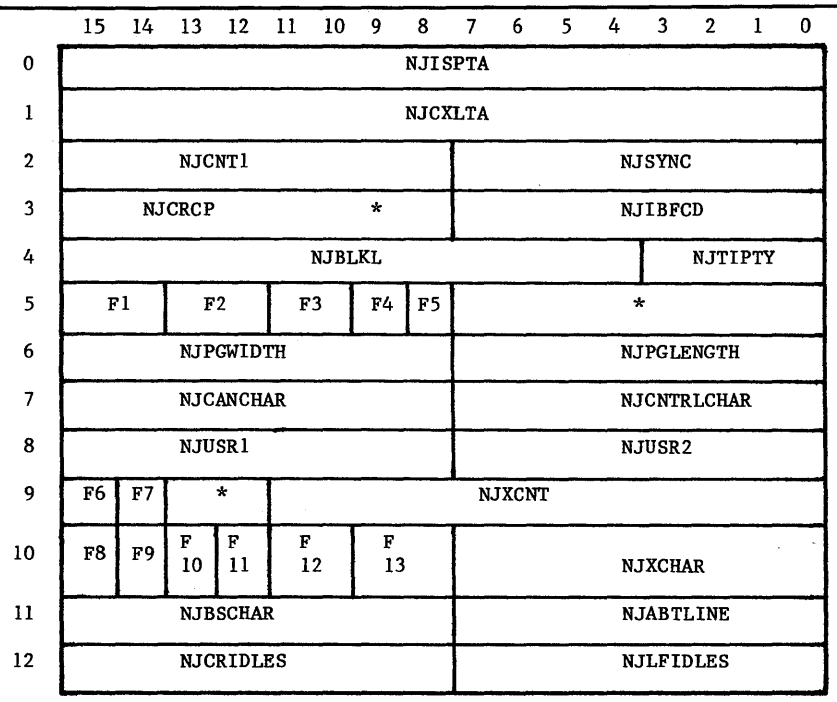
```
  Name            Type              Description

NAION        :  BOOLEAN;      INPUT ON FLAG
NAOON        :  BOOLEAN;      OUTPUT ON FLAG
NAISON       :  BOOLEAN;      INPUT SUPERVISION
NALCBUP      :  BOOLEAN;      LCB ASSIGNED FLAG
NAISR        :  BOOLEAN;      CLA STATUS PENDING
NALTYP       :  B05BITS;      LINE TYPE
NAHARDER     :  BOOLEAN;      HARD ERROR IN PROGRESS
NASPILL      :  B05BITS;      CLA STATUS COUNT
NALCBP       :  NOLCBP;       POINTER TO MUX LCB
NAOBTCMD     :  B08BITS;      CLA TURNAROUND COMMAND
NANDCD       :  BOOLEAN;      DCD DROPPED
NAMTO        :  BOOLEAN;      MODEM TIMEOUT IN PROGRESS
NAWAIT       :  BOOLEAN;      FLAG FOR FIRST OVFL. TIMEOUT
NAOVFE       :  BOOLEAN;      FIRST STATUS OVFL. WL RCV
NAMSI        :  B04BITS;      MODEM STATE INDEX
NAMSPTA      :  INTEGER;      MODEM STATE POINTER TABLE
NAFCCST      :  NFCCSE;       CLA COMMAND WORDS
NASTAT       :  INTEGER;      CLA STATUS
```

Figure 5-7. Port Table

Word	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	F1	F2	*	F3 BJIVTSIZE				F4		F5		BJLISTIX				
1	BJDFTC															
2	BJPTIMRTN															
3	BJETIMETN															
4	BJJFTD															
5	BJFDT															
6	BJJAT															
7	BJAT															
8	BJTPMUX2															
9	BJTEMUX2															
10	BJTCBPINIT															
11	BJTCBEINIT															
12	BJTXTPAGE															
13	BJTXTENT															

Flag	Field		Type	Description
F1	BJOBT	:	BOOLEAN;	GENERATE OUTPUT BUFFER TRANSMITTED FLAG
F2	BJBZL	:	BOOLEAN;	RESET TIMER FLAG ON OBT
	BJIVTSIZE	:	B04BITS;	NO. WORDS IN TCB/TCT IVT
F4	BJTCBSIZE	:	BOBUFSIZES;	TCB BUFFER SIZE
F5	BJQTYPE	:	BOOPTYPES;	TYPE OF QUEUE FOR DATA
	BJLISTIX	:	BOWKLSTS;	WORKLIST INDEX
	BJDFTC	:	NOTCLASS;	DEFAULT TERM CLASS FOR PTENABL
	BJPTIMRTN	:	INTEGER:	TIP TIMAL ROUTINE - PAGE ADDR
	BJETIMRTN	:	INTEGER;	TIP TIMAL ROUTINE - ENTRY ADDR
	BJJFDT	:	DDFDTPTR;	TCB FIELD DESCRIPTOR TABLE ADDR
	BJFDT	:	DDFDTPTR;	LCB FIELD DESCRIPTOR TABLE ADDR
	BJJAT	:	DFATPTR;	TCB ACTION TABLE ADDRESS
	BJAT	:	DFATPTR;	LCB ACTION TABLE ADDRESS
	BJTPMUX2	:	INTEGER;	TIP LEVEL TWO PAGE ADDRESS
	BJTEMUX2	:	INTEGER;	TIP LEVEL TWO ENTRY ADDRESS
	BJTCBPINIT	:	INTEGER;	TCB INIT ROUTINE PAGE
	BJTCBEINIT	:	INTEGER;	TCB INIT ROUTINE ENTRY
	BJTXTPAGE	:	INTEGER;	TEXT PROCESSING PAGE
	BJTXTENT	:	INTEGER;	TEXT PROCESSING ENTRY

Figure 5-8. TIP Type Table



Flag	Field		Type	Description
	NJISPTA	:	INTEGER;	INPUT STATES PTR TBL ADDRESS
	NJCXLTA	:	INTEGER;	CODE TRANSLATE TABLE ADDR
	NJCNT1	:	B08BITS;	INPUT CHARACTER COUNT 1
	NJSYNC	:	B08BITS;	SYNC CHARACTER
	NJCPCP	:	B05BITS;	CRC POLYNOMIAL INDEX
	NJIBFCD	:	B08BITS;	FCD OF FIRST BUFFER
	NJBLKL	:	B012BITS;	BLOCK SIZE
	NJTIPTY	:	NOTIPTY;	TIP TYPE
F1	NJPARITY	:	B02BITS;	PARITY
F2	NJCHLEN	:	B02BITS;	CHARACTER LENGTH
F3	NJPARTYPE	:	B02BITS;	PARITY TYPE FOR ASYNC TIP
F4	NJPGWAIT	:	BOOLEAN;	PAGE WAIT MODE
F5	NJXPARENT	:	BOOLEAN;	TRANSPARENT INPUT MODE
	NJPGWIDTH	:	B08BITS;	PAGE WIDTH
	NJPGLENGTH	:	B08BITS;	PAGE LENGTH
	NJCANCHAR	:	CHAR;	CANCEL INPUT LINE CHAR
	NJCNTRLCHAR	:	CHAR;	CONTROL CHARACTER
	NJUSR1	:	CHAR;	USER BREAK ONE
	NJUSR2	:	CHAR;	USER BREAK TWO
F6	NJXTO	:	BOOLEAN;	TIMEOUT IS EXPECTED DELIMITER
F7	NJXCHRON,			XPT CHAR IS A DELIMITER
	NJXCNT	:	B012BITS;	TRANSPARENT CHARACTER COUNT
F8	NJCRCALC	:	BOOLEAN;	CR IDLE CNT TO BE CALCULATED
F9	NJLFCALC	:	BOOLEAN;	LF IDLE CNT TO BE CALCULATED
F10	NJECHOPLX	:	BOOLEAN;	ECHOPLEX MODE
F11	NJINDEV	:	BOOLEAN;	INPUT DEVICE (F=KB, T=PT)
F12	NJOUTDEV	:	B02BITS;	OUTPUT DEVICE (PRT,DISPLAY,PT)
F13	NJAPL	:	B02BITS;	APL (0=NO,1=YES,2=SPECIAL)
	NJXCHAR	:	CHAR;	TRANSPARENT DELIMITER CHAR
	NJBSCHAR	:	CHAR;	BACKSPACE CHARACTER
	NJABTLINE	:	CHAR;	ABORT OUTPUT LINE CHARACTER
	NJCRIDLES	:	B08BITS;	COUNT OF IDLES AFTER CR
	NJLFIDLES	:	B08BITS);	COUNT OF IDLES AFTER LF

Figure 5-9. Terminal Characteristics Table

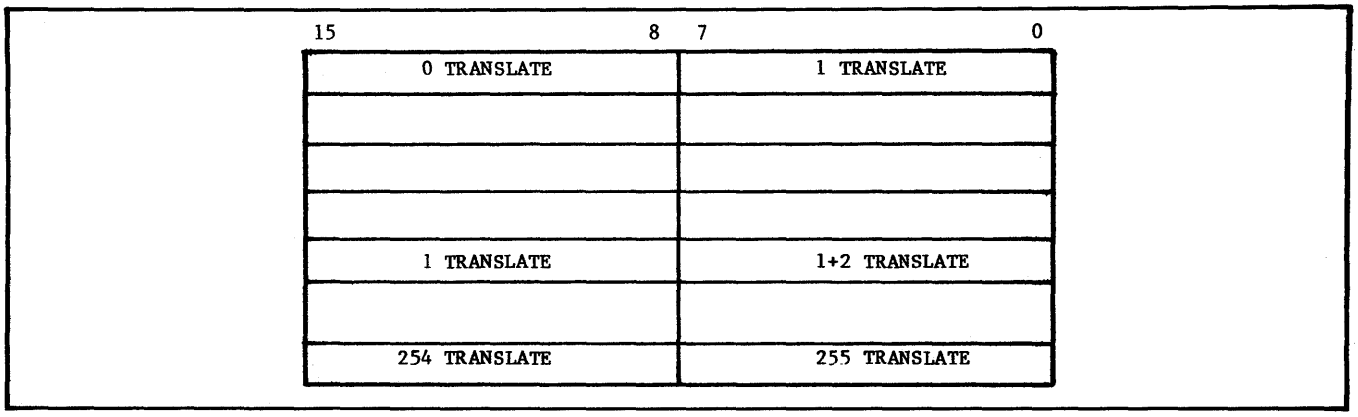


Figure 5-10. Example of a Code Translate Table

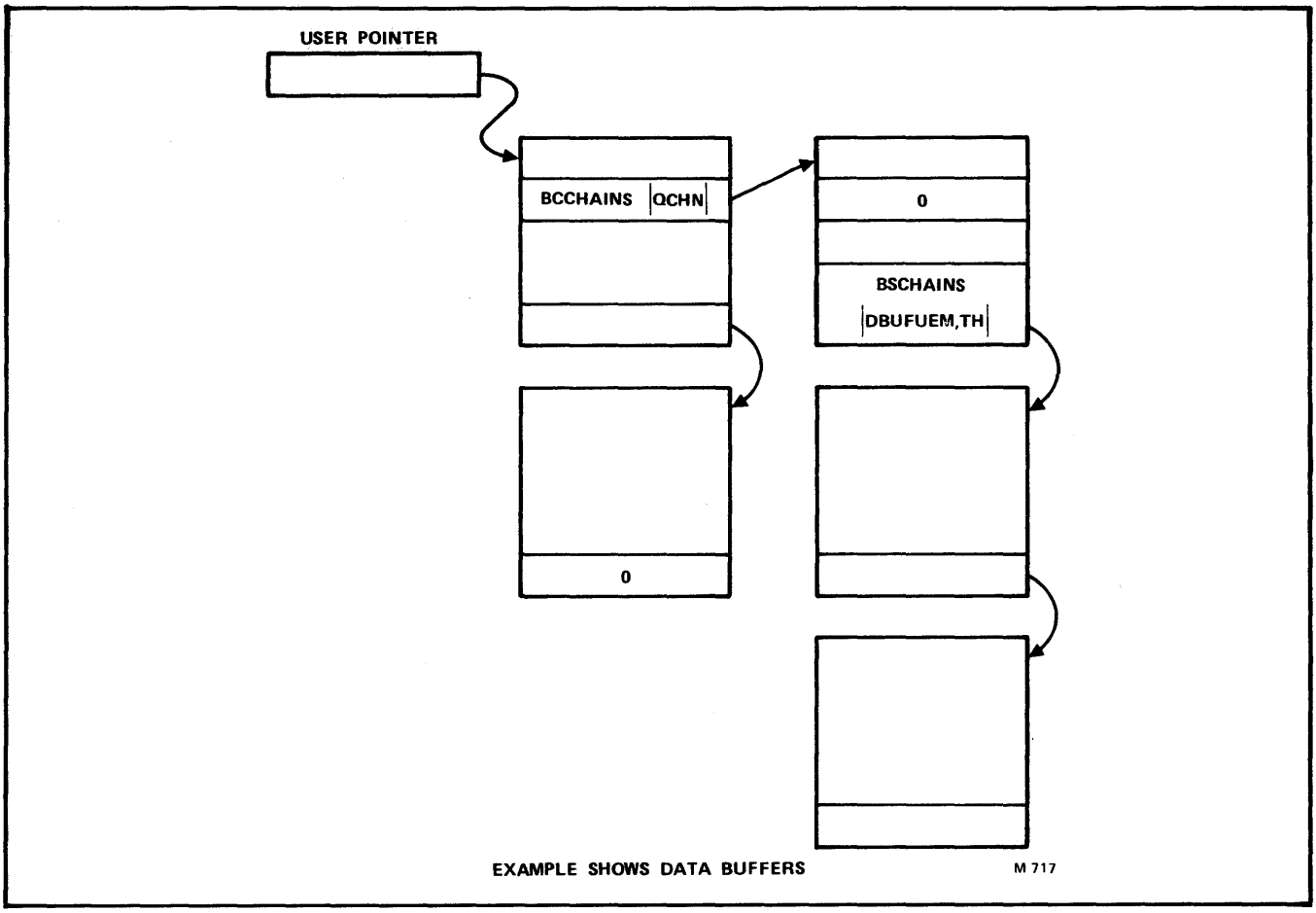
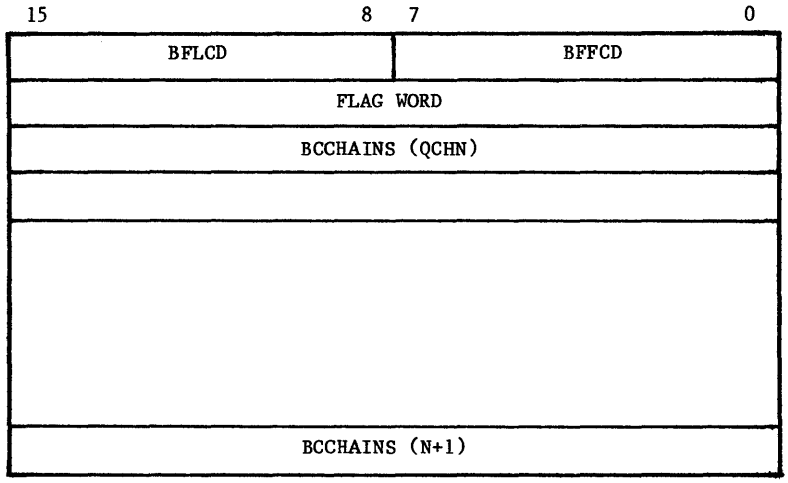
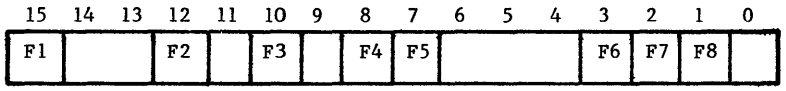


Figure 5-11. Queue of Chained Buffers



where flags are as follows:



where:

F1 = BFEOTFLG - set by input state instruction, used for data sent across coupler, not maintained by TIP

F2 = BFSUPCHAIN - set to inform firmware that buffer is of non-standard length; that is, don't look for chain word

F3 = BFINTBLK - set to inform TIP that block did not originate in host, stops BIP from sending an acknowledge to host

F4 = BFPERM - set to indicate an array being used as a buffer, used to inhibit any attempt to release buffer

F5 = BFLNKQ - set to indicate a valid address is present in the queue word

F6 = BFSP7 - TIP defined, can be set by input state instruction

F7 = BFSP8 - TIP defined, can be set by input state instruction

F8 = BFSP9 - TIP defined, can be set by input state instruction

All other flag bits should not be used by the TIP.

Field Name	Type	Purpose
BFLCD, BFFCD	SUBRANGE	Specify the last character and the first character of data in the buffer. They are of subrange type.
BCCHAINS	BOBUFFPTR	Array for accessing the queue and chain words, index beginning with 1.

Alternate types for buffer structure:

BFDATAC	CHAR	Character array for accessing each 8-bit character in buffer, beginning with zero.
BSTCB	BOTCB	Packed record used for accessing TCBs which are assigned to data buffers.
BIINT	INTEGER	Array for accessing each word in buffer, usually to clear fields initially, index beginning with 1.

Figure 5-12. Data Buffers (1 of 2)

Constants available for buffer manipulation:

QCHN	INTEGER	Selects the word in a data buffer used for queuing.
DATA	INTEGER	Character displacement for first data character in first buffer of a chain.
FBYTE	INTEGER	Character displacement for first data character in subsequent buffers of a chain.
BTPT	INTEGER	Character displacement of the block type byte of a message.
DBC	INTEGER	Character displacement of the data block clarifier byte of a message.
DBUFLENGTH	INTEGER	Length of data buffer in words.
J1LASTCHAR	INTEGER	Last usable character in a data buffer.
BOS32	INTEGER	32
BOS64	INTEGER	64
BETPSIZE	INTEGER	Pointer to control tables for buffers used for TPCB.
TPBUFLENGTH	INTEGER	Size of TPCB buffer in characters.

Figure 5-12. Data Buffers (2 of 2)

```
**************************************************************************************
*                                                                                    *
*                          MULTIPLEX SUBSYSTEM COMMANDS                              *
*                THESE ARE COMMANDS ISSUED TO THE MUX COMMAND DRIVER                 *
*                                                                                    *
**************************************************************************************

               NKINPT        =  6;   INPUT
               NKDOUT        =  7;   DIRECT OUTPUT
               NKINOUT       =  9;   INPUT AFTER OUTPUT
               NKENDIN       = $A;   TERMINATE INPUT
               NKENDOUT      = $B;   TERMINATE OUTPUT
               NKDISL        = $C;   DISABLE LINE
               NKCLRL        = $D;   CLEAR LINE
               NKCONTROL     = $E;   CONTROL

**************************************************************************************
*                                                                                    *
*                                  EVENT WORK CODES                                  *
* THESE WORK CODES APPEAR IN THE WORK CODE FIELD OF THE EVENT PACKET RETURNED TO     *
* THE EVENT WORKLIST QUEUE.  THEY SPECIFY THE NATURE OF THE INFORMATION CONTAINED    *
* IN SAID PACKET.  CODE VALUES OF 1 THROUGH $1E ARE RESERVED FOR MUX USE EXCLUSIVELY *
*                                                                                    *
**************************************************************************************

               MMOBUX        =  2;   OUTPUT BUFFER TRANSMITTED
               MMBUTCH       =  3;   BUFFER THRESHOLD CHANGED
               MMFES         =  8;   FRAIMING ERROR STATUS
               MMCHOUT       =  9;   CHARACTER TIMEOUT
               NMINEND       = $C;   INPUT TERMINATED
               NMOTEND       = $D;   OUTPUT TERMINATED
               MMBREAK       = $E;   ASYNC BREAK DETECTED
               MMHARDERR     = $F;   HARD ERROR
```

Figure 5-13. System Constants Used by the TIP (1 of 8)

```
*****************************************************************************
*                                                                           *
*                      LINE SPEED INDEX TABLE (ASYNCHRONOUS)                *
*                                                                           *
*****************************************************************************

              INDEX                       BAUD RATE

              NO800          =  0;   800
              NO110          =  1;   110
              NO134          =  2;   134.5
              NO150          =  3;   150
              NO300          =  4;   300
              NO600          =  5;   600
              NO1200         =  6;   1200
              NO2400         =  7;   2400
              NO4800         =  8;   4800
              NO9600         =  9;   9600
              NODIAG         = 10;   DIAGNOSTICS CLASS


*****************************************************************************
*                                                                           *
*                                CODE SET KEYS                              *
*                                                                           *
*****************************************************************************

              KEY                                   CODE SET

              NOBCD          =  1;   BCD - MODE 4A BCD
              NOASCII        =  2;   ASCII - ASCII FOR ASYNC OR MODE 4A ASCII
              NOMODE4C       =  3;   MODE 4C
              NOTYPEPAPL     =  3;   TYPEWRITER-PAIRED APL-ASCII
              NOBITPAPL      =  4;   BIT-PAIRED APL-ASCII
              NOEBCDAPL      =  5;   EBCD
              NOEAPLAPL      =  6;   EBCD APL
              NOCORR         =  7;   CORRESPONDENCE
              NOCORAPL       =  8;   CORRESPONDENCE APL
              NOXBCD         =  9;   EBCDIC

*****************************************************************************
*                                                                           *
*                                DEVICE TYPE                                *
*                                                                           *
*****************************************************************************

              N1CON          =  0;   CONSOLE
              N1CR           =  1;   CARD READER
              N1LP           =  2;   LINE PRINTER
              N1CP           =  3;   CARD PUNCH
              N1PLOT         =  4;   PLOTTER
              N1INTDEV       =  7;   INTERNAL DEVICE

*****************************************************************************
*                                                                           *
*                             MODEM CONTROL STATES                          *
*                                                                           *
*****************************************************************************

              MSTIDL         =  4;   STATE 4, LINE IDLED
```

Figure 5-13. System Constants Used by the TIP (2 of 8)

```
*****************************************************************************
*                                                                           *
*                            OPS LEVEL WORKCODES                            *
*                THE FOLLOWING WORKCODES ARE USED FOR BASE SYSTEM,          *
*                TIP, SVM, AND APPLICATION FUNCTION.                        *
*                                                                           *
*****************************************************************************

*                          --SYSTEM WORKCODES FOR TIP--

              AOHARDERR     = $0F;  HARD ERROR          - FROM MUX SUBSYSTEM
              AOTIMEOUT     = $10;  LINE TIMER EXPIRED  - FROM LCB SCAN
              AOQUEOUT      = $11;  OUTPUT IN QUEUE     - FROM INTERNAL P.
              AOSMEN        = $12;  ENABLE LINE         - FROM LINE INIT.
              AOSMDA        = $13;  DISABLE LINE        - FROM SVM
              AOSMTCB       = $14;  TCB BUILT           - FROM SVM
              AOSMDLTCB     = $15;  DELETE TCB          - FROM SVM
              AOSMRCTCB     = $16;  RECONFIGURE TCB     - FROM SVM

*                                --MISCL.--

               AOBREAK       = $19;  DOWNLINE BREAK      - FROM SVM
               AOOBUX        = $1A;  OUTPUT BUFFER XMIT  - FROM TIP TO TIP

*                       --WORKCODES TO THE SERVICE MODULE--

               COLINOP       = $20;  LINE OPERATIONAL    - FROM TIP
               COLNINOP      = $21;  LINE INOPERATIVE    - FROM TIP
               COLNDA        = $22;  LINE DISABLED       - FROM TIP
               CODLTCB       = $23;  TCB DELETED         - FROM TIP
               CORCTCB       = $2C;  TERMINAL RECONFIGD  - FROM TIP

*                    --TIP GENERATED WORKCODES FROM INPUT STATES--

               AOWK1         = $21;  TIP WORKCODE  1
               AOWK2         = $22;  TIP WORKCODE  2
               AOWK3         = $23;  TIP WORKCODE  3
               AOWK4         = $24;  TIP WORKCODE  4
               AOWK5         = $25;  TIP WORKCODE  5
               AOWK6         = $26;  TIP WORKCODE  6
               AOWK7         = $27;  TIP WORKCODE  7
               AOWK8         = $28;  TIP WORKCODE  8
               AOWK9         = $29;  TIP WORKCODE  9
               AOWK10        = $2A;  TIP WORKCODE 10
               AOWK11        = $2B;  TIP WORKCODE 11
               AOWK12        = $2C;  TIP WORKCODE 12
               AOWK13        = $2D;  TIP WORKCODE 13
               AOWK14        = $2E;  TIP WORKCODE 14
               AOWK15        = $2F;  TIP WORKCODE 15
               AOWK16        = $30;  TIP WORKCODE 16
               AOWK17        = $31;  TIP WORKCODE 17
               AOWK18        = $32;  TIP WORKCODE 18
               AOWK19        = $33;  TIP WORKCODE 19
               AOWK20        = $34;  TIP WORKCODE 20
               AOWK21        = $35;  TIP WORKCODE 21
               AOWK22        = $36;  TIP WORKCODE 22
               AOWK23        = $37;  TIP WORKCODE 23
               AOWK24        = $38;  TIP WORKCODE 24
               AOWK25        = $39;  TIP WORKCODE 25
               AOWK26        = $3A;  TIP WORKCODE 26
               AOWK27        = $3B;  TIP WORKCODE 27
               AOWK28        = $3C;  TIP WORKCODE 28
               AOWK29        = $3D;  TIP WORKCODE 29
               AOWK30        = $3E;  TIP WORKCODE 30
               AOWK31        = $3F;  TIP WORKCODE 31
```

Figure 5-13. System Constants Used by the TIP (3 of 8)

```
*****************************************************************************
*                                                                           *
*                          BLOCK PROTOCOL CONSTANTS                         *
*                                                                           *
*****************************************************************************

          HTBLK       = $1;   BLOCK TYPE
          HTMSG       = $2;   MESSAGE TYPE
          HTBACK      = $3;   BACK TYPE
          HTCMD       = $4;   COMMAND TYPE
          HTBREAK     = $5;   BREAK TYPE
          HTSTOP      = $6;   STOP TYPE
          HTSTRT      = $7;   START TYPE
          HTRESET     = $8;   RESET TYPE
          HTINIT      = $9;   INITIALIZE TYPE

*****************************************************************************
*                                                                           *
*                          BLOCK WORD DEFINITIONS                           *
*                                                                           *
*****************************************************************************

          QCHN        =  3;   WORD INDEX FOR QUEUE

*****************************************************************************
*                                                                           *
*                              TERMINAL CLASS                               *
*                                                                           *
*****************************************************************************

          NOTMLIA     =  0;   MLIA
          NOM33       =  1;   ASYNC  - M33,M35,M37,M38
          NO713       =  2;          - CDC 713
          NOM1240     =  3;          - MEMOREX 1240
          NO2741      =  4;          - IBM 2741
          NOM40       =  5;          - M40
          NOH2000     =  6;          - HAZELTINE 2000
          NO751       =  7;          - CDC 751-1
          NOT4014     =  8;          - TEKTRONIX 4014
          NOHASP      =  9;   HASP
          NO200UT     = 10;   MODE 4 - 200UT
          NO214       = 11;          - 214
          NO711       = 12;          - 711-10
          NO714       = 13;          - 714
          NO731       = 14;          - 731
          NO734       = 15;          - 734

*****************************************************************************
*                                                                           *
*                                TIP TYPES                                  *
*                                                                           *
*****************************************************************************

          NOHDLC      = 0;    HDLC LIP
          N1ASYNC     = 1;    ASYNC TIP
          N1M4        = 2;    MODE 4 TIP
          N1HASP      = 3;    HASP TIP
          NO780       = 4;    2780/3780
          NOMLIA      = 5;    MLIA ERROR HANDLER
          NOCONSOLE   = 6;    NPU CONSOLE
          NOCOUPLER   = 7;    CYBER COUPLER
          NOLINIT     = 8;    LINE INITIALIZER
          NOOLDIAG    = 9;    ON-LINE DIAGS **LAST TIP TYPE ALWAYS**
```

Figure 5-13. System Constants Used by the TIP (4 of 8)

```
**************************************************************************************
*                                                                                    *
*                                  SUB TIP TYPES                                     *
*                                                                                    *
**************************************************************************************

            NOAS110       = 1;    110 BAUD ASCII
            NOAS150       = 2;    150 BAUD ASCII
            NOAS300       = 3;    300 BAUD ASCII
            NOE2741       = 4;    2741 EBCD
            NOC2741       = 5;    2741 CORRES

            NOM4A         = 1;    MODE4A
            NOM4C         = 2;    MODE4C

**************************************************************************************
*                                                                                    *
*                            INPUT STOPPED REASON CODES                              *
*                                                                                    *
**************************************************************************************

            B9SIRESPONSE  = 0;    STOP INPUT RESPONSE
            B9NOTREADY    = 1;    INPUT DEVICE NOT READY
            B9CSLIP       = 2;    CARD SLIP ERROR
            B9EOI         = 3;    EOI IN INPUT DATA
            B9INTACT      = 4;    INTERACTIVE INTERRUPT

**************************************************************************************
*                                                                                    *
*                                 COMMAND PFC/SFC                                    *
*                                                                                    *
**************************************************************************************

            D8CTRL        = $01   CONTROL PFC
            D9START       = $05   START INPUT SFC
            D9STOP        = $06;  STOP INPUT SFC
            D9STPD        = $07;  INPUT STOPPED SFC
            D9DEF         = $04;  DEFINE TERMINAL CHARACTERISTICS


**************************************************************************************
*                                                                                    *
*                            UPLINE BREAK REASON CODES                               *
*                                                                                    *
**************************************************************************************

            B9USR1        = 1;    USER BREAK 1
            B9USR2        = 2;    USER BREAK 2
            B9ODEVNOTRDY  = 3;    OUTPUT DEVICE NOT READY
            B9FORMAT      = 4;    BAD BLOCK FORMAT

**************************************************************************************
*                                                                                    *
*                             UPLINE STOP REASON CODES                               *
*                                                                                    *
**************************************************************************************

            B9TBUSY       = 1;    TERMINAL BUSY
            B9TFAILURE    = 2;    TERMINAL FAILURE
            B9BINTERRUPT  = 3;    BATCH INTERRUPT

**************************************************************************************
*                                                                                    *
*                            LINE STATUS RESPONSE CODES                              *
*                                                                                    *
**************************************************************************************

            C6LOPER       = 0;    LINE OPERATIONAL
            C6LINOP       = 4;    LINE INOPERATIVE
            C6LNORING     = 5;    NO RING INDICATOR
```

Figure 5-13. System Constants Used by the TIP (5 of 8)

```
*****************************************************************************
*                                                                           *
*                              IVT BUILD PARAMETERS                         *
*                                                                           *
*****************************************************************************

            B7AIMAX       = 20;   MAX AUTO INPUT PARAMETERS
            B7PWDEF       = 140;  DEFAULT INPUT BLOCK SIZE

*****************************************************************************
*                                                                           *
*                              IVT PARITY OPTIONS                           *
*                                                                           *
*****************************************************************************

            B7ZERO        = 0;    ZERO PARITY
            B7ODD         = 1;    ODD PARITY
            B7EVEN        = 2;    EVEN PARITY
            B7NONE        = 3;    NO PARITY


*****************************************************************************
*                                                                           *
*                              IVT APL OPTIONS                              *
*                                                                           *
*****************************************************************************

            B8APLNO       = 0;    APL MODE = NO
            B8APLYES      = 1;    APL MODE = YES
            B8APLSPEC     = 2;    APL MODE = SPECIAL

*****************************************************************************
*                                                                           *
*                          IVT OUTPUT DEVICE OPTIONS                        *
*                                                                           *
*****************************************************************************

            B8PRNTR       = 0;    PRINTER
            B8DISPLAY     = 1;    DISPLAY
            B8PTAPE       = 2;    PAPER TAPE

*****************************************************************************
*                                                                           *
*                          IVT MISCELLANEOUS CONSTANTS                      *
*                                                                           *
*****************************************************************************

            B8NOPAGING    = 0;    PAGE WIDTH = 0 (NO PAGING)

            B7ODDPAR      = 0;    BSPARITY = ODD PARITY
            B7NOPAR       = 1;    BSPARITY = NO PARITY
            B7EVPAR       = 2;    BSPARITY = EVEN PARITY

*****************************************************************************
*                                                                           *
*                          IVT CHARACTER DEFINITIONS                        *
*            THE FOLLOWING CHARACTERS ARE DEFINED FOR THE INTERACTIVE       *
*            VIRTUAL TERMINAL                                               *
*                                                                           *
*****************************************************************************

*                              --FORMAT EFFECTORS--

            I9FSS         = $20;  FE - SINGLE SPACE
            I9FDS         = $30;  FE - DOUBLE SPACE
            I9FTS         = $2D;  FE - TRIPLE SPACE
            I9FNS         = $2B;  FE - START OF CURRENT LINE
            I9FRS         = $2A;  FE - TOP OF FORM
            I9FPE         = $31;  FE - HOME CURSOR AND CLEAR
            I9FNA         = $20;  FE - NO ACTION
            I9PSS         = $2E;  FE - POST PRINT SINGLE SPACE
            I9PNS         = $2F;  FE - POST PRINT START OF CURRENT LINE
```

Figure 5-13. System Constants Used by the TIP (6 of 8)

```
*                                --CONTROL CODES--

               I9LF          = $0A;  LINE FEED
               I9CR          = $0D;  CARRIAGE RETURN
               I9US          = $1F;  LOGICAL LINE SEPARATOR

*******************************************************************************
*                                                                             *
*                          BVT CHARACTER DEFINITIONS                          *
*                THE FOLLOWING CHARACTERS ARE DEFINED FOR THE                 *
*                BATCH VIRTUAL TERMINAL                                       *
*                                                                             *
*******************************************************************************

               B9BVT         = $FF;  BVT ESCAPE CODE

*                                  --MODE CHANGE--

               B9MCNO        = $00;  MODE CHANGE - NONE
               B9MC29        = $01;  MODE CHANGE - 029
               B9MC26        = $02;  MODE CHANGE - 026
               B9MCOT        = $03;  MODE CHANGE - OTHER

*                               --INFORMATION SEPARATORS--

               B9EOM         = $0A;  END OF MEDIA
               B9EOR         = $0B;  END OF RECORD
               B9EOIN        = $0C;  END OF INFORMATION

*                                 --COMPRESSED BLANKS--

               B9CB02        = $12;  2 COMPRESSED BLANKS
               B9CB31        = $2F;  31 COMPRESSED BLANKS

*                                 --COMPRESSED ZEROES--

               B9CZ02        = $32;  2 COMPRESSED ZEROES
               B9CZ15        = $3F;  15 COMPRESSED ZEROES

*                                 --REPLICATION COUNT--

               B9RC02        = $42;  2 REPEATED CHARACTERS
               B9RC79        = $8F;  79 REPEATED CHARACTERS

*                                  --STRING LENGTH--

               B9SLEN        = $90;  INDETERMINATE LENGTH
               B9SL01        = $91;  STRING OF LENGTH 1
               B9SL63        = $CF;  STRING OF LENGTH 63



*                                  --FORMS CONTROL--

                                     PREPRINT          POSTPRINT

               B9FCS1        = $E0;  SPACE 1           NO SPACE
               B9FCS2        = $E1;  SPACE 2           NO SPACE
               B9FCS3        = $E2;  SPACE 3           NO SPACE
               B9FCSS        = $E3;  SUPPRESS SPACE    NO SPACE
               B9FCPE        = $E4;  PAGE EJECT        NO SPACE
               B9FCBP        = $E5;  BOTTOM OF PAGE    NO SPACE
               B9FCC6        = $E6;  CHANNEL 6         NO SPACE
               B9FCC5        = $E7;  CHANNEL 5         NO SPACE
               B9FCC4        = $E8;  CHANNEL 4         NO SPACE
               B9FCC3        = $E9;  CHANNEL 3         NO SPACE
               B9FCC2        = $EA;  CHANNEL 2         NO SPACE
               B9FC11        = $EB;  CHANNEL 11        NO SPACE
               B9FCC7        = $EC;  CHANNEL 7         NO SPACE
               B9FCC8        = $ED;  CHANNEL 8         NO SPACE
               B9FCC9        = $EE;  CHANNEL 9         NO SPACE
```

Figure 5-13. System Constants Used by the TIP (7 of 8)

```
B9FC10          = $EF;  CHANNEL 10          NO SPACE
B9PPPE          = $F0;  NO SPACE            PAGE EJECT
B9PPBP          = $F1;  NO SPACE            BOTTOM OF PAGE
B9PPC6          = $F2;  NO SPACE            CHANNEL 6
B9PPC5          = $F3;  NO SPACE            CHANNEL 5
B9PPC4          = $F4;  NO SPACE            CHANNEL 4
B9PPC3          = $F5;  NO SPACE            CHANNEL 3
B9PPC2          = $F6;  NO SPACE            CHANNEL 2
B9PP11          = $F7;  NO SPACE            CHANNEL 11
B9PPC7          = $F8;  NO SPACE            CHANNEL 7
B9PPC8          = $F9;  NO SPACE            CHANNEL 8
B9PPC9          = $FA;  NO SPACE            CHANNEL 9
B9PP10          = $FB;  NO SPACE            CHANNEL 10
B9FCR1          = $FC;  RESERVED
B9FCR2          = $FD;  RESERVED
B9FCR3          = $FE;  RESERVED

**************************************************************************************
*                                                                                    *
*                                  DATA BLOCK FORMAT                                 *
*                                                                                    *
**************************************************************************************

DN              =  8;   DESTINATION NODE
SN              =  9;   SOURCE NODE
CN              = 10;   CONNECTION NUMBER
BTPT            = 11;   BLOCK TYPE/BSN/PRIORITY
P1              = 12;   PARAMETER 1
P2              = 13;   PARAMETER 2
P3              = 14;   PARAMETER 3
P4              = 15;   PARAMETER 4
P5              = 16;   PARAMETER 5
P6              = 17;   PARAMETER 6
P7              = 18;   PARAMETER 7
P8              = 19;   PARAMETER 8
P9              = 20;   PARAMETER 9
P10             = 21;   PARAMETER 10
P11             = 22;   PARAMETER 11
P12             = 23;   PARAMETER 12
P13             = 24;   PARAMETER 13
P14             = 25;   PARAMETER 14
P15             = 26;   PARAMETER 15
P16             = 27;   PARAMETER 16
P18             = 29;   PARAMETER 18
P20             = 31;   PARAMETER 20
P21             = 32;   PARAMETER 21
P22             = 33;   PARAMETER 22
P24             = 35;   PARAMETER 24
FBYTE           = DN;     FCD FOR FIRST BYTE OF DATA

**************************************************************************************
*                                                                                    *
*                         BLOCK PROTOCOL MESSAGE PARAMETERS                          *
*                                                                                    *
**************************************************************************************

BLOCK           = DN;   FCD OF 1ST BYTE OF BLOCK HEADER
DBC             = P1;   DATA BLOCK CLARIFIER
DATA            = P1;      FCD OF 1ST BYTE OF DATA

**************************************************************************************
*                                                                                    *
*                                  SM PARAMETER DEFNSE                               *
*                                                                                    *
**************************************************************************************

PFC             = P1;   PRIMARY FUNCTION CODE
SFC             = P2;   SECONDARY FUNCTION CODE
```

Figure 5-13. System Constants Used by the TIP (8 of 8)

```
FIELD LENGTHS:

      B01BIT        = 0..1;
      B02BITS       = 0..3;
      B03BITS       = 0..7;
      B04BITS       = 0..15;
      B05BITS       = 0..31;
      B06BITS       = 0..63;
      B07BITS       = 0..127;
      B08BITS       = 0..255;
      B09BITS       = 0..511;
      B010BITS      = 0..1023;
      B011BITS      = 0..2047;
      B012BITS      = 0..4095;
      B013BITS      = 0..8191;
      B014BITS      = 0..16383;
      B015BITS      = 0..32767;

BIT ELEMENTS:

      ELEMENTS = (BIT0,BIT1,BIT2,BIT3,BIT4,BIT5,BIT6,BIT7,
                  BIT8,BIT9,BIT10,BIT11,BIT12,BIT13,BIT14,BIT15);

GENERAL MASK WORD:

      SETWORD = SET OF ELEMENTS;

POINTERS:

      BOBUFPTR = BOBUFFER;  BUFFER POINTER
      BZLCBP   = BZLCB;     LCB POINTER



*****************************************************************************
*                                                                           *
*                    TERMINAL TYPE (TT)/DEVICE TYPE (DT)                    *
*                                                                           *
*****************************************************************************

      NODEVTYPE     = 0..7;               DEVICE CODE
      NOTCLASS      = 0..19;              TERMINAL CLASS
      NPDT          = PACKED RECORD       DEVICE TYPE (DT)
                      CASE X: INTEGER OF
                      1: (NPSPR1    : B08BITS;     SPARE BYTE
                          NPDEV     : NODEVTYPE;   DEVICE TYPE
                          NPTCLASS  : NOTCLASS);   TERMINAL CLASS
                      2: (NPSPR2,                  SPARE BYTE
                          NPCHAR    : CHAR);       CHARACTER OVERLAY
                      3: (NPINT     : INTEGER);    INTEGER OVERLAY
                      END;

      NOTIPTYPE     = 0..9;               TIP TYPE
      NOSUBTIP      = 0..7;               TIP SUB TYPE

      NPTT          = PACKED RECORD
                      CASE X: INTEGER OF
                      1: (NPSPR3    : B08BITS;     SPARE BYTE
                          NPAUTO    : BOOLEAN;     AUTO RECOGNITION
                          NPTIPTYPE: NOTIPTYPE;    TIP TYPE FIELD
                          NPSUBTIP : NOSUBTIP);    TIP SUB TYPE FIELD
                      2: (NPSPR4,                  SPARE BYTE
                          NPCHR     : CHAR);       CHARACTER OVERLAY
                      3: (NPSPR5    : B08BITS;     SPARE BYTE
                          NPLS      : B04BITS;     LINE SPEED
                          NPCD      : B04BITS);    CODE SET
                      END;
```

Figure 5-14. System Types Used by the TIP (1 of 3)

```
*******************************************************************************
*                                                                             *
*                              BLOCK TYPE BYTE                                *
*                                                                             *
*******************************************************************************

          BLKTYPE       = PACKED RECORD
                          CASE DMY : INTEGER OF
                          1: (
                             BTSPR1 : B08BITS;
                             BTPRID : BOOLEAN;      PRIORITY DESIGNATOR
                             BTBSN  : B03BITS;      BLOCK SERIAL NUMBER
                             BTYPE  : B04BITS;      BLOCK TYPE
                          2: (
                             BTSPR2,
                             BTCHR  : CHAR);        CHARACTER OVERLAY
                          END;

          BOLINO        = PACKED RECORD
                          CASE BOLINTAG : INTEGER OF
                          1: (BOLINO:INTEGER);      FULL WORD
                          2: (BDPORT    ,           PORT NUMBER
                              BDSUBPORT:B08BITS);   SUB PORT NUMBER
                          END;
*******************************************************************************
*                                                                             *
*                          DATA BLOCK CLARIFIER (DBC)                         *
*                                                                             *
*******************************************************************************

     DBDBC = PACKED RECORD CASE DBTAG : INTEGER OF
          1: (DBDM1   : CHAR;                   DUMMY
              DBCHAR  : CHAR);                  CHARACTER OVERLAY
          2: (DBDLFILL: B08BITS;                DOWNLINE DBC
              DBDLS1,                           SPARE
              DBDLS2,                           SPARE
              DBDLS3,                           SPARE
              DBDLS4,                           SPARE
              DBDLFE,                           FORMAT EFFECTORS
              DBDLXPT,                          TRANSPARENT
              DBDLS5,                           SPARE
              DBDLAUTO : BOOLEAN);              AUTO INPUT
          3: (DBULFILL: B08BITS;                UPLINE DBC
              DBULS1,                           SPARE
              DBULS2,                           SPARE
              DBULS3,                           SPARE
              DBULS4,                           SPARE
              DBULS5,                           SPARE
              DBULXPT,                          TRANSPARENT
              DBULCAN,                          CANCEL
              DBULPERR: BOOLEAN);               PARITY ERR

          4: (DBSP1 : B010BITS;                  SPARE
              DBCCF : B02BITS;                   CODE CONVERSION FIELD
              DBBSF : BOOLEAN;                   BACKSPACE PRESENT FLAG
              DBDBT : B03BITS;                   DATA CLARIFIER
          5: (PDPM1 : B08BITS;                  DUMMY
              PDPRUB: BOOLEAN;                  PRUB BLOCK
              PDBANB: BOOLEAN;                  BANNER BLOCK
              PDSP1 : B04BITS;                  NOT USED
              PDEOI : BOOLEAN;                  MESSAGE CONTAINS EOI
              PDXPAR: BOOLEAN);                 TRANSPARENT DATA
          6: (DBF1  : B08BITS;                   FILL
              DBPRUB,                           PRUB BLOCK
              DBBANNER,                         BANNER BLOCK
              DBSP2,                            NOT USED
              DBSP3,                            NOT USED
              DBSP4,                            NOT USED
              DBSP5,                            NOT USED
              DBEOI,                            EOI(1) OR EOR(0) BLOCK
              DBCXPT: BOOLEAN);                 TRANSPARENT DATA
            END;
```

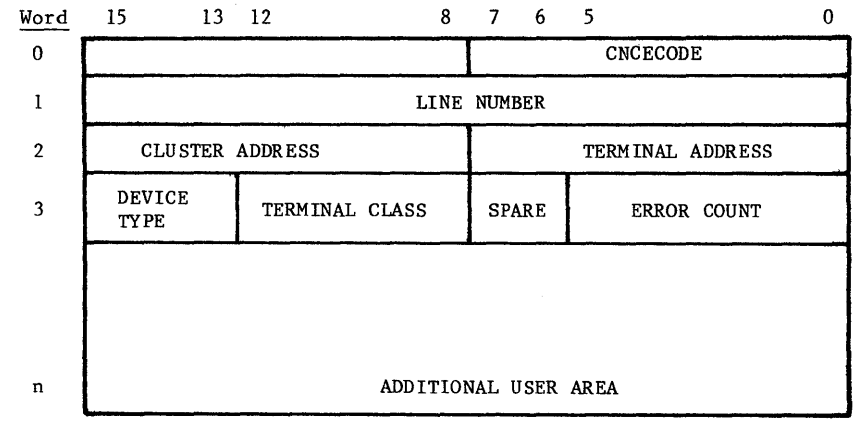
Figure 5-14. System Types Used by the TIP (2 of 3)

```
INPUT REGULATION OPTIONS FOR PTREGL:

      REGLTYPES = (RELOGLNK,              LOGICAL LINK REGULATION
                   RELOCAL,               LOCAL BUFFER LEVELS
                   REABL,                 ALLOWABLE BLOCK LIMIT
                   REACPINP);             ACCEPT INPUT

      REGLSET   = SET OF REGLTYPES;
```

Figure 5-14. System Types Used by the TIP (3 of 3)

```
DATA BUFFER SIZE:

      BEDBSIZE ; BECTPTR;                 DATA BLOCK SIZE

BUFFER SIZE-INDEX PARAMETERS

      BOS8,                               EIGHT WORDS
      BOS16,                              SIXTEEN WORDS
      BOS32,                              THIRTY-TWO WORDS
      BOS64,                              SIXTY-FOUR WORDS

*****************************************************************************************
*                                                                                       *
*                                ONE SECOND CLOCK                                       *
*                                                                                       *
*****************************************************************************************

      CASECNTR      :  INTEGER;

TEXT PROCESSING BUFFER SIZE:

      BETPSIZE : BECTPTR;
      TPBUFLENGTH : INTEGER;              TEXT PROCESSING BUFFER
                                             LENGTH

BUFFER LENGTHS ARRAY:

      BUFLENGTH : ARRAY (BOBUFSIZES) OF INTEGER;

BUFFER CHARACTER LIMITS:

      BUFLCD : ARRAY (BOBUFSIZES) OF INTEGER;

BUFFER SIZE MASKS:

      BUFMASKS : ARRAY (BOBUFSIZES) OF SETWORD;
```

Figure 5-15. System Variables Used by the TIP



Field Name	Type	Purpose
CNCECODE	INTEGER	specification of the type of error event to be logged in SEF
LINE NUMBER	BOLINO	line number
CLUSTER ADDRESS	B08BITS	cluster address
TERMINAL ADDRESS	B08BITS	terminal address
DEVICE TYPE	NODEVTYPE	device type
TERMINAL CLASS	NOTCLASS	terminal class
SPARE	B02BITS	spare
ERROR CODE	B06BITS	number of times this error event occurred

Figure 5-16. System Engineering File Work Area Entry

Block protocol is used to communicate commands and information between the NPU and the host. Blocks are composed of consecutive bytes. The shortest block consists of only a header (four bytes); the longest block consists of 2047 bytes, including the four-byte header.

Block protocol assumes that the logical connection between processes in the host and the NPU is error free (a supportive, lower level protocol provides delivery assurances between the processes). However, the logical connection can be abnormally broken, either process can fail, or the processes can become temporarily congested, leading to regulation of information transfer.

Failure of a process is usually reported by means of a service message. Temporary bottlenecks at a destination process are usually a result of inability to deliver data to an associated terminal or to the host. Block handling provides a standard method for informing the transmitting process of a temporary problem so that any subsequent data transfers on that connection can be held in abeyance until the problem is corrected.

The paths between the two processes are fully symmetrical as shown in figure 6-1. Blocks belong to one of three categories:

- Forward supervision (FS) functions are performed by INIT and RST blocks.
- Reverse supervision (RS) functions are performed by BACK, BRK, STRT, and STP blocks.
- Forward data (FD) functions are performed by BLK, MSG, and CMD blocks.

BLOCK FORMAT

The first two bytes of any block are reserved for a link header (which is used when sending/receiving data from a remote NPU). The next four bytes of any block constitute the block header. Format of the block header is as shown in figure 6-2.

The current release consists of nine principal block types plus an additional assurance control block type used only for NPU to NPU transmissions. Characteristics of each type are summarized in table 6-1.

The first three bytes of the block header provide a standard network address. Byte 4 contains the block

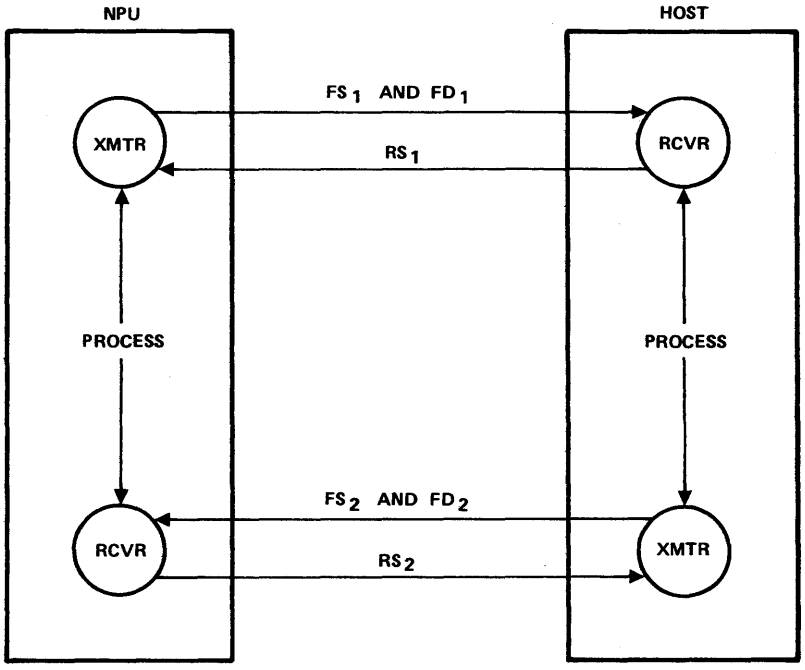


Figure 6-1. Sample Block Paths Between NPU and Host

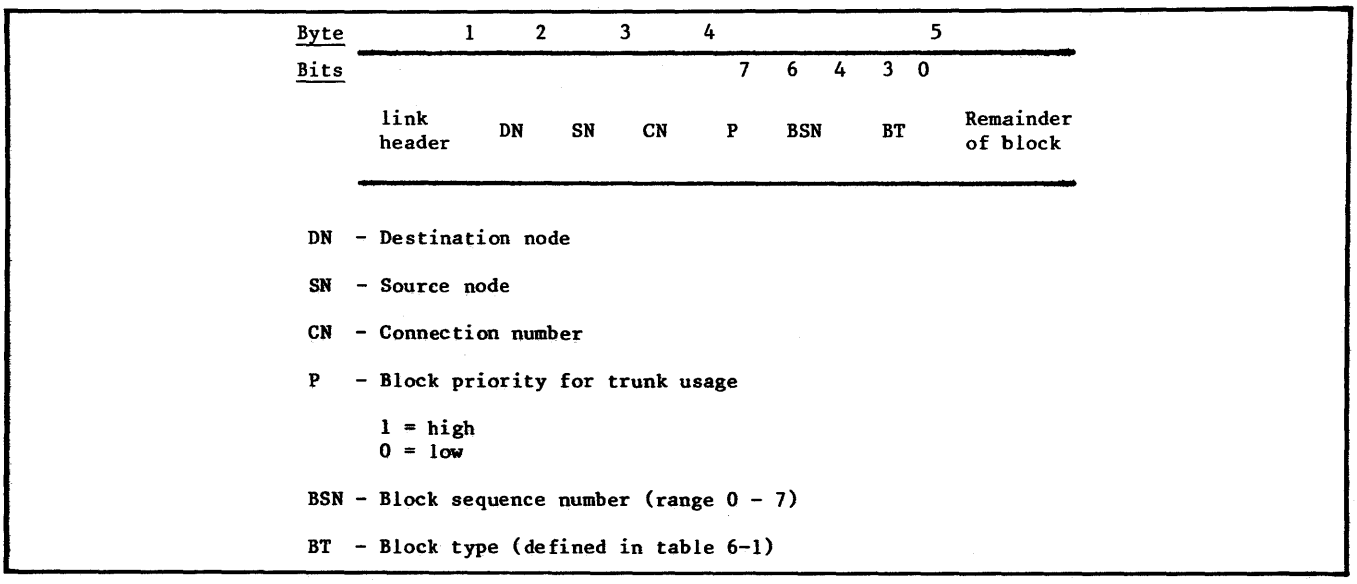


Figure 6-2. Block Header Format

TABLE 6-1. BLOCK TYPES

Mnemonic	Name	Block Type	Traffic Type	General Function
BLK	Block	1	FD	Any data block which is not the EOM block of a multiblock message
MSG	Message	2	FD	Data block which is the EOM block of a multiblock message or the only block of a message
BACK	Block Acknowledgment	3	RS	Block acknowledgment for block transmitted in opposite direction
CMD	Command	4	FD	Command
BRK	Break	5	RS	Indicates a discontinuity in the data stream traveling in the opposite direction
STP	Stop	6	RS	Forward data stream is undeliverable and should be stopped
STRT	Start	7	RS	Forward data stream can be started
RST	Reset	8	FS	Transmitter has cleared logical connection after receiving a BRK or STRT
INIT	Initiate	9	FS	Initiate a logical connection
--	--	10 . . . 14		Reserved

priority (P), block sequence number (BSN), and block type (BT). The content of the remainder of the block, if any, varies with the block type.

The priority of the block is only significant when the block is required to traverse a network trunk. Priority provides for preferential treatment for high-priority blocks when trunk queueing occurs. (Trunk queueing is a part of priority assignment.) All blocks (regardless of type) containing the same address must be assigned the same priority.

The BSN supplied in a downline block of type MSG, BLK, or CMD must be returned in the BSN field of the upline BACK which acknowledges that block. When a BRK or STP is sent, the BSN field must contain the BSN which was contained in the last BACK sent for this connection. The BSN is always zero on other upline and downline blocks.

ADDRESS

The address contains the node IDs for the source and destination of the block plus a connection number.

NODE

Each NPU has a unique node ID; each interface between a host and an NPU has a unique node ID; the host has two unique node IDs. Node ID = 0 is reserved for the Network Supervisor (NS) in the host. One nonzero node (usually node 1) is reserved for the Communications Supervisor (CS). The remaining node IDs (between 2 and 255) are build time parameters. For example, in a single-host, single-NPU system, the host interface (coupler of the local NPU) might be node ID two, and the terminal node (interface to the terminals) might be node ID three; this pair of nodes forms a logical link. Thus, traffic going upline (from a terminal to the host) has a destination node ID of two and a source node ID of three. Traffic going downline from NS to the NPU has a destination node ID of two and a source node ID of zero.

CONNECTION NUMBER

A logical connection is the association between a terminal device on an NPU and an application process in the host, by which traffic is communicated between the terminal (or a device at that terminal) and the applicable process. The connection number is one byte long, and has a range of values between 1 and 255. Every block traveling downline to a terminal device or upline from a terminal device bears the connection number. Unique connection numbers are assigned to all terminal devices on a given logical link.

SERVICE CHANNEL

A block having a connection number of zero is called a service message, and the logical connection over which it is communicated is called the service channel. Unlike logical connections that can be dynamically created and released, the service channel always exists. Service messages include commands, requests for status, error information, statistics information, or replies to one of these three message categories. Commands traveling via the service channel establish logical connections and communicate control, status, and error data. The complete summary of service messages is found in appendix C of the CCP System Programmer's Reference Manual.

BLOCK TYPES

The block types are described in detail below.

BLK - BLOCK OF A MESSAGE

A BLK block is a data block containing a portion, but not the last segment, of a data message. All data blocks contain from 1 to 2043 bytes of data immediately following the four-byte header. The content of the data field is determined arbitrarily by the communicating processes; that is, the TIP and the host application.

MSG - MESSAGE OR LAST BLOCK OF A MESSAGE

A message is a self-contained unit of data communications and is transmitted as data stream. An end-of-message indicator always terminates the message's data stream.

If a message is 2043 bytes or less in length, it can be transmitted within a single MSG block. If a message is longer than 2043 bytes or if, as is usual, the message is segmented by the terminal or by the TIP, all segments but the last are transmitted within BLK blocks. The last segment is transmitted within a MSG block.

BACK - BLOCK ACKNOWLEDGMENT

A BACK block is the acknowledgment of a received block. It is returned to the transmitter by the receiver as BLK, MSG, and CMD blocks are processed to allow the transmitter to adjust the rate of issuing data to the rate of delivery to the receiver. The transmitter should not issue unacknowledged blocks in excess of a network block limit (NBL) for each connection. The BACK block that acknowledges a previously transmitted block allows the transmitter to maintain an outstanding block count to ensure that the NBL is not exceeded. NBL is established by the connection as a part of the configuration process. Note that no data bytes are associated with a BACK block.

CMD - COMMAND BLOCK

A CMD block carries a network command. It allows connected processes to communicate outside of the data stream but synchronous with that stream. The command is received by the destination process in the same ordering sequence to the data stream, or other commands, as existed at the source. If CN is 0, the command is a service message. The data bytes of the message are highly structured. Rather than using BACK blocks as acknowledgment, service messages use other service messages as acknowledgments. See appendix C.

BRK - BREAK IN MESSAGE STREAM

The BRK block indicates a discontinuity (break) in the data stream and travels in the opposite direction. The receiving process responds with an RST to specify the point in the data stream where the action caused by the BRK block occurred. The sender of the BRK block discards all blocks received before the RST block. A further BRK or STP block must not be sent before the RST block is received.

A single data byte, the reason code (RC), follows the BRK block header and specifies the reason for breaking the transmission. The RC byte is defined as follows:

1 = User Break 1 received (typically means queue abort occurred)

2 = User Break 2 received (typically means job abort occurred)

3 = Output device not ready

4 = Illegal or invalidly formatted block received from host

The receiver sends the BRK block when that device is unable to determine when data can again be delivered. The sender must use a higher level protocol than the break condition to determine recovery.

STP - STOP MESSAGE TRAFFIC

The STP (Stop) block indicates that no further blocks should be sent until a STRT block is received.

The STP block is used when a process is unable to deliver data to the final destination but is able to determine when data delivery can resume. The STP block is used to release data buffered in the network; STP blocks are not required for temporary delays. A reason code follows the header. This code is passed to the connected process. The sender of the STP block discards all sequenced blocks received. The RC byte is interpreted as follows:

1 = Terminal busy
2 = Terminal failure
3 = Batch interrupted by interactive input or output

STRT - START MESSAGE TRAFFIC

The STRT (Start) block is used after a STP block to allow resumption of data flow to the destination sending the STRT block. The receiving process responds with a RST. No data bytes are associated with this block.

RST - RESET BLOCK

The RST (reset) block is sent in response to either a BRK or STRT block. It serves to delimit the data stream and indicate the point at which the BRK or STRT block occurred. From the time the BRK or STRT block was sent until the receipt of the RST block, all unacknowledged blocks and all new blocks are discarded. No data bytes are associated with this block.

INIT - INITIALIZE TRAFFIC

The INIT (initiate) block delimits the new data boundaries when a connection is first made. Newly established connections discard blocks from the logical connection until the INIT protocol is completed. The second end of the connection to be set up immediately sends an INIT block. Upon receipt of the INIT block, the first end to be set up responds with an INIT block and starts accepting blocks over the logical connection. Upon receipt of the responding INIT block, the second end of the connection to be set up also starts to accept blocks over the logical connection. No data bytes are associated with this block.

BAD BLOCKS DETECTED BY NPU

When NPU software detects a bad block, the NPU discards the block. If the block is bad due to a bad block type or to a bad sequence number, a BRK block is sent to the host. If the block is a BLK, CMD, or MSG, no BACK block is sent to the host. For any other block type, no action solicited by the block is taken and the block is not acknowledged.

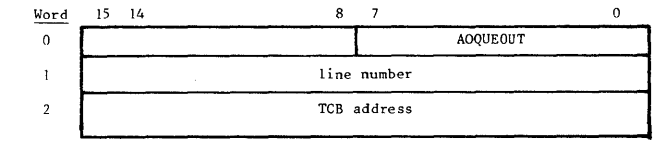
The Block Protocol Interface Package (BIP) controls the transportation of data and control blocks between the TIP and the application program. BIP provides a set of modules which the TIP uses to receive data, to send data, and to regulate the flow of data. Two independent processes for data and commands are provided: one for upline and the other for downline.

DOWNLINE DATA AND COMMANDS

After line and terminal configuration has been completed, the host software and the BIP initialize the connection for both upline and downline data. This initialization discards any residual messages that might have been left in the network from a previous use of the connection (CN). This initialization is invisible to the TIP. The host application programs are required to originate a downline message to begin data exchange. The TIP uses this first downline message to indicate that the connection has been initialized, and to begin I/O tasks.

Whenever an output message arrives from the host, BIP queues the message to the appropriate TCB. Whenever BIP places a message in an empty TCB queue, BIP also generates a worklist entry for the TIP associated with that TCB. (Note that BIP does not generate a worklist entry to the TIP if there are already messages in the TCB queue.) This worklist can be used to initiate output. After the output operation is completed, the TIP must check to find if there is more output queued for delivery. Format of the output queued worklist entry to the TIP is shown below:

Output Queued WLE:



PREOUTPUT POINT OF INTERFACE

The TIP checks for pending output by calling the preoutput point of interface procedure, PBPROPOI. If output is available, PBPROPOI returns the address of the chain of message buffers to the TIP. Otherwise, a nil pointer is passed to the TIP.

The principal characteristics of PBPROPOI are as follows:

```
Procedure name:     PBPROPOI

Input parameter:    (B1TCB) = address of TCB

Return parameter:   (B1BUFF) = address of first
                    buffer in chain (or nil if
                    there is nothing in queue)

Calling sequence:   PBPROPOI

Example of call:

B1TCB : = (tcbpointer); Address of TCB to be
                        checked
PBPROPOI;               Get next output message
IF B1BUFF = NIL         If no messages,...
THEN
  .
  .
  .
ELSE                    ELSE, IF OUTPUT EXISTS...
  .
  .
  .
```

The message which was removed from the output queue must be acknowledged before calling PBPROPOI again. The message can be acknowledged by calling the post-output point of interface (PBPOPOI), PTBACK, PTSTOP, or PTBREAK. Message and terminal conditions determine which routine is called. PBPOPOI is used to acknowledge data messages (BLK or MSG blocks). Commands are acknowledged by calling PTBACK. PTBREAK (a negative acknowledgment) is called if the message can be neither delivered nor actioned. Such conditions could be caused either by syntax errors in the message or a device-not-ready condition at the terminal. The BRK block is used for device-not-ready only if the TIP cannot determine when the device-ready condition occurs. If the TIP can resolve the not ready condition, PTSTOP is called instead. The STP block does not acknowledge a message if the message was unqueued.

POST OUTPUT POINT OF INTERFACE

After successfully transmitting data to a terminal, the TIP calls the post output point of interface program, PBPOPOI. This program performs the following functions:

- It acknowledges to the host application program that the block was successfully transmitted
- It accumulates line and terminal statistics
- It releases the data buffers

CCP must acknowledge all the blocks that an application program sends and it must acknowledge them only once. It is possible for a network block to be fragmented into two or more blocks. Each of these internally generated blocks is identified by a BFINTBLK flag which is set to TRUE in the first buffer of the block. The TIP must pass this flag to the transmission block so PBPOPOI can recognize that it is not to be acknowledged. When PBPOPOI processes one of these internally generated blocks, it accumulates statistics for the block. However, since only one fragment of the block has BFINTBLK set to FALSE, PBPOPOI acknowledges that transmission block only.

Note that PBPOPOI must not be called more than once for any network block without the BFINTBLK flag set, since this would cause multiple acknowledgment of a single network block.

PBPOPOI's principal characteristics are as follows:

```
Procedure name:     PBPOPOI

Input parameter:    (B1TCB) = address of TCB

Return parameter:   (B1BUFF) = address of buffer
                    chain

Calling sequence:   PBPOPOI

Example of call:

B1TCB : = (tcbpointer);      TCB address
B1BUFF : = (bufferpointer); Buffer chain address
PBPOPOI;                     Acknowledge block
```

COMMAND ACKNOWLEDGMENT

After the TIP processes a command, the TIP acknowledges command (CMD) blocks by calling PTBACK. This procedure must be called only once for each CMD block.

PTBACK's principal characteristics are as follows:

```
Procedure name:     PTBACK

Input parameter:    (B1TCB) = address of TCB

Return parameter:   none

Calling sequence:   PTBACK

Example of call:

B1TCB : = (tcbpointer); TCB address
PTBACK;                 Acknowledge the CMD block
```

NEGATIVE ACKNOWLEDGMENT OF BLOCKS AND BREAKS

A negative acknowledgement takes the form of a break (BRK) block, which is generated by the PTBREAK routine. Negative acknowledgement is sent to the host application program whenever the TIP cannot deliver a data (MSG or BLK) block, or is unable to execute a CMD block. PTBREAK is called when the TIP detects a syntax error or a sequence error in either a data or a command block. PTBREAK is also called when the data cannot be delivered for an indeterminate period, as when the receiving terminal is not ready. The host application program must provide the logic to restart the output operation when the terminal is again able to accept data.

BRK blocks have a second use: they can carry user data upline for special processing. For example, when a user break is entered at a terminal, a break block is generated for the application program. In this case, BRK block is inserted into the data stream to carry upline data; it is therefore not synchronous with other upline network blocks. Since the BRK block can never be synchronous with the normal data stream, it can be received and processed ahead of any data or commands that were queued for upline transmission prior to the BRK block, but which the application has not yet received.

If (1) delivery assurance is required, and (2) the output block was fragmented, and (3) the TIP did acknowledge the network block without delivering all of the fragmented blocks, then the TIP must hold the fragmented blocks either until they can be delivered or until the connection is broken. However, if the failure to deliver all of the fragments was caused by a user break received from the terminal, all queued blocks are discarded; there is no delivery assurance for this condition.

NOTE

The BACK block is used only for data flow control. Delivery assurance requires some other mechanism, such as passing CMD blocks between nodes. None of the standard CCP TIPs use delivery assurance.

PTBREAK's principal characteristics are as follows:

```
Procedure name:     PTBREAK

Input parameters:   (B1TCB) = address of TCB
                    (reason code) = integer
                    variable:
                    Type of break condition:
                    B9FORMAT = syntax or sequence
                    error
                    B9ODEVNOTRDY = Output device
                    not ready
                    B9USR1 = user break 1
                    B9USR2 = user break 2

Return parameter:   none

Calling sequence:   PTBREAK (reason code);

Example of call:

B1TCB : = (tcbpointer); TCB address
PTBBREK(B9FORMAT);      Syntax error in message
```

STOP OUTPUT

The TIP can stop the downline data and command flow at any time by calling PTSTOP to generate a STP block. A stop block causes the host application to suspend output until a start condition is sent upline. The STP block acts as a negative acknowledgment if a network block was unqueued but not acknowledged. The host application repeats all unacknowledged messages when the downline data flow is restarted. The TIP must not send a BRK block while a STP block is outstanding (that is, no STRT block has been generated to restart output from the host application program).

PTSTOP's principal characteristics are as follows:

```
Procedure name:     PTSTOP

Input parameters:   (B1TCB) = address of TCB
                    (reason code) = reason code
                    for the stop
                    Type of stop condition:
                    B9TBUSY = terminal busy
                    B9TFAILURE = terminal failed

Return parameter    none

Calling sequence:   PTSTOP (reason code);

Example of call:

B1TCB : = (tcbpointer); TCB address
PTSTOP(B9BUSY);         Stop downline message
                        flow
```

START OUTPUT

To restart an output stream that was previously stopped by the call to PTSTOP, the TIP issues a call to PTSTART. That routine sends a STRT block upline if, and only if, the last block sent upline was a STP block.

PTSTART's principal characteristics are:

```
Procedure name:     PTSTART

Input parameters:   (B1TCB) = address of TCB
                    (reason code) = constant of
                    type INTEGER, which is always
                    set to a zero value

Return parameter:   none

Calling sequence:   PTSTART (reason code); reason
                    code is always zero

Example of call:

BEGIN
    B1TCB : = (tcbpointer);  TCB address
    PTSTART(0);              Start downline
                             message flow
END;
```

UPLINE DATA AND COMMANDS

UPLINE DATA

After terminal configuration is completed, it is possible for the TIP to initiate and to process upline data. However, the network cannot accept upline data until the connection to the host application program is initialized. Synchronous and asynchronous protocols should start processing data in different ways:

- Synchronous TIPs should wait for the first downline data to indicate that the connection has been completed.
- Uncontrolled protocols which are used with asynchronous terminals can lose the first few data characters if the TIP waits for an indication that the connection has been completed. For such protocols, the first output from the application program to the terminal is a prompt, indicating a successful connection.

Before starting any input processing, or releasing any upline data into the network, the TIP must make an input data regulation check to find if input is allowed. The TIP calls PTREGL to determine the current input conditions. If input is allowed, the TIP requests input from the terminal or allows already gathered messages to be released to the network. The TIP calls the post input point of interface, PBPIPOI, to transform the data into network MSG or BLK blocks, and to pass the blocks upline to the network.

UPLINE COMMANDS

Commands can be sent upline either as a response to a downline command, or as a result of upline data. In either case, the TIP calls the Internal Input Point of Interface, PBIIPOI, to send a command.

POST INPUT POINT OF INTERFACE

Whenever the TIP has data that is ready to be sent to the host application program, the TIP calls the post input point of interface routine, PBPIPOI. This POI performs three tasks: it creates a network block, it routes the block to the host, and it gathers line and terminal statistics for the data.

PBPIPOI's principal characteristics are as follows:

```
Procedure name:     PBPIPOI

Input parameters:   (B1BUFF) = address of data
                    buffer chain
                    (B1TCB) = address of TCB
                    (B1BT) = block type:
                         HTMSG = message
                         HTBLK = block

Return parameter:   none

Calling sequence:   PBPIPOI

Example of call:

B1BUFF : = (bufferpointer); Address of input data
B1TCB : = (tcbpointer);     TCB address
B1BT := HTMSG;              Message block
PBPIPOI;                    Send message to
                            application program
```

Prior to calling PBPIPOI, the TIP should call PTREGL to assure that the connection has been initialized and can accept data. Otherwise, data could be lost.

The first character displacement (FCD) of the first data buffer must point to the first data character and not to the first header byte. The first data character is defined to be the data block clarifier (DBC). If the DBC is in the proper relative location in the buffer, the system adds the network header to the first buffer. If the DBC is in any other byte, the system allocates a new buffer for the network header and chains the data buffers to the new header. This inefficient use of data buffers should be avoided. The TIP should normally place the DBC in the byte indexed by the global constant DATA. (The global variable DBC is equated to DATA to allow more descriptive code.) Following the DBC, the first transformed data byte from the terminal in placed at location (DATA + 1). The TIP normally prepares the FCD and DBC as shown:

```
VAR
    (dbc) : DBDBC      Data block clarifier
WITH B1BUFF ↑ DO       With the data buffer
BEGIN                  To be sent
    BFFCD : = DATA;    Indicate the location of
                       the DBC
    BFDATAC [DBC] : =
      (dbc).DBCHAR;    Set the message DBC
END;
```

INTERNAL INPUT POINT OF INTERFACE

To send a command to the host, the TIP constructs the command in a data buffer and calls the internal input point of interface, PBIIPOI, to format the command into a network recognized CMD block, and to send the block upline. The first data byte of the command should begin at an index specified by the global constant DATA (this avoids inefficient use of data buffers).

PBIIPOI'S principal characteristics are as follows:

```
Procedure name:     PBIIPOI

Input parameters:   (B1BUFF) = address of command
                    (B1TCB) = address of TCB
                    (B1BT) = HTCMD - Network
                    command block type

Return parameter:   none

Calling sequence:   PBIIPOI

Example of call:

B1BUFF := (bufferpointer); Address of command
                           data
B1TCB : = (tcbpointer);    TCB address
B1BT := HTCMD;             Command block
PBIIPOI;                   Send command to host
```

MESSAGE SEQUENCING

Each downline BLK, MSG, or CMD block is given a sequence number by the network software. This sequence number is returned to the host in the acknowledgment for the block. BIP maintains the sequence numbers, but the the TIP must assure that interface procedures are called in the proper order. Failure to maintain this order causes the host software to report an NPU failure. This will lead to reloading the NPU and causes the loss of all current messages, as well as a loss of time in reloading the NPU and reconfiguring all its lines.

ERROR PROCESSING

Illegal parameters passed to the BIP routines will result in the eventual failure of the NPU. The BIP does not check input parameters to protect the system against failures resulting from erroneous TIP calls.

SERVICE MODULE INTERFACE 8

The Service Module (SVM) provides the interface for supervisory (service) messages between the TIP and the host.

Downline, a service message from the host originates as a request to change the status of a line or a terminal. The service module forwards the request to the TIP for processing or for TIP concurrence. In most cases the TIP must reply to the service module request. However, the TIP need not reply immediately. Instead, the TIP can save the request and continue processing until an appropriate time is found to send the reply. Such delays are often necessary to complete a protocol cycle so that the terminal is left in a known state rather than in some error or recovery state.

Upline, the TIP can generate a service message when the TIP detects a change in the status of a line or terminal. In this case, the TIP calls the service module to forward the change of status to the host in the form of a supervisory message. The TIP also gathers statistics for the host's engineering file, and calls the service module indirectly to deliver this information to the host. Note that this information is independent of the application programs. Such information is for diagnostic purposes; it is independent of online message processing.

There are three methods of interfacing to the service module:

- Most communication between TIP and service module is initiated by worklist entries.
- The service module makes a direct call to the TIP's TCB initialization routine during terminal configuration.
- The TIP calls other routines which in turn call the service module, thereby providing indirect calls to the SVM.

The following tasks use the TIP/SVM interface:

- Worklist call from SVM to TIP to enable a line. The TIP must reply to the SVM with a line enabled or a line inoperative worklist.
- Worklist call from SVM to TIP to disable a line. The TIP must reply to the SVM with a line disabled worklist.
- Direct call from SVM to TIP to configure user TIP defined fields of the TCB. The TIP must return control to the SVM by completing the return jump.
- Worklist call from SVM to TIP to start terminal operation following configuration. There is no reply to this service message.
- Worklist call from SVM to TIP to change a data connection from the terminal to another application program. The TIP must reply to the SVM with a terminal reconfigured worklist.
- Worklist call from SVM to TIP to delete a terminal. The TIP must reply to the SVM with a terminal deleted worklist.
- Indirect call to SVM via PNSMGEN to report a change of terminal status (such as a terminal failure or recovery).
- Indirect call to SVM via PNCEFILE to report statistical performance data.

ENABLE LINE

A line enable worklist is sent to the TIP when a line is being initialized for use and the modem signals are correct. The TIP initializes the TIP-defined fields, if necessary. The enable line worklist is sent to the TIP for both the initial line configuration and for reinitialization. Reinitialization can result from a line enable or a disconnect line request from the host.

The TIP must reply to the SVM with either a line enabled worklist (workcode = COLINOP), or a line inoperative worklist (workcode = COLNINOP), depending on the TIP's success in enabling the line.

The TIP must do additional processing on autorecognition lines (the LCB's BZAUTO field = TRUE). The TIP communicates with the terminal, and the operator replies by pressing certain keys which send information for the TIP to interpret. By using autorecognition methods the TIP can determine the following:

- Line speed
- SubTIP type
- Code set
- Cluster address(es)
- Terminal address(es)
- Device type(s)

The TIP places the detected information in specified fields of a data buffer. This buffer is passed to the SVM along with the worklist entry for the line enabled reply. If the TIP failed to generate any autorecognition information, the address of the autorecognition buffer will be NIL. SVM passes all autorecognition information to the host.

NOTE

Implementing new autorecognition parameters requires host program changes. New use of existing autorecognition fields may also require host program changes.

If the TIP writer uses a dummy TCB during autorecognition, the TIP itself must both create and maintain this TCB.

The service module assures that only one enable line worklist is sent to the TIP for each line. Terminal related requests are not sent to the TIP until SVM receives the TIP's reply to the enable line message. However, SVM can

send the TIP a disable line worklist to cancel the autorecognition procedures at any time.

Figure 8-1 shows the worklist entry format for the enable line request and for the TIP's worklist replies to this request.

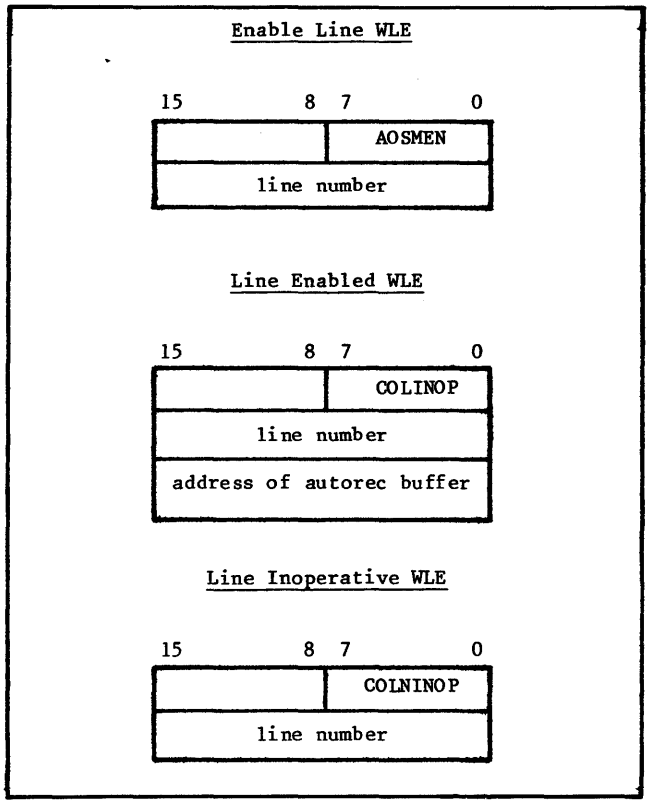


Figure 8-1. Enable Line Worklists and Replies

DISABLE LINE

When the host decides to discontinue use of a line, it sends a disable or disconnect request to the NPU. The service module processes its part of the request and passes a disable line worklist to the TIP. The TIP terminates any activities on the line and then sends a line disabled worklist to the service module. The TIP need not return the reply immediately, however. Instead it may delay any reply until the terminal is left in a known state. The TIP must release all internally maintained buffers for the line, including all the buffers held in each TCB on the line. The service module releases the TCBs.

The format of the disable line worklist to the TIP, and the format of the TIP's reply worklist to the service module are shown in figure 8-2.

CONFIGURE TERMINAL

After the line is enabled, the host can request that the terminals on that line be configured. The service module handles a large part of this configuration process and then calls the TIP directly to finish configuring the terminal.

The TIP must contain a separate subroutine to handle this call. The TIP's part of the configuration process normally consists of initializing the value of TIP-defined fields in the TCB (and in some cases the LCB).

After initializing the fields, the TIP completes the return jump to the service module.

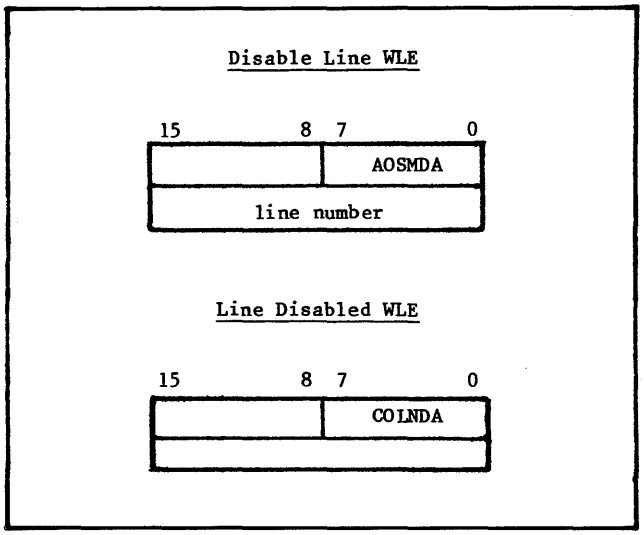


Figure 8-2. Disable Line Worklists

NOTE

If the TIP returns control to the monitor from this subroutine, the NPU will fail.

The TIP must not begin I/O at this time, as the connection is not yet open and data cannot be sent from the terminal to the host.

After both the TIP and the service module have completed the terminal configuration, the service module notifies the host and sends a TCB-configured worklist to the TIP. When the TIP receives this worklist, the TIP writer has a choice of coding options:

- The TIP can be coded so that the TIP initiates I/O with the terminal at this time. However, note that it may still be necessary to inhibit input because the connection is not yet initialized. The TIP discovers whether or not input can be accepted by calling PTREGL directly.

- The TIP can be coded to wait for the first output block from the host as the event which initiates both input and output for the terminal. This first output block is either (1) the request from the network for log on information, or (2) the first message from a user-written application program. In either case, the BIP sends a worklist to the TIP since this is the first block in the TCB's output data queue.

Note that the TIP does not return a worklist to the service module to reply to the TCB-configured worklist.

The principal characteristics of the configure TCB process are as follows:

```
Procedure name:      Chosen by the TIP writer

Input paramters:     (B1TCB) = address of TCB

Output parameters:   None
```

Processes:	Initialize values of TIP-defined TCB fields, if necessary
Required system table changes:	Address of TCB configuration routine must be initialized in the TIP type table.

Figure 8-3 shows the format of the TCB configured worklist sent from the service module to the TIP.

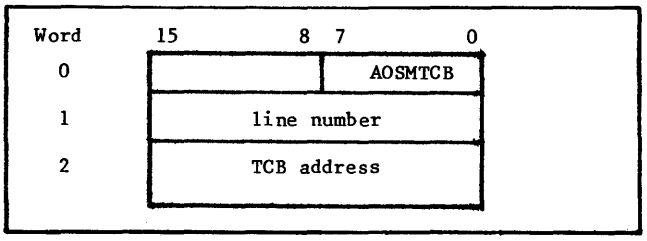


Figure 8-3. TCB Configured Worklist

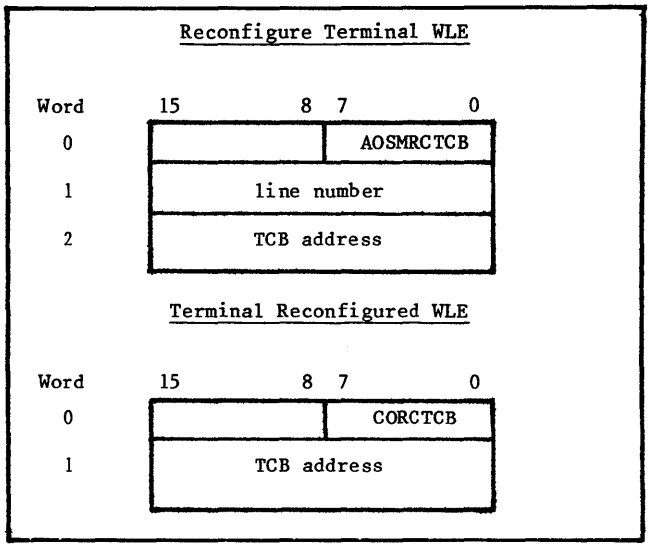


Figure 8-4. Reconfigure TCB Worklists

RECONFIGURE TERMINAL

A reconfiguration request from the service module indicates that the host is switching the data connection from one application program to another. The TIP must return the TCB to the same initialized state as if it were newly configured. When the TIP has finished its part of the reconfiguration process, the TIP sends a TCB reconfigured worklist to the service module.

The TIP reconfiguration process consists of the following actions:

- All internally held data buffers are released.
- All TIP-defined fields in the TCB are initialized.
- The TIP initiates I/O on the connection in the same way it initialized I/O for a configuration process.

It is especially important that the terminal be left in a known state since this request is a normal part of the switching procedure between application programs. All residual effects of the old application program should be eliminated and all current protocol cycles should be allowed to go to completion. The TIP should delay the reply to the reconfiguration request until all the above actions are completed. However, the TIP must reply to the request even if the line fails and the service module is sent a line-inoperative status worklist.

The service module will not send a second reconfiguration request for the same terminal until the TIP replies to the first reconfiguration request. Furthermore, if the TIP has not replied to a reconfiguration request, the service module will not send a line disable request for that terminal's line. Note, however, that there is no limitation on sending configure TCB, reconfigure TCB, or disable line requests for any other terminal controlled by the TIP.

If a terminal has failed, the TIP must retain this information. A request to change a connection does not change the error status of the terminal.

Figure 8-4 shows the worklist formats for the reconfiguration requests.

DELETE TERMINAL

When the host requests that a terminal or a terminal device be deleted, the service module sends a delete terminal worklist to the TIP. To process this request the TIP must accomplish the following:

- Release all internally held buffers
- Save any data in the TCB which the TIP has reason to retain
- Unqueue the TCB from the chain of active TCBs for the line
- Reply that the above actions have been completed by sending a terminal deleted worklist to the service module. The service module will not send a delete line request for this terminal's line if the TIP has not replied to a previous delete terminal request.

The TCB unqueuing process proceeds as follows:

- The TIP finds the pointer to the first TCB linked to the line. This pointer is held in the LCB field, BZTCBPTR.
- If this pointer is for the TCB to be deleted, the TIP moves the pointer to the following TCB into BZTCBPTR. This pointer is contained in the TCB field, BSCHAIN.
- If this pointer is not for the TCB to be deleted, the TCB chain in each TCB's BSCHAIN field is searched until the TCB preceding the TCB to be deleted is found. That TCB points to the TCB to be deleted. That TCB's BSCHAIN field is loaded with the BSCHAIN field from the TCB to be deleted.

Figure 8-5 shows the delete terminal worklist formats.

TERMINAL STATUS CHANGES

If a TIP detects changes in a a terminal's status, the TIP passes the information indirectly to the service module by

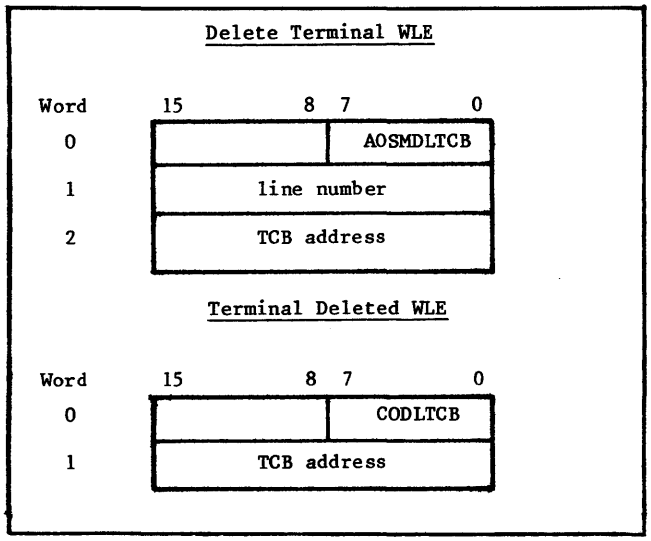


Figure 8-5. Delete Terminal Worklists

means of a direct call to PNSMGEN. The service module subsequently sends the appropriate service message to the host.

The procedures for reporting status changes are as follows:

- The TIP detects an irrecoverable error such as a protocol failure.
- The terminal must be configured at the time.
- The TIP collects the status change information and calls the SVM indirectly.
- When the error condition is resolved and the terminal can again be used, the TIP again reports the change of status via PNSMGEN.
- The host supervision does not have to act on a failing terminal status change. Application programs can be written which will not relinquish a connection even though the terminal has failed.
- The TIP must also initiate sending a STP block to the application by means of a direct call to PTSTOP. This will stop transfers on the connection. The application program can be written in either of two ways: (1) The application can relinquish the connection to the network, or (2) the application can retain the connection and wait for a STRT block to be sent, indicating that the terminal has recovered.

If the application relinquishes the connection, the host supervisor can send a message downline to reconfigure the terminal without initializing the data path (that is, the old connection is ended, but no new connection is defined). In this case, protocol blocks are inhibited from using the path, since there is no receiving application program. The host supervisor then waits for an operational status service message before any new connection is established. (The TIP notifies PNSMGEN that the terminal's status has changed to operational; the service module sends the message upline.) The host supervisor establishes the new connection by a second reconfigure message for the same terminal.

If the TIP cannot detect that the failed terminal has recovered, both the connection and the terminal are unusable until the network operator intervenes. The TIP receives notification of the operator's intervention in the form of a delete terminal request or a disable line request.

The TIP must retain the operational or nonoperational status of the terminal throughout the reconfiguration process. The TIP must not reissue a terminal inoperative status as a result of the reconfiguration request. Since the STP block sent on the original connection is not transferred to the new connection, the STRT block must be cancelled.

The principal characteristics of the terminal status change process are shown in figure 8-6.

SYSTEM ENGINEERING FILE ENTRIES

Entries are made in the system engineering file for periodic statistical analysis. Entries help the analyst to diagnose patterns of failure and to identify marginal components.

The TIP constructs the entry format in a work area and calls PNCEFILE to generate and to send the appropriate type of CE ERROR message to the host. The TIP writer must define the various fields in the entry. See section 5 for the recommended field definitions and field types. Additional user information can be added to the entry; however, new formats may require corresponding changes in the engineering file analyzer. In addition, the engineering file analayzer may need changes if the TIP writer adds new error file entry codes. No provision is made for the creation of new alarm messages to the network operator as a result of newly defined engineering file entries.

The principal characteristics of the system engineering file entry generating process are as follows:

Procedure name:	PNCEFILE
Input parameters:	(length) = length of the prototype entry in eight bit bytes
	(CNCEOVLY) = prototype entry constructed prior to the call to PNCEFILE
Return parameters:	None
Calling sequence:	Initialize the prototype entry in the work area
	Call PNCEFILE, passing the entry length as a parameter:
	PNCEFILE (length)

Figure 8-7 shows an example of calling the service module to generate a CE error service message.

ERROR PROCESSING

The system may fail in the following cases:

- The TIP sends a worklist to SVM which is not in reply to a SVM worklist, and is not one of the acceptable unsolicted worklists (unsolicited line and terminal status requests are acceptable).

- Line number or TCB address in the worklist to SVM is incorrect. In most cases, the illegal value causes destructive changes to memory which leads to an eventual NPU failure. If SVM detects the error, that module executes an immediate system halt. It should be apparent that the TIP writer must carefully validate his SVM interface.

```
Procedure name:     PNSMGEN

Input paramters:    (GENPFC) = D8STATUS
                    (GENSFC) = D9TMLSTAT
                    (OP)     = TCB address
                    (BSINOP) = Terminal status (operational = TRUE, inop = FALSE)

Output parameters:  None

Calling sequence:   The global variables GENPFC and GENSFC are INTEGER type. The global constants D8STATUS
                    and D9TMLSTAT are provided for use with this call. The global variable must be set up
                    immediately preceding the call.

                    The global OP is of BOBUFPTR type.

                    The boolean variable BSINOP is a flag in the TCB. The TIP sets and clears this flag.

                    The TIP cannot call the service module directly. Therefore an indirect call is made to
                    PNSMGEN. The array BRTNJUMP, the constant C1PNSMGEN, the fields JENTADDR and JPAGEVAL
                    in the array BRTNJUMP and the procedure PBXFER are all system defined. The TIP writer
                    should use the code shown in figure 8-6 to make the call.
```

Figure 8-6. Terminal Status Change Call to Service Module

```
WITH <tcbpointer> ↑Dd
   WITH CNCE OVLY |<level>| Dd
   BEGIN
      CNCECODE           :=  <error file entry code>; ↱entry code       ↓
      <line number>      :=  <lcbpointer>↑.BZLINO;    ↱line number      ↓
      <cluster address>  :=  BSCA;                    ↱cluster address  ↓
      <terminal address> :=  BSTA;                    ↱terminal address ↓
      <device type>      :=  BSDEVTYPE;               ↱device type      ↓
      <terminal class>   :=  BSTCLASS;                ↱terminal class   ↓
      <error count>      :=  <occurrence count>;      ↱occurrence count ↓
      PNCEFILE (<length>);                            ↱issue SEF entry  ↓
   END;

   where:

      <tcbpointer>              =   pointer variable to TCB
      <level>                   =   interrupt level constant
                                      OPS  = OPS level (normal) entry to TIP
                                      MUX2 = MUX level entry to TIP
      <error file entry code>   =   defines entry
                                      user should select values X'40 to X'4F
      <line number>             =   line number and terminal reporting error
      <lcb pointer>             =   pointer variable to LCB
      <cluster address>         =  )
      <terminal address>        =  } entries from TCB defining
      <device type>             =  } terminal with problem
      <terminal class>          =  )
      <error count>             =   count field for this error
      <occurence count>         =   value for count field
```

Figure 8-7. Sample Call to SVM to Generate a CE Error Message

CCP provides a variety of subroutines to perform functions that are common to several TIPs. These common functions are as follows:

- To check if input regulation should occur
- To provide reentrant code entry and exit procedures
- To process IVT commands
- To gather error processing statistics

INPUT REGULATION, PTREGL

Before a TIP can solicit or allow upline data, the TIP must check the upline traffic conditions to find if input should be regulated (inhibited). Four types of conditions can cause the TIP to regulate input. The TIP has the choice of checking for any of these conditions or any combination of these conditions:

- The data path from the terminal is congested with too many upline blocks. This condition occurs when the number of network blocks sent to the host, but not yet acknowledged, exceeds the user defined threshold value.
- The data path from the host to an NPU attached to the terminals is congested because of a failed intermediate node or a shortage of data buffers in an intermediate node. An intermediate node is defined to be any NPU that is upline from an NPU which has attached terminals.
- The node attached to the terminals is itself congested, and has too few assignable data buffers to allow further input from the terminals.
- The data path from the terminal to the host application program has not been initialized.

The TIP issues a call to the boolean function, PTREGL, to determine if one or more of these regulation causing conditions exists.

The principal characteristics of the regulation checking subroutine are as follows:

```
Procedure name:     PTREGL

Input parameters:   (tcbpointer) = address of the
                    TCB

                    (regltype) = variable of type
                    SETWORD with the following
                    values:

                      EGLOGLNK = logical link
                      regulation

                      REABL = network block limit

                      REACPINP = connection
                      initialized status

                      RELOCAL = data buffer
                      shortage in this NPU

Output parameter:   input regulation status, type
                    BOOLEAN.

                    TRUE = input allowed

                    FALSE = at least one of the
                    requested types of regulation
                    condition exists.

Calling sequence:   (regvalue) := PTREGL
                    ((tcbptr),(regtype))

Example of call:

IF PTREGL(B1TCB, RELOGLNK,REABL,REACPINP,RELOCAL )
THEN
BEGIN               ALLOW INPUT FOR THIS
    .               TERMINAL
    .
    .
END;
```

For controlled protocols, the TIP may discover the status of all the possible conditions prior to permitting input. For uncontrolled protocols, the terminal's input may be accepted by the TIP and then discarded. The terminal must be notified that the input was discarded in these cases.

REENTRANT CODE ENTRY AND EXIT PROCEDURES

If the TIP supports more than one line, the TIP must use reentrant coding to use the network effectively . It is possible that the reentrant code could be written for each entry to the TIP so that the TIP would relinquish control with a normal exit. Given the modular structure of TIPs, it is more likely that the TIP will relinquish control within a subroutine while waiting for an external event to occur. The occurrence of the event will cause the TIP to reenter the subroutine at the point where control was relinquished, and to continue processing. The type of reentrancy cannot be handled by the push down stack logic which the PASCAL compiler provides.

The common subroutine is written so that the TIP executes code to set up to await the external event, and then exits to the system by calling one of two exit procedures: PTSV1LCB or PTSV2LCB. Since each routine saves the address of its call in the line control block (LCB), only one call may be outstanding for each routine at any one time. Only the return address is saved; the TIP

TABLE 9-1. CCP INTERRUPT LEVELS

Hardware Line No.	Software Priority	Interrupt Description
0	P1	INTERNAL
1	P6	TELETYPE (CCP CONSOLE)
2	P2	MULTIPLEX LOOP ERROR
3	P3	MUX-2 LEVEL
4	P16	1742-30 LINE PRINTER (CONSOLE OPTION)
5	P5	SPARE
6	P7	COUPLER
7	P8	SPARE
8	P9	REAL TIME CLOCK
9	P10	1742 LINE PRINTER (CONSOLE OPTION)
10	P11	SPARE
11	P12	SPARE
12	P13	MLIA OUTPUT DATA DEMAND (ODD)
13	P14	MLIA INPUT LINE FRAME
14	P15	SPARE
15	---	HARDWARE BREAKPOINT
--	P17	OPS LEVEL PROGRAMS (OPERATION BELOW THE LOWEST INTERRUPT LEVEL)

must be coded to handle restoration of other necessary information, such as local variables, program return address, program initialization and formal parameter linkage. Saving all this information would normally require the TIP writer to have an intimate knowledge of the compiler generated code; however, coding conventions are employed to avoid the need for such knowledge.

There are sixteen interrupt levels in the CCP system, and a seventeenth (lowest) level called the OPS-level. The TIP is coded to permit all its PASCAL subroutines to execute at the same interrupt level (OPS-level). (Note, however, that a TIP can have one or more independent PASCAL routines that are coded to execute on Mux-2 level. This is explained elsewhere.) Table 9-1 shows the interrupt levels of the standard CCP system.

The OPS-level TIP retains control of the processor until it voluntarily relinquishes control to the system. This allows the TIP to use local variables without being concerned about their destruction until control is relinquished. Therefore, the TIP saves only those local variables that will be needed after reentry. This convention allows the TIP to be compiled by PASCAL without specifying the PASCAL reentrancy compiler option. The compiler uses this option to create and to use its own local variables.

The compiler uses two registers (R1 and R2) in the generated code. These registers are initialized by a WITH statement of a pointer variable or a formal parameter. The code assumes the correct content of these registers within the domain of the WITH statement. If three or more WITH statements are nested, the compiler creates local variables for use as substitute registers. To simplify these conditions, the convention is adopted to close all but two WITH statements prior to making an exit for for a subsequent reentrance. Further, the rule of closure is invoked if a procedure is called which ultimately exits for a subsequent reentry. If one or two WITH statements are not closed, they are always the WITH of the LCB and the TCB, in that order. This rule defines the contents of R1 and R2 for all modules.

The convention of using the TCB and the LCB is arbitrary. The TIP writer can use any other specification. However, it is convenient to always use a fixed constant for these registers.

If a procedure which exits for subsequent reentry is called from more than one place, the return address must be saved. This return address may be obtained by the PASCAL procedure RETADR. The TIP must save this adddress in table space associated with the terminal, usually in the TIP defined area of the TCB.

If a procedure uses a WITH statement of a pointer variable, the procedure entry code saves the current contents of R1 and R2. The initialization code uses local variables and restores the values of the last call to the procedure, not the values of the current caller.

Therefore, the TIP must execute code to return directly to the called procedure, thereby avoiding register restoration. This convention is used only if the values of R1 and R2 are always known to be the result of a WITH of the LCB and the TCB respectively. The restoration of registers may be avoided by defining a procedure which executes the PASCAL procedure RETURN without specifying a WITH statement.

The register restoration avoidance procedure is as follows:

```
PROCEDURE (special exit)((return address):
  INTEGER):
BEGIN
    RETURN (returnaddress);
END;
```

The following is an example of the use of the procedure:

```
PROCEDURE (tip module);
BEGIN
    WITH(tcbpointer)↑.BSTCB,      TCB POINTER
        (lcbpointer)↑DO           LCB POINTER
    BEGIN
    RETADR(<return address>);     PROCEDURE RETURN
        .
        .
        .
    END;  WITH (tcbpopinter)
    (special exit)(<return address>);
END; (tip module)
```

Formal parameters cause the use of local variables and cannot be referenced following an exit for subsequent reentry. Therefore, all return parameters must be passed back to the calling procedure in dynamic tables such as the TCB. Further, all references to the content of formal input parameters must precede the exit. This rule normally results in passing only the TCB and the specification of the TCB in the WITH as the first statement.

The principal characteristics of exiting for subsequent reentry are as follows:

```
Procedure name:     PTSV1LCB or PTSV2LCB

Input parameter:    (HALCBP[HLOPS]) = LCB address

Return parameter:   none

Calling sequence:   PTSV1LCB
                    PTSV2LCB

Example of call:

PTSVxLCB;           Wait for event
```

At some later time, the TIP receives control from the system to process the worklist, which indicates that the awaited event has occurred. The TIP must recognize this solicited worklist so that it can call the reentrancy procedure PTRT1LCB or PTRT2LCB. These procedures transfer control to the subroutine which last executed the related PTSV1LCB or PTSV2LCB procedure. The contents of registers R1 and R2 are not altered. Therefore, the TIP must specify the WITH of the LCB and TCB prior to executing the reentry.

The principal characteristics of executing a reentry are as follows:

```
Procedure name:     PTRT1LCB or PTRT2LCB

Input parameter:    (HALCBP[HLOPS]) = LCB address

Return parameter:   none

Calling sequence:   PTRT1LCB
                    PTRT2LCB

Example of call:

WITH (tcbpointer)↑.BSTCB,
     (lcbpointer)↑.DO
BEGIN
    PTRT1LCB;       Reenter the TIP module
END;
```

IVT COMMAND PROCESSING

When the TIP detects either an upline or downline interactive virtual terminal (IVT) command, the TIP calls function PTIVTCMD to check the syntax of the request and to process the command. If the command is entered from a terminal, PTIVTCMD decodes the command and changes the appropriate fields in the IVT parameter area of the TCB. On return, a reply to the terminal operator is provided in IVT format. The TIP will subsequently convert and transmit this information to the terminal. PTIVTCMD performs similar processing for downline commands from a host application program, but in this case the reply to the host takes the form of a BACK or BRK block. If the command was accepted, a BACK block is sent; otherwise, a BRK block is sent upline.

The function PTIVTCMD returns a boolean value to inform the TIP whether the command was accepted or not. A value of TRUE indicates that the command was valid. The TIP must initialize any internal or external conditions which are affected by the IVT parameter. The specific IVT command and the communications line adapter or terminal define the actual process to be performed. For most TIPs, nothing is required. The buffers containing the command are always released, whether or not the command was accepted.

The TIP is not informed which IVT command was detected. If any IVT command causes changes to internal tables or to the communications line adapter, the TIP must perform all the actions for any command.

The principal characteristics of a call to PTIVTCMD are as follows:

```
Procedure name:     PTIVTCMD

Input parameters:   (source) = source of command

                      C9TERM = upline command
                      from terminal

                      C9APPL = downline command
                      from application program

                    (B1BUFF) = address of data
                    buffers holding the command
```

```
Return parameters:  (PTIVTCMD) = BOOLEAN
                    condition of results:

                       TRUE = command accepted
                       FALSE = command rejected

                    (B1BUFF) = pointers to the
                    buffers with reply to
                    terminal for an upline
                    command, in IVT format.  The
                    internal block flag
                    (BFINTBLK) is set TRUE.

Calling sequence:   (result) = PTIVTCMD(source);

Example of call:

VAR
    (result) : BOOLEAN;              COMMAND RESULT
BEGIN
    B1BUFF := (bufferpointer);       BUFFER WITH
                                     COMMAND
    (result) := PTIVTCMD (B9TERM); PROCESS
                                     COMMAND
END;
```

STATISTICS

The system accumulates line and terminal statistics for periodic reports to the host's engineering file. Statistics are normally gathered by the POI procedures PBPOPOI and PBPIPOI for downline and upline traffic respectively. The TIP must also gather error statistics. For each block which had errors that required retransmitting the block, the TIP calls PNSGATH to record the event.

The principal characteristics of a call to PNSGATH are as follows:

```
Procedure name:     PNSGATH

Input parameters:   (tcbpointer) = address of TCB
                    (buffer pointer) = NIL
                    (statistics type)=JOBADBLK

Return parameters:  none

Calling sequence:

    PNSGATH((tcbpointer),(bufferpointer),
    (statisticstype));

Example of call:

VAR
    (tcbpointer) : BOBUFFPTR;      TCB ADDRESS
BEGIN
    (tcbpointer) := (buffer address);
    PNSGATH((tcbpointer),NIL,JOBADBLK);
END;
```

The CCP operating system has several subroutines which provide services for the TIP. The TIP calls these subroutines directly, or directly references tables and variables kept by these subroutines. The services are as follows:

- Placing the TIP into control with a worklist-defined task. The base system initiates TIP execution and provides a work area for the worklist.
- Providing queuing of TIP-generated worklists, to itself, or to other OPS-level modules
- Assignment and release of buffers
- Timing services
- Finding the address of a line control block from the line number

TIP EXECUTION STARTED BY A WORKLIST

At the OPS level, the operating system monitor places the TIP into execution when that TIP is the next module on the OPS-monitor worklist table, and the TIP has a worklist queued to it. The OPS monitor scans the worklist control blocks for all OPS-level entry modules on a round robin basis, so that all of these procedures have equal priority.

Before placing the TIP in control of the NPU, the monitor moves the worklist entry into a global variable. The global variable is a complex type. Two of its variants apply to TIPs: one of these is for OPS-level entries, the other is for Mux-2 level entries. The two types of worklist entries used for TIPs are shown below. Since it is not possible to determine which type of processing to use before the workcode field is decoded, the TIP uses the multiplex subsystem overlay exclusively. The SVM data structure overlay is used when the TIP sends a worklist to the service module.

OPS-level reference to the worklist is as follows:

```
VAR
    (wc) : INTEGER;                          Workcode
BEGIN
    WITH BWWLENTRY[OPS].CMSMLEY DO
    BEGIN
        (wc) : = CMWKCODE;
    END;
END;
```

Mux-2 level reference:

```
VAR
    (wc) : INTEGER;                          Workcode
BEGIN
    WITH BWWLENTRY[MUX2].BOEWLQ DO
    BEGIN
        (wc) := MMWKCOD;
    END;
END;
```

SENDING A WORKLIST ENTRY

The TIP calls PBLSPUT to send a worklist entry to the TIP itself or to another subsystem such as the service module.

The principal characteristics of a call to PBLSPUT are as follows:

```
Procedure name:     PBLSPUT

Input parameters:   (WLEptr) = Address of
                    worklist entry

                    (WLCBpointer) = Address of
                    the worklist control block

Return parameters:  none

Calling sequence:   PBLSPUT( WLEptr ,  WLCBptr );

Example of call:

WITH BWWLENTRY[OPS].CMSMLEY DO   Type of worklist
BEGIN
    CMWKCODE : = (wc)
    CMDATA : = (data)
END;
PBLSPUT (BWWLENTRY[OPS],
    BYWLCB[BOSMWL]);                 Generate a
                                     worklist for
                                     the service
                                     module
```

Note that the TIP is needed to set the worklist fields prior to calling PBLSPUT.

BUFFER MANAGEMENT

The base system provides subroutines to assign and release buffers for the TIP.

ASSIGNMENT OF A BUFFER

The TIP may need to create messages or table structures in buffers. The TIP calls PBGET1BF to assign a buffer for such a specific function. PBGET1BF returns the address of the buffer which has been reserved for the TIP.

The call to PBGET1BF to assign a buffer is as follows:

```
VAR
    (bufptr) : BOBUFPTR;              Buffer address
BEGIN
    (bufptr) : = PBGET1BF(bufsize);
END;
```

Bufsize is a set of system defined global variables. The use of any sized buffer but those that are system defined causes a system halt. The sizes are as follows:

Variable	Size
BOS8	8 word buffer
BOS16	16 word buffer
BOS32	32 word buffer
B064	64 word buffer
BEDBSIZE	Data buffer size
BETPSIZE	TPCB buffer size; text processing control block

Bufptr contains the assigned buffer address when control returns to the TIP. Note that most data buffer assignment is done for the TIP, both during text processing and during input state program processing. In both cases, the firmware program (text porcessor or input data processor) does the buffer asignment.

BUFFER RELEASE

Two base subroutines are provided for releasing buffers; one for releasing a single buffer, and the other for releasing a chain of buffers.

Single Buffer Release

The TIP calls PBREL1BF to release a single buffer. The calling procedure is as follows:

```
VAR
    (bufptr) : BOBUFFPTR;          Address of buffer
BEGIN
    PBREL1BF((bufptr),(bufsize));
END;
```

Bufptr must contain a valid buffer address. Bufsize must be one of the valid system global variables defined for buffer size (see above).

If the buffer that is being released has other buffers chained to it, only the first buffer in the chain is released. The address of the next buffer in the chain is returned in the bufptr parameter. If the TIP passes a bad address to PBREL1BF in bufptr or a bad buffer size in bufsize, this will cause destruction of memory, and lead to an eventual system halt. Note that bufadr equal zero is valid for PBRELZRO but invalid for PBREL1BF.

Release of a Chain of Buffers

The TIP calls PBRELZRO to release a chain of buffers. Format of the call is as follows:

```
VAR
    (bufptr) : BOBUFFPTR;  Address of first buffer
BEGIN
    PBRELZRO((bufptr),
        (bufsize));
END;
```

Bufptr contains the address of the first buffer in the chain. Bufsize contains the size of each buffer in the chain (chained buffers are always uniform in size). It must be one of the globally-defined buffer sizes.

On return to the TIP, bufptr always will be nil. If the TIP passes PBRELZRO, a bad address in bufptr, or a bad buffer size in bufsize, this will cause destruction of memory, and lead to an eventual system halt.

NOTE

Use of PBRELZRO protects against any attempt to release a nil bufptr.

TIMING SERVICES

The base system provides two timing functions for user written TIPs. These are:

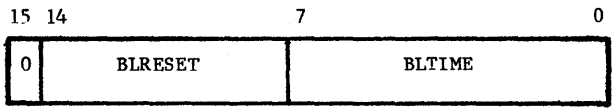
- A variable containing a one second clock which the TIP can reference to measure timed intervals.
- An array of decremented timers with an entry for each line in the system.

ONE SECOND CLOCK

The system variable CASECNTR is incremented once a second by the base system timer. This INTEGER type variable is set to zero during system initialization and is never reset. The time recycles to zero at 65,536 seconds.

LINE TIMER

The line timer provides a set of functions so the TIP writer can time line related events. The timer control table is an array indexed by port number. The port number is obtained from the line control block. Each entry in the array is of the form:



The timing routines scan the table for a nonzero value for each enabled line in the system. Each half-second the timer decrements the value in BLTIME until the value reaches zero. At that time the system sends a timer worklist (A0TIMOUT) to the TIP that controls the line. The TIP can set the BLTIME value at any time by placing a nonzero value in the field. The TIP can cancel the timer at any time by setting BLTIME to zero.

If the TIP sets a nonzero value in BLRESET, the multiplex subsystem will transfer this value to BLTIME (thereby starting the timer), whenever output is successfully sent over the line. This feature is useful for TIPs, which support a half-duplex protocol where a terminal response to the output block is expected to begin within a fixed interval, such a two seconds. An input state instruction can set BLTIME to zero or to a larger value once the first input character has been returned from the terminal. The system never changes the value in BLTRESET.

The following example shows a use of the line timer:

```
CONST
    (rstime) = (rstvalue);   Link turnaround time
    (setime) = (setvalue);   Timer interval
VAR
    (lcbptr) : BZLCBP;
```

```
BEGIN
    WITH HALCB|HLOPS|↑,
        BLTIMTBL↑|port| DO
    BLTIME : = (setvalue);    Starts the timer
    BLTREST : = (rstvalue);   Set turnaround time
    BLTIME : = 0;             Clear timer
END;
```

The line timer is often used to assure that an expected event occurred. If the TIP is waiting for an expected event that will generate a worklist, the timer can be started to time the event (note that an expected event and a soliticited worklist are different terms for the same action). If the timer worklist entry is received by the TIP, the expected event failed to occur. If the expected event worklist is received by the TIP, the event did occur during the permitted period.

Since line inputs or outputs are usually the events to be timed, the TIP can set flags in the LCB which can be referenced by the system to terminate input and/or output before sending a worklist to the TIP. This feature is selected by setting boolean values in the LCB to true, as shown below:

```
VAR
    (lcbptr) : BZLCBP;          LCB address
BEGIN
    WITH (lcbptr)↑ DO
    BEGIN
        BZINPUT : = TRUE;       Terminate input on
                                timeout
        BZOUTPUT : = TRUE;      Terminate output
                                on timeout
    END;
END;
```

It is possible for an expected event to occur before the timer is cancelled. This causes both worklists to be sent: the timer worklist, and the expected event worklist. To take care of this condition, the timeout worklist can be marked with a sequence number, BZWTCOUNT, supplied by the TIP in the LCB. This value is placed in the MMWTCOUNT field of the worklist. If the TIP increments BZWTCOUNT when the timer is cancelled, the timer worklist can be discarded as extraneous, since MMWTCOUNT is not equal to BZWTCOUNT. If, however, the expected event worklist follows the timer worklist, the former worklist cannot be detected as the timed event. Use of the routines PTSV1LCB and PTSV2LCB causes the LCB value, BZWTCOUNT, to be incremented. Caution should be exercized for handling any reentrancy conditions for an expected worklist following a timeout worklist. Most conditions can be avoided by using the terminate input and terminate output features of the timeout event. However, it is also possible to generate a TIP-defined continue worklist in the TIP's own queue. Then the TIP can discard any expected event worklists until the continue worklist is received. Note that the TIP must not discard any other worklist entries; other worklist entires must be processed normally.

LINE CONTROL BLOCK (LCB) ADDRESS

The TIP calls the base subroutine PBLCBP to find the address of the line table when the TIP has the line number.

The principal characteristics of a call to PBLCBP are as follows:

```
Procedure name:     PBLCBP

Input parameter:    (lineno) = Line number

Return parameter:   (LCBptr) = Address of LCB

Calling sequence:   PBLCBP((lineno),(LCBptr));

Example of call:

    VAR
        (LCBptr) : BZLCBP           Address of LCB
        (line no) : INTEGER         Line number
    BEGIN
        PBLCBP((line no),(lcbptr));
    END;
```

The line number is usually specified for the TIP by the worklist entry.

The multiplex subsystem contains the hardware, microprograms, and software elements necessary to provide data and control paths for information interchange between the TIPs and all communications lines. Design of the subsystem is based on the multiplex loop concept, which is a demand-driven system for gathering input data and status from the communications lines, and distributing output data and control information to the communications lines. All of this is done on a real-time basis. Figure 11-1 shows the basic elements of the multiplex subsystem.

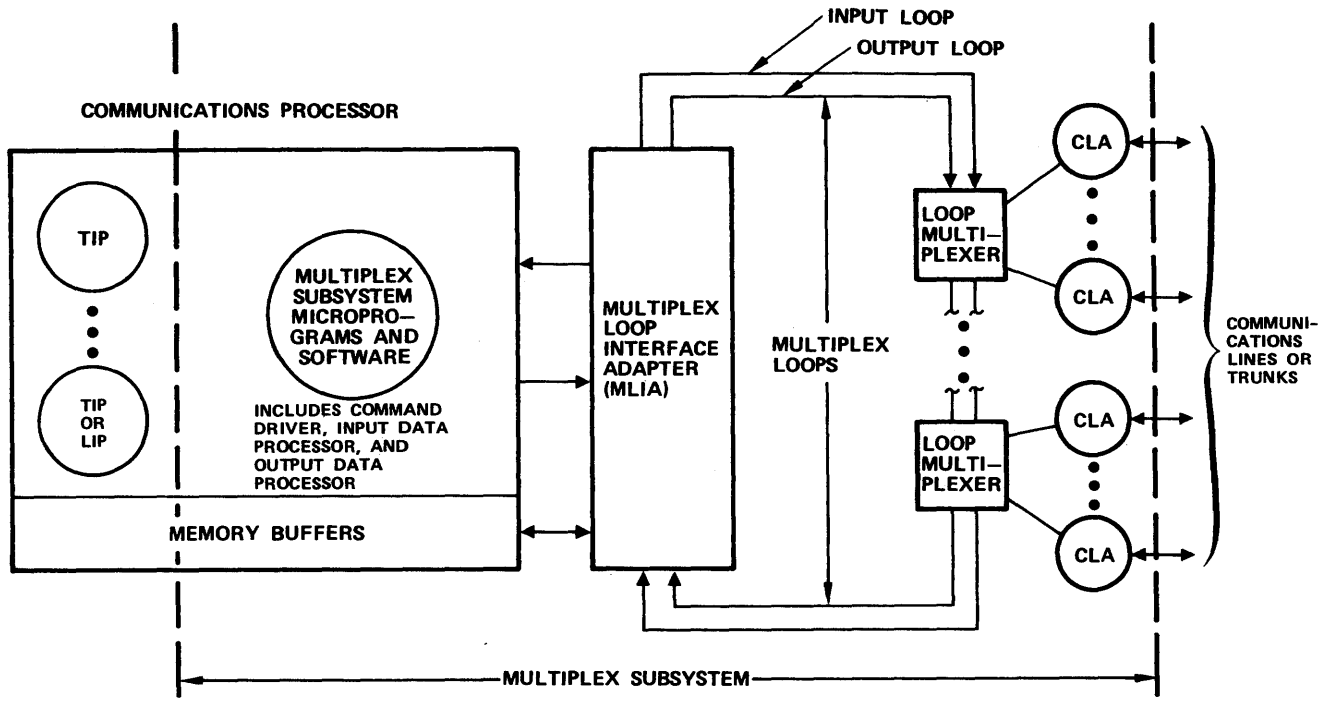
A major purpose of the multiplex subsystem is to transfer the task of processing lines according to physical characteristics from the TIPs to the multiplex subsystem programs. The TIPs need only command the multiplex subsystem according to the logical characteristics of a line; the physical characteristics are handled by the multiplex subsystem and are transparent to the TIPs.

Line-oriented input and output buffers provide temporary storage for data. The input data is placed in the circular input buffer (CIB) from which it is later extracted (demultiplexed), transformed to IVT/BVT ASCII format by the appropriate TIP, and moved into a line-oriented input buffer. The part of the TIP that does this (called input state programs) is controlled by the multiplex subsystem. The OPS-level TIP informs the command driver where the programs are located; the multiplex subsystem's input processor controls execution of the input state programs.

Output data is read by the output processor from an output data buffer. The address of this buffer and other transfer information is supplied by the OPS-level TIP to the command driver. Data is in terminal format.

The multiplex subsystem is event-driven by interrupts: an output data demand (ODD) for the next character of output data, or the input line frame received interrupt which indicates that data (and possibly CLA status) is contained in the CIB ready for demultiplexing.

The interrupts are handled with global information stored in various tables. The subsystem processes data on a character-by-character basis while TIPs process data on a message or block basis. Circuit, modem, and subsystem status is detected and transferred to the TIPs using OPS-level worklist calls. Control information is received from the TIPs in the form of a call to the command driver with an attached command packet. This command packet



CLA – COMMUNICATIONS LINE ADAPTER
TIP – TERMINAL INTERFACE PROGRAM
LIP – LINK INTERFACE PROGRAM

M-166

Figure 11-1. Basic Elements of the Multiplex Subsystem

is used to set up the multiplex LCB (NCLCB), which is the principal table used to control the transfer.

HARDWARE COMPONENTS

The multiplex subsystem includes the multiplex loop interface adapter (MLIA), loop multiplexers, and communications line adapters (CLAs).

MULTIPLEX LOOP INTERFACE ADAPTER

The MLIA provides hardware interface between the multiplex input/output loops and the multiplex subsystem software. The major functions are as follows:

- Management of the I/O loops
- Input data buffering - compensates for the difference in rate at which characters are removed from the input loops and the rate at which they are stored in the main memory
- Output data demand (ODD) detection and buffering
- Multiplex loop error detection
- Generation of interrupts for the multiplex subsystem microprograms and software for functions such as:

  - Output data demand received
  - Line frame received
  - Loop error conditions

LOOP MULTIPLEXERS

Each loop multiplexer provides an interface between a group of as many as 32 CLAs and the demand-driven multiplex loop. Its primary function is to receive parallel data from the CLAs and present it to the serial input loop in the loop cell format. Conversely, it assembles serial data in the loop cell format from the output loop and presents it to the CLAs in parallel form.

COMMUNICATIONS LINE ADAPTERS (CLA)

The CLAs provide the interface between the loop multiplexers and the communications lines. The primary functions of the CLAs are to assemble serial data from the communications line into parallel data and present this data to the loop multiplexer or, conversely, to disassemble parallel data from the loop multiplexer and present it in serial form to the communications line. The CLA operating characteristics can be altered under program control for such functions as signal rate, character length, parity, and stop bit duration.

TIP INTERFACES

The multiplexer interface to the TIPs is defined in the following paragraphs.

A TIP is a multilevel program that executes at three processing levels:

- Multiplex level 1 (firmware or microcode level)
- Multiplex level 2 (macrocode level)
- OPS level (processing to satisfy network protocol such as service message handling and timing)

MULTIPLEX LEVEL 1 - INPUT STATE PROGRAMS

All incoming characters are processed by the multiplex subsystem, as directed by the input state programs. These

TABLE 11-1. MUX-2 LEVEL WORKCODES

Workcode	Description
MMOBT	If the output buffer terminate flag (BJOBT) is set in the TIP-type table, the multiplex subsystem will send this workcode upon completion of each output block. The TIP uses this workcode as a trigger to start another output, thereby assuring maximum line utilization. All text processing of output data must be done at the OPS level, and not in response to this workcode.
MMFES	If an asynchronous CLA reports a framing error status, this workcode is sent to the TIP.
MMBREAK	If an asynchronous CLA reports a break, and the break condition is maintained for more than 200 ms, the multiplex subsystem sends this workcode to the TIP.
MMCHOUT	This workcode is sent to the TIP when the character timer expires. The timer has a resolution of 100 ms and is used only with asynchronous CLAs.
A0WK1 to A0WK31	The TIP's input state programs can send a workcode to the Mux-2 level procedure with one of these user-defined codes. This procedure can disrupt the system's normal multiprogramming methods. It should be avoided.
MMBUTCH	When the system is critically low on buffers and a new buffer is required to process an incoming character, the TIP's input state programs are given control. A predefined state (state 3) is entered. This state can issue a MMBUTCH workcode to the TIP's level-2 procedure. The multiplex subsystem detects this workcode and releases all of this line's input buffers. It also stops processing of any other characters from other input lines.

TABLE 11-2. OPS-LEVEL WORKCODES

Workcode	Description
A0HARDERR	The multiplex subsystem has detected an irrecoverable modem signal error. The TIP should report a line failure to the host.
A0WK1 to A0WK31	The TIP's input state program creates a worklist entry to send character-related event information to the OPS-level TIP. These codes are TIP-defined. These codes may also be used to indicate a terminate input or terminate output request.

TIP-supplied programs direct the translation of terminal data into network data format. The input state programs are constructed of a special set of state programming language instructions which are interpreted by the input data processor of the multiplex subsystem. Usually, the entire translation of terminal data to virtual terminal format can be accomplished by the proper use of state programming instructions. The instructions for handling and generating these codes are described in detail in the State Programming Language Reference Manual. Table 11-1 summarizes workcode functions for Mux-2 level and table 11-2 describes the workcode functions for OPS level.

MUX-2 LEVEL

Each TIP can provide a procedure to receive control from the multiplex subsystem by means of a direct call. This procedure is to process high priority tasks. The OPS-level portion of the TIP can be interrupted to process the event leading to this priority call. Therefore, any procedures that are common between the OPS level and the Mux-2 level of the TIP must be compiled with the reentrant code option selected. It is very desirable to avoid sharing subroutines between the two levels.

The Mux-2 level procedure is entered for four system-defined events. The entry parameters appear in the form of a worklist, with the workcode and parameters contained in a work array, BWWLENTRY. The four system workcodes are defined in table 11-1. In addition, the TIP can send workcodes to its Mux-2 level procedure for user defined events. The worklist entries are entered in the multiplex subsystem event work list queue (MMEWLQ). Those worklist entries not recognized by the multiplex subsystem (A0WK1 through A0WK31) are passed on to the TIP. Each input state instruction which creates a worklist (BLDWL and TIBWL) requires the address of the worklist control block owning the particular queue. This address is calculated by an MPEDIT expression which utilizes the worklist index. For the multiplex subsystem queue itself, the work list index used in the expression is MMEWLQ.

OPS LEVEL

The TIP must provide a primary entry point to receive control for all OPS-level worklist entries. These worklist entries are generated by any of several system modules as well as by the multiplex subsystem. The multiplex subsystem originates a worklist whenever it is requested to do so by one of the TIP's input state programs. These worklists indicate the occurrence of character-related events, such as the start of a message, the end of a message, the end of a line, or other protocol related events. Table 11-2 defines the workcodes sent to the OPS level by the multiplex subsystem

COMMAND DRIVER

The TIP can issue commands to the multiplex subsystem to request the execution of an input, output, or control function for the line. These commands can be issued from either the OPS level or the Mux-2 level of the TIP. The commands indicate the type of function to be performed and are used to initialize user fields in the mux LCB which the TIP's input state programs will need to use.

The command driver procedure used by the OPS-level TIP is PBCOIN (general format of the command packet that accompanies the call is shown in figure 11-2); the procedure used by the Mux-2 level TIP is PMCDRV. Two procedures are provided to avoid the complexities of reentrant coding.

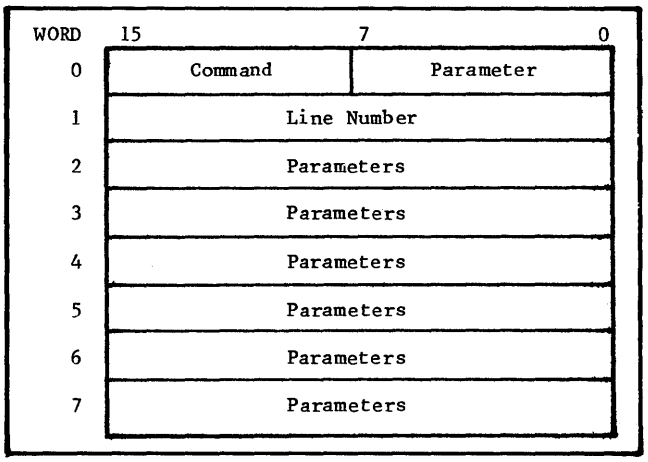


Figure 11-2. Command Packet General Format

Principal characteristics of the command driver call are as follows:

Procedure name: PBCOIN or PMCDRV

Input parameter: (command packet) = Data record of type NKCMD with control and data parameters

Output parameter: None

Calling sequence: PBCOIN (command packet);
PMCDRV (command packet);

Example of a call:

```
VAR
    (command packet) : NKINCOM;      Command
                                     packet
```

```
    (output buffer)  : BOBUFPTR;     Address of
                                      output buffer
BEGIN
    WITH(command packet) DO           Set up
                                      output command

    BEGIN
        NKCMD  : = NKDOUT;            Command code
        NKLINO : = (line number);     Line number
        NKOBP  : = (output buffer);   Address of
                                      data
    END;
    PBCOIN(command packet);           Initiate
                                      output
END;
```

The following commands are available to the TIP writer for controlling the flow of data to and from the communications lines:

- NKCONTROL - Control line
- NKINPT - Input
- NKDOUT - Output
- NKINOUT - Input after output
- NKENDIN - Terminate input
- NKENDOUT - Terminate output
- NKDISL - Disable Line

Control Command

The control command (NKCONTROL) serves a twofold purpose. It can define the character transmission characteristics of a given line according to the terminal class (NKTCLS) for input/output signaling rate, character length, parity type, stop bit duration, and sync character. The command can also specify up to five modem/circuit

WORD	15 14		7 6	0
0	NKCMD		NKTCLS	
1	NKLINO			
2	F1	NKFUN1	F2	NKFUN2
3	F3	NKFUN3	F4	NKFUN4
4	F5	NKFUN5	NKZERO	

Field	Type	Description
NKCMD	B08BITS	Command code (NKCONTROL)
NKTCLS	B08BITS	New terminal class. If nonzero, the character parity, character length, and sync character are set up for this new terminal class from NJTECT.
NKLINO	INTEGER	Line number
F1 thru F5 and NKFUN1 thru NKFUN5	BOOLEAN	Optional modem/circuit function; if the associated flag (NKSRF1 - NKSRF5) is set, the function is to be implemented. 1 = Function to be performed 0 = Function disabled
NKZERO	B08BITS	Delimits end of options. NKZERO is placed in the byte following the last requested modem/circuit function; five functions can be specified.

OPTIONAL MODEM/CIRCUIT FUNCTIONS

Function Mnemonic	Function Provided	Description
NOBREAK	BREAK	Send break
NOECHO	ECHO	Echoplex mode
NOPON	PON	Parity on
NOPSET	PSET	Parity set (1 = even, 0 = odd)
NOCLLS	CLLS	Character length (LSB)
NOCLMS	CLMS	Character length (MSB)

Pulsed functions provide momentary signal and need not be reset

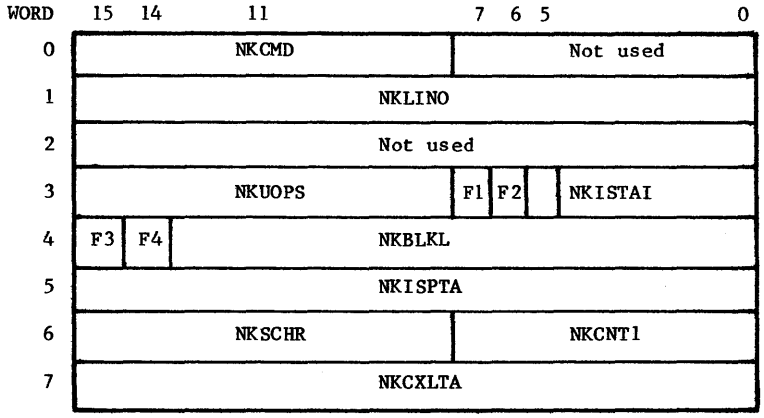
Figure 11-3. Control Command Format

control functions, such as echo, break, terminal busy, or resync. Such control functions are specified in the optional fields of the command packet.

Generally, the command is used to initialize or alter the character transmission characteristics of the line or to generate circuit control functions. The control command format is as shown in figure 11-3. Optional modem/ circuit functions are defined in the figure.

Input Command (NKINPT)

The input command directs the multiplex subsystem to initiate the processing of data on the specified input line (i.e., to turn on the input side of the communications line adapter). The processing functions provided by the subsystem are determined by the input processing state program index. Additional information is passed by a pointer table address for the input processing states. If



Field	Type	Description
NKCMD	B08BITS	Command code (NKINPT)
NKLINO	INTEGER	Line number
NKUOPS	BOOLEAN	Eight user flags (NKUOP1 - NKUOP8). NKUOP1 is bit 15 in the MLCB user flag field,...NKUOP8 is bit 8 in that field. NKUOPS is moved into MLCB if NKMVB is 1.
F1	BOOLEAN	NKMVB, move block of user flags into MLCB
F2	BOOLEAN	NKRPRT, strip parity flag; value must be specified: 1 = strip parity 0 = do not strip parity } Actually, bit 7 of the character, whether it is parity or excess
NKISTAI	B06BITS	Input state program index; must be set
F3	BOOLEAN	NKNOXL, code translate flag 1 = translate 0 = do not translate
F4	BOOLEAN	NKSCENBL, change special character flag
NKBLKL	B012BITS	Character count 2. If nonzero, this sets the character count of the block and the initialization value.
NKISPTA	INTEGER	Pointer to input state program pointer table. If zero, the existing value is not changed. The original value is specified in NJTECT.
NKSCHR	CHAR	Special character, moved to MLCB if NKSCENBL flag is set.
NKCNT1	B08BITS	Character count 1. If nonzero, character count 1 and its initialization value are set to this value.
NKCXLTA	INTEGER	Code translation table address. If zero, the existing value is not changed. The original value is specified in NJTECT.

Figure 11-4. Input Command Format

this option is not used, the information is taken from the terminal class table (NJTECT). Format of the input command is shown in figure 11-4.

OUTPUT COMMAND (NKDOUT)

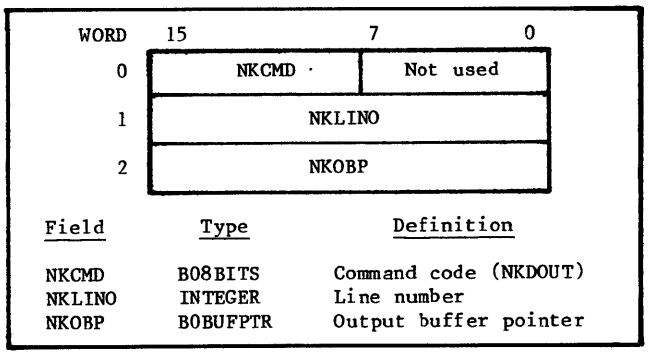
The output command permits output messages to be directed to a specified output line. Line, modem, and control functions, as defined in the line type tables, are generated by the subsystem as a function of the physical line requirements.

Output continues until the character specified by the last character displacement is transmitted. At that point, the subsystem chains to the next output buffer, if the chain address in the buffer is nonzero. Output stops if the chain address is zero or if the suppress chaining flag (BFSUPCHAIN) is set in the flag word of the first output buffer.

The subsystem generates an optional worklist entry for the user program for each data block output by the subsystem.

The terminate output or the disable command can also be used to terminate output processing functions on a specified line. Receipt of either command causes the subsystem to immediately cease all processing functions associated with the specified line.

The format of the output command is shown in figure 11-5.



Field	Type	Definition
NKCMD	B08BITS	Command code (NKDOUT)
NKLINO	INTEGER	Line number
NKOBP	BOBUFPTR	Output buffer pointer

Figure 11-5. Output Command Packet

Input After Output (NKINOUT)

This command permits interactive terminals (such as a display/keyboard combination) to be immediately ready to receive input data in response to a message displayed at the terminal. An index to the input state process table indicates the treatment of the returned data. The format for this command is shown in figure 11-6.

If the buffer output is the last data buffer of a transmission block, and line turnaround is required, (1) the subsystem generates the proper modem control signals to turn the line around, (2) monitors modem status for line turnaround, and (3) notifies the appropriate terminal-dependent subroutine that the line is ready for input. Modem signals and modem status analysis functions are specified by the line-type tables.

Terminate Input Command (NKENDIN)

This command enables the TIP to direct the multiplex subsystem to immediately stop input processing functions on the specified line. Buffers are optionally discarded. The TIP program can, by issuing an input command, direct the subsystem to resume input on the line. Transmission line characteristics are not altered by the terminate input command and, therefore, the TIP need not generate a control command. The format for the terminate input command is shown in figure 11-7.

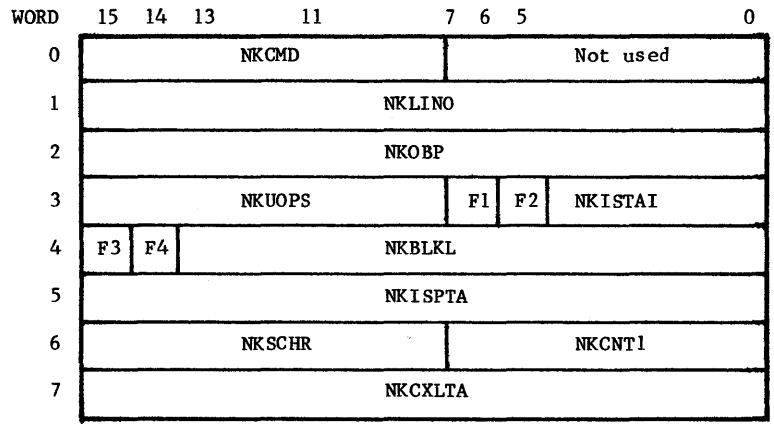
After processing the terminate input command, the subsystem optionally generates a worklist entry to the TIP as specified in the worklist and workcode.

Terminate Output Command (NKENDOUT)

This command enables the TIP to direct the multiplex subsystem to terminate output processing functions on the specified line immediately. After processing the terminate command, an optional worklist entry is generated to the TIP, using the specified worklist and workcode. This command is used when the TIP interrupts an outgoing message for a higher priority message, or when an abnormal line condition occurs. The format of the terminate output command is shown in figure 11-8.

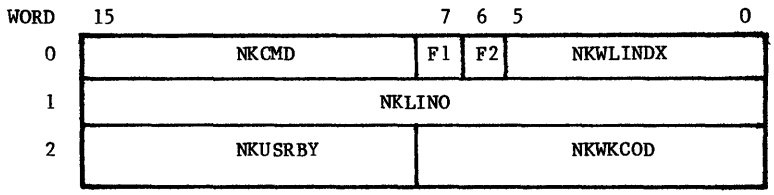
Disable Line Command (NKDISL)

The disable line command directs the multiplex subsystem to terminate all processing functions of the specified line. Modem control signals are generated to inhibit further exchange between the local modem and the communications line. The subsystem also releases all data structures defining the character processing functions for the line. To reactivate the functions, a control, initialize, and enable command, followed by either an input or output command, must be issued. The format for the disable line command is shown in figure 11-9.



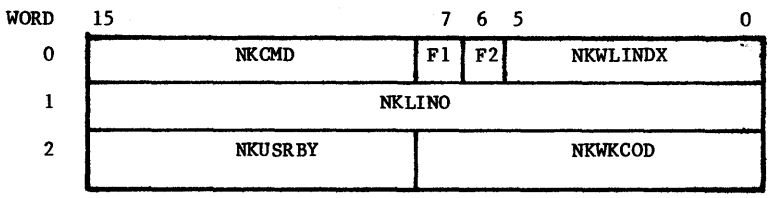
Field	Type	Description
NKCMD	B08BITS	Command code (NKINOUT)
NKLINO	INTEGER	Line number
NKOBP	B0BUFPTR	Output buffer pointer
NKUOPS	BOOLEAN	Eight user flags (NKUOP1 - NKUOP8). NKUOP1 is bit 15 in the MLCB user flag word; NKUOP8 is bit 8 in that word. NKUOPS is moved into MLCB if NKMVB is 1.
F1	BOOLEAN	NKMVB, move block of user flags into MLCB
F2	BOOLEAN	NKRPRT, strip parity flag; value must be specified: 1 = strip parity 0 = do not strip parity
NKISTAI	B06BITS	Input state program index; must be set
F3	BOOLEAN	NKNOXL, code translate flag 1 = translate 0 = do not translate
F4	BOOLEAN	NKSCENBL, change special character flag
NKBLKL	B012BITS	Character count 2. If nonzero, this sets the character count of the block and the initialization value.
NKISPTA	INTEGER	Pointer to input state pointer table. If zero, the existing value is not changed. The original value is specified in NJTECT.
NKSCHR	CHAR	Special character, moved into MLCB if NKSCENBL flag is set
NKCNT1	B08BITS	Character count 1 . If nonzero, the character count 1 and its initialization value are set to this value.
NKCXLTA	INTEGER	Code translation table address. If zero, The existing value is not changed. The original value is specified in NJTECT.

Figure 11-6. Input after Output Command Format



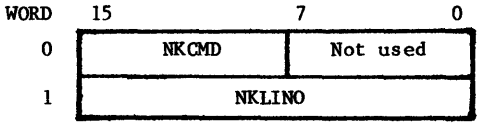
Field	Type	Description
NKCMD	B08BITS	Command code (NKENDIN)
F1	BOOLEAN	NKRELBFS, release buffer flag (release buffer if set)
F2	BOOLEAN	NKWKFL, send worklist to user (if set)
NKWLINDX	BOWKLSTS	Worklist index, used if NKWKFLG is set. The TIP uses one of the user-defined values BOVWL1 through BOVWL4.
NKLINO	INTEGER	Line number
NKUSRBY	B08BITS	User-supplied byte, returned in field MMWTCOUNT in worklist
NKWKCOD	NKWKCOD	User workcode in worklist (MMWKCOD). One of the values AOWK1 through AOWK31 is used.

Figure 11-7. Terminate Input Command Format



Field	Type	Description
NKCMD	B08BITS	Command code (NKENDOUT)
F1	BOOLEAN	NKRELBFS, releases buffer when flag is set; these are buffers specified in BZLBTOMUX
F2	BOOLEAN	NKWKFLG, sends worklist to user when set
NKWLINDX	BOWKLSTS	Worklist index; used if NKWKFLG is set. The TIP uses one of the user-defined values BOVWL1 theough BOVWL4.
NKLINO	INTEGER	Line number
NKUSRBY	B08BITS	User-supplied byte to be returned in field MMWTCOUNT in worklist
NKWKCOD	B08BITS	User workcode in worklist (MMWKCOD). One of the values AOWK1 through AOWK31 is used.

Figure 11-8. Terminate Output Command Format



Field	Type	Description
NKCMD	B08BITS	Command code (NKDISL)
NKLINO	INTEGER	Line Number

Figure 11-9. Disable Line Command Format

Virtual terminal format allows host application programs to expect only two types of input: ASCII input from a standardized interactive terminal (IVT), or ASCII input from a standardized batch terminal (BVT).

Each TIP is responsible for converting terminal code and format to and from ASCII virtual terminal formats. Downline, this is handled entirely by text processing state programs. If the TIP handles several types of terminals with dissimilar characteristics, the TIP will probably need one set of state programs to handle the conversions for each terminal type.

Since the techniques to format for IVT and BVT have little in common, the two types of terminals are discussed individually.

BATCH VIRTUAL TERMINAL (BVT)

Batch virtual terminal provides the standard interface which permits application programs in the host to exchange information with remote batch terminals, without regard to specific terminal characteristics.

BATCH VIRTUAL TERMINAL CHARACTERISTICS

The BVT is deemed to be a multi-device terminal operating remotely from the host. BVT is connected to the NPU by a synchronous modem using a medium speed or high speed line. (Although the protocol on the line may differ by equipment type, BVT is always assumed to be a block transfer terminal).

A separate logical connection exists for each device on a line. Device types that can exist at remote terminals include card readers, printers, plotters, and card punches.

BVT is capable of supporting data compression, printer carriage control, code conversion, transparent data, mode control, and file structure. For downline blocks, the host process assured that downline network blocks are not longer than the allowable block size for the terminal device, allowing space for TIP-inserted control information. The host application program must also assure that the width of an output print line does not exceed the device's printline width.

The host process is also responsible for compressing data. Downline, only blank and zero compression is allowed. Any other type of attempted compression will cause the block to be rejected by a BRK block from CCP. The terminal determines the degree of upline compression allowed. Full compression is assumed.

The BVT data is contained within MSG and BLK blocks; the CMD blocks are used to pass BVT command information. The standard remote batch facility (RBF) uses the BVT data format and defines the sequence of block types for transferring data and control. Since a user-defined batch TIP can be designed to accept data connections with RBF, the rules for RBF are discussed here.

SUMMARY OF RBF RULES

Control Console and Batch Devices

RBF assumes that a batch terminal consists of one or more batch devices (such as a card reader or a printer) and an interactive console. The interactive console provides any control commands necessary for RBF operation. The interactive console uses the IVT interface. It is possible to simulate all the necessary RBF control commands that are normally provided by an interactive console, by use of control functions entered through a card reader. The TIP must separate the control information from the batch card input and send the control as IVT data on the simulated console connection.

Batch Input

Batch input from a card reader should not be started until a start input command is received from RBF. This command assures that RBF will accept the data and no information will be lost. Card reading can begin and network blocks can be constructed by the TIP. Usually, one transmission block is created for BVT and sent to the host as one network block. The TIP does not perform data compression. If the BVT syntax provides the appropriate representation, the TIP is not required to perform data expansion. The TIP should discard all trailing blanks on each card.

If an end of information card (EOI) is detected in the upline transmission block, The TIP must fragment the data into one or more network blocks. The EOI card must be the last card of a MSG block. Subsequent EOI cards are discarded until a non-EOI card is detected. If such a non-EOI card is found, the remaining data is send to the host as the first block of the next message.

Whenever the card reader has run out of source data and has informed the TIP of this condition, the TIP must generate an input stopped command to follow the last network block. This informs RBF that the input device is not ready. The TIP must await a start input command from RBF before card reading can resume.

The input stopped command can also indicate a slipped card error. Either the slipped card or the end of source condition will cause RBF to generate a message to the batch terminal's console.

A stop input command may be sent downline to the TIP as a result of an RBF command. In this case, the TIP must stop reading the cards, and must also send an input stopped command upline to acknowledge that the requested action was performed. The stopped input command follows the last network block containing data.

Batch Output

The output data stream from RBF does not contain commands. A network output data block can be transformed

into a single transmission block and sent to the terminal. RBF must contain the necessary information so that it can create the proper sized and formatted network blocks.

If an output device is not ready to receive data, the block is undeliverable and the TIP replies to the host with a BRK block. The BRK block's reason code prompts RBF to create a message for the interactive console which controls this device. It is assumed that the operator will enter the proper control commands so that output can resume.

Output Block Size

The TIP writer has two choices for defining network block construction for the new TIP: (1) he can modify RBF by adding a new set of rules suited to this TIP's terminal buffer sizes, or (2) he can cause the TIP to fragment a network block for the device into two or more transmission blocks. If the latter choice is made and the TIP has to generate a BRK block because the device was not ready, the TIP is responsible for saving the unprinted fragments. Normally, the TIP would assure that these unprinted fragments are neither duplicated nor lost. However, if such delivery assurance is not required, the fragments can be discarded.

Line utilization is affected if the chosen fragment size is not the maximum size allowed by the terminal's buffers. It should be noted that the selection of block size is not a significant factor for low speed line utilization.

Line Folding and Compression

Line folding and compression of zeroes or blanks is performed by RBF. The TIP need not perform any further compression, but it is permitted to do so. Compresssion of characters in RBF can be suppressed if character compression is not supported by the terminal, or the TIP can perform character expansion prior to transmission.

STP and STRT Blocks

The STP block can be sent on a batch connection if the TIP needs to stop downline data or commands. The TIP must detect a restart condition and send a STRT block upline to indicate that the data stream has been restarted. This command sequence is especially useful when batch devices are physically dependent and cannot be used simultaneously.

BVT SYNTAX

Table 12-1 defines the syntax for BVT data and commands. The table is presented in a modified Bachus Naur format. The rules for this type of table are given on the first page of the table. Table 12-2 describes the print actions for the defined forms-control characters. All TIPs should support forms-control characters E0 through E4. All other values can be treated as preprint single space, if the printer does not support paper motion functions.

INTERACTIVE VIRTUAL TERMINAL

The interactive virtual terminal (IVT) is defined to provide common interface-to-host application programs for a variety of interactive terminal types. The TIP is responsible for converting the IVT interface to the terminal's real characteristics, and the converse. The output transform is usually simple, as the IVT interface is defined for display characters and is not extended to the various control capabilities of physical terminals. The interface requires that specific functions be performed both upline and downline. The IVT syntax is described in table 12-3. This is also presented in modified Bachus Naur format. The rules for this format are given in table 12-1.

DOWNLINE IVT TRANSFORM

The TIP receives three types of downline blocks that are associated with IVT processing: BLK, MSG, and CMD blocks. BLK and MSG blocks contain data to be sent to the terminal. CMD blocks contain control information for the TIP. Host application programs originate all three types of blocks, as required. A BLK block indicates that a MSG block must follow to complete the message for the terminal. One or more BLK blocks can precede the MSG block. A MSG block can contain all of the message or only the final block of the message.

Time of Transferring Downline Data

The TIP translates the data in a downline IVT block to a format and code set that can be recognized by the terminal. The TIP can be designed in either of two ways: (1) the TIP can immediately send the data downline to the terminal and send an acknowlegement upline to the host that the data was sent, or (2) the TIP can collect blocks and wait until the MSG block is received before sending the data to the terminal. When the TIP receives a message block, it must send the data, since the network cannot assure the TIP that more data will follow.

Downline data is normally to be translated into terminal format and code. However, it is possible for the host to send blocks of transparent data for the terminal. The TIP must send the transparent data to the terminal immediately; the TIP can neither fragment blocks nor collect blocks to send as a complete message.

Downline Data Format

Downline IVT data consists of one or more logical lines. A logical line consists of an optional format effector, a string of ASCII data or IVT control characters, and a terminator. Two conditions are possible for format effectors: either the format effectors are present in every logical line of a MSG or BLK block, or none of the lines contain format effectors. If a suppression bit is set in the data block clarifier (DBC), none of the lines have format effectors.

IVT Format Effectors

The host application program assumes that a given format effector always causes a consistant type of operation at each terminal. However, since some terminals inhibit certain operations, the TIP must provide default actions to cover these cases. Format effectors are described in table 12-4.

For each input operation from a terminal, the TIP positions the cursor to the first character position for the next line.

NOTE

In the following discussion, cursor positioning should be taken to mean either positioning a true cursor (display symbol) or positioning of the next character to be

TABLE 12-1. BVT SYNTAX

Following is a description of the metasymbols belonging to the BNF formalism used to describe IVT, BVT, block protocol, and commands.

Symbol	Meaning
::=	This metasymbol indicates that the syntactic construct to the left is defined as the string or alternation of strings to the right.
|	The vertical line demarcates alternatives.
<>	The angular brackets enclose syntactic constructs denoted by English phrases.
{ }	Braces enclose optional expressions.
*	An asterisk exponent indicates zero or more repetitions.
+	A plus exponent indicates one or more repetitions.
( )	Parentheses are used to factor alternatives or expressions formed from them.

<BVTBLKDATA>$_1$	::=	<UPLINEBVTBLK> | <DOWNLINEBVTBLK>
<BVTMSGDATA>$_1$	::=	<UPLINEBVTMSG> | DOWNLINEBVTMSG>
<UPLINEBVTBLK>	::=	<CARDREADERBLK>
<DOWNLINEBVTBLK>	::=	<PRINTBLK> | <PUNCHBLK> | <PLOTBLK>
<UPLINEBVTMSG>	::=	<CARDREADERMSG>
<DOWNLINEBVTMSG>	::=	<PRINTMSG> | <PUNCHMSG> | <PLOTMSG>
<CARDREADERBLK>	::=	<CARDREADERDBC> <UPLINECARD>*
<CARDREADERMSG>	::=	<CARDREADERDBC> <UPLINECARD> * <ENDOFINFORMATION>
<PRINTBLK>	::=	<PRINTDBC> <PRINTLINE>*
<PRINTMSG>	::=	<PRINTDBC> <PRINTLINE>* {<ENDOFINFORMATION>}
<PUNCHBLK>	::=	<PUNCHDBC> <DOWNLINECARD>*
<PUNCHMSG>$_2$	::=	<PUNCHDBC> {<DOWNLINECARD> * <ENDOFINFORMATION>}
<PLOTBLK>	::=	<PLOTDBC> PLOTLINE>*
<PLOTMSG>	::=	<PLOTDBC> <PLOTLINE> * {<ENDOFINFORMATION>}
<UPLINECARD>	::=	<CARDREADERDATA> | <ENDOFRECORD>
<PRINTLINE>	::=	<PRINTDATA> | <ENDOFRECORD>
<DOWNLINECARD>	::=	<PUNCHDATA> | <ENDOFRECORD>
<PLOTLINE>	::=	<PLOTDATA> | <ENDOFRECORD>
<CARDREADERDBC>	::=	X'00
<PRINTDBC>	::=	X'00
<PLOTDBC>	::=	X'00
<PUNCHDBC>	::=	<SPARE> <SPARE> <SPARE> <SPARE> <SPARE> <SPARE> <BANNERCARD> <SPARE>

TABLE 12-1. BVT SYNTAX (Contd)

<SPARE>$_3$	::=	0
<BANNERCARD>$_3$	::=	<PUNCHBANNERCARD> | <DONTPUNCHBANNERCARD>
<PUNCHBANNERCARD>$_3$	::=	1
<DONTPUNCHBANNERCARD>$_3$	::=	0
<CARDREADERDATA>	::=	{<MODECHANGE>} <UPLINECOMPRESSEDDATA>* <ENDOFMEDIA>
<PRINTDATA>	::=	{<MODECHANGE>} <FORMSCONTROL> <DOWNLINEDOMPRESSEDDATA>* <ENDOFMEDIA>
<PUNCHDATA>	::=	{<MODECHANGE>} <DOWNLINECOMPRESSEDDATA>* <ENDOFMEDIA>
<PLOTDATA>	::=	{<MODECHANGE>} <DOWNLINECOMPRESSEDDATA>* <ENDOFMEDIA>
<UPLINECOMPRESSEDDATA>	::=	<COMPRESSEDZEROS> | <COMPRESSEDBLANKS> | <REPLICATIONCOUNT> <BYTE> | <STRINGINDICATOR> <INDETERMINATELENGTHSTRING> | <STRINGLENGTH> <FIXEDLENGTHSTRING>
<DOWNLINECOMPRESSEDDATA>	::=	<COMPRESSEDZEROS> | <COMPRESSEDBLANKS> | <STRINGLENGTH> <FIXEDLENGTHSTRING>
<FORMSCONTROL>$_4$	::=	<BVTESCAPE> (X'E0 | X'E1 | ... | X'FE)
<COMPRESSEDZEROS>$_5$	::=	<BVTESCAPE> (X'32 | X'33 | ... | X'3F)
<COMPRESSEDBLANKS>$_6$	::=	<BVTESCAPE> (X'12 | X'13 | ... | X'2F)
<REPLICATIONCOUNT>$_7$	::=	<BVTESCAPE> (X'42 | X'43 | ... | X'8F)
<BYTE>	::=	X00 | X'01 | ... | X'FF
<BVTESCAPE>	::=	X'FF
<MODECHANGE>$_8$	::=	<ASCII029> | <ASCII026>
<ASCII029>	::=	<BVTESCAPE> X'00
<ASCII026>	::=	<BVTESCAPE> X'03
<STRINGINDICATOR>$_9$	::=	<BVTESCAPE> X'90
<INDETERMINATELENGTHSTRING>	::=	$\overset{+}{<\text{STRINGSEQUENCE}>}$ 10
<STRINGSEQUENCE>	::=	<DATAFF> | <STRINGBYTE>
<DATAFF>	::=	<BVTESCAPE> X'FF
<STRINGBYTE>	::=	X'00 X'01 ... X'FE
<STRINGLENGTH>$_{11}$	::=	<BVTESCAPE> (X'91 X'92 ... X'CF)
<FIXEDLENGTHSTRING>	::=	$\overset{n}{<\text{BYTE}>}$ 12
<ENDOFMEDIA>$_{13}$	::=	<BVTESCAPE> X'0A
<ENDOFRECORD>$_{14}$	::=	<BVTESCAPE> X'0B <LEVELNUMBER> <LEVELNUMBER> <ENDOFMEDIA>
<LEVELNUMBER>	::=	<BYTE>

TABLE 12-1. BVT SYNTAX (Contd)

<ENDOFINFORMATION>15	::=	<BVTESCAPE> X'0C <ENDOFMEDIA>
<CMDBLKDATA>	::=	<STOP INPUT> | <STARTINPUT> | <INPUTSTOPPED>
<STOPINPUT>	::=	<CTRLPFC> <STOPINPUTSFC>
<STARTINPUT>	::=	<CTRLPFC> <STARTINPUTSFC>
<INPUTSTOPPED>	::=	<CTRLPFC <INPUTSTOPPEDSFC> <INPUTSTOPPEDREASONCODE>
<CTRLPFC>	::=	X'C1
<STOPINPUTSFC>	::=	5
<STARTINPUTSFC>	::=	6
<INPUTSTOPPEDSFC>	::=	7
<INPUTSTOPPEDREASONCODE>	::=	<STOPINPUTRESPONSE> | <INPUTDEVICENOTREADY>| <CARDSLIPERROR> | <EOIINPUT>
<STOPINPUTRESPONSE>	::=	0
<INPUTDEVICENOTREADY>	::=	1
<CARDSLIPERROR>	::=	2
<EOIINPUT>	::=	3

NOTES:

1. <BVTBLKDATA> and <BUTMSGDATA> represent the data portion of a network data block, i.e., that portion of the block starting with the DBC.

2. <ENDOFINFORMATION> should be required in a <PUNCHMSG> but the R4 syntax allows a <PUNCHMSG> consisting of <PUNCHDBC> only.

3. These syntactic constructs represent a single bit.

4. <FORMSCONTROL> which are not supported by a specific device result in a preprint single space.

5. <COMPRESSEDZEROS> is used to represent 2 through 15 zeros.

6. <COMPRESSEDBLANKS> is used to represent 2 through 31 blanks.

7. <REPLICATIONCOUNT> is the number of times (2 through 79) the following character is to be repeated.

8. Each device type supported by the BVT is assigned a data mode which, in most cases, is unchangeable. However, downline data to a card punch may contain a <MODE CHANGE> requesting the TIP to perform the appropriate code translation to generate the desired punched cards. The mode selected stays in effect until the next <MODE CHANGE> or an <ENDOFINFORMATION>, which causes the data mode to be returned to the default for the device. For all other downline data and all upline data, <MODE CHANGE> is ignored.

   <ASCII-029> indicates that the data should be interpreted as ASCII, but that only the 64 character subset will appear. The data will be translated by the TIP to produce 029 cards. Similarly, <ASCII-026> will produce 026 cards.

9. <STRINGINDICATOR> indicates a character string of 1 to 80 characters following which is terminated by the first non-data FF.

10. A <STRINGSEQUENCE> is restricted to 1 to 80 characters.

11. <STRINGLENGTH> indicates a character string following of 1 to 63 bytes.

TABLE 12-1. BVT SYNTAX (Contd)

12. n represents repetition from 1 to 63 times based on the value of <STRINGLENGTH>.

13. <ENDOFMEDIA> not preceded by data causes a blank record to be generated.

14. <ENDOFRECORD> causes a 789 card to be punched for card punch. It is ignored for printer or plotter. Upline the level numbers, represent the contents of columns 2 and 3 of 789 card and columns 6 and 7 of a /*EOR card.

15. <ENDOFINFORMATION> is ignored in print and plot MSG blocks. It causes a /*EOI card to be generated for a HASP card punch.

TABLE 12-2. FORMSCONTROL VALUES FOR BVT BLOCKS

<FORMSCONTROL> (HEX)	Action Before Printing	Action After Printing
E0 (1)	Space 1	No Space
E1 (1)	Space 2	No Space
E2 (1)	Space 3	No Space
E3 (1)	Suppress Space	No Space
E4 (1)	Skip to Channel 1 (2)	No Space
E5	Skip to Channel 12 (3)	No Space
E6	Skip to Channel 6	No Space
E7	Skip to Channel 5	No Space
E8	Skip to Channel 4	No Space
E9	Skip to Channel 3	No Space
EA	Skip to Channel 2	No Space
EB	Skip to Channel 11	No Space
EC	Skip to Channel 7	No Space
ED	Skip to Channel 8	No Space
EE	Skip to Channel 9	No Space
EF	Skip to Channel 10	No Space
F0	No Space	Skip to Channel 1 (2)
F1	No Space	Skip to Channel 12 (3)
F2	No Space	Skip to Channel 6
F3	No Space	Skip to Channel 5
F4	No Space	Skip to Channel 4
F5	No Space	Skip to Channel 3
F6	No Space	Skip to Channel 2
F7	No Space	Skip to Channel 11
F8	No Space	Skip to Channel 7
F9	No Space	Skip to Channel 8
FA	No Space	Skip to Channel 9
FB	No Space	Skip to Channel 10
FC-FE	Reserved	

Notes:

1. Supported on all devices
2. Page eject
3. Bottom of page

TABLE 12-3. IVT SYNTAX

<DOWNLINECONSOLEBLK>	::=	<DOWNLINECONSOLEDBC> <DOWNLINECONSOLELINES>
<UPLINECONSOLEBLK>	::=	<UPLINECONSOLEDBC> <UPLINE FRAGMENT>
<DOWNLINECONSOLEMSG>	::=	<DOWNLINECONSOLEDBC> <DOWNLINECONSOLELINES>
<UPLINECONSOLEMSG>	::=	<UPLINECONSOLEDBC> <UPLINEFRAGMENTEND>
<UPLINEFRAGMENT>	::=	<TRANSPARENTFRAGMENT> | IVTFRAGMENT>
<TRANSPARENTFRAGMENT>	::=	<BYTE>*
<IVTFRAGMENT>	::=	<UPLINEIVTBYTE>*
<UPLINEFRAGMENTEND>	::=	<TRANSPARENTFRAGMENT> | <IVTFRAGMENT>
<DOWNLINECONSOLELINES>	::=	<TRANSPARENTLINE> | <DOWNLINEIVTLINES>
<TRANSPARENTLINE>	::=	<BYTE>*
<UPLINEIVTLINE>	::=	<UPLINEIVTBYTE>*
<DOWNLINEIVTLINES>	::=	(<FORMATEFFECTOR> <DOWNLINEIVTBYTE> * <US>)*| (<DOWNLINEIVTBYTE> * <US>)*
<DOWNLINEIVTBYTE>$_5$	::=	X'00 | X'01 | ... X'09 | X'0B | X'0C | X'0E | ... | X'1E | X'20 | ... | X'7E | X'7F | <CR> | <LF>
<US>	::=	X'1F
<CR>	::=	X'0D
<LF>	::=	X'0A
<UPLINEIVTBYTE>$_5$	::=	X'00 | X'01 | ... | X'7E | X'7F
<BYTE>	::=	X'00 | X'01 | ... | X'7E | X'FF
<FORMATEFFECTOR>$_2$	::=	<BYTE>
<UPLINECONSOLEDBC>	::=	<SPARE> <SPARE> <SPARE> <NOTUSED> <NOTUSED> <TRANSPARENT> <CANCELLED> <PARITYERROR>
<SPARE>	::=	0
<NOTUSED>	::=	0
<TRANSPARENT>$_3$	::=	<DATAISNOTTRANSPARENT> | <DATAISTRANSPARENT>
<CANCELLED>	::=	<PREVIOUSDATAISNOTCANCELLED> | <PREVIOUSDATAISCANCELLED>
<PARITYERROR>	::=	<DATARECEIVEDWITHOUTERROR> | <DATARECEIVEDWITHPARITYERROR>
<DATAISNOTTRANSPARENT>	::=	0
<DATAISTRANSPARENT>	::=	1
<PREVIOUSDATAISNOTCANCELLED>	::=	0
<PREVIOUSDATAISCANCELLED>	::=	1
<DATARECEIVEDWITHOUTERROR>	::=	0
<DATARECEIVEDWITHPARITYERROR>	::=	1
<DOWNLINECONSOLEDBC>	::=	<SPARE> <SPARE> <SPARE> <SPARE> <FORMATEFFECTORUSAGE> <TRANSPARENT> <NOTUSED> <AUTOINPUT>

TABLE 12-3. IVT SYNTAX (Contd)

<FORMATEFFECTORUSAGE>	::=	<FORMATEFFECTORSUSED> | <FORMATEFFECTORSNOTUSED>
<AUTOINPUT>	::=	<OUTPUTISAUTOINPUT> | <OUTPUTISNOTAUTOINPUT>
<FORMATEFFECTORISUSED>	::=	0
<FORMATEFFECTORNOTUSED>	::=	1
<OUTPUTISAUTOINPUT>16	::=	1
<OUTPUTISNOTAUTOINPUT>	::=	0
<CONSOLECMDDATA>	::=	<DOWNLINECONSOLECMD> | <UPLINECONSOLECMD>
<DOWNLINECONSOLECMD>	::=	<STOPINPUT> | <STARTINPUT> | <IVTCOMMAND>
<UPLINECONSOLECMD>	::=	<INPUTSTOPPED>
<STOPINPUT>	::=	<CTRLPFC> <STOPINPUTSFC>
<STARTINPUT>	::=	<CTRLPFC> <STARTINPUTSFC>
<IVTCOMMAND>	::=	<CTRLPFC> <IVTCOMMAND> <IVTCOMMANDTEXT>4
<INPUTSTOPPED>	::=	<CTRLPFC> <INPUTSTOPPEDSFC> <INPUTSTOPPEDREASONCODE>
<CTRLPFC>	::=	X'C1
<STOPINPUTSFC>	::=	X'05
<STARTINPUTSFC>	::=	X'06
<TERMINALDEFINITIONSFC>	::=	X'04
<INPUTSTOPPEDSFC>	::=	X'07
<INPUTSTOPPEDREASONCODE>	::=	<STOPINPUTRESPONSE>
<STOPINPUTRESPONSE>	::=	X'00

NOTES:

1. See table 12-4 for a definition of the format effectors.

2. All data in the upline block marked with the DBC flag is considered to be transparent data. For downline data, the transparent flag takes precedence over the format effector selection.

3. The actual text of (IVTCOMMANDTEXT) is the responsibility of PTIVTCMD; it is not the TIP's responsibility.

4. IVT bytes are ASCII characters, both upline an downline. The parity bit is ignored by the TIP for downline data. Upline, the parity bit must always be zero.

5. A network block specifying autoinput in the DBC may contain only one logical line. It must be a MSG block.

TABLE 12-4. FORMAT EFFECTORS

FE	Preprint Action	Postprint Action
space	single space	no space
0	double space	no space
-	triple space	no space
+	position to start of current line	no space
*	position to top of form or home cursor	no space
1	home cursor and clear (page eject)	no space
,	no space	no space
.	no space	single space
/	no space	position to start of current line

written on the platen of a typewriter. The latter naturally includes platen (vertical) movement as well as head (horizontal) movement.

If output follows input, the TIP processes format effectors to account for this repositioning. For example, a single space preprint format effector is treated as a null operation when that format effects immediately follows an input operation.

Following output of a logical line, the cursor is left where the data line terminated. Any subsequent output must therefore process the format effector as specified; that is, a single space preprint format effector causes the cursor to move to the first character position of the next line. Then the additional data can be transmitted.

Some terminals cause special difficulties, since the cursor is not left at the end of the data line but is moved automatically to some other position. For example, on a Mode 4 display unit, the protocol requires an end of text symbol at the end of the logical line. The terminal displays this EOT symbol. However, this displaces the cursor to one character position past the last user character location. Following each output operation on such a terminal, the TIP can return the cursor to the first character position of the following line. Subsequent format effectors must account for the post output repositioning.

The TIP is required to display the data in the same way whether the network block was composed of a single logical line or of many logical lines. If the TIP repositions the cursor following each output, the TIP must also reposition the cursor at the end of each logical line.

Whenever the TIP cannot perform the specific action directed by a format effector, the TIP must perform the same action as for a preprint single space. Also, the TIP must supply any idle characters that are needed to allow the terminal to perform a physical repositioning action. These idle characters, of course, follow the format effector. The rule for inserting idle characters is given below in the explanation of adding idle characters for line feeds and carriage returns.

The TIP does not supply preprint or postprint cursor motion for downline IVT data which has no embedded format effectors. If the cursor cannot be left in the position following the last user character, the TIP returns the cursor to the first character of the next line, for every output physical line.

IVT Character Processing

An IVT data line can use any character in the standard 128 character ASCII code set (see appendix A of the CCP 3 Reference Manual). The TIP must be able to translate any of these characters to an equivalent terminal code character.

There is one IVT code for a carriage return (CR) and another IVT code for a line feed (LF). When the TIP recognizes either of these codes it must be able to do two things: (1) issue a command to position the cursor to the proper location, and (2) issue characters that will allow the terminal sufficient time to position the cursor. The TIP provides this additional time by inserting idle characters following the carriage return or line feed. Then the TIP may resume outputting data characters.

The idle character is defined by the terminal: it may be any non-displayed character that does not affect platen motion or paper movement, and does not have any other terminal function. The number of idle characters to be inserted is determined by the IVT commands.

The sequences of CR LF or LF CR can be replaced by the new line function, if the terminal supports that function. If separate LF or CR functions are not supported by the terminal, the TIP can interpret the CR as a new line request and can discard all LF codes.

Autoinput

If the autoinput flag is set in the data block clarifier, the TIP must save the first 20 characters of the downline source data. The format effector and the unit separator must be discarded, as well as any characters following the twentieth. An autoinput block must be a message block,

and must contain a single logical line. When in autoinput mode, the TIP outputs the logical line, saves the first twenty characters, and waits for an input line. This input line is appended to the saved output characters, and the entire new line of data is returned to the application program in the host. It is possible to autoinput transparent data.

Note that input data is required following this downline autoinput block. Note, also, that the entire output is delivered to the terminal, even if page wait logic would normally require fragmentation.

Paging

When the output data appears on a hard copy device such as a printer or a teletype, the TIP must provide paging for the data. The TIP counts input and output lines and inserts six new line requests to provide page-to-page spacing. This spacing is intended to provide a fixed length for each page of a message. When a format effector selects a new page, a sufficient number of new line requests are inserted to position the page to the bottom of the page. Then the six new line requests are generated.

If the printer supports a top-of-form or a page-eject function, that device feature should be used instead of new line requests.

The TIP's interpretation of format effectors must allow for platen positioning caused by paging logic. For example, a preprint format effector which occurs at the top-of-form can be ignored. Also, if a format effector causes multiple lines that would span a page boundary, the TIP treats the format effector as a top-of-form.

The number of lines on a page is selectable by the IVT command %PL=nn. When the page length is nonzero and the output device is a hardcopy printer, paging is required. Page length on a display specifies the number of lines on the screen.

Page Wait

The IVT command %PG = Y selects the page-wait feature for display devices. This feature requires the TIP to construct and deliver one, and only one, full page of data to the terminal. In page-wait mode, the TIP must wait for input data after sending each full page of data. The TIP can construct a full page of data in either of two ways: (1) it can gather enough data to make a full page of data within the TIP, and send this data as a single transmission to the terminal, or (2) it can send multiple outputs to the terminal, so that the terminal constructs the page on the output device.

A page is considered full whenever the TIP has transmitted PL - 1 lines. Note that the last line of the screen is reserved. This line is used by the terminal operator to enter the awaited input (page turn indication) so that more output can be sent later. A page is also considered to be full if the format effector requests a top-of-form or a page-eject. In such a case, the TIP must indicate that more data is to follow, even though the current page does not appear to be full. The TIP should send the message OVER so that it appears on the line following the last line of the output text. This alerts the operator that more user data will follow after the operator enters the page turn signal. If the page length is zero, only a top-of-form or page-eject format effector will cause a page-wait condition.

The following things must be considered when calculating a full page:

- IVT control codes CR and LF
- The page width

The IVT command %PW = nn specifies the page width and indicates the physical boundary of a line. For hardcopy devices, the TIP must insert the proper new line codes to return the platen to the first character position of the next line. For display devices, page width specifies the column at which the terminal automatically inserts a new line; the TIP does not need to insert any codes.

The TIP is not responsible for suppression of double spacing if the character following an automatic new line (or an inserted new line) is an IVT control code or the end of a logical line.

The page width specification may cause line folding. The effect of line folding must be considered in calculating both paging and page wait. If a folded line would appear on the last line of a page (or both there and on subsequent lines), the line is fragmented. The second fragment of the line is saved; that fragment appears on the next page.

For transparent output, page wait occurs following each message block.

Terminal Control Codes

Downline data can contain ASCII control codes or escape sequences which cause actions to be performed by the terminal. The TIP writer has two choices for handling these: (1) the TIP may exclude the codes by substituting a blank for each such code, or (2) the TIP may transmit the codes unchanged. In the latter case, the TIP can elect to account for the cursor motion caused by the codes, or it can ignore the cursor motion and all its side effects. However, the TIP writer should note that such cursor motion can affect both paging and page wait. In any case, the TIP writer must assure that any such codes passed downline do not cause a terminal failure, result in error reports, or disconnect the user.

Data Block Clarifier

The data block clarifier (DBC) is used to indicate the format of the data. The format should remain consistant throughout a complete message. A message fragmented into a sequence of BLK blocks and a MSG block should all have the same value for the fragment or format effector flag. From message to message, however, the format may be changed as the application program desires.

CMD Blocks

Command (CMD) blocks are defined both upline and downline. Only one of them is required by a user TIP; that is the downline CMD block containing an IVT command. Other CMD blocks are used by standard TIPS and by CDC application programs. These CMD blocks are a specially defined use and do not occur in any other circumstances. It is possible that a user TIP and a user application program could define such a special use of a CMD block.

UPLINE IVT TRANSFORMS

The upline IVT transform converts terminal data to ASCII data. Each sequence of characters is accumulated and

transferred to the host as a single message. The input sequence begins with the NPU detecting a character on the line, and the sequence ends with the NPU detecting a terminating character or sequence of characters. After the data is converted to IVT format, the data is checked for IVT functions before it is sent upline.

The upline IVT transform discards any leading or trailing characters that are automatically supplied by the terminal, or required by the terminal's protocol. The terminating character is always sent by the terminal for its carriage return function. For block mode terminals, the terminating character can be any key which releases the data for transmission (such as an ETX). This terminating character or character sequence is always discarded.

Backspace Processing

If the terminal sends a backspace code in the data stream, the TIP's input state programs usually process the character. Backspace can be used only to correct characters in the same physical input line; that is, the user cannot backspace across a carriage return, line feed, or page-width boundary.

Data Fragmenting

The TIP can fragment upline data into multiple network blocks. All such fragments except the last are sent upline as BLK blocks. The last fragment is sent as a MSG block. The TIP can use any reasonable fragmentation rule, but the following factors should be considered:

- Fragments should be at least 150 characters in length. A smaller fragment unduly raises the overhead cost of network block handling.
- Fragmentation should be avoided unless characters are being accumulated in the NPU memory at typing speeds.
- Fragments must not be released for upline transmission unless the backspace cannot alter the data in the fragment.
- Fragmentation performed at the input state level can be reduced by concatenating blocks at the OPS level before passing the blocks upline to the host.

Upline IVT Processing

Before sending data to the host, the TIP must check the following IVT functions in the order listed (all checks are made after processing the data for a possible backspacing operation):

- Page turn prompt
- Abort input line
- Cancel input
- IVT command
- User break 1 or user break 2
- Autoinput

Page Turn Prompt

If the page wait feature is selected, the TIP must check for a page turn prompt. The prompt has two forms: (1) a message containing no data, or (2) a message containing only the IVT control code. The null line is treated as a prompt only if there is additional output available for transmission to the terminal. In this case, the prompt is discarded and the TIP resumes sending data to the terminal. If output is not available, a null message is sent to the host. If autoinput is being held, a null line satisfies the input, if it is not also a page turn prompt.

Abort Line

If output is interrupted by a break key or a full duplex input, the current output line is discarded, if the next input is an abort line. If any character except the abort output character is found on the input line, output is resumed starting from the beginning of the line. If a logical line consists of several physical lines, only the interrupted physical line is discarded.

Cancel Line

The last or only character of a line is checked to find if it is a cancel character. If it is, the message is discarded and the TIP sends a *DEL* message to the terminal. If an IVT command was being accumulated when cancelled, the command is discarded. If data was being accumulated and previous fragments have already been sent to the host, the TIP sends a null MSG block upline with the cancel flag set in the data block clarifier.

If a cancel operation occurs during autoinput, the TIP discards any autoinput data it is saving. Note that the TIP continues to wait for input to complete the autoinput operation. Cancelling the autoinput data does not satisfy the requirement for input.

IVT Command

If the first character of a message is the IVT control character, the TIP calls PTIVTCMD to process the command which follows. The TIP accumulates the entire command before sending it to PTIVTCMD; that is, all fragments must be reassemebled, and none of the fragments are sent to the host. If the command exceeds 150 characters in length, the remaining characters are discarded.

If any of the IVT commands that are implemented require functional changes of the hardware (such as a communications line adapater command), the TIP must send all possible functions to the hardware, following each IVT command. The TIP does not parse or act on the IVT command. The TIP merely senses that a command occurred, and reinitializes all the hardware functions.

PTIVTCMD determines the correct action to be taken for each IVT command, by reference to parameters in the field descriptor and field action tables. It is preferrable to accept all IVT commands, but to perform an action only for those which are appropriate for the type of terminal. The IVT commands are listed and described in the CCP System Programmer's Reference Manual.

User Breaks

The user break code must be the first and only character of a message. If a user break is found in a message, the data is discarded and the TIP generates a BRK block which is sent to the host. All internally held downline data (including autoinput data) is discarded.

Autoinput

If the TIP is holding autoinput when input arrives, the input data is chained to the autoinput data. The result is sent to the host as one input message. After autoinput has been sent to the host, the next output can be sent to the terminal. Autoinput data can be cancelled by the cancel character; however, the TIP must still wait for upline data before permitting the next output transmission.

Transparent Input Data

An IVT command (either from the terminal or sent downline from the host) can cause the TIP to process input data in transparent mode. The command applies to the next input message. Successful receipt of input data cancels transparent mode, so a new command is required for each line that is to be processed as transparent data. The TIP must inspect the BSXPARENT field of the TCB prior to each input operation to determine the processing mode. After sending the input line to the host, the TIP must set the BSXPARENT flag to FALSE. During transparent mode processing, the TIP checks for the page turn prompt. Autoinput is permissible in transparent mode. Only a null line input with data available for output is considered to be a valid page turn prompt in tranparent mode; the IVT control code is not considered a page prompt.

CCP provides a text processor for processing the various character translations that are necessary when network data is converted to terminal data. The network data is in interactive virtual terminal (IVT) format, in batch virtual terminal (BVT) format, or is transparent data. Terminal data is normally in a different format. The usual principal differences are as follows:

- A different code set than the network's
- A terminal transmission buffer size different than the network block sizes
- Terminal operating characteristics that are not identical to IVT or BVT

Text processing is usually required downline. A TIP can be written to perform upline text processing also.

TEXT PROCESSING STATE PROGRAMS

The TIP is responsible for making the translations, using a specialized language for character processing. This language contains most of the same instructions that are available to the input state programs for upline character processing . The instructions are described in detail in the State Programming Language Reference Manual. A few state programming instructions are available only for text processing, and cannot be used by the input state programs. All instructions are written into firmware level text processing state programs, and are interpreted by the text processor, when the TIP calls that firmware module via PTTPINF.

When the TIP is ready to translate a source message, the TIP sets up a control table called the Text Processing Control Block (TPCB), which is an overlay of the Mux LCB (NCLCB). This TPCB contains the pointers to the source data buffers, the instructions to be interpreted (that is, the address of a pointer table to the TIP's firmware level text processing state programs), and other TIP-defined fields. The TPCB was discussed in section 6, principal data structures. When the table is ready, the TIP makes a direct call to PTTPINF. This transfers control to the text processor, which controls the TIP's own text processing state programs.

The principal characteristics of executing PTTPINF are:

Procedure name:	PTTPINF
Input parameter:	(TPCBptr) = address of the TPCB
Return parameter:	(errcode) = text processing failure indicator
Calling Sequence:	PTTPINF(TPCBPTR);

Example of a call:

```
WITH BITCB .BSTCB, (TPCBptr) DO     SPECIFY TPCB
                                    ADDRESS
BEGIN
    NCaaa : =  value;
        .
        .
        .                           SET UP ALL
                                    THE NECESSARY
                                    TPCB FIELDS
    NCzzz : =  value;
    PTTPINF(TPCBptr);
END;
```

The text processor supplies destination buffers to receive the translated characters, as necessary. The text processing state programs handle each source character and determine its disposition. In general, the translated characters will be sequentially placed in the destination buffers. On return to the TIP, the TPCB holds pointers to the source data buffers and to the destination buffers.

The text processor is normally called for downline data. In all cases, the text processor is called only from the OPS-level TIP. Text processing translations are performed at a lower priority level than input state program processing, so input character translations can interrupt text processing translations. However, no other OPS-level program can gain control of the NPU while a text processing operation is in progress. Therefore, there is no need for reentrant coding in the TIP's firmware-level text processing state programs.

DOWNLINE TEXT PROCESSING FOR LOW AND MEDIUM-SPEED LINES

For TIPs which operate on lines with baud rates at or below 9600, text processing is done on demand; that is, one block of data is removed from the TIP's output queue and a sufficient amount of that data is translated and given to the multiplex subsystem for transmission. On completion of the transmission, the TIP translates more output data and passes it to the multiplex susbsystem. This process continues until there is no more data to be transmitted. In demand mode, there is never any output on the line during a text processing operation.

DOWNLINE TEXT PROCESSING FOR HIGH-SPEED LINES

For a TIP to efficiently utilize a line that operates at higher baud rates, text processing and output transmission are performed simultaneously. To accomplish this, the TIP must remove enough data from the output queue to create two transmission blocks. As soon as this amount of data is text processed, the first transmission block is

passed to the multiplex subsystem for sending. As soon as the first block is successfully sent, the TIP passes the second block to the multiplex subsystem. Then the TIP immediately gets another block from its output queue and starts text processing that block into another transmission block.

Except for starting a message, the TIP must not remove more than one block from its output queue without sending an acknowledgment to the host (the TIP has the choice of delaying the acknowledgement for fragmented blocks). In the special case where the first two transmission blocks must be translated before either of them is sent, the TIP may remove more than one network block from its output queue. The BIP routine PTBACK can be called to acknowledge to the host that the first block was unqueued. This allows the second block to be unqueued.

If PTBACK is used in this manner the postoutput POI (PBPOPOI) must be used only for accounting purposes, and not for control. To do this, the TIP sets the internal block flag (BFINTBLK) to inhibit PBPROPOI from transmitting another BACK block upline. If the TIP is to accomodate the usual application program delivery assurance requirements, the TIP must itself preserve the data acknowledged but not sent to the terminal, even though a BRK or STP block is sent upline. The application program will not resend those blocks which were acknowledged by the TIP.

UPLINE TEXT PROCESSING

Text processing may also be performed on upline data. In most cases, the TIP's input state programs will have already translated the data during input processing, as the characters were being demultiplexed from the circular input buffer. If a complete data translation cannot be performed at this time, the TIP must supply upline text processing programs to finish the text translation. The input data buffers supplied by the multiplex subsystem become the source buffers for this upline text processing operation. The call to PTTPINF is the same as for downline text processing. When text procesing is complete, the TIP calls the postinput POI to send the blocks upline.

Note that two-pass text processing has a considerable effect on NPU utilization, and should be avoided if at all possible.

There are two ways to initialize PASCAL variables for the TIP: by using VALUE statements, or by using executable initialization code. Use of VALUE statements is limited by restrictions on the syntax. It also causes hard-coded numeric fields to be used, instead of the symbolic names which are more appropriate for PASCAL. Initialization by executable code requires that the TIP maintain code that is used only during system initialization.

To resolve both these problems, an editing capability was created for use during the build process. This editing process is called MPEDIT. MPEDIT allows value initialization using symbolic names, and it provides value initializations of addresses which are known only after a system is linked and absolutized.

MPEDIT DIRECTIVES

MPEDIT directives are used to initialize variables and fields for PASCAL globals. If the user is not thoroughly familiar with MPEDIT directives, he is urged to review the edit phase section of the CYBER Cross System Link Editor and Library Maintenance Reference Manual. The insertion of MPEDIT decks is shown in section 4 of this manual.

The codes described below may be necessary for modifying a TIP or for writing a new TIP. The MPEDIT program structure is shown in figure 14-1. A / preceding a symbolic name indicates a symbol defined only to MPEDIT. Symbols not preceded by a / are defined for PASCAL or assembly language or MPLINK.

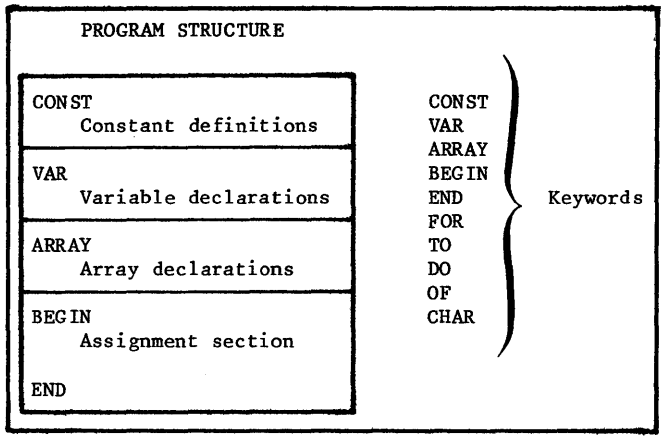


Figure 14-1. MPEDIT Program Structure

MPEDIT CONSTANTS

Constant definitions are added by the TIP writer to create symbolic names which are subsequently used by MPEDIT statements during value initialization. The constants are usually default values for:

- Cancel character
- IVT control character
- User break 1 character
- User break 2 character
- Transparent delimiter character
- Backspace character
- Character length in bits

Constants are defined by:

/CONSTNAME = nr;

where CONSTNAME is the TIP's local mnemonic. nr is the value assigned to that mnemonic. ; ends the constant definition.

Constant definitions are contained in the user supplied COMDECK ZCNtip.

MPEDIT VARIABLES

MPEDIT variables define symbolic names which are defined only to MPEDIT, and are used for numeric calculations. Probably the user neither needs to add to, nor change, any of the standard CCP edit phase variables. If it becomes necessary to add a variable, the format of the variable declaration is:

/VARNAME;

where VARNAME is the variable name. ; ends the variable declaration.

Variables /I, /J, /K, and /L are available for TIP use without special definition. User-defined variables are contained in the user-supplied COMDECK VARtip.

MPEDIT ARRAYS

To initialize values in arrays, the TIP writer must define for MPEDIT both an index range and a length of element (these parameters are not placed in the symbol table by PASCAL). Array declarations have the format:

ARRAYNAME [RANGE] OF SIZE;

where ARRAYNAME is the symbolic name of the array declared in the PASCAL global variable declaration section. ARRAYNAME may be any entry point name which allows initialization of arrays in assembly language programs. Range is a subrange declaration of the form:

M..N

where M and N are numeric constants or symbols. SIZE is the number of words in an array entry. ; ends the variable declaration.

Array declarations are contained in the user COMDECK ZARtip.

ASSIGNMENT SECTION

Each TIP has a section of code in the MPEDIT assignment section. The assignment statements most likely to be required are as follows:

- LCB descriptor table
- TCB descriptor table
- Worklist control block
- TIP type table
- Terminal characteristics table
- TCB/LCB action tables
- State program address linkage

Each of these is discussed separately later.

WORKLIST CONTROL BLOCK

Each TIP defines a worklist control block. Format of the block's initialization code is as follows:

```
BYWLCB [BOUSERn].BYPRADDR    := /ENTRY
                                (procname);
BYWLCB [BOUSERn].BYPAGE      := /PGNUM
                                (procname);
BYWLCB [BOUSERn].BYWLINDEX   := BOUSERn;
BYWLCB [BOUSERn].BYMAXCNT    := 1
BYWLCB [BOUSERn].BYINC       := 3
BYWLCB [BOUSERn].BYWLREQ     := /TRUE
```

where BOUSERn is the index of the worklist in the OPS MONITOR TABLE (BYWLCB), with n ranging from 1 to A. Procname is the procedure name containing the OPS-level entry point of the TIP.

TIP TYPE TABLES

Each TIP requires a TIP type table. Almost every field of this table must be initialized. The typical definition for an entry is:

```
BYTIPTYPT  NOUTTn].field := value;
```

where NOUTTn is the TIP type definition index. Standard TIPs use n values of 1 through 4. The TIP writer must define a new value for n for his TIP. Field is one of the fields as defined in the TIP type table. The fields usually defined are shown in table 14-1. Value may be locally defined, or it may be a global symbol.

TABLE 14-1. TIP-TYPE TABLE DEFINITIONS

Field	Range	Comments
BJOBT	TRUE or FALSE	Output buffer terminated worklist is to be generated (not usually needed for controlled protocols)
BJBZL	TRUE or FALSE	Reset timer flag when output buffer termination occurs. Controlled protocols reset timer after a successful output in order to time the terminal's response.
BJIVTSIZE	BSLFIDLES to BSPGWIDTH + 1	
BJTCBSIZE	/BOS32	TCB for a TIP always requires a 32 word buffer
BJQTYPE	/BOQNEXTSEG	
BJLISTIX	BOUSERn	Index to the worklist control block
BJDFTC	NOUSER1 - NOUSER5 for user terminal classes	Default terminal class when line is enabled
BJJFDC	VATnFDT for n = 1 to 4	Address of TIP's TCB descriptor table
BJFDT	VALnFDT for n = 1 to 4	Address of TIP's LCB descriptor table
BJJAT	VATnAT for n = 1 to 4	Address of TIP's TCB action table
BJAT	VALnAT for n = 1 to 4	Address of TIP's LCB action table
BJTIPMUX2	/PGNUM(procname)	Page number of the Mux-2 level entry to the TIP
BJTEMUX2	/ENTRY(procname)	Entry point for the Mux-2 level entry to the TIP. Defined only for those TIPs that have a routine exclusively for handling Mux-2 level inputs.
BJTCBPINIT	/PGNUM(procname)	Page number for the TIP's TCB initialization routine (direct call from service module)
BJTCBEINIT	/ENTRY(procname)	Entry point for the TIP's TCB initialization routine

See figure 14-2 for sample MPEDIT statements to initialize TIP type table.

```
TIP-TYPE TABLE FOR USER TIP 1

BJTIPTYPT NOUTT1    .BJOBT       := /TRUE;
BJTIPTYPT NOUTT1    .BJBZL       := /FALSE;
BJTIPTYPT NOUTT1    .BJTCBSIZE   := /BJS32;
BJTIPTYPT NOUTT1    .BJQTYPE     := /BJQNEXTSEQ;
BJTIPTYPT NOUTT1    .BJLISTIX    := /BOUWL1;
BJTIPTYPT NOUTT1    .BJDFTC      := /NOUTC1;
BJTIPTYPT NOUTT1    .BJJFDT      := /UAT1FDT;



BJTIPTYPT NOUTT1    .BJFDT       := UAL1FDT;
BJTIPTYPT NOUTT1    .BJJAT       := UAT1AT;
BJTIPTYPT NOUTT1    .BJAT        := UAL1AT;
BJTIPTYPT NOUTT1    .BJTEMUX2    := /ENTRY (procname);
BJTIPTYPT NOUTT1    .BJTPMUX2    := /PGNUM (procname);
BJTIPTYPT NOUTT1    .BJIVTSIZE   := BSLFIDLES -
                                    BSPGWIDTH + 1;
BJTIPTYPT NOUTT1    .BJTCBPINIT := /ENTRY (procname);
BJTIPTYPT NOUTT1    .BJTCBPINIT := /PGNUM (procname);
```

Figure 14-2. MPEDIT Statements to Initialize TIP Type Table (Sample)

TERMINAL CLASS TABLE

A terminal class table must be defined for each of the TIP's terminal classes. Two catagories of tables are usually defined: one for the base fields and one for the IVT overlay fields. The content of these tables may vary widely from TIP to TIP. For the same TIP, however, the same fields are set for each terminal class. Format of a table entry initialization is:

NJTECT [NoUTTn] . field := value;

where NOUTTn is the terminal class with n ranging from 1 through 4. Field is the NJTECT field mnemonic. Table 14-2 shows the fields defined for TIPs. Value is a symbolic or absolute value.

See figure 14-3 for sample MPEDIT statement to initialize terminal class table.

```
TERMINAL CHARACTERISTICS (BASE FIELDS) FOR TERM CLASS NOUTC1

NJTECT NOUTC1    .NJISPTA  := UAISPTR;          INPUT STATE ADDR
NJTECT NOUTC1    .NJTIPTY  := NOUTT1;
NJTECT NOUTC1    .NJPARITY := B7EVPAR;
NJTECT NOUTC1    .NJCHLEN  := /CLSEVEN;

IVT TERM CHARACTERISTICS FOR TERM CLASS NOUTC1

NJTECT NOUTC1    .NJPGWAIT    := /FALSE;
NJTECT NOUTC1    .NJPGWIDTH   := 72;
NJTECT NOUTC1    .NJPGLENGTH  := 0;
NJTECT NOUTC1    .NJCANCHAR   := /I31CANCHR;
NJTECT NOUTC1    .NJCNTRLCHAR := /I31CNTRLCHR;
NJTECT NOUTC1    .NJUSR1      := /I31UB1;
NJTECT NOUTC1    .NJUSR2      := /I31UB2;
NJTECT NOUTC1    .NJXCHAR     := /I31XCHAR;
NJTECT NOUTC1    .NJBSCHAR    := /I31BSCHAR;
NJTECT NOUTC1    .NJABTLINE   := /I31ABTLINE;
NJTECT NOUTC1    .NJXPARENT   := /FALSE;
NJTECT NOUTC1    .NJXTO       := /FALSE;
NJTECT NOUTC1    .NJXCNT      := 2043;
NJTECT NOUTC1    .NJCRCALC    := /TRUE;
NJTECT NOUTC1    .NJCRIDLES   := 2;
```

Figure 14-3. MPEDIT Statement to Initialize Terminal Class Table (Sample)

FIELD DESCRIPTOR TABLE DEFINITION

Normally, the TIP writer is not required to add new field name/field value (fn/fv) pairs to line or terminal configuration. These changes require host code changes. Only those fn/fv values that are defined for existing TIPs are described here. The TIP-defined line field descriptor table defines the line speed option. The TCB field descriptor is nil. The example shown in figure 14-4 is for USER TIP 1 and is contained in user supplied COMDECK ZEXtip.

```
*****************************************
*                                       *
*   USER TIP 1 TCB FIELD DESCRIPTOR TABLE   *
*                                       *
*****************************************

UAT1FDT 0 .ODNUMENT := 0;        NUMBER OF ENTRIES


*****************************************
*                                       *
*   USER TIP 1 LCB FIELD DESCRIPTOR TABLE   *
*                                       *
*****************************************

UAL1FDT 0 .DDNUMENT := 1;        NUMBER OF ENTRY

/BZLNSPD             := 21;
UAL1FDT 1 .DDFSTRT   := /START(BZLNSPD);
UAL1FDT 1 .DDFLNTH   := /LENGTH(BZLNSPD);
UAL1FDT 1 .DDFDISP   := BZLNSPD;
```

Figure 14-4. TCB and LCB Field Description Table Initialization (Sample)

ACTION TABLE INITIALIZATION

The following checks are manditory during a terminal configuration operation:

- /BSTCLASS - terminal class. Upper and lower bounds must be checked (D2VU and D2VL). If this is a reconfigure terminal service message, a variant TCB (D2TCBDFLT) should be established.
- /BSCN - change CN. The connection number must be processed (D2ACN).
- /SN - data path between host and terminal. The directory change must be processed (D2LLCB).
- /DN - new path to terminal for data from host. The directory change must be processed (D2LLCB).
- /BSNBL - network block limit.
- /BSIPRI - data priority.
- Internal (0) - three internal actions are possible. The new TIP must determine which of these internal actions is required. The first two or three action table entries are reserved for these internal actions (as was described earlier in the direct SVM to TIP call in the service module interface). Normally, D2TCBCHN, D2TCBINIT, and D2INIT, are used in that order. The remaining actions are descretionary; these usually check the upper and/or lower bound of the field to be initialized. The bound must be supplied in the fourth word of the entry (.DFPARAM := x).

TABLE 14-2. TERMINAL CHARACTERISTICS TABLE DEFINITIONS

Field	Value/Comments
NJISPTA	address of input states pointer table
NJCXLTA	address of code translate table
NJCNT1	initial value for counter 1 (0-255)
NJSYNC	SYNC character for protocol
NJCRCP	CRC polynomial index (see SPRM)
NJIBFCD	character displacement for first character to be stored in first buffer
NJBLKL	reset value for count 2 (0-16383)
NJTIPTY	TIPTYPE
NJPARITY	parity option B7ODDPR : odd parity B7NOPAR : no parity B7EVPAR : even parity
NJCHLEN	character length (see CLA Reference Manual for values) - normal 1 for 8-bit characters
NJPARTYPE	parity type B7ZERO : zero parity B7ODD : odd parity B7EVEN : even parity B7NONE : no parity
NJPGWAIT	page wait option (/TRUE,/FALSE)
NJXPARENT	Transparent input

IVT Parameters - The following fields must start on a word boundary:

```
NJPGWIDTH    :  B08BITS;          PAGE WIDTH
NJPGLENGTH   :  B08BITS;          PAGE LENGTH
NJCANCHAR    :  CHAR;             CANCEL INPUT LINE CHAR
NJCNTRLCHAR  :  CHAR;             CONTROL CHARACTER
NJUSR1       :  CHAR;             USER BREAK ONE
NJUSR2       :  CHAR;             USER BREAK TWO
NJXTO        :  BOOLEAN;          TIMEOUT IS EXPECTED DELIMITER
NJXCHRON                          XPT CHAR IS A DELIMITER
NJXCNT       :  B012BITS;         TRANSPARENT CHARACTER COUNT
NJCRCALC     :  BOOLEAN;          CR IDLE CNT TO BE CALCULATED
NJLFCALC     :  BOOLEAN;          LF IDLE CNT TO BE CALCULATED
NJECHOPLX    :  BOOLEAN;          ECHOPLEX MODE
NJINDEV      :  BOOLEAN;          INPUT DEVICE (F=KB, T=PT)
NJOUTDEV     :  B02BITS;          OUTPUT DEVICE (PRT, DISPLAY, PT)
NJAPL        :  B02BITS;          APL (0=NO, 1=YES, 2=SPECIAL)
NJXCHAR      :  CHAR;             TRANSPARENT DELIMITER CHAR
NJBSCHAR     :  CHAR;             BACKSPACE CHARACTER
NJABTLINE    :  CHAR;             ABORT OUTPUT LINE CHARACTER
NJCPIDLES    :  B08BITS;          COUNT OF IDLES AFTER CR
NJLFIDLES    :  B08BITS);         COUNT OF IDLES AFTER LF
```

Note: The IVT parameters are used to initially set the IVT fields in the TCB if the device type is a console. The permissable values for these parameters are defined by the IVT specifications (see the CCP System Programmer's Reference Manual).

Figure 14-5 shows an example of an action table initialization. All the remaining values in the example may be checked; but there is no requirement to check them. These fields permit optional terminal specification; in the terminal definition statements of NDL, or by IVT commands from a terminal or from an application program. Omission indicates that processing is skipped. The example shows a complete set, whether it is needed by the TIP or not.

```
**********************************
*                                *
*   USER TIP 1 TCB ACTION TABLE  *
*                                *
**********************************

/L := 1;

UAT1AT /L .DFFN    := 0;            INTERNAL ACTION
UAT1AT /L .DFRKEY  := D2SKP;
UAT1AT /L .DFCKEY  := D2TCBCHN;
UAT1AT /L .DFPARAM := 0;                          /L := /L + 1

UAT1AT /L .DFFN    := 0;
UAT1AT /L .DFRKEY  := DZSKP;
UAT1AT /L .DFCKEY  := DZTCBINIT;
UAT1AT /L .DFPARAM := 0;                          /L := /L + 1

UAT1AT /L .DFFN    := 0;            INTERNAL ACTION
UAT1AT /L .DFRKEY  := D2INIT;
UAT1AT /L .DFCKEY  := D2INIT;
UAT1AT /L .DFPARAM := 0;                          /L := /L + 1

UAT1AT /L .DFFN    := /BSTCLASS;
UAT1AT /L .DFRKEY  := D2VUL;
UAT1AT /L .DFCKEY  := D2VUL;
UAT1AT /L .DFPARAM := $801                        /L := /L + 1

UAT1AT /L .DFFN    := /BSTCLASS;
UAT1AT /L .DFRKEY  := D2VM;
UAT1AT /L .DFCKEY  := D2NA;
UAT1AT /L .DFPARAM := $204                        /L := /L + 1

UAT1AT /L .DFFN    := /BSTCLASS;
UAT1AT /L .DFRKEY  := D2TCBDFLT;
UAT1AT /L .DFCKEY  := D2NA;
UAT1AT /L .DFPARAM := 0:                          /L := /L + 1

UAT1AT /L .DFFN    := /BSOWNER;
UAT1AT /L .DFRKEY  := D2VL;
UAT1AT /L .DFCKEY  := D2VL;
UAT1AT /L .DFPARAM := 0:                          /L := /L + 1

UAT1AT /L .DFFN    := /BSCN;
UAT1AT /L .DFRKEY  := D2ACN;
UAT1AT /L .DFCKEY  := D2ACN;
UAT1AT /L .DFPARAM := 0:                          /L := /L + 1

UAT1AT /L .DFFN    := /DN;
UAT1AT /L .DFRKEY  := D2ADN;
UAT1AT /L .DFCKEY  := D2SKP;
UAT1AT /L .DFPARAM := 0:                          /L := /L + 1

UAT1AT /L .DFFN    := /SN;
UAT1AT /L .DFRKEY  := D2LLCB;
UAT1AT /L .DFCKEY  := D2LLCB;
UAT1AT /L .DFPARAM := 0:                          /L := /L + 1

UAT1AT /L .DFFN    := /BSNBL;
UAT1AT /L .DFRKEY  := D2VU;
UAT1AT /L .DFCKEY  := D2VU;
UAT1AT /L .DFPARAM := 7:                          /L := /L + 1
```

Figure 14-5. MPEDIT Initialization of USER TIP 1 TCB Action Table (Sample; 1 of 3)

```
UAT1AT /L .DFFN    := /BSIPRI;
UAT1AT /L .DFRKEY  := D2VU;
UAT1AT /L .DFCKEY  := D2VU;
UAT1AT /L .DFPARAM := 3:                          /L := /L + 1

UAT1AT /L .DFFN    := /BSIPRI;
UAT1AT /L .DFRKEY  := D2VL;
UAT1AT /L .DFCKEY  := D2VL;
UAT1AT /L .DFPARAM := 1:                          /L := /L + 1

UAT1AT /L .DFFN    := /BSCANCHAR;
UAT1AT /L .DFRKEY  := D2VU;
UAT1AT /L .DFCKEY  := D2VU;
UAT1AT /L .DFPARAM := 127;                        /L := /L + 1

UAT1AT /L .DFFN    := /BSBSCHAR;
UAT1AT /L .DFRKEY  := D2VU;
UAT1AT /L .DFCKEY  := D2VU;
UAT1AT /L .DFPARAM := 127;                        /L := /L + 1

UAT1AT /L .DFFN    := /BSCNTRLCHAR;
UAT1AT /L .DFRKEY  := D2VU;
UAT1AT /L .DFCKEY  := D2VU;
UAT1AT /L .DFPARAM := 127;                        /L := /L + 1

UAT1AT /L .DFFN    := /BSCRIDLES;
UAT1AT /L .DFRKEY  := D2VU;
UAT1AT /L .DFCKEY  := D2VU;
UAT1AT /L .DFPARAM := 99;                         /L := /L + 1

UAT1AT /L .DFFN    := /BSLFIDLES;
UAT1AT /L .DFRKEY  := D2VU;
UAT1AT /L .DFCKEY  := D2VU;
UAT1AT /L .DFPARAM := 99;                         /L := /L + 1

UAT1AT /L .DFFN    := /BSCRCALC;
UAT1AT /L .DFRKEY  := D2VU;
UAT1AT /L .DFCKEY  := D2VU;
UAT1AT /L .DFPARAM := 1;                          /L := /L + 1

UAT1AT /L .DFFN    := /BSCRCALC;
UAT1AT /L .DFRKEY  := D2CRIDLE;
UAT1AT /L .DFCKEY  := D2CRIDLE;
UAT1AT /L .DFPARAM := 0;                          /L := /L + 1

UAT1AT /L .DFFN    := /BSLFCALC;
UAT1AT /L .DFRKEY  := D2VU;
UAT1AT /L .DFCKEY  := D2VU;
UAT1AT /L .DFPARAM := 1;                          /L := /L + 1

UAT1AT /L .DFFN    := /BSLFCALC;
UAT1AT /L .DFRKEY  := D2LFIDLE;
UAT1AT /L .DFCKEY  := D2LFIDLE;
UAT1AT /L .DFPARAM := 0;                          /L := /L + 1

UAT1AT /L .DFFN    := /BSAPL;
UAT1AT /L .DFRKEY  := D2VU;
UAT1AT /L .DFCKEY  := D2VU;
UAT1AT /L .DFPARAM := 2;                          /L := /L + 1

UAT1AT /L .DFFN    := /BSAPL;
UAT1AT /L .DFRKEY  := D2APL;
UAT1AT /L .DFCKEY  := D2APL;
UAT1AT /L .DFPARAM := 0;                          /L := /L + 1

UAT1AT /L .DFFN    := /BSXPARENT;
UAT1AT /L .DFRKEY  := D2VU;
UAT1AT /L .DFCKEY  := D2VU;
UAT1AT /L .DFPARAM := 1;                          /L := /L + 1
```

Figure 14-5. MPEDIT Initialization of USER TIP 1 TCB Action Table (Sample; 2 of 3)

```
UAT1AT /L .DFFN    := /BSXCHM;
UAT1AT /L .DFRKEY  := D2VU;
UAT1AT /L .DFCKEY  := D2VU;
UAT1AT /L .DFPARAM := 15;                          /L := /L + 1

UAT1AT /L .DFFN    := /BSXTO;
UAT1AT /L .DFRKEY  := D2VU;
UAT1AT /L .DFCKEY  := D2VU;
UAT1AT /L .DFPARAM := 1;                           /L := /L + 1

UAT1AT /L .DFFN    := /BSINDEV;
UAT1AT /L .DFRKEY  := D2VU;
UAT1AT /L .DFCKEY  := D2VU;
UAT1AT /L .DFPARAM := 1;                           /L := /L + 1

UAT1AT /L .DFFN    := /BSOUTDEV;
UAT1AT /L .DFRKEY  := D2VU;
UAT1AT /L .DFCKEY  := D2VU;
UAT1AT /L .DFPARAM := 2;                           /L := /L + 1

UAT1AT /L .DFFN    := /BSECHOPLX;
UAT1AT /L .DFRKEY  := D2VU;
UAT1AT /L .DFCKEY  := D2VU;
UAT1AT /L .DFPARAM := 1;                           /L := /L + 1

UAT1AT /L .DFFN    := /BSPGWAIT;
UAT1AT /L .DFRKEY  := D2VU;
UAT1AT /L .DFCKEY  := D2VU;
UAT1AT /L .DFPARAM := 1;                           /L := /L + 1

UAT1AT /L .DFFN    := /BSPARITY;
UAT1AT /L .DFRKEY  := D2VU;
UAT1AT /L .DFCKEY  := D2VU;
UAT1AT /L .DFPARAM := 3;                           /L := /L + 1

UAT1AT /L .DFFN    := /BSPARITY;
UAT1AT /L .DFRKEY  := D2PARITY;
UAT1AT /L .DFCKEY  := D2PARITY;
UAT1AT /L .DFPARAM := 0;                           /L := /L + 1

UAT1AT /L .DFFN    := /BSABTLINE;
UAT1AT /L .DFRKEY  := D2VU;
UAT1AT /L .DFCKEY  := D2VU;
UAT1AT /L .DFPARAM := 127;                         /L := /L + 1

UAT1AT /L .DFFN    := /BSUSR1;
UAT1AT /L .DFRKEY  := D2VU;
UAT1AT /L .DFCKEY  := D2VU;
UAT1AT /L .DFPARAM := 127;                         /L := /L + 1

UAT1AT /L .DFFN    := /BSUSR2;
UAT1AT /L .DFRKEY  := D2VU;
UAT1AT /L .DFCKEY  := D2VU;
UAT1AT /L .DFPARAM := 127;                         /L := /L + 1

UAT1AT /L .DFFN    := /BSCODE;
UAT1AT /L .DFRKEY  := D2VL;
UAT1AT /L .DFCKEY  := D2VL;
UAT1AT /L .DFPARAM := 4;                           /L := /L + 1

UAT1AT /L .DFFN    := /BSCODE;
UAT1AT /L .DFRKEY  := D2VU;
UAT1AT /L .DFCKEY  := D2VU;
UAT1AT /L .DFPARAM := 5;                           /L := /L + 1

UAT1AT /L .DFFN    := /BSXCHRON;
UAT1AT /L .DFRKEY  := D2VU;
UAT1AT /L .DFCKEY  := D2VU;
UAT1AT /L .DFPARAM := 1;                           /L := /L + 1

UAT1AT /L .OFEND   := $7FFF:          END OF TABLE
```

Figure 14-5. MPEDIT Initialization of USER TIP 1 TCB Action Table (Sample; 3 of 3)

Table 14-3 defines all the terminal configuration field numbers and the associated mnemonic names. In initialization statements, a / precedes the field name, since these MPEDIT variables are defined for the TIP writer. The set of terminal initialization statements are placed in the user supplied COMDECK ZEXtip.

The following check is mandatory for line configuration:

- /BZOWNER - owning host for this line.

An optional check can be made for line speed, if the line is asynchronous with a fixed speed, and this speed is defined in the line definition:

- /BZLNSPD - line speed.

TABLE 14-3. TCB FIELD DESCRIPTOR TABLE

FN	Field	Comments
5	BSTCLASS	Terminal class. A new class must be defined if the terminal does not use a protocol identical to a current terminal (exceptions - minor variations are available by use of IVT command parameters).
12	BSOWNER	Node ID of CS in the host.
13	BSCN	Connection Number defined by initialization for this terminal.
14	BSLLCB (upper)	The destination node (DN) byte in the LLCB that is used as a directory.
15	BSLLCB (lower)	The source node (SN) byte in the LLCB that is used as a directory.
16	BSABL	Available block limit - the number of blocks that can be used by a TIP before triggering regulation if the TIP requests PTREGL to set regulation on available block count.
19	BSIPRI	Input priority. The priority of an input block that will be accepted without triggering regulation if the TIP requests PTREGL to set regulation on input priority.
28	BSPGWIDTH	Default page width (in characters) for the device. See note 1.
29	BSPGLENGTH	Default page length (in lines) for the device. See note 1.
30	BSCANCHAR	Character used to cancel an input line from the terminal. See note 2.
31	BSBSCHAR	Character used for backspacing on the device. See note 2.
32	BSCNTRLCHAR	Control character for the device. See note 2.
33	BSCRIDLES	Carriage return idle count for the device. See note 2.
34	BSLFIDLES	Line feed idle count for the device. See note 2.
35	BSCRCALC	Flag to calculate the CR idle count for the device.
36	BSLFCALC	Flag to calculate the LF idle count for the device.
37	BSAPL	APL mode flag. See note 2.
38	BSXPARENT	Transparent text delimiting flag.
39	BSXCHM	Most significant 4 bits of the transparent character count (BSXCNT). See note 3.
40	BSXCHL	Least significant 8 bits of the transparent character count (BSXCNT). See note 3.
41	BSXCHAR	Transparent text delimiting character. See note 3.
42	BSXTO	Transparent message delimited by timeout flag. Value is 300 $\pm$ 100 ms. See note 3.

TABLE 14-3. TCB FIELD DESCRIPTOR TABLE (2 of 2)

FN	Field	Comments
43	BSINDEV	Input device. See note 2.
44	BSOUTDEV	Output device. See note 2.
45	BSECHOPLX	Echoplex mode flag. See note 2.
46	BSPGWAIT	Page Wait Mode flag. See note 2.
47	BSPARITY	Parity type selector.
48	BSABTLINE	Abort output line character. See note 2.
49	BSUSR1	User break character 1. See note 2.
50	BSUSR2	User break character 2. See note 2.
51	BSCODE	TIP code set. Not used if autorecognition is used. See TIP type table in appendix E of SPRM. See note 4.
52	BSXCHRON	Transparent message delimited by a delimiter character. See note 3.

NOTES:

1. For interactive devices, this value can be altered later by an IVT command. A message is sent upline changing this value in CS, but not in the load file. At reload time, the value reverts to this initialization value.

2. For interactive devices, this value can be altered later by an IVT command. The value is kept locally in the TCB. Any action that causes release of the TCB loses the changed value. Reconfiguration of the TCB from the host uses the initialization value.

3. For interactive devices, the character count, timeout flag or transparent delimiting character in the TCB INT overlay can be changed by an upline IVT parameter change message from the interactive device. All of the values revert to default value after the current transparent message is processed.

4. New codes will need to be defined for new TIP that services a multi-code terminal group.

Figure 14-6 shows a sample of the LCB action table initialization statements.

The MPEDIT statements for LCB action table initialization are included in the user supplied COMDECK ZEXtip.

STATE PROGRAM ADDRESS LINKAGE

A TIP can be written with various state programs and code conversion tables used dynamically, depending on the situation at the terminal. If this is done, the TIP writer must initialize globals with addresses, so the PASCAL code can reference the content.

```
**********************************
*                                *
*   USER TIP 1 LCB ACTION TABLE  *
*                                *
**********************************

UAL1AT 1 .DFFN     := /BZOWHER;
UAL1AT 1 .DFRKEY   := D2VL;
UAL1AT 1 .DFCKEY   := D2VL;
UAL1AT 1 .DFPARAM := 1:

UAL1AT 2 .DFFN     := /BZLNSPO;
UAL1AT 2 .DFRKEY   := D2VU;
UAL1AT 2 .DFCKEY   := D2VU;
UAL1AT 2 .DFPARAM := 8:

UAL1AT 3 .OFEND    := $7FFF;          END OF TABLE
```

Figure 14-6. MPEDIT Initialization Statements for LCB Action Table (Sample)

BLOCK PROTOCOL ACKNOWLEDGMENT SCHEME A

The nine block types plus four special commands (block type 4) are used to establish and maintain communication between the host and the NPU. Tables A-1 and A-2 show the block acknowledgment scheme. In general, the following applies:

- BACK blocks are never acknowledged
- MSG and BLK blocks are always BACKed
- CMD blocks to a TIP (CN≠0) are BACKed
- CMD blocks (service messages) downline are acknowledged by an upline service message reply
- BRK blocks downline cause an RST block to be sent upline
- BRK upline causes host to notify the application program which sent the MSG, BLK, or CMD block. Sender decides whether to retransmit block
- INIT, RST, and STP downline are not acknowledged
- STRT downline causes an RST block to be sent upline.
- STP block upline is not acknowledged
- INIT upline causes an INIT downline
- STRT block upline causes an RST downline

TABLE A-1. NPU ACKNOWLEDGMENT OF DOWNLINE BLOCKS

Block Type	Queued To	Acknowledged By	Remarks
BACK	Not queued	Not acknowledged	BIP decrements outstanding block count and releases the BACK block.
Bad block	Not queued	Not acknowledged	BIP counts this as a bad address input block and discards.
CMD with CN=0	SVM	Service module with a reply SM	Response service message is the acknowledgment.
CMD with CN≠0	TCB's output queue	TIP (may be thru PTIVTCMD) with a BACK block	If application to TIP CMD, TIP calls PTBACK. TIP also generates input stopped CMD block messages. If downline IVT CMD block, PTIVTCMD sends BACK block if parameters successfully changed, otherwise, BRK block is sent. BRK block is sent for any unintelligible CMD block received by TIP.
MSG/BLK	TCB's output queue	BIP	TIP calls BIP when message is transmitted to terminal, BIP sends BACK.
BRK	TIP WLE to OPS-level	BIP RST block	BIP sends a RST block; sends WLE to TIP (allows higher priority than data).
INIT	Not queued	Not acknowledged	Processed by BIP which sets accept input (AI) and accept output (AO) flags. BIP also sets outstanding block limit (OBL) counter to 0.
RST	Not queued	Not acknowledged	Processed by BIP which sets accept output (AO) flag.
STRT	Not queued	BIP with RST block	BIP sets accept input (AI) flag and sends the reset block.
STOP	TIP WLE to OPS-level	Not acknowledged	BIP clears accept input (AI) flag. However, WLE allows processing the communications line of messages with higher priority than data.

TABLE A-2. ACKNOWLEDGMENT OF TIP-GENERATED UPLINE BLOCKS

Block Types	Acknowledgment Required	Remarks
BACK	No	BACK block is acknowledgment to previous MSG, CMD or BLK block sent to host.
CMD CN=0	No	This is either an unsolicited SM that requires specifications from host or is the solicited answer to host's previous SM to the NPU (SVM processes). Unsolicited SMs are repeated after a time interval until the required action from the host is detected.
CMD CN≠0	Yes	Host responds with a BACK block.
BRK	Yes	Host responds with a RST. MSG/BLK/CMD blocks not acknowledged may be repeated.
BLK or MSG	Yes, BACK block	Internal BID process handles the BACK block. TIP is not notified.
STRT	Yes	Host replies with RST to start output.
STP	No	Accept output flag is reset. TCB's output data queue is purged.

INDEX

COMMENT SHEET

MANUAL TITLE: CCP3 TIP Writer's Guide

PUBLICATION NO.: 60474600 **REVISION:** B

NAME:_____

COMPANY:_____

STREET ADDRESS:_____

CITY:_____ **STATE:**_____ **ZIP CODE:**_____

This form is not intended to be used as an order blank. Control Data Corporation welcomes your evaluation of this manual. Please indicate any errors, suggested additions or deletions, or general comments below (please include page number references).

CUT ALONG LINE

AA3419 REV. 4/79 PRINTED IN U.S.A.

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

FOLD ON DOTTED LINES AND TAPE

TAPE

TAPE

FOLD

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 8241 MINNEAPOLIS, MINN.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION

Publications and Graphics Division
P. O. Box 4380-P
Anaheim, California 92803

CUT ALONG LINE

FOLD

FOLD

CORPORATE HEADQUARTERS, P.O. BOX 0, MINNEAPOLIS, MINN. 55440
SALES OFFICES AND SERVICE CENTERS IN MAJOR CITIES THROUGHOUT THE WORLD

LITHO IN U.S.A.

CONTROL DATA CORPORATION