AD-A038 344 BOLT BERANEK AND NEWMAN INC CAMBRIDGE MASS F/G 9/2
THE TERMINAL INTERFACE MESSAGE PROCESSOR PROGRAM.(U)
FEB 77
UNCLASSIFIED TECHNICAL INFORMATION-91 DAHC15-69-C-0179
NI




AD A038344

(14) Technical Information-91

BOLT BERANEK AND NEWMAN INC.

(15) DAHC15-69-C-0179,
F08606-75-C-0032

(6) THE TERMINAL INTERFACE MESSAGE PROCESSOR PROGRAM,

(11) Feb[redacted]1977

(12) 228 p.

Update for TIP Version 410

February 1977

DDC
APR 14 1977
C

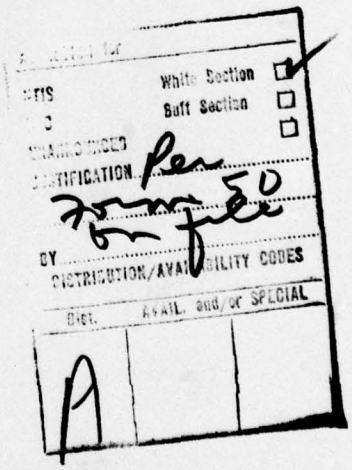
ACCESSION for
NTIS White Section
DDC Buff Section
UNANNOUNCED
JUSTIFICATION per form 50 on file
BY
DISTRIBUTION/AVAILABILITY CODES
Dist. AVAIL. and/or SPECIAL
A

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

060 100

Table of Contents

1. Overview of the Terminal IMP Hardware
2. Software Summary
3. Performance Summary
4. Summary of Protocol Design Decisions and Protocol Deviations
   - 4.1 Design Decisions
   - 4.2 Deviations from Protocols
5. References and TIP Bibliography
6. Storage Layout
7. Data Structures
8. Detailed Software Description
   - 8.1 Outline of Program's Functional Structure
   - 8.2 Detailed Descriptions
   - 8.3 Index to Detailed Descriptions



1. OVERVIEW OF THE TERMINAL IMP HARDWARE

Understanding the Terminal IMP software depends on an understanding of the hardware environment in which the software resides. A summary of the Terminal IMP hardware follows.

Up to 63* terminals, either remote or local, of widely diverse characteristics may be connected to a given TIP and thereby "talk" into the network. It is also possible to connect a Host to a TIP in the usual way a Host is connected to an IMP.

The TIP is built around a Honeywell H-316 computer with 28K of core. It embodies a standard 16-port multiplexed memory channel with priority interrupts and includes a Teletype for debugging and program reloading. Other features of the standard IMP also present are a real-time clock, power-fail and auto-restart mechanisms, and a program-generated interrupt feature. As in the standard IMP, interfaces are provided for connecting to high-speed (50-kilobit, 230.4-kilobit, etc.) modems as well as to Hosts.

Aside from the additional 12K of core memory, the primary hardware feature which distinguishes the TIP from a standard IMP is a Multi-Line Controller (MLC) which allows for connection of terminals to the IMP. Any of the MLC lines may go to local terminals or via modems to remote terminals. As shown in Figure 1-1 the MLC consists of two portions, one a piece of central logic which handles the assembly and disassembly of characters and transfers them to and from memory, and the other a set of individual Line Interface Units (all identical except for small number of option jumpers) which synchronize reporting to individual data bits between the central logic and the terminal devices and provide for control and status information to and from the modem or device. Line Interface Units may be physically incorporated one at a time as required.

---

*There are 64 hardware lines but line Ø is logically reserved by the program for special use.

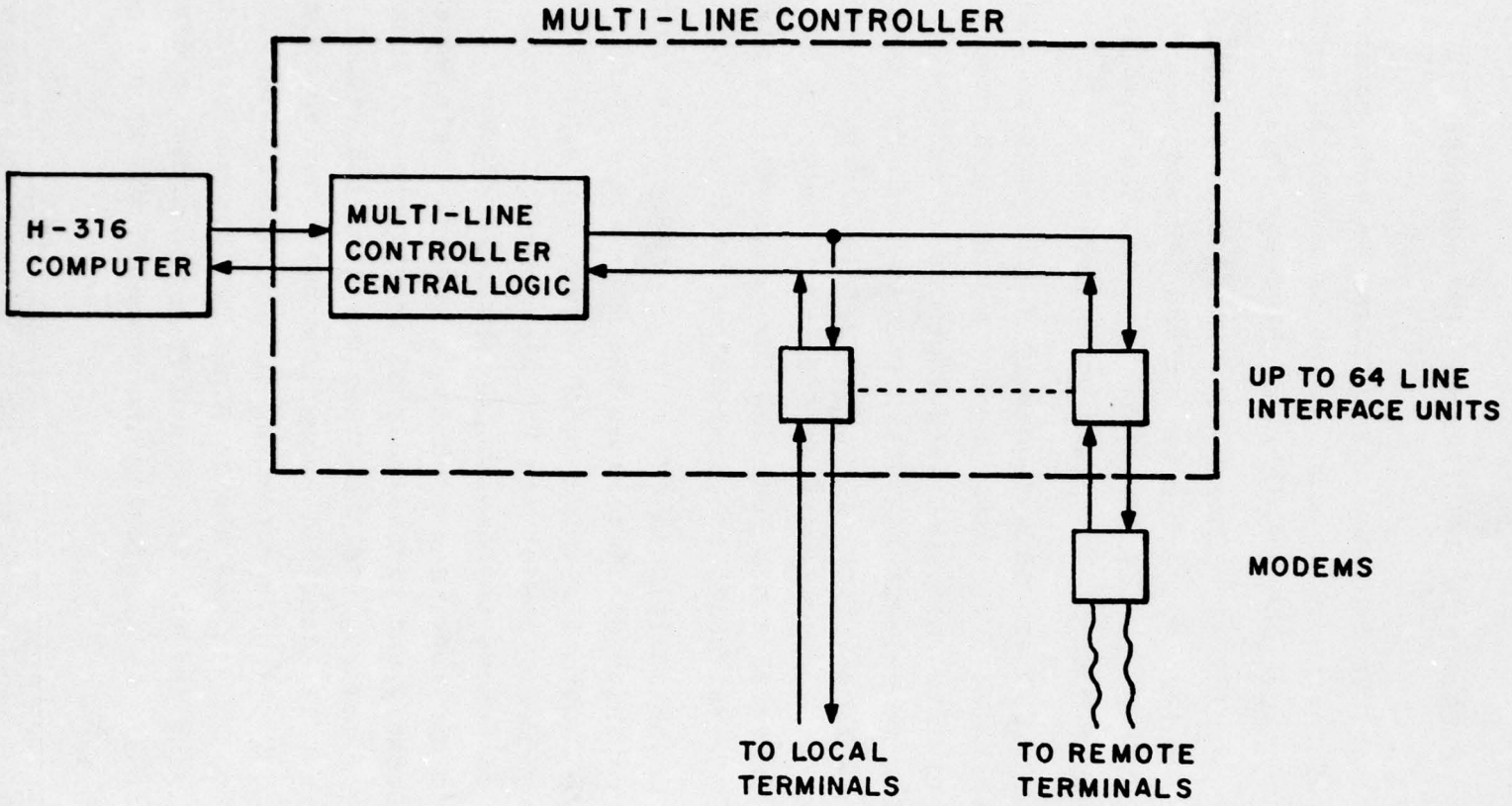


FIGURE 1-1 BLOCK DIAGRAM OF THE TIP HARDWARE

The MLC connects to the high-speed multiplexed memory channel option of the H-316, and uses three of its channels as well as two priority interrupts and a small number of control instructions.

In order to accommodate a variety of devices, the controller handles a wide range of data rates and character sizes, in both synchronous and asynchronous modes. Data characters of length 5, 6, 7, or 8 bits are allowed by the controller. Since no interpretation of characters is done by the hardware, any character set, such as ASCII or EBCDIC, may be used.

The following is a list of data rates accepted by the controller.

SYNCHRONOUS	ASYNCHRONOUS (Nominal Rates)	
Any rate up to and including 19.2 Kb/s	75	1200
	110	1800
	134.5	2400
	150	4800 }
	300	9600 } output only
	600	19200 }

All above in bits/second

The data format required of all devices is bit serial; each character indicates its own beginning by means of a start bit preceding the data and includes one or more stop bits at the end of the character.

Given these characteristics, then, the controller will connect to the great majority of normal terminal devices such as Teletypes, alphanumeric CRT units, and modems, and also (with suitable remote interface units) to many peripheral devices such as card readers, line printers, and graphics terminals. Either full or half duplex devices can be accommodated. The standard TIP program cannot deal with a magnetic tape unit through the MLC. However, as a special

option, and with the use of additional core memory, standard Honeywell tape drives can be connected to the TIP as normal peripherals.

The individual terminal line levels are consistent with EIA RS-232C convention. Data rates and character length are individually set for each line *by the program.* For incoming asynchronous lines, the program includes the capability for detecting character length and line data rate as discussed below.

Logically, the controller consists of 64 input ports and 64 output ports. Each input/output pair is brought out to a single connector which is normally connected to a single device. However, by using a special "Y" cable, the ports may go to completely separate devices of entirely different properties. Thus, *input* port 16 may connect to a slow, asynchronous, 5-bit character keyboard while *output* port 16 connects to a high speed, synchronous display of some sort. In order to achieve this flexibility, the MLC stores information about each input and each output port and the program sets up this information for each half of each port in turn as it turns the ports "ON."

Several aspects of the MLC design are noteworthy. The central logic treats each of the 64 ports in succession, each port getting 800 ns of attention per cycle. The port then waits the remainder of the cycle (51.2 μs) for its next turn. For both input and output, two full characters are buffered in the central logic, the one currently being assembled or disassembled and one further character to allow for delays in memory accessing.

During input, characters from the various lines stream into a tumble table in memory on a first come, first served basis. Periodically a clock interrupt causes the program to switch tables and look for input.

Output characters are fed to all lines from a single output table. Ordering the characters in this table in such a way as to keep a set of lines of widely diverse speeds solidly occupied is a difficult task. To assist the program in this, a novel mechanism has been built into the MLC hardware whereby each line, as it uses up a character from the output table, enters a request consisting of its line number into a "request" table in memory. This table is periodically inspected by the program and the requests are used in building the next output table with the characters in proper line sequence.

The design of the terminal interface portion of the MLC is modular. Each Line Interface Unit (LIU) contains all the logic required for full duplex, bit serial communication and consists of a basic bi-directional data section and a control and status section. The data section contains transmit and receive portions each with clock and data lines. For asynchronous devices the clock line is ignored and timing is provided by the MLC itself. (For received asynchronous characters, timing is triggered by the leading edge of the start bit of each character.)

The control and status monitor functions are provided for modems as required by the RS-232C specification. Four outputs are available for control functions and six inputs are available to monitor status. The outputs are under program control and are available for non-standard functions if the data terminal is not a modem. For example, these lines could be used to operate a local line printer. RS-232C connectors are mounted directly on the LIU cards. To allow for variations in terminal and modem pin assignments, the signals are brought to connector pins via jumpers on the card.

2. SOFTWARE SUMMARY

Because the terminals connected to a TIP communicate with Hosts at remote sites, the TIP, in addition to performing the IMP function, also acts as intermediary between the terminal and the distant Host. This means that network standards for format and protocol must be implemented in the TIP. One can thus think of the TIP software as containing both a very simpleminded mini-Host and a regular IMP program.

Figure 2-1 gives a simplified diagrammatic view of the program. The lower block marked "IMP" represents the usual IMP program. The two lines into and out of that block are logically equivalent to input and output from a Host. The code conversion blocks are in fact surprisingly complex and include all of the material for dealing with diverse (and perverse) types of terminals.

As the user types on the keyboard, characters go, via input code conversion, to the input block. Information for remote sites is formed into regular network messages and passes through the OR switch to the IMP program for transmittal. Command characters are fed off to the side to the command block where commands are decoded. The commands are then "performed" in that they either set some appropriate parameter or a flag which calls for later action. An example of this is the LOGIN command. Such a command in fact triggers a complex network protocol procedure, the various steps of which are performed by the PROTOCOL block working in conjunction with the remote Host through the IMPs. As part of this process an appropriate special message will be sent to the terminal via the Special Messages block indicating the status (success, failure, etc.) of the procedure.

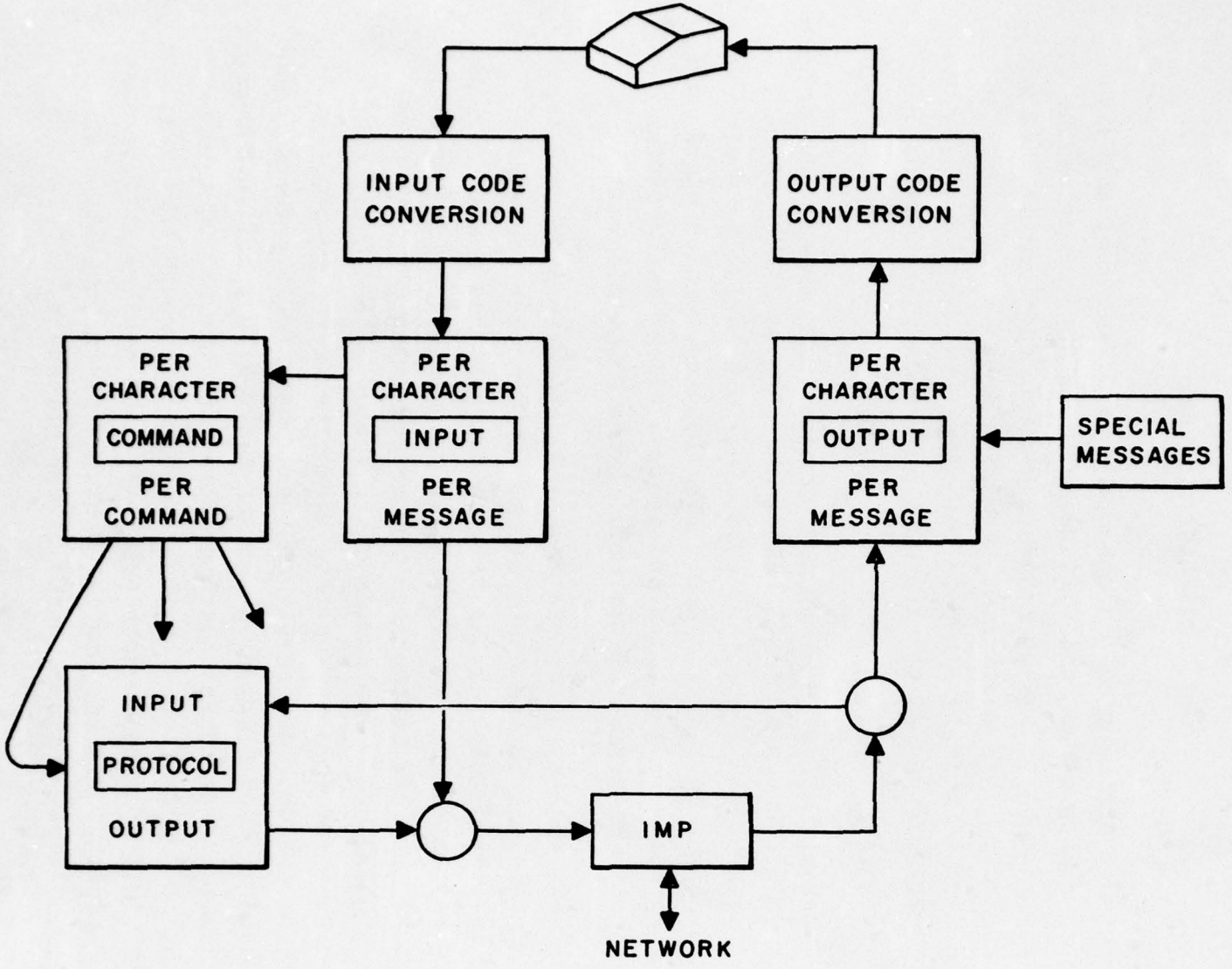


FIGURE 2-1 BLOCK DIAGRAM OF TIP PROGRAM

Once connection to a remote Host is established, regular messages flow directly through the Input block and on through the IMP program. Returning responses come in through the IMP, into the OUTPUT program where they are fed through the OUTPUT Code Conversion block to the terminal itself.

3. PERFORMANCE SUMMARY

The program can handle approximately 100 kilobits of one-way terminal traffic provided the message size is sufficiently large that per-message processing is amortized over many characters. Overhead per message is such that if individual characters are sent one at a time there is a loss of somewhere between a factor of ten and twenty in bandwidth. A different way to look at program performance is to observe that the per-character processing time is about 75 μs.

These figures ignore the fact that the machine must devote some of its bandwidth to acting as an IMP, both for terminal traffic and for regular network traffic. About 5% of the machine is lost to acting as an IMP, even in the absence of traffic. If there is network traffic, more of the machine bandwidth is used up. Five hundred kilobits of two-way phone line and Host traffic saturates the machine without any terminal traffic*.

In addition to bandwidth which goes into the IMP part of the job, another 10 percent of the (total) machine is taken up simply in fielding clock interrupts from the Multi-Line Controller. This again is bandwidth used in idling even with no actual terminal traffic.

The following formula summarizes, approximately, the bandwidth capabilities:

$$P + H + 15T \leq 600$$

*The number 500 kilobits is for full size (8000 bit) messages. Shorter messages use up more capability per bit and thus reduce the overall bandwidth capability.

where:

P = total phone line traffic (in kilobits/sec) wherein, for example, a full duplex 50 Kb phone line counts as 100;

H = total Host traffic (in kilobits/sec) wherein the usual full duplex Host interface, with its usual 10 μs/bit setting, counts as 200; and

T = total terminal traffic (in kilobits/sec) wherein an ASCII terminal such as a (110-baud) full-duplex Teletype (ASR-33) counts as twice its baud rate (i.e., 0.220 Kb).

This means that it takes fifteen times as much program time to service every terminal character as it does to service a character's worth of phone line or Host message.

A further factor that influences terminal traffic handling capability has to do with the terminals themselves. Certain types of terminals require more attention from the program than others, independent of their speed but based rather on their complexity. In particular, for example, while an IBM 2741 nominally runs at 134.5 bits per second, the complexity is such that it uses nearly three times the program bandwidth that would be used in servicing a half-duplex ASCII terminal of equivalent speed. Allowances for such variations must be made in computing the machine's ability to service a particular configuration. It must be borne in mind that all of these performance figures are approximations and that the actual rules are extremely complex.

4. SUMMARY OF PROTOCOL DESIGN DECISIONS AND PROTOCOL DEVIATIONS

This section discusses the Terminal IMP's implementation of its Network Control Program (NCP) for the Host/Host Protocol [1], Initial Connection Protocol (ICP) [2], and TELNET [3]. Included are descriptions of the design decisions made where such decisions are permitted by the protocols, and of instances of non-compliance with the protocols.

Most of the choices made during protocol implementation on the Terminal IMP were influenced strongly by storage limitations. The Terminal IMP has no bulk storage for buffering, and has only 12 kilowords of 16-bit words available for both device I/O buffers and program. The program must drive up to 63 terminals which will generally include a variety of terminal types with differing code sets and communication protocols (e.g., the IBM 2741 terminals). In addition, the Terminal IMP must include a rudimentary language processor which allows a terminal user to specify parameters affecting his network connections. Since the Terminal IMP exists only to provide access to the network for 63 terminals, it must be prepared to maintain 126 (simplex) network connections at any time; thus each word stored in the NCP tables on a per-connection basis consumes a significant portion of the Terminal IMP memory.

It should be remembered that the Terminal IMP is designed to provide access to the network for its users, not to provide service to the rest of the network. Thus the Terminal IMP does not contain programs to perform the "server" portion of the ICP; in fact, it does not have a "logger" socket.

4.1 Design Decisions

4.1.1. The Terminal IMP ignores incoming ERR commands and does not output ERR commands.

4.1.2. The Terminal IMP assumes that incoming messages have the format and contents demanded by the relevant protocols. For example, the byte size of incoming TELNET messages is assumed to be 8. The major checks which the Terminal IMP does make are:

1) if an incoming control message has a byte count greater than 120 then it is discarded.

2) if a control command opcode greater than 18 is found during the processing of a control message then the remainder of the control message is discarded.

3) if an incoming data message has a byte count indicating that the bit allocation for the connection is exceeded (based on the assumed byte size) then the message is discarded.

4.1.3 "Unsolicited" control messages, including RST commands, are saved for response to the limit of the unsolicited reply queue, which currently has eight slots. Unsolicited messages are discarded if there is no space available when they arrive.

4.1.4 Socket numbers and the link for receive connections are pre-assigned based on the hardware "physical address" (in the terminal multiplexing device) of the terminal. The high order 16 bits of the socket number give the device number (in the range 0-63) and the low order 16 bits are normally 2 or 3 depending on the socket's gender (zero is also used during ICP); the TIP requires servers to send data to it on link "device number" +2 [range 2-65].

4.1.5 During ICP, with the Terminal IMP as the user site, the Terminal IMP follows the "Listen" option rather than the "Init" option (as described at the top of page 3 of [2]). In other words, the Terminal IMP does not issue the RFCs involving sockets U+2 and U+3 except in response to incoming RFCs involving those sockets.

4.1.6 The TIP clears out its connection tables as it sends an RST or an RRP and then immediately continues operations with the affected Host, except in the case of the "initial" RST sent by the TIP when there has been no recent activity with that Host; the TIP then waits for the RRP before allowing new logins to procede.

4.2 Deviations from Protocols

4.2.1 The Terminal IMP does not guarantee to issue CLS commands in response to "unsolicited" RFCs. There are currently several ways to "solicit" an RFC, as follows:

1) A terminal user can tell the Terminal IMP to perform the ICP to the TELNET Logger at some foreign Host. This action "solicits" the RFCs defined by the ICP.

2) A terminal user can send an RFC to any particular Host and socket he chooses. This "solicits" a matching RFC.

3) A terminal user can set his own receive socket "wild." This action "solicits" an STR from anyone to his socket. Similarly, the user can set his send socket "wild" to "solicit" an RTS.

If the terminal IMP receives a "solicited" RFC, it handles it in accordance with the protocol. For unsolicited RFCs, the Terminal IMP maintains a special message queue. When an unsolicited RFC is received, permanent information is placed on the queue if there is room; if not, the RFC is ignored. In the "background", the queue is emptied by constructing the appropriate CLSs.

4.2.2. After issuing a CLS for a connection, the Terminal IMP will not wait forever for a matching CLS. There are two cases:

1) The Terminal IMP has sent an RFC, grown tired of waiting for a matching RFC, and therefore issued a CLS.

2) The Terminal IMP has sent a CLS for an established connection (matching RFCs exchanged).

In either of these cases the Terminal IMP will wait for a matching CLS for a "reasonable" time (probably 30 seconds to one minute) and will then "forget" the connection. After the connection is forgotten, the Terminal IMP will consider both sockets involved to be free for other use.

Because of program size and table size restrictions, the Terminal IMP assigns socket numbers to a terminal as a direct function of the physical address of the terminal. Thus (given this socket assignment scheme) the failure of some foreign Host to answer a CLS could permanently "hang" a terminal. It might be argued that the Terminal IMP could issue a RST to the offending Host, but this would also break the connections of other terminal users who might be performing useful work with that Host.

4.2.3 The Terminal IMP ignores all RET commands. The Terminal IMP cannot buffer very much input from the network to a given terminal because of core size limitations. Accordingly, the Terminal IMP allocates only one message and a small number of bits on each connection for which the Terminal IMP is the receiver. The Terminal IMP attempts to keep the usable bandwidth as high as possible by sending a new allocation, which brings the total allocation up to the maximum amount, each time that:

1) one of the two buffers assigned to the terminal is empty, and

2) the allocations are below the maxima.

Thus, if a spontaneous RET were received, the reasonable thing for the Terminal IMP to do would be to immediately issue a new ALL. However, if a foreign Host had some reason for issuing a first spontaneous RET, it would probably issue a second RET as soon as it received the ALL. This would be likely to lead to an infinite (and very rapid) RET-ALL loop between the two machines, chewing up a considerable portion of the Terminal IMP's bandwidth. Since the Terminal IMP can't "afford" to communicate with such a Host, it ignores all RETs.

4.2.4. The Terminal IMP ignores all GVB commands. Implementation of GVB appears to require an unreasonable number of instructions and, at the moment at least, no Host appears to use the GVB command. If we were to implement GVB we would always RET all of both allocations and this doesn't seem very useful.

4.2.5. The Terminal IMP does not handle a total bit-allocation greater than 65,534 (2^{16}-2) correctly. If the bit-allocation is ever raised above 65,534 the Terminal IMP will treat the allocation as infinite. This treatment allows the Terminal IMP to store the bit allocation for each connection in a single word, and to avoid double precision addition and subtraction. Our reasons for this decision are:

1) We save more than 100 words of memory which would be required for allocation tables and for double precision addition/subtraction routines.

2) Our experience indicates that very few Hosts (probably one at most) ever raise their total bit allocation above 65,534 bits.

3) We expect that any Host that ever raises its bit allocation above 65,534 would probably be willing to issue an infinite bit allocation if one were provided by the protocol. Once the bit allocation is greater than about 16,000, the message allocation (which the Terminal IMP handles correctly) is a more powerful method of controlling network loading of a Host system than bit allocation.

4.2.6 ECO's are handled correctly if they are small in number. If there is room in the special message queue (which is also used for RST, RRP, NXR, NXS, and CLS in response to an unsolicited RFC), the ERP is entered into the queue; otherwise it is discarded.

4.2.7 INSs and DMs are counted only mod4 with the counter nominally at Ø and INSs decrementing it and DMs incrementing it. When the counter is at 2 or 3, output is suppressed. Thus, if there are a large number of TELNET SYNCHs in progress (to a single port) simultaneously, some of the SYNCHs will not have been handled correctly.

The reason for this method of implementation is that protocol requires a potentially infinite counter to keep track of INSs that have not yet been matched. Further, since this method will always fall back in synch when the matching DMs eventually do come in, the only effect of this is occasionally not suppressing output when it should have been.

4.2.8 Host-Host protocol allows only one outstanding message on the control link between the TIP and any given Host. The TIP packs up all requested control commands to a Host at the time it sends such a message, but even so delays arise. A send-allocate request for a device must wait for the RFNM to return if a link Ø message has already been sent. For large round-trip times and/or many devices connected to the same Host, such delays can become noticeable as stutter.

The TIP allows more than one message outstanding on link Ø. To maintain an ability to retransmit allocates, the TIP uses an extension to protocol which identifies a "sublink" on link Ø. Allocates are sent with saved, identifiable sublinks of 1-7; other control commands are sent on sublink Ø, or else no record is kept of a sublink. RFNMs and INCs are returned with the same sublink information so that the TIP can take appropriate action.

5. References and TIP Bibliography

[1] NIC 8246, Host/Host Protocol for the ARPA Network, January 1972.

[2] NIC 7101, Official Initial Connection Protocol, June 1971.

[3] NIC 9348, Ad Hoc Telnet Protocol, April 1972.

[4] The BBN TIP Hardware Manual, BBN Report No. 2184.

[5] Specifications for the Interconnection of Terminals and the Terminal IMP, BBN Report No. 2277.

[6] Terminal Interface Message Processor User's Guide, BBN Report No. 2183.

[7] S.M. Ornstein et al, The Terminal IMP for the ARPA Computer Network, Proc. AQIPS 1972 SJCC., pp. 243-254.

[8] N.W. Mimno et al, Terminal Access to the ARPA Network: Experience and Improvements, Proc. Seventh Annual IEEE Computer Society International Conference, pp. 39-43.

[9] R.E. Kahn, Terminal Access to the ARPA Computer Network, Computer Networks, Courant Computer Symposium 3, Courant Institute, New York.

[10] TIP User's Group Notes, Available from the Network Information Center, Stanford Research Institute, Menlo Park, California 94025.

6. Storage Layout

The TIP occupies upper core in Honeywell 316 IMPs, starting at location 37777. The version number of the running TIP system appears in location 37777 (also 40000 for historical reasons); it takes the form NNNMM (octal) where NNN is the basic TIP system version and MM is the patch version, if any.

TIP code follows to approximately 60000 (octal), varying somewhat from version to version. Most tables follow the code; a few are interspersed with code for efficiency or other reasons.

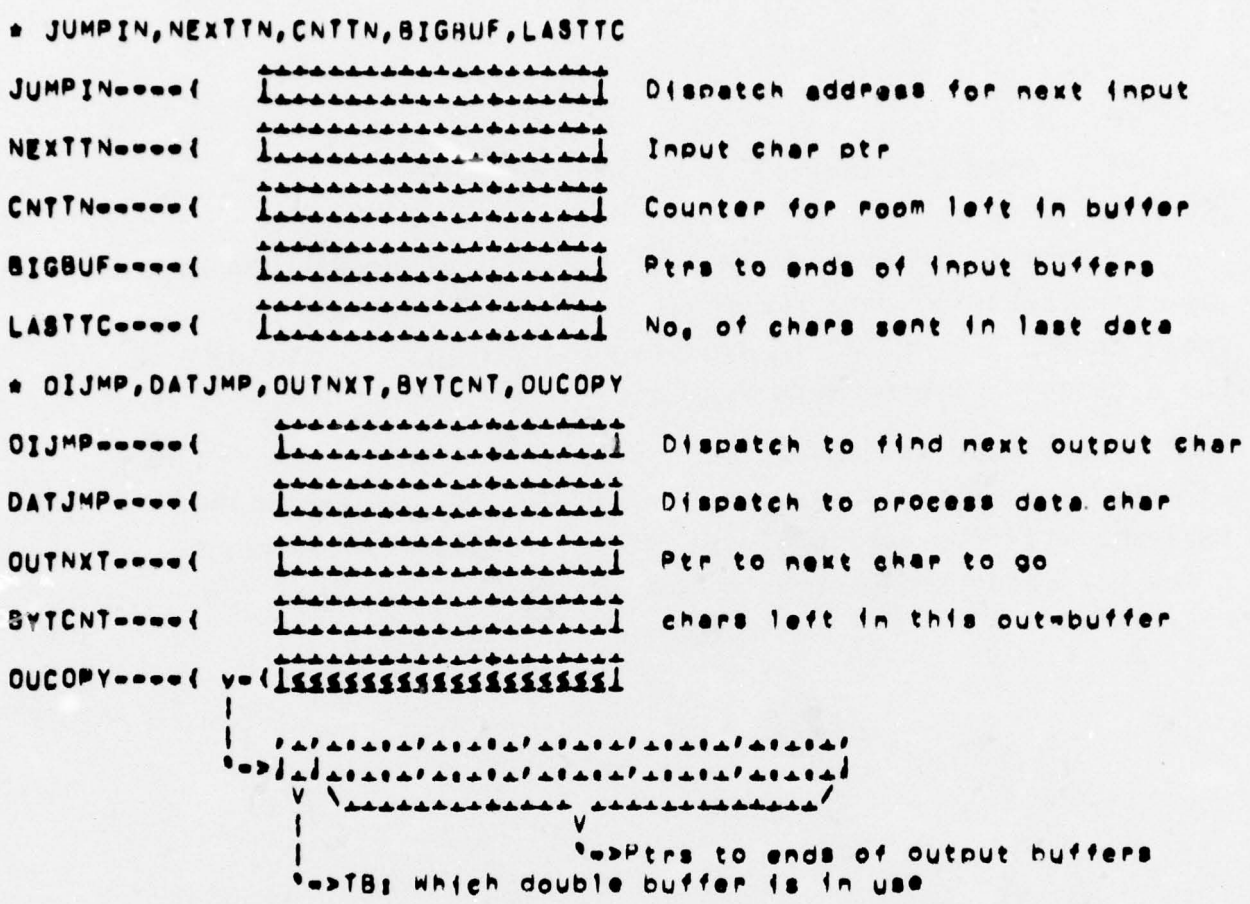
The remainder of the space is allocated to terminal buffers in two blocks, for input and output respectively. Within each block, the terminal buffers are fixed and contiguous; that is, the buffer for device 6, say, follows the buffer for device 5 and is directly followed by that of device 7. The specific allocation of space among terminals is determined on a site basis and loaded with the system as part of a site specific parameters file. This buffer assignment may be changed at any time but does require reassembly of the parameters file and reloading of the TIP.

The magnetic tape option, if present, occupies locations 70000 to 77777 for its buffers and code.

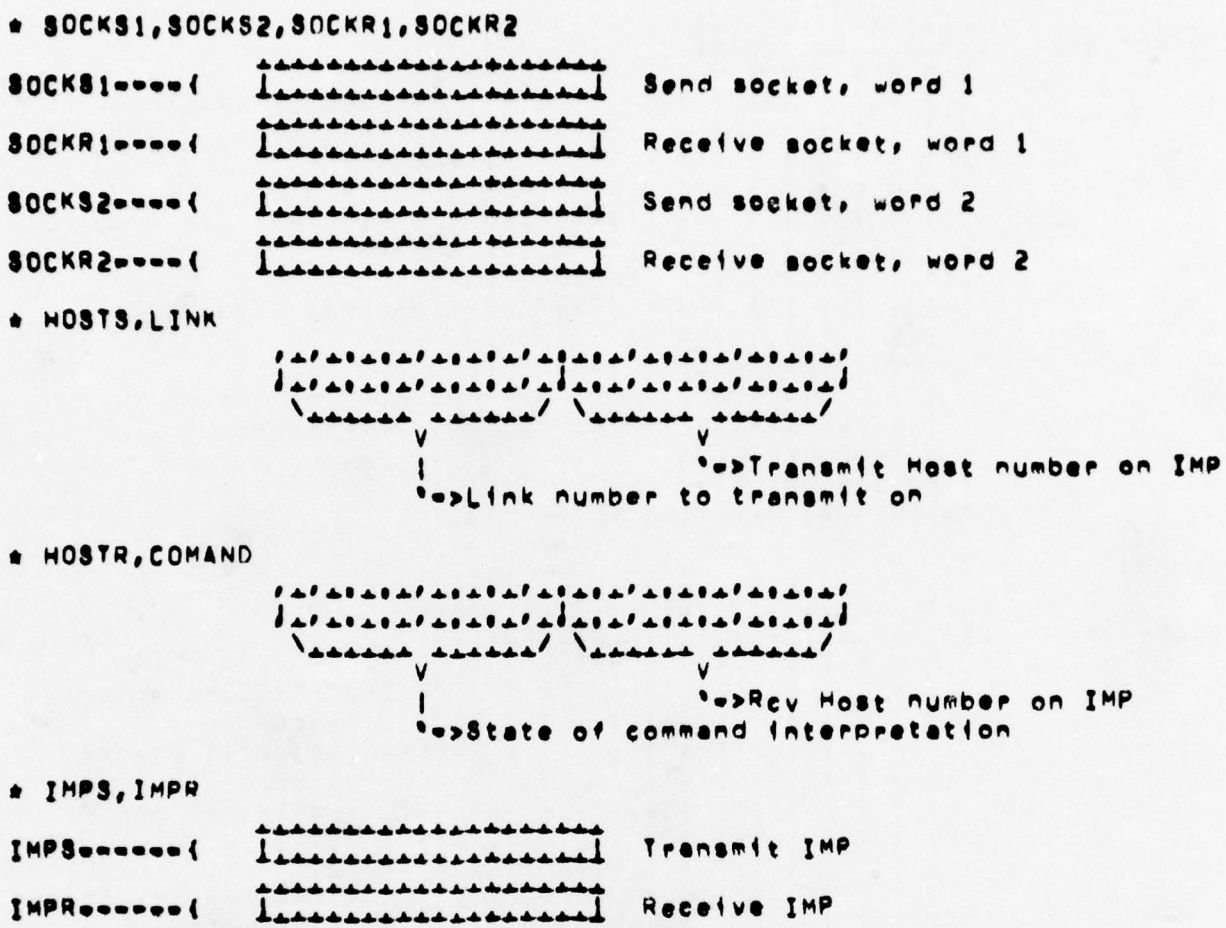
7. Data Structures

Following are the formats of the individual entries of every important table or data structure in the TIP software system. These charts are useful in studying the TIP system listing and the detailed software descriptions.

The tables are invariably sixty-four entries long or, as is occasionally necessary, made up of two or more contiguous tables sixty-four entries long. Often several tables only a few bits wide are packed in one word.



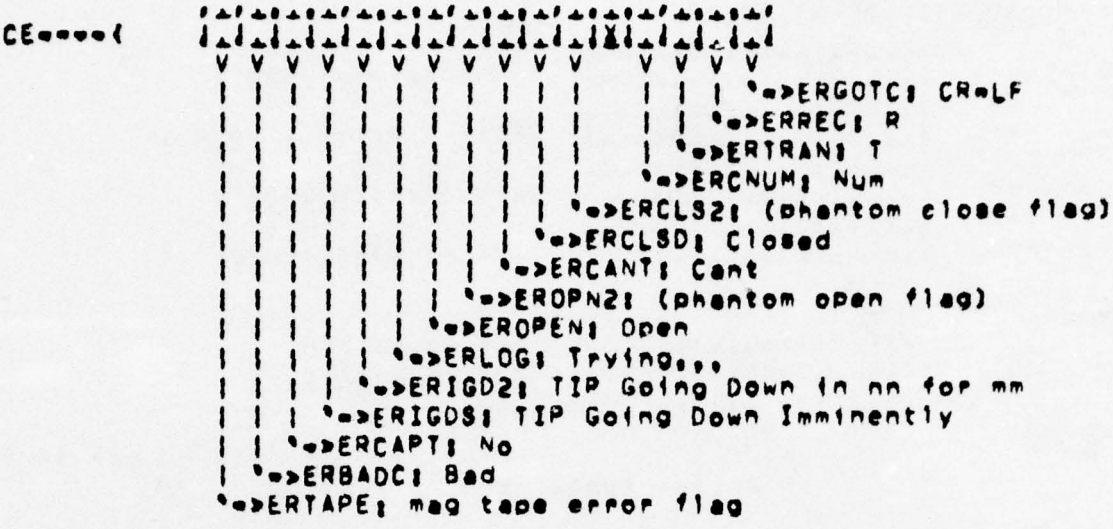
BEST AVAILABLE COPY



BEST AVAILABLE COPY

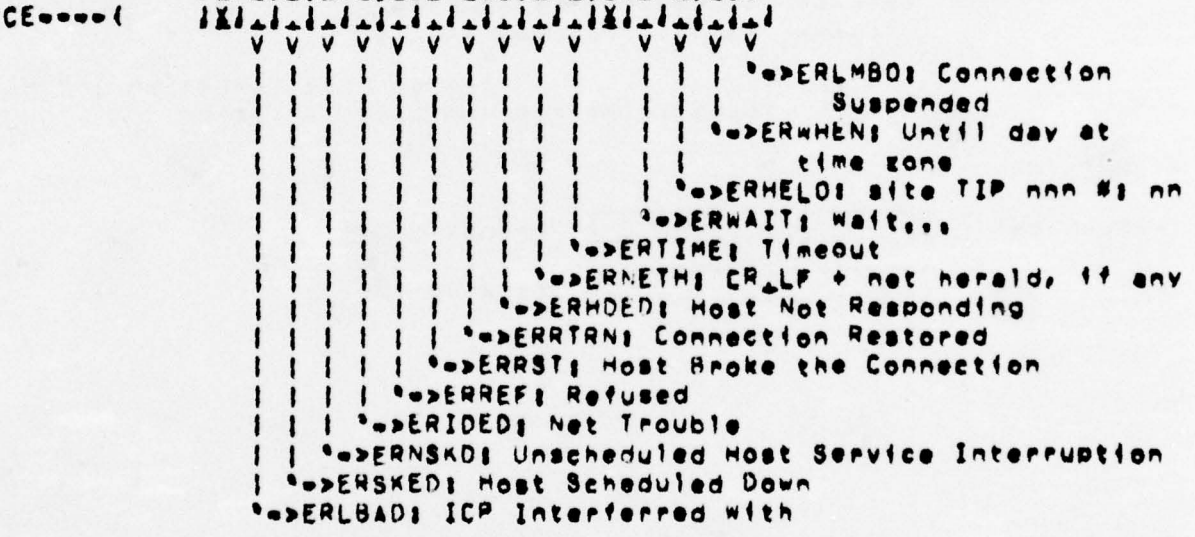
```
* ERROR

DEVICE----(
```




```
* ERROR2

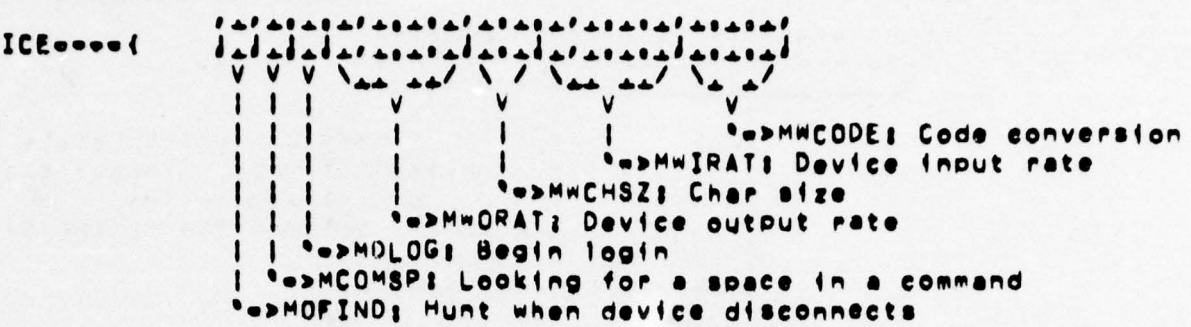
DEVICE----(
```



BEST AVAILABLE COPY

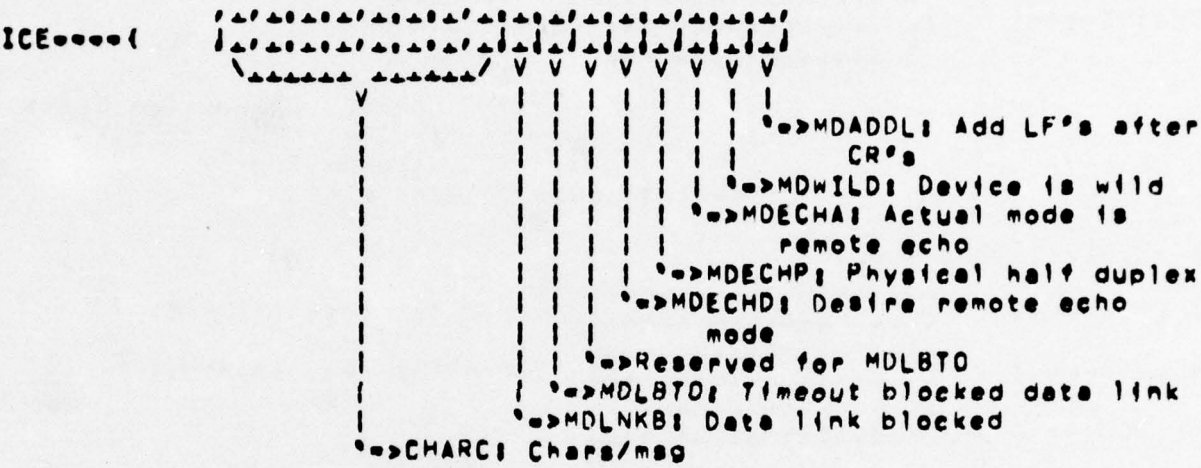
```
* RATE,CODE,SIZE,RBITS

DEVICE----(
```

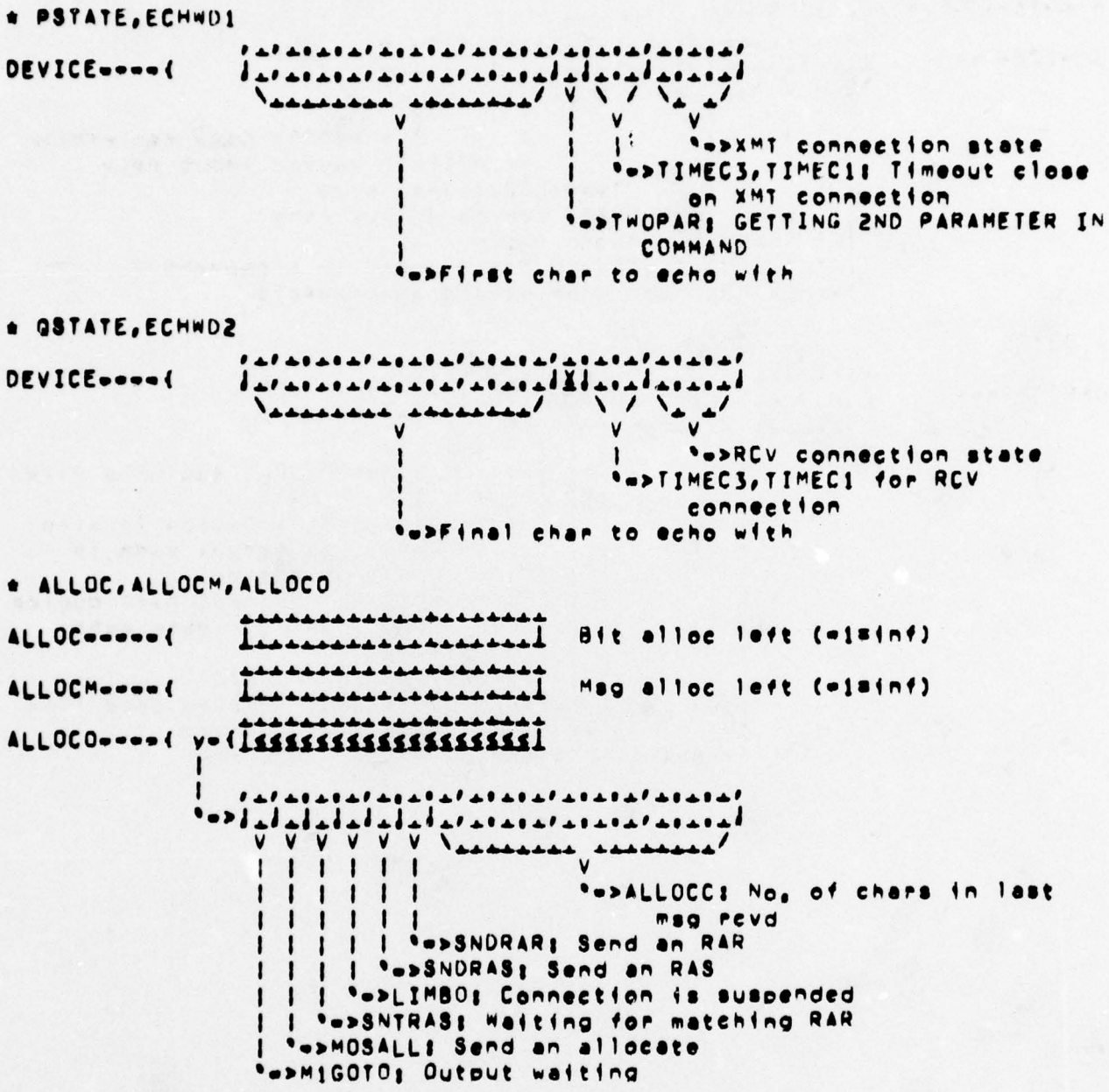



```
* DBITS

DEVICE----(
```

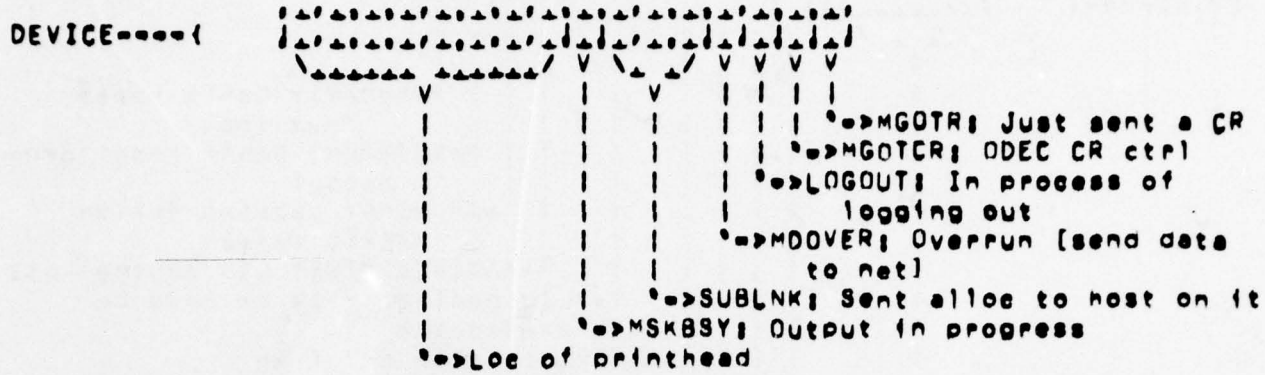


BEST AVAILABLE COPY

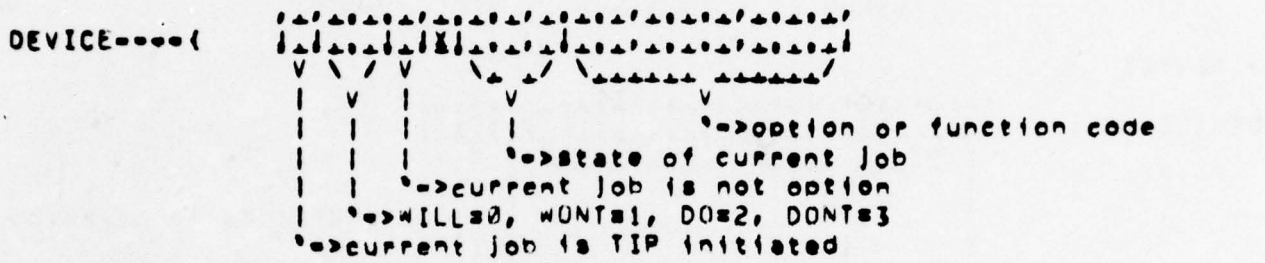


BEST AVAILABLE COPY

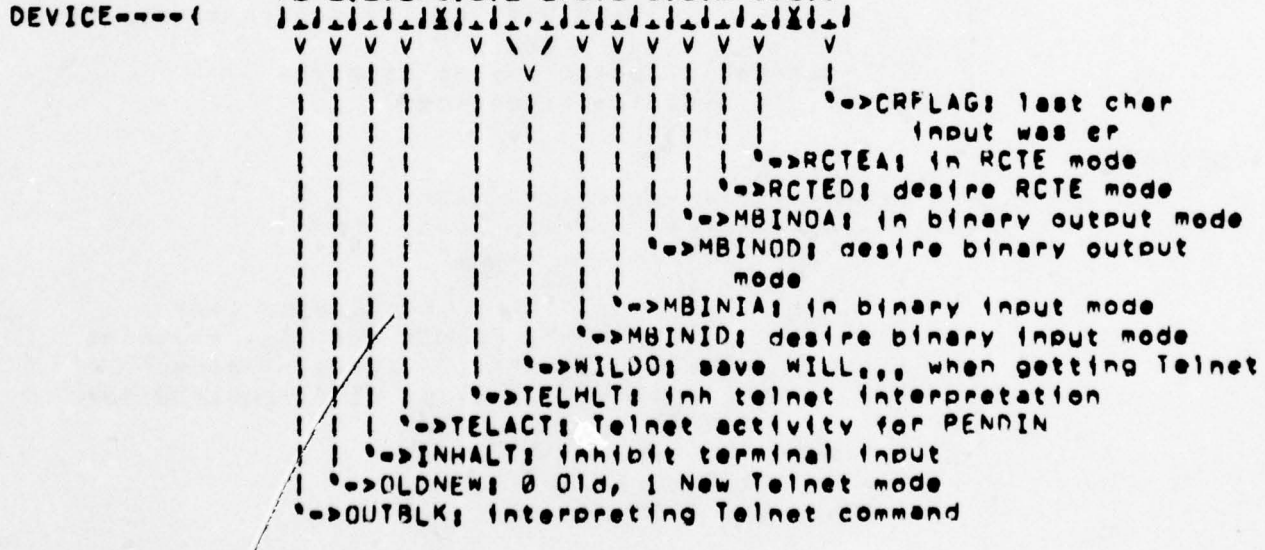
* M



* TELDIS

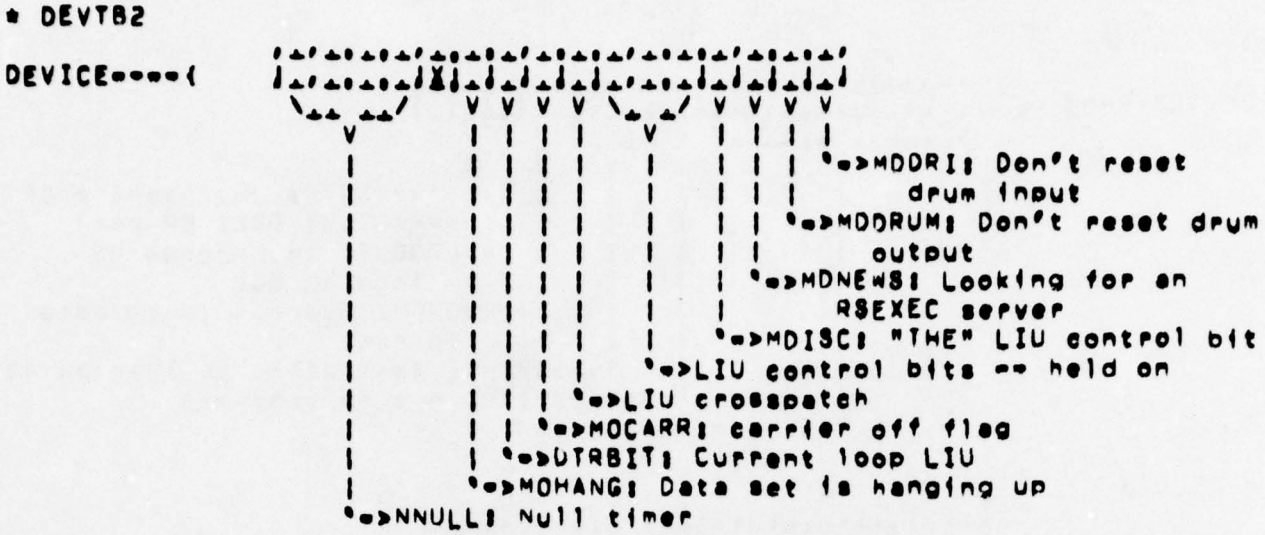


* DEVTBI

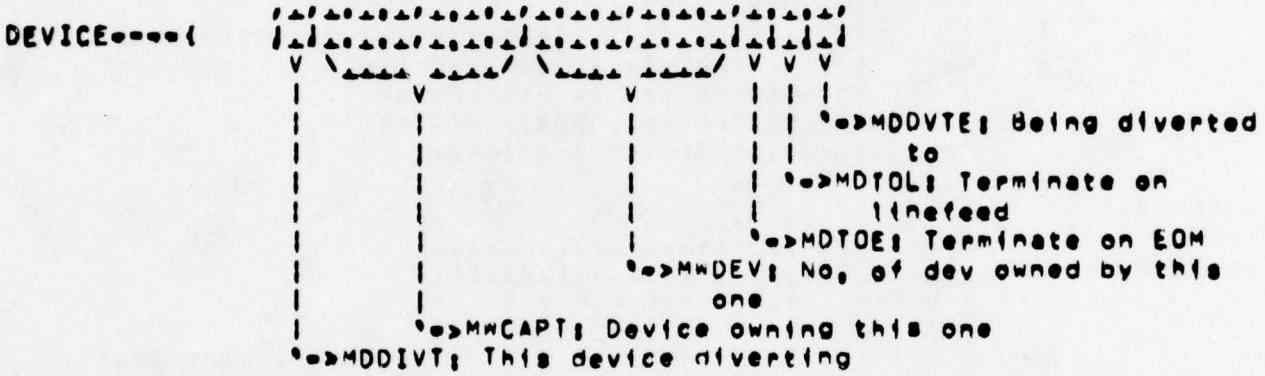


BEST AVAILABLE COPY

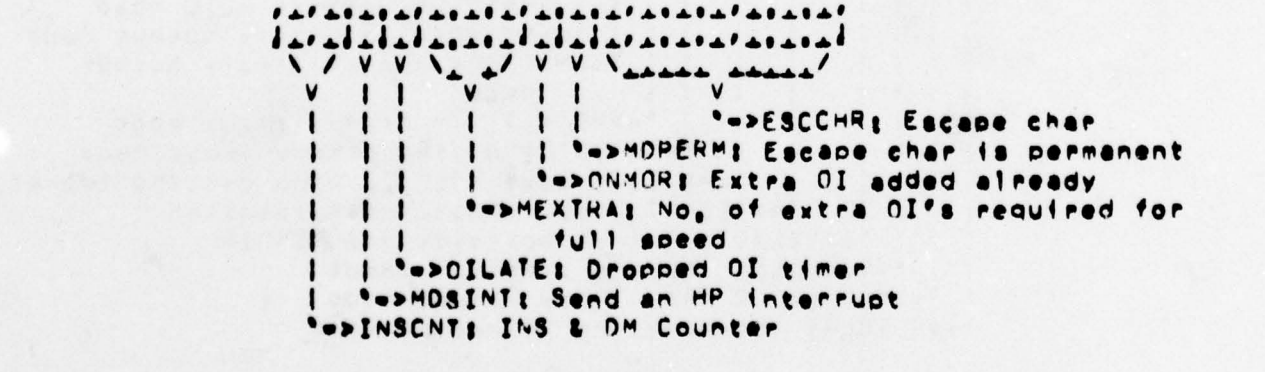
* DEVTB2



* DEVTB3

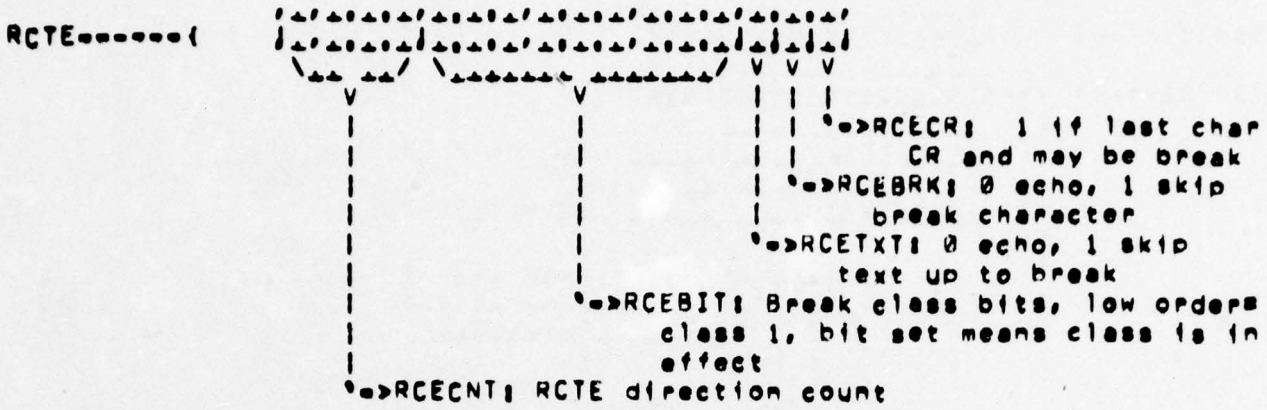


* DEVTB5

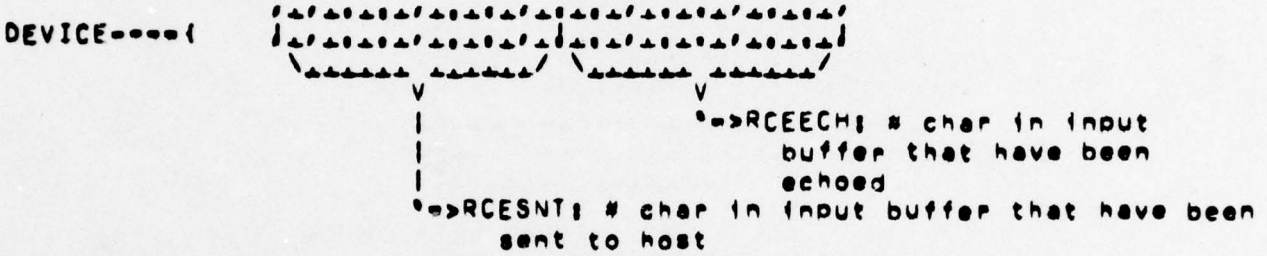


BEST AVAILABLE COPY

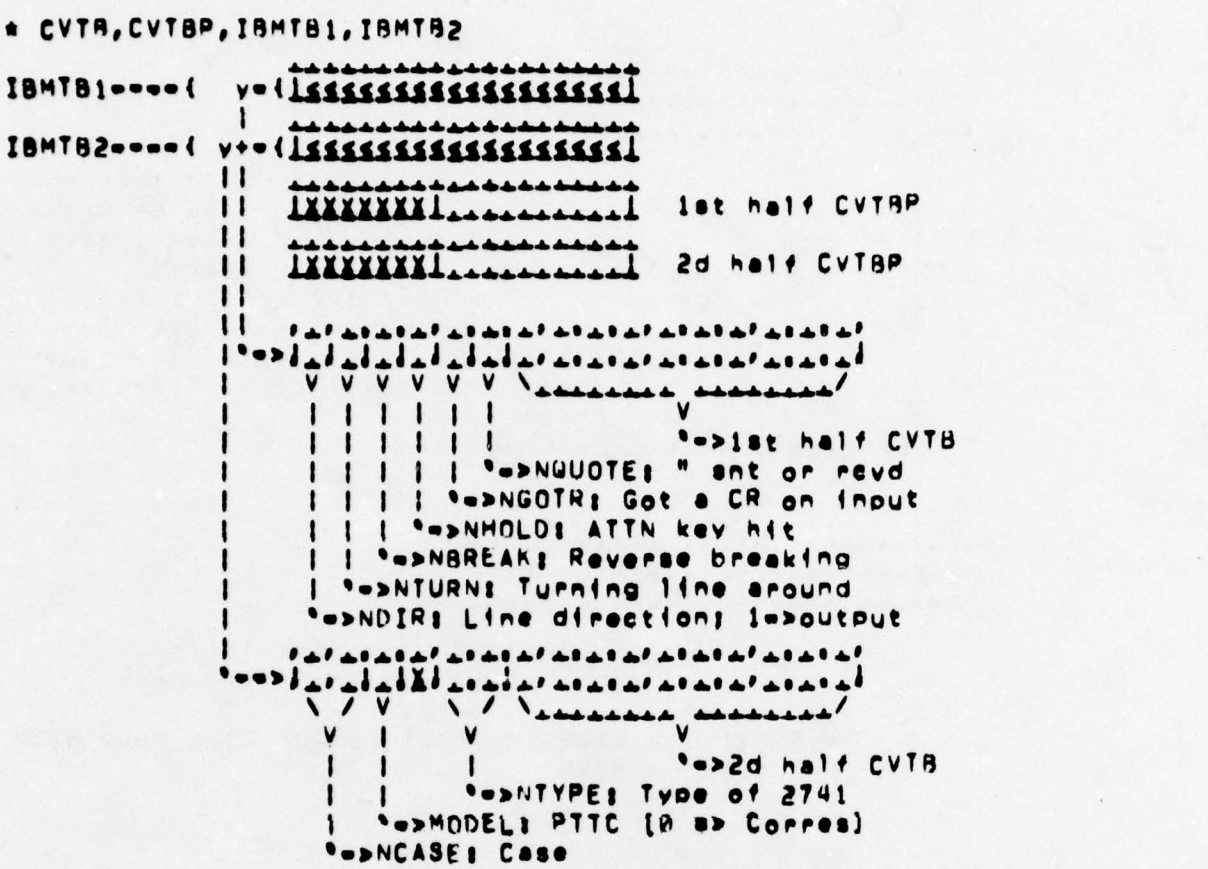
* NN



* RCECHO



BEST AVAILABLE COPY



BEST AVAILABLE COPY

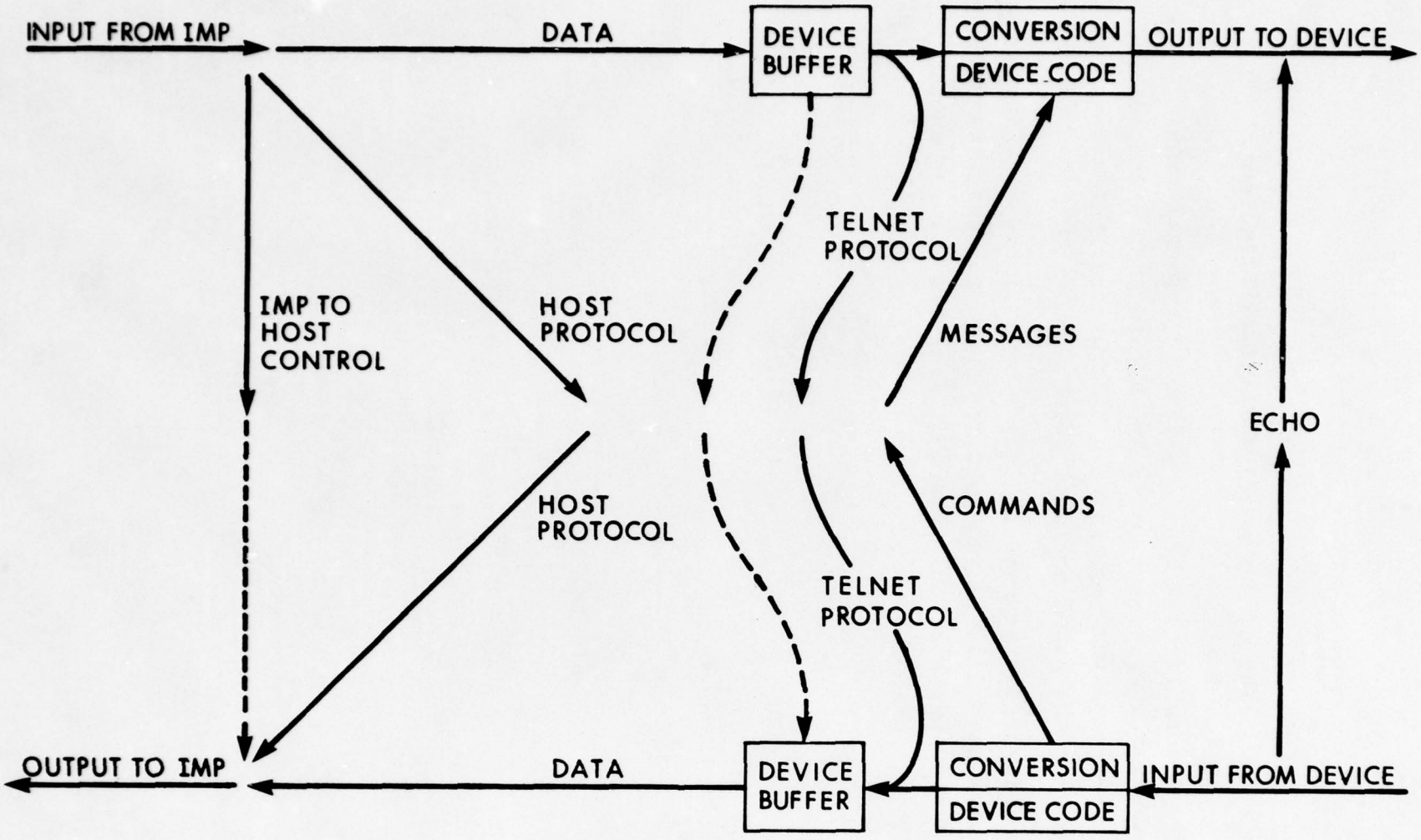
8. Detailed Software Description

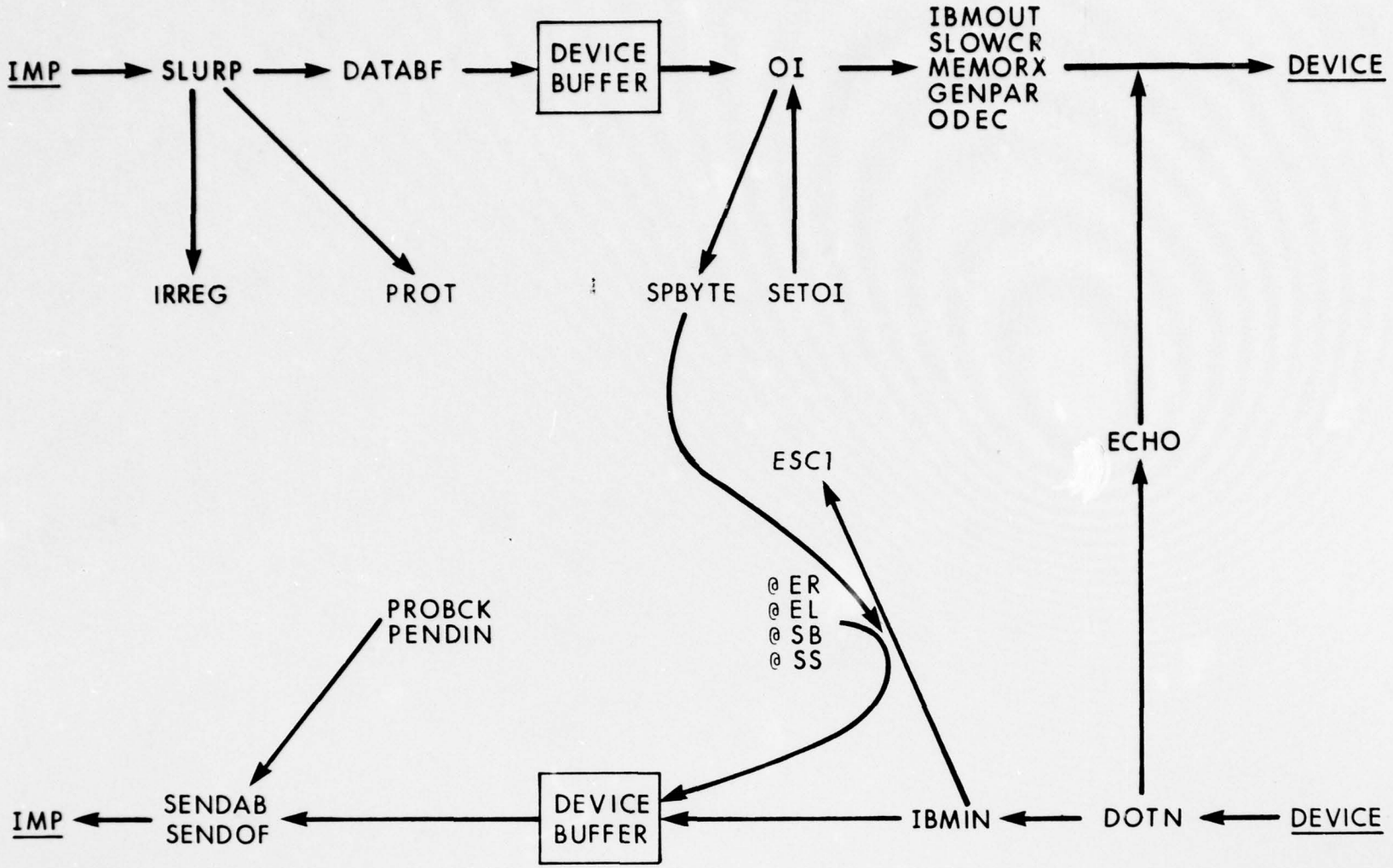
Section 8.1 gives an overview of the various functional modules in the TIP program and the main routines which perform these functions. Section 8.2 contains the bulk of the material in Section 8, the detailed software descriptions themselves. Section 8.3 is an index of the "labels" used in Section 8.2.

8.1 Outline of Program's Functional Structure

The figure gives a broad outline of the data paths in the IMP system software. Data paths are marked with directed arcs. A few important paths (marked with dashed arcs) which control the flow over the paths are also shown.

The figure after next indicates some of the routines along the data paths and serves as an overview of the detailed descriptions of the next section. The following two figures should be studied together.


INPUT FROM IMP
DATA
DEVICE BUFFER
CONVERSION
DEVICE CODE
OUTPUT TO DEVICE
TELNET PROTOCOL
IMP TO HOST CONTROL
HOST PROTOCOL
MESSAGES
ECHO
HOST PROTOCOL
COMMANDS
TELNET PROTOCOL
OUTPUT TO IMP
DATA
DEVICE BUFFER
CONVERSION
DEVICE CODE
INPUT FROM DEVICE


IMP
SLURP
DATABF
DEVICE BUFFER
OI
IBMOUT
SLOWCR
MEMORX
GENPAR
ODEC
DEVICE
IRREG
PROT
SPBYTE
SETOI
ECHO
ESC1
@ ER
@ EL
@ SB
@ SS
PROBCK
PENDIN
IMP
SENDAB
SENDOF
DEVICE BUFFER
IBMIN
DOTN
DEVICE

8.2 Detailed Descriptions

The detailed descriptions in the rest of this section are a mixture of descriptive prose and diagrams. Very few of the diagrams are precise flow charts of the actual code; instead, the diagrams vary from functional descriptions of what the routine under consideration does, to state diagrams for the conceptual finite state machines that the routine implements, to logic diagrams for the routine. Study of the remainder of this section assumes concurrent study of the program listing and the formats of the program data structures. In the following diagrams, wherever reasonable, the diagram labels are keyed to the program labels found in the listing for convenient cross-referencing.

Many of the diagrams are given in an ALGOL-like notation where labels appear in the left column and "goto"s and "if"s are used to indicate branches in program flow. This notation is used instead of a more conventional two-dimensional notation to facilitate computer maintenance of the diagrams. In the absence of "goto"s, "call"s, etc. program flow is assumed to go down the page and from one page to the next.

The program descriptions in the rest of this section are generally organized by program priority level (initialization, MLC clock interrupt, background, MLC output interrupt, and mag tape option) as shown in the following hierarchical outline. Again, this organization is to facilitate cross-referencing the descriptions with the listing.

8.2.1	Initialization
8.2.2	MLC CLOCK Level
8.2.2.1	MLC Input
8.2.2.1.1	Packaging Data into INPUT Buffers
8.2.2.1.2	Echo initiation
8.2.2.1.3	User command processing
8.2.2.1.3.1	Command interpretation
8.2.2.1.3.2	Command execution
8.2.2.1.4	Rate initialization for "hunting" devices
8.2.2.1.5	Input Conversion
8.2.2.1.5.1	EBCDIC to ASCII Conversion Logic
8.2.2.2	MLC Output (OUTINs)
8.2.2.2.1	2741 Line Control
8.2.2.2.2	TELNET Protocol Handler
8.2.2.2.2.1	FIXECH, FIXECO, FIXJMP
8.2.2.2.3	Echoes
8.2.2.2.4	Messages to devices
8.2.2.2.5	Output conversion
8.2.2.2.5.1	ASCII to EBCDIC Conversion Logic
8.2.2.2.5.2	Slow Carriage Return
8.2.2.2.5.3	Even Parity
8.2.2.2.5.4	Line Printers
8.2.2.2.5.4.1	MEMORX
8.2.2.2.5.4.2	ODEC
8.2.2.3	Background Timer (BTIME)
8.2.3	Background
8.2.3.1	Connection and Host Functions (PENDIN)
8.2.3.1.1	Device Related Functions
8.2.3.1.1.1	Process Telnet queue entries
8.2.3.1.1.2	Transfer data to IMP
8.2.3.1.1.3	Allocates
8.2.3.1.1.4	News

8.2.3.1.2	Host related functions
8.2.3.1.2.1	Send INS's
8.2.3.1.2.2	Send RASs and RARs
8.2.3.1.2.3	Check Reset Table for New RST to go
8.2.3.1.2.4	Send Messages from Special Message Queue
8.2.3.2	Connection Control (PROBCK)
8.2.3.3	Initial Connection Protocol (Logger)
8.2.3.3.1	Timeout missing events: TIMCLS
8.2.3.4	Send data to IMP
8.2.3.4.1	Link 0
8.2.3.4.2	Not link 0
8.2.3.5	Accept Data from IMP (SLURP)
8.2.3.5.1	Irregular Messages
8.2.3.5.2	Regular Messages
8.2.3.5.2.1	Link 0 messages
8.2.3.5.2.1.1	Handling RFCs and CLSs
8.2.3.5.2.1.2	Handling ECO, RST, RRP
8.2.3.5.2.1.3	Handling ALLOCATES
8.2.3.5.2.1.4	Counting INSs
8.2.3.5.2.1.5	Handling RASs, RARs, and RAPs
8.2.3.5.2.1.6	Handling NXR and NXS
8.2.3.5.2.2	Message Not On Link 0
8.2.3.5.2.2.1	OUNEW
8.2.3.5.2.3	Accounting Data
8.2.3.6	Modem and LIU Control
8.2.3.7	Dump Requests
8.2.4	MLC Output Interrupt Routine
8.2.5	Magnetic tape option
8.3	Index to detailed descriptions

8.2.1 Initialization

Initialization comes in two classes: complete TIP initialization (done by the routine GOGGLE), and device initialization (done by the routine RESET).

GOGGLE is called at the top of the TIP background loop if the TIP (re)initialization flag (TIPFLG) is set. RESET is called by GOGGLE as part of complete TIP initialization, and by MODEMC, HUNT, and RC for device initialization, the latter being the RESET command.

GOGGLE shut off TIP interrupts

reset the MLC

set up Page Ø literals

clear special message queue

clear Reset Table

mark to send RSTs to all news Hosts

initialize device stuff including
call to RESET

zero various flags

initialize MLC output buffer pointers

initialize various co-routines

initialize various timers

set up correct priority
interrupt mask for clock level

initialize MLC IN and OUTIN buffer pointers

initialize Mag tape option if it exists

set up interrupt entrances

return from GOGGLE

The subroutine RESET, following, initializes a device. It is called by various routines, including: GOGGLE (initialization), MODEMC, HUNT, and the user RESET command. RESET makes a distinction between hunting and non-hunting devices; the latter keep certain parameters set even through a reset. On entry, the device to be reset is indicated by the index register, and the AC is negative if "HELLO..." should not be typed out.

RESET performs the following functions for all devices, hunting or not:

- initializes internal status bits (allocates, blocked links, etc.);
- sets the escape character to the default unless marked as permanent;
- initializes connections (allocates and links) by calling SHUTDN if not open, or if an open connection exists, initiates a close;
- aborts any login, open, or news request in progress;
- initializes buffer pointers and count;
- releases any devices captured by this one, and uncaptures this device if captured;
- sets up the appropriate dispatch for input and output characters;
- puts the terminal in Old Telnet protocol mode.

In addition, RESET performs the following functions for hunting devices only:

- changes the device rate to the nominal hunting rate;
- resets the "connection" parameters to their default condition of not wild, non-binary, transmit on every character (and not specifically on an EOM or EOL), local echo mode, will accept remote echo mode, and add a linefeed after a carriage return.

At the moment, only the call from MODEMC suppresses the HELLO message.

```
RESET     if device is invalid, return from
          RESET

          initialize status bits

          put into Old Telnet mode

          if escape character is not permanent
          reset escape character to @

RESET5    initialize (close) connections

          initialize buffer pointers
          and count

          uncapture the device and any
          other devices it may have captured

          if device is hunting device,
          goto RESETa

          set up correct input and
          output dispatches

          goto RESET2


RESETa    set the rate to hunting

          initialize the connection
          parameters

          set input dispatch to HUNT
```

```
RESET2    if HELLO message is suppressed,
          return from RESET

          convert device # to ASCII and
          place in network data buffer

          if device is non-hunting, mark
          to type out "LATEST NET NEWS...",
          if any

          Mark to type out "HELLO..."
          followed by network data

          Return from RESET
```

8.2.2 MLC CLOCK Level

The MLC clock level routine, CLOCK, is mostly a straight series of calls to other clock level routines and inline code.

CLOCK	save registers and keys and mask
	set up new mask
	if 40 ticks of PEND clock have not gone by, goto CLOCK4
	reset PEND clock
	reset LOG clock
	reset PEND flag
	mark that TIP is ready (for NCC)
	increment data set control timer
CLOCK4	do MLC input
DOTN2	pack data into device input buffers

```
OOPS    call CLKOI

        call BTIME

        if output not busy, call TOUT
        to restart it

        restore registers, mask, and keys

        return from CLOCK
```

8.2.2.1 MLC Input

The inline code following CLOCK4 in CLOCK performs MLC input as shown.

CLOCK4 swap buffers

set up MLC input
buffer pointers to
new input buffer

set up pointer
to input buffer to
be emptied

goto DOTN2

8.2.2.1.1 Packing Data into INPUT Buffers

The input routine starting at DOTN is called on an interrupt basis. Characters input to the MLC are stored in a (double-buffered) tumble table where each entry consists of the character and an index to the originating device. Each clock time of 3.3 msec, one of these tumble tables is processed by the input routines, character-by-character. The disposition of input is indicated by the JUMPIN table which contains a dispatch (jump instruction) for each device; specific dispatches are detailed in the following flow charts. Generally they distinguish from each other: data, commands, echoing or not echoing, ASCII input and input requiring code conversion, input considered 8-bit binary, and the special case of the initial character(s) from a hunting device. Clearly, the dispatch changes as a user sets up and then executes a dialogue with a Host system.

DOTN — increment to get next entry from tumble table

DOTN2 — pick up a character and device number from tumble table and save them

if character is a break, goto BREAK

dispatch via table entry for this device to one of CONECO, CONVT, CONEEE, CONESC, IBMEEE, IBMESC, IBMECO, IBMCON, BINECO, BINCON, or HUNT

BREAK

if device is still attempting to hunt,
goto HUNT

if device is a 2741 and TIP site supports them,
goto IBMBRK

BREAK3

if device does not have an open transmit
connection, goto BREAK1

convert character to appropriate TELNET break
character

goto NOPE (old Telnet) or DOTN (New Telnet)

BREAK1

if device is non-hunting, goto DOTN

call RESET

goto DOTN

IBMBRK

if 2741 is in input mode (user can type),
then goto BREAK3

if 2741 is in output mode and the line is
trying to turn to input mode,
then goto BREAK1

goto IBBRAK

CONECO	call ECHO to echo character
CONVT	mask character to 7 bits
	goto NOPE
BINECO	call ECHO to echo character
BINCON	if in Old Telnet mode, goto NOPE2
	if in New Telnet mode, goto NOPE15
IBMEEE IBMESC	call IBMIN to convert character and do protocol
	goto ESC
IBMECO IBMCON	call IBMIN to convert character and do protocol
	goto REG

REG	if device has suppressed intercepting commands, goto REGa
	if character is the device's escape character, goto ESCAPE
REGa	if connection is in RCTE mode, goto RCTEIN
	if character is an LF, goto FEED
	if character is an EOM, goto EOM
NOPE	if device is using New Telnet, goto NOPE1Ø
NOPE2	if there is no space in buffer, goto ECHBEL
	store the character in buffer
	if there is no space left, call SENDIT to mark it to be sent to net
ADDLF	if not in add LF mode, goto DOTN
	if character is a CR, change character in tumble table to a LF
NOPE5	save character
	if device half-duplex, echo it
	dispatch via table entry for this device to one of CONECO, CONVT, CONEEE, CONESC, IMBEEE, IBMESC, IBMECO, IBMCON, BINECO, BINCON

ECHBEL	don't echo the character, but send out a BEL code (upshift for IBMs) to show character was discarded
	goto DOTN
FEED	if device not set to transmit on LF, goto NOPE
	goto EOMa
EOM	if device not set to transmit on EDM, goto NOPE
EOMa	set up counter to make buffer look full
	goto NOPE
RCTEIN	if character is not a break character, goto NOPE
	start any output or echo to the terminal by calling OUNEW
	goto EOMa

CONEEE	call ECHO to echo character
CONESC	mask character to seven bits
ESC	if character is a CR, goto ADDLF
	call ESC1 to interpret command character
	if return 1 from ESC1, goto DOTN
ESC2	set up table dispatch to expect data
	if character is an esc which is to be data, goto REGa
	goto DOTN
NOPE1Ø	if holding off input, goto ECHBEL
	if char is not a CR, goto NOPE2Ø
	if not $\geq$2 spaces in buffer, goto ECHBEL
	store CR in buffer
	change character in tumble table to LF if in add LF mode, or to NULL if not
	goto NOPE5

```
NOPE15      if holding off input, goto ECHBEL

            if character is IAC, double it

            if not >2 spaces in buffer,
            goto ECHBEL

            store extra IAC in buffer

            goto NOPE3Ø


NOPE2Ø      if there is no space in buffer,
            goto ECHBEL

NOPE3Ø      store character in buffer and
            if no space is left, call SENDIT to
            mark it to be sent to net
```

8.2.2.1.2 Echo initiation

The ECHO subroutine performs local echoing for the TIP. It has table storage space allowing two echo characters to be saved for each device, tables ECHWD1 and ECHWD2. Echo characters take priority over regular data for output.

```
ECHO    if OI dispatch is ECHL,
        goto DOTN

        if OI dispatch is ECHR, reset
        OI dispatch to ECHL

        if OI dispatch is neither ECHL or ECHR,
        set OI dispatch to ECHR

        copy ECHWD2 to ECHWD1

        put new character in ECHWD2

        restart output to the terminal
        by calling OUNEW

        return from ECHO
```

8.2.2.1.3 User command processing

8.2.2.1.3.1 Command interpretation

The subroutine ESC1 interprets local TIP commands. Each time through, ESC1 interprets one character which must be in the AC when it is called. The command status of each device is saved in a table called COMAND and converted to the temporary parameter COM when ESC1 is called. Only the first character of a word is significant; the command @INSERT LINEFEED, for example is equivalent to @I L. Extra spaces and the rest of each word are ignored by ESC1. By default, a command affects the device from which it was given. If, however, a command is preceded by a number, as in @2 I L, a command will affect the device so designated (in this case device 2). In this case the affected device is captured by the commanding device (which may be itself) and cannot be commanded by any other device until it is "given back" with a GIVE BACK or RESET command. In general, there are two kinds of commands, those with and those without parameters. Once the command is accepted and interpreted, ESC1 dispatches to specific code for each command. Parameters, if any, follow the command itself (e.g., @L 69) and are interpreted by this specific command code. Descriptions of these specific commands appear in the next section.

```
ESC1    initialize SKIPS, LFFLG; save character

        if mag tape command, goto MC

        look for a special character

        if character is not a space,
        goto ESC1a

        ignore character; reset flag

        return 1 from ESC1 to wait for next
        character
```

ESC1a if character is a LF, goto RETURN

if character is TELNET control, return 1
from ESC1 to wait fof next character

if character is rubout, goto EXIT

if character is lower case, convert
it to upper case

```
Q2      look at status now

        if a space is needed, return 1 from ESC1
        to wait for next character

        if a command has already been matched,
        goto PAR

        if we are interpreting a command,
        goto SYNTAX

        if still looking for number, goto Q3

        if character is an esc, goto EXIT

        undivert output

        goto Q3


EXIT    clean up

        return 2 from ESC1 to indicate done
```

```
RETURN    if have nothing yet, goto EXIT

          if haven't got a dispatch yet,
          goto RETURNa

          set up dispatch address

          goto DISP


RETURNa   reset space flag, set LF flag

          goto SSYN1
```

```
PAR         set up parameter
            dispatch

            goto DISPC

                ..

Q3          if character is a number, goto
            Q3a

SSYN1       correct status

            goto SYNTAX


Q3a         accumulate a diverting device

            set status to looking
            for number

            return 1 from ESC1 to wait
            for next character
```

```
SYNTAX      interpret command syntax

Q6          pick up the next character
            to match

            if this one is not to
            be skipped, goto Q6a

            adjust SKIPs flag

            goto Q6


Qba         if not at end of
            tree, goto Q8

            if a parameter is needed, goto  LFCHRa

LFCHR       if character is a LF, goto ERR1

LFCHRa      set up dispatch

DISPC       go do it

            goto EXIT


ERR1        send out "BAD"

            goto EXIT
```

```
Q8      if character is
        LF, goto Q9

        if character doesn't match, goto Q9

        save new status

        return 1 from ESC1 to
        get next character


Q9      if there is more to match,
        goto Q6

        goto ERR1
```

8.2.2.1.3.2 Command execution

Local commands interpreted by the TIP subroutine ESC1 cause the TIP itself to take the actions noted below. Most generally, commands affect either local terminal handling functions or network connection functions. Where parameters or other information are required it is so noted. The commands are listed in alphabetical order and cross reference other commands as appropriate.

BINARY INPUT END
: Leave 8-bit binary input mode; this command must be given from another device since commands are not recognized in binary input mode.

BINARY INPUT START
: Enter 8-bit binary input mode; in New Telnet, set bit for binary input desired, negotiate with Host, and enter mode if Host agrees.

BINARY OUTPUT END
: Leave 8-bit binary output mode.

BINARY OUTPUT START
: Enter 8-bit binary output mode; in New Telnet, set bit for binary output desired, negotiate with Host, and enter mode if Host agrees.

BINARY OUTPUT END
: Leave 8-bit binary output mode.

CLEAR DEVICE WILD
: Set device to be unwild; i.e., stop accepting RFC's from any Host.

CLEAR INSERT LINEFEED
: Stop inserting linefeed after carriage-return.

CLOSE
: Close all outstanding connections or abort current Host login.

DEVICE CODE 37
: Establish parity computation for Model 37 Teletype for output to this device.

DEVICE CODE ASCII
: Establish code conversion for an ASCII terminal.

DEVICE CODE EXTRA-PADDING
: Establish code conversion for a terminal with slow CR; i.e., supply padding for correct carriage control.

DEVICE CODE OTHER-PADDING

Establish code conversion for a line printer, i.e., supply padding for correct carriage control.

DEVICE RATE #

is a 10-bit code specifying hardware rate and character size settings for the device commanded.*

DIVERT OUTPUT

Capture device # and divert this terminal's output to it. # is an octal number.

ECHO ALL

Same as ECHO LOCAL

ECHO HALFDUPLEX

Commands the TIP to expect terminal-generated echo -- TIP echoes nothing except internally generated characters; TIP will refuse New Telnet echo option.

ECHO LOCAL

Commands the TIP to perform local TIP-generated output -- TIP echoes everything. If the connection is open, also sends the Telnet "ECHO LOCAL" command. This mode is the default in both Old and New Telnet.

ECHO NONE

Same as ECHO REMOTE.

ECHO REMOTE

Commands the TIP to request remote Host-generated echo for data -- TIP echoes commands only and not data. If the connection is open, also sends the Telnet "ECHO REMOTE" command. In Old Telnet these actions simply take place. In New Telnet this mode is optional and may be refused by the remote Host; but the TIP will remember that the terminal desires to be in this echo mode, and renew the request when a new connection opens.

ECHO REMOTE CONTROLLED

Commands the TIP to request Remote Controlled Transmission and Echoing (RCTE) mode. TIP echoes commands plus data as instructed by the remote Host. This mode is cleared by typing some other echo command. RCTE mode is available only under New Telnet protocol; it is an option, as discussed in the ECHO REMOTE command.

FLUSH

Delete all unsent characters in this device's input buffer.

GIVE BACK

Release control of captured device #, where # is an octal number.

- - - - - - - - - - - - - - -

* # denotes a decimal number unless otherwise stated.

HOST #
Simultaneous "@S T H" and "@R F H"; manual initialization of send and receive Host parameter to #.

INITIAL CONNECTION PROTOCOL
Start the initial connection protocol.

INSERT LINEFEED
Commands the TIP to insert linefeed after carriage-returns.

INTERCEPT #
Use # as TIP command character instead of 64 [i.e., @].

INTERCEPT ESC
Leave 7-bit binary mode; this command must be given from another device since a device in 7-bit binary mode ignores commands.

INTERCEPT NONE
Enter 7-bit binary mode; the TIP will ignore commands given by this device.

LOGIN #
An obsolete form of OPEN.

M # #
Mag tape command # with argument #.

NETWORK-VIRTUAL-TIP-EXECUTIVE
Connects the user to the Network-Virtual-Tip-Executive.

NEW TELNET
Sets the terminal to use New Telnet protocol; not allowed when a connection is open.

OLD TELNET
Sets the terminal to use Old Telnet protocol; not allowed when a connection is open.

OPEN #
Start the initial connection procedure with Host # to get Telnet connections -- equivalent to "@S T H" followed by "@R F S n" followed by "@I C P", where n is 1 for Old Telnet and 27 octal for New Telnet.

PROTOCOL BOTH
Simultaneous "@P T T" and "@P T R".

PROTOCOL TO RECEIVE
Manually initiate protocol to open a receive connection.

PROTOCOL TO TRANSMIT
Manually initiate protocol to open a transmit connection.

RECEIVE FROM HOST #
Establish Host # parameter for manual initialization of both send and receive Host.

RECEIVE FROM SOCKET #
Establish socket # parameter for manual initialization of the receive connection--socket # is given in octal.

RECEIVE FROM WILD
Equivalent to "@R F S <any>"; Host remains as specified.

RESET
Reinitialize a particular TIP port.

RESET NCP
Resets NCP of the Host parameter associated with this device.

SEND ABORT OUTPUT
Send the New Telnet AO function code; types "Can't" if Old Telnet.

SEND ARE YOU THERE
Send the New Telnet AYT function code; types "Can't" if Old Telnet.

SEND BREAK
Send the Telnet "BREAK" command, appropriate to Old or New Telnet.

SEND COMMAND
Send the command escape character; only in Old Telnet.

SEND ERASE CHARACTER
Send the New Telnet EC function code; types "Can't" if Old Telnet.

SEND ERASE LINE
Send the New Telnet EL function code; types "Can't" if Old Telnet.

SEND INTERRUPT PROCESS
Send the New Telnet IP function code; types "Can't" if Old Telnet.

SEND SYNC
Send the Telnet "SYNC" command and an "INTERRUPT SENDER" message, appropriate to Old or New Telnet.

SEND TO HOST #
Establish Host # parameter for manual initialization of the transmit connection.

SEND TO SOCKET #
Establish socket # parameter for manual initialization of the transmit connection--socket # is given in octal.

SEND TO WILD
Equivalent to "@S T S <any>"; Host remains as specified.

SET DEVICE WILD
Equivalent to the commands "@R F H <any>", "@S T H <any>", "@S T S <any>", and "@R F S <any>", and, in addition, leave these in effect after a connection is closed.

TRANSMIT EVERY #
Send off the input buffer at least every #th character where 0 <# <256.

TRANSMIT NOW
Send off the input buffer now.

TRANSMIT ON LINEFEED
Send the input buffer every time a linefeed is encountered.

TRANSMIT ON MESSAGE-END
Send the input buffer every time an end-of-message is encountered.

8.2.2.1.4 Rate initialization for "hunting" devices

The HUNT routine determines the rate of a device. Certain TIP ports may always have the same devices attached to them, such as line printers, and may be pre-initialized to the particular requirements of those devices. Other TIP ports, however, will be used by very simple devices or by different devices, as in the case of modems: these ports are called "hunting." The first character typed on such a device should be the hunt character for that device (see TIP User's Guide). Hunting devices are initially (and after an @R command) set to a nominal rate of 134.5 baud; the expected character will thus vary depending on the actual rate of transmission. The HUNT routine compares the first character with the various possibilities to determine the correct rate.

HUNT compare character with expected responses

if no match, goto HUNTa

call RESET

set up the right rate & code*

adjust the table dispatch accordingly

goto DOTN

HUNTa wait for another character, try again

goto DOTN

* This includes the following:
for a hunt to 15Ø baud, the port is set to @D C 3
for a hunt to 3ØØ baud, the port is set to @D C E

8.2.2.1.5 Input Conversion

8.2.2.1.5.1 EBCDIC to ASCII Conversion Logic

At present the TIP supports only one input conversion, from EBCDIC to ASCII. This conversion package is by default loaded with the TIP, but is optional on a site by site basis. A site preferring to use the space for buffers may so request the NCC. This section discusses the logic for converting from EBCDIC to ASCII for four types of PTTC and four types of Correspondence model 2741 terminals.

The EBCDIC to ASCII code conversion is handled by the routine called IBMIN using the main conversion tables labeled TAB1 and TAB1P and the auxiliary tables labeled TAB2, TAB3, and DISPIN, along with some state information in the tables IBMTB1 and IBMTB2.

The TAB1 conversion table is for the correspondence EBCDIC to ASCII conversion and is 128 elements long. The first half of TAB1 is used for direct table lookup of the ASCII equivalents of the lower case correspondence EBCDIC characters, and the second half of TAB1 is used for the upper case correspondence EBCDIC characters. The individual elements in TAB1 contain either the direct ASCII equivalent of the EBCDIC character to be converted, a pointer to the auxiliary conversion table TAB3, or an offset with which to dispatch through the address table DISPIN. If the preceding character was a quote ("), the character looked up in TAB1 is used as the index for a further lookup in the retranslation table TAB2, or the character looked up in TAB1 is mapped into some other character in TAB1.

The TAB1P conversion table is used identically to TAB1 except for PTTC EBCDIC characters.

The detailed logic of IBMIN follows:

```
IBMIN       get input character
            and save it
            (in IBCHAR)

            if line is in output mode, go to IBMQ7

            get upper/lower case bit
            for terminal and append
            it to left of character;
            save result (in MODELT)

            get model of terminal
            and use it to select table
            to access (TAB1 for Correspondence,
            TAB1P for PTTC)

IBMQ1       pull character out of correct
            position in correct table

IBMQ2       save character (in IBDATA)

            if character special
            (>200), goto IBMQ6

            if last character received
            was a quote, goto IBMQ3

            increment head position
            counter

            get saved character
            (from IBDATA)

IBMQ9       pass character on

            return from IBMIN
```

```
IBMQ3     mark that last
          character received was
          not quote

          if character = 140, goto IBMQ10

          if character >140, goto IBMQ5

          if character >37, goto IBMQ4

IBMQ10    character input has no
          translation, skip it

          goto DOTN


IBMQ4     build pointer to access
          character in retranslation
          table (TAB2)

          goto IBMQ1


IBMQ5     if character >172, goto IBMQ10

          character was a lower case
          letter preceded by quote

          map character into control
          lower case letter

          goto IBMQ2
```

IBMQ6 if dispatch through DISPIN
is called for (character >360),
goto IBMQ8

if character is a shift up
character (character = 360),
goto INUC

character requires lookup in
an auxiliary table

build pointer to auxiliary table
(TAB3) based on terminal model and type

goto IBMQ1

IBMQ8 dispatch on character through DISPIN (to INBS, INLC, INNL, INTAB, INCC, INDQ, INLF, or INERR)

IBMQ7 If character is a circle-D, mark line in input mode

goto IBMQ1Ø

INUC prepare a 1

goto INUCa

INLC prepare a Ø

INUCa jam Ø or 1 in NCASE

goto IBMQ10

```
INLF    mark that last character
        was not quote

        increase null counter
        by a bunch (call UPNULL)

        pick up a linefeed

        goto IBMQ9
```

INDQ	complement mark indicating whether last character received was quote
	increment head position counter
	if this is not the second of a pair of double quotes, goto IBMQ10
	pick up a quote
	goto IBMQ9

```
INBS    if last character was not quote,
        goto INBS2

        clear mark that last
        character was quote

        goto IBMQ1Ø


INBS2   subtract one from head
        position counter if head
        not at left margin

        pick up a backspace

        goto IBMQ9
```

```
INNL    zero head position
        counter

        increase null counter
        (call UPNULL)

        mark that last character
        was not quote

        mark that carriage
        return received

        pick up carriage-return

        goto IBMQ9


INCC    if attempting to enter output mode,
        goto INC2

        mark that line is attempting to enter
        input mode

        restart output [to get circle-C sent]

        goto INC4


INC2    unblock output [clear NHOLD]

INC4    pick up a line feed

        goto IBMQ9
```

```
INTAB       mark that last character received
            was not quote

            increment null counter
            a little

            increment head position
            to next multiple of 8

            pick up tab

            goto IBMQ9
```

8.2.2.2 MLC Output (OUTINs)

CLKOI	swap buffers and restart OI
	put a "device Ø" at end of buffer
	save address of end of buffer in PSOI
	call OI
	return from CLKOI
OI OIL2	pickup next device number and dispatch to one of OIDISP, DISPAT, ECHL, ECHR, or return from OI (only device Ø does the latter).

```
OIDISP    call LINSBY

          if in RCTE mode and doing echoing,
          goto OIECHO

          if in middle of typing something,
          goto NEWB2

          if currently interpreting a Telnet
          command, skip errors and goto OI21Ø

          if any ERROR2 errors are pending,
          goto SETOI2

          if we were last typing errors,
          goto OID2Ø4

          if any ERROR errors pending
          (besides "CLOSED", "T", "R", or
          "memory bits"), goto SETOI

          if net data is waiting, goto NEWBUF

OID2Ø4    if there are any other errors pending,
          goto SETOI

OID21Ø    if net data is waiting,
          goto NEWBUF

          if terminal is not EBCDIC, goto OID31Ø

          if terminal is not in output mode,
          goto OID31Ø

          mark as "turning around"

OID22Ø    set delay timer and send
          a circle-C

          goto OIL1

OID3ØØ    set dispatch to OIDISP

OID31Ø    mark terminal inactive

          if device is not high speed (>24ØØ baud),
          goto OIL1
```

```
                increment number of allowed extra OIs

                if this was not an "added" OI, goto OIL1

                clear memory of extra OI [ONMOR]

                goto OIL1

NEWBUF          mark to send allocate

                set up pointer to proper half of
                buffer and set up byte count

                if interpreting a Telnet command,
                goto DISGO1

                if next character might be Telnet,
                goto NEWB3a

NEWB3           call DIRCHK to turn line around if
                IBM device

NEWB3a          set dispatch (from OI) to be DISPAT
```

```
DISPAT    get next character

          if character might be a TELNET
          character, goto DISTEL

DISP1     if in SYNCH-Datamark sequence,
          discard data, goto OIL4

DISGO1    dispatch by table entry to one
DISGO2    of DISGET (determine terminal type),
          DISREG (regular ASCII), DISPAR (parity),
          IBMOUT (code conversion), ODEC, MEMRX,
          or SLOWCR (special CR control), or
          several stages of Telnet command
          interpretation

OIECHO    if background is moving data in getting
          RFNM, wait, goto OIL1

          get next character for echo, if none
          goto OID31Ø

          if character is a break character,
          switch to "do output" mode

          if character is to be skipped,
          goto OIL2

          output character and goto OIL1
```

DISGET	set up table entry (DATJMP) according to device type; binary is set to ASCII type
	goto DISGO2
DISREG NCH2 DISR1	format character for MLC and send it
	if device is high speed (>24ØØ baud), adjust counters appropriately
	increment byte count
	if done, set top level dispatch (from OI) to be OIDISP
	goto OIL1

OIL1	step to next OI device number
	goto OIL2
NCHAØ	revise buffer pointers back to do same character next time
NCHA1	format character for MLC
NCHA4	goto DISR1

8.2.2.2.1 2741 Line Control

The 2741 line control code is by default loaded into a TIP but is optional on a site by site basis; any site may choose not to support the 2741 terminal, thus gaining extra space for buffers.

A) Keyboard is unlocked, TIP is in input mode accepting data (NDIR=NTURN=Ø). One of two things eventually happens:

1) A Circle-C comes in (because a CR or ATTN was typed). We immediately revert to input mode by marking the line in "output turning" (NDIR=1, NTURN=1). OI will notice this state and will send a Circle-C to the 2741. The 2741 will then eventually unlock its keyboard and it will then send a Circle-D which will cause the line to be back in input mode.

2) Some output comes along.

NBREAK is set and the line is held open. After an appropriate wait, we send a char of "all mark" and set the line to "input turning" (NDIR=Ø, NTURN=1). OI will then send a Circle-D and mark the line to be in output mode. Output may then proceed.

B) Keyboard is locked, output is in progress (NTURN=Ø, NDIR=1). One of two things may happen:

1) There is no more output to send. The line is marked as in "output turning". The same code as for A.1, above, will send a Circle-C, and the eventual Circle-D will put the line into input mode.

2) A break comes in (user hit ATTN). NHOLD is set and then proceed as in B.1. NHOLD will prevent any output from happening (i.e., you can only go down the A.2 path if hold is clear). NHOLD is cleared by the input of a Circle-C.

```
DIRCHK    if device is not EBCDIC,
          return from DIRCHK

          if device is in output mode,
          return from DIRCHK

          send reverse break and mark
          device as "turning around"

          goto OIL1
```

```
LINBSY    if device is not EBCDIC,
          return from LINBSY

          if is not turning around,
          holding off output, or
          timing a reverse break, return
          from LINBSY

          if device not in "input turning
          around to output" mode,
          goto LINBSYa

          mark in output mode and
          send a circle-D

          goto OIL1


LINBSYA   if timing a reverse break
          or holding off output,
          goto LCHAR3

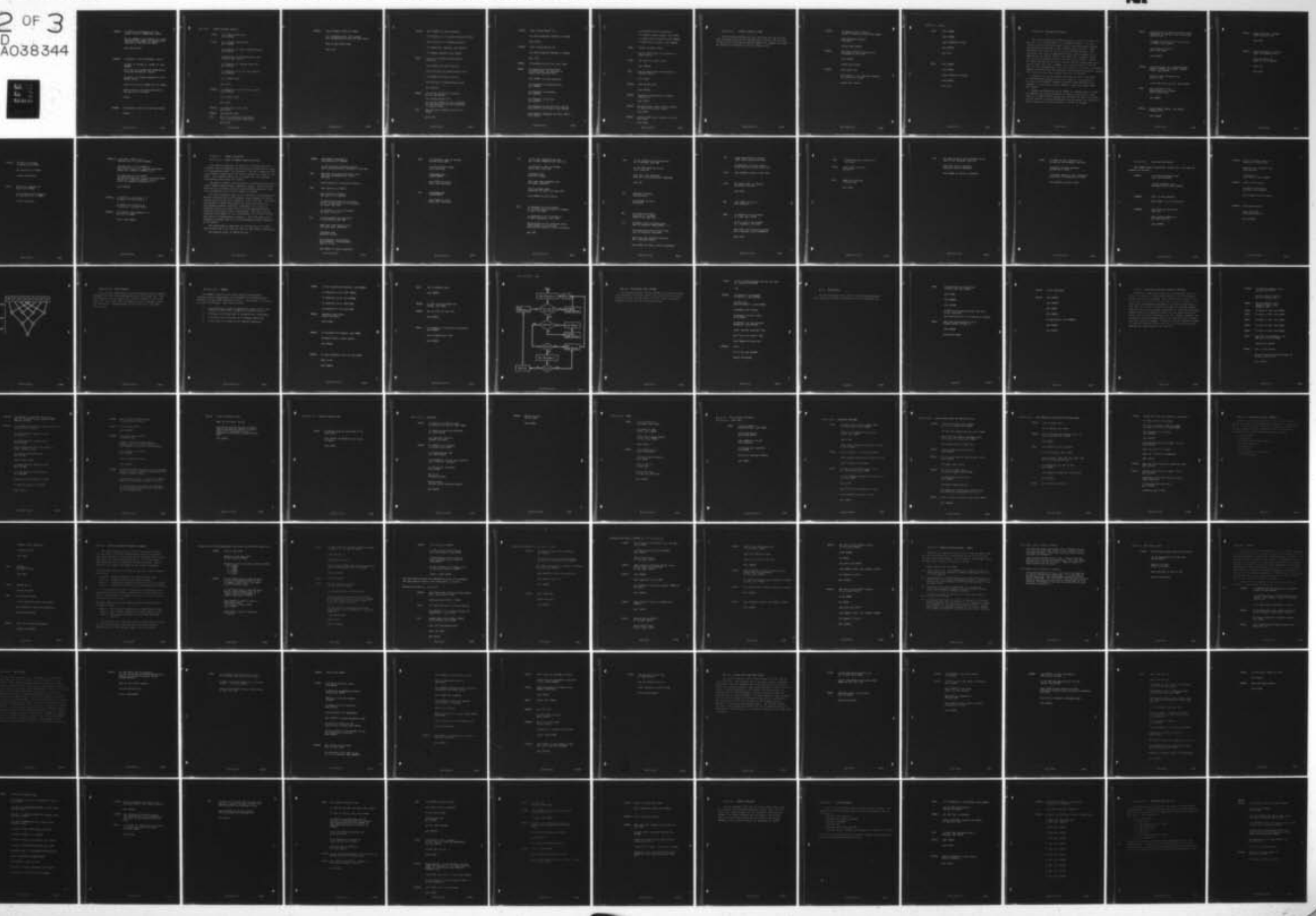
          now turning from output to
          input

          if delay time ≠ Ø, goto LCHAR3

          goto LAST8
```

AD-A038 344 BOLT BERANEK AND NEWMAN INC CAMBRIDGE MASS F/G 9/2
THE TERMINAL INTERFACE MESSAGE PROCESSOR PROGRAM.(U)
FEB 77
UNCLASSIFIED TECHNICAL INFORMATION-91 DAHC15-69-C-0179
NI

2 OF 3
AD
A038344



MIFAST If device is synchronous or its output rate is ≤ 24ØØ baud, return

set up INDMAX to the address of a word containing the number of parallel OIs required to achieve full speed

give skip return

FASTER if device is not high-speed, return

if this is already an "added" OI goto RESMOR

if an OI has already been added during this pass through OI, return

if device is already operating at full speed, return

count one less OI needed for full speed

add an extra OI for this terminal on the end of the OI table

return

RESMOR clear memory that an OI has been added

return

8.2.2.2.2 TELNET Protocol Handler

```
DISTEL    if in New Telnet mode,
          goto NEWTEL

OLDTEL    if in binary output mode,
          goto DISREG

          if character is "sent authentication",
          goto SENDID

          if character is "authentication code
          coming", goto GETID

          if character is "command from net",
          goto NETCOM

          if character is not an "echo remote",
          goto SPBYTEa

          call FIXECO (FDX)

          goto OIL4

SPBYTEa   if character is not an "echo local",
          goto SPBYTEc

          call FIXECO (HDX)

          goto OIL4

SPBYTEc   if character is not a DM,
          goto OIL4

SPBYDM    bump INS/DM count

OIL4      set OI to reprocess this device
          (i.e., step to next character)

          goto OIL2
```

```
SENDID      mark transmit buffer as empty

            put "authentication code coming"
            and authentication code into new buffer

            mark to get buffer sent

            goto OIL4
```

```
NETCOM    call GETBYT for next character

          if character is CR, ignore and goto NETCOM

          pass character to command processor

          if command not complete, goto NETCOM

          if command complete, goto TELDON

GETID     if port is already authenticated,
          goto OIL4

          call GETBYT for next character

          store 1st byte of authentication code

          call GETBYT for next character

          add 2nd byte to authentication code

          goto TELDON

GETBYT    if another character in buffer,
          get it and return

          set "doing Telnet" bit

          put return address in table (DATJMP)
          so next data byte will be dispatched
          to correct place

OIL5      set top level dispatch (from OI) to
          OIDISP

          goto OIL2
```

```
TELNOT    clear "doing Telnet" bit

          set table dispatch (DATJMP) to DISGET

          goto DISP1

TELDON    clear "doing Telnet" bit

          set table dispatch (DATJMP) to DISGET

          goto OIL4

NEWTEL    if character is not IAC, goto DISP1

NEWTØ     if temporarily halting Telnet
          interpretation, set dispatch to
          return here and goto OIL1

          call GETBYT for next character

          if character is Subnegotiation,
          goto NEWTSB

          if character is Datamark,
          goto NEWTDM

          if character is 2nd IAC,
          goto TELNOT

          if character is none of above and not
          WILL, WON'T, DO or DON'T, goto TELDON

          save whether character was WILL, WON'T,
          DO or DON'T
```

```
                call GETBYT for next character
                if unknown option request. goto TELQ
                if binary option request, goto NEWT3Ø
                if RCTE option request, goto NEWT4Ø

TELQ1           inhibit terminal input

                set up queue entry for response to
                known option

TELQ2           put entry on Telnet queue

                goto TELDON

TELQ            set up queue entry for response to
                unknown option

                goto TELQ2

NEWTDM          bump INS/DM count

                goto TELDON

NEWTSB          interpret subnegotiation command
                (RCTE only)

                goto TELDON

NEWT3Ø          process binary output option request
                to synch with output stream

                goto TELQ1

NEWT4Ø          process RCTE option request to synch
                with data

                goto TELQ1
```

8.2.2.2.2.1 FIXECH, FIXECO, FIXJMP

The following routines are also called by the code for the user commands which set the echoing mode; however, the inclusion of the routines with the preceding TELNET echo command code seems as natural as including it with the command code.

FIXJMP	if input is not currently going to network, return from FIXJMP
	set new input dispatch from mode
	return from FIXJMP
FIXECO	set actual Network echo mode as indicated by ACN entry
	call FIXJMP
	return from FIXECO
FIXECH	clear RCTE mode
	call FIXECO to set mode and dispatch as indicated on entry
	return from FIXECH

8.2.2.2.3 Echoes

```
ECHL    call LINBSY
        call DIRCHK
        reset DISPATCH to ECHR
        send ECHWD1
        goto OIL1

ECHR    call LINBSY
        call DIRCHK
        reset dispatch to OIDISP
        send ECHWD2
        goto OIL1
```

8.2.2.2.4 Messages to devices

The routines SETOI and SETOI2 are used to detect if there are any bits set in ERROR (SETOI) or in ERROR2 (SETOI2). If a bit is set, a pointer and byte count for the associated error message are retrieved and passed to the terminal output routine. SETOI and SETOI2 are called by the MLC clock interrupt routine. The only thing less than straightforward about these routines is the handling of the ERDED2, EROPN2, and ERCLS2 bits by SETOI. These bits are where memory is kept of the fact that a "DEAD", "OPEN", or "CLOSE" message was recently printed, and are cleared when there are no "real" bits left in ERROR. Also, if both the "T" and "R" bits are set simultaneously, both are cleared. Of course, the ERDED2, EROPN2, and ERCLS2 bits are prevented from themselves causing a message to be printed since they actually represent no message.

ERDED2 is set by the routine DEAD when either a "HOST DEAD" or "NET TROUBLE" message is set to be printed. Its purpose is to cancel the "REFUSED" message to prevent "DEAD REFUSED" when logging to a dead Host, if the "REFUSED" comes soon enough after the "DEAD".

EROPN2 and ERCLS2 are set by ERRTEL to remember when an "OPEN" or "CLOSED" was recently printed, in order to prevent the "OPEN OPEN" or "CLOSED CLOSED" message when "CLOSED" or "OPEN" is printed but "R" and "T" are canceled because both are there, but they didn't arrive together so another "CLOSED" or "OPEN" is present.

SETOI build pointer to base of correct table depending on whether error is mag tape error or not

if both "T" and "R" bits are not set in ERROR, goto SETOIa

clear both "T" and "R" bits from ERROR

goto OIDISP

SETOIa if no bits are set in ERROR besides ERDED2, EROPN2, and ERCLS2 "memory bits", goto SETOIB

get the first non-memory bit which is set

if no other bits are set, goto SETOIc

SET6 get a pointer to the associated error message and its byte count

goto NEWB3

SETOIb clear ERDED2, EROPN2, and ERCLS2 "memory bits"

goto OIDISP

SETOIc clear first bit in ERROR which was found set

goto SET6

SETOI2 build a pointer to correct table for ERROR2 errors

find the first bit which is set in ERROR2

clear bit

goto SET6

The three routines ERRTEL, HOLLER, and HOLL2 are used to set flags which the terminal output routines detect and then print the indicated message on the TIP terminal. Two words of message flags are associated with each device; these are in the tables ERROR and ERROR2. HOLLER sets flags in ERROR; HOLL2 sets flags in ERROR2. ERRTEL is called rather than HOLLER for those messages which may have an associated "T" or "R", e.g., "OPEN T". The handling of the "CLOSED" and "OPEN" messages by ERRTEL is a little complicated. If either of these bits is set in the call to ERRTEL, the associated "memory bit," ERCLS2 or EROPN2, is set to remember that "CLOSED" or "OPEN" was just printed. Making use of this memory, if ERCLS2 or EROPN2 is set when ERRTEL is called with a "CLOSED" or "OPEN", that bit is cleared to prevent two "CLOSEDs" or "OPENs" from being printed in a row. This is done by clearing the "OPEN" or "CLOSED" bit in the argument to ERRTEL if the associated memory bit is set before the argument is passed to HOLLER.

HOLLER	OR bits in argument to HOLLER into ERROR
	set CR-LF bit in ERROR
	return from HOLLER
HOLL2	OR bits in argument to HOLL2 into ERROR2
	call HOLLER with Ø argument to set CR-LF bit in ERROR
	return from HOLL2

```
ERRTEL(X)  if neither CLOSED-bit or
           OPEN-bit set in X, goto ERRTELa

           for whichever of the CLOSED or
           OPEN bits is set in X, set the associated
           memory bit, ERCLS2 or EROPN2, in X

           for whichever of the CLOSED
           or OPEN bits was set in X, if the associated
           memory bit, ERCLS2 or EROPN2, is set,
           clear the CLOSED or OPEN bit in X

           goto ERRTELb


ERRTELa    if ERRTEL was called by a "T
           routine," OR T-bit into X

           if ERRTEL was called by an
           "R routine," OR R-bit into X

ERRTELb    call HOLLER with argument X to
           set bits in ERROR

           return from ERRTEL
```

8.2.2.2.5 Output conversion

8.2.2.2.5.1 ASCII to EBCDIC Conversion Logic

This section discusses the logic of converting from ASCII to EBCDIC (TIP to terminal) for four types of PTTX and four types of Correspondence model 2741 terminals. The 2741 conversion code is by default loaded into a TIP but is optional on a site by site basis. Any site may choose not to support the 2741 terminal, thus gaining extra space for buffers.

The ASCII to EBCDIC code conversion is handled by the routine called IBMOUT using the main conversion tables labeled CVTB and CVTBP and the auxiliary conversion table SCVTB, along with some state information in the tables IBMTB1 and IBMTB2.

The CVTB conversion table is for the ASCII to Correspondence EBCDIC conversion and is 128 elements long, suitable for direct indexed access by the ASCII character to be converted. The individual element in CVTB either gives the direct conversion to a Correspondence EBCDIC character, indicates that a quote (") character should be printed before the Correspondence EBCDIC character and perhaps further indications. Each entry in the auxiliary table SCVTB has four subelements, one for each of the four types of Correspondence terminals. The CVTB entries also indicate the proper case for the Correspondence EBCDIC character to be printed.

The CVTBP table is identical in form and use to the CVTB table except that it is used for ASCII to PTTC EBCDIC conversions.

The detailed logic of IBMOUT follows:

```
IBMOUT      save ASCII character to
            be converted (in CHARIB)

            if the previous character printed
            was a carriage-return (MGOTR≠ 0), goto PØ3

PØ4         get base of proper conversion table,
            CVTB for Correspondence, CVTBP
            for PTTC

            build pointer to converted character

PØ2         save pointer (in IBMT2)

            get converted character
            and save it (in HOLDIT)

            if auxiliary lookup is to be done
            and if character not to be preceded
            by quote, goto PØ1

            if character is to be preceded
            by quote, goto P11

P13         if case matters for character
            to be printed, goto PØ9

            mark that quote was not last
            character sent (NQUOTE←Ø)

            increment head
            position counter

            get character to be printed
            (from HOLDIT); mask character
            down to 7 bits

            goto NCHA1 (to print character)
```

```
PØ9     if terminal's case is already
        correct, goto PØ8

        if case should be upper
        case, goto PlØ

        complement case
        (NCASE←NCASE)

        goto NCHAØ (to print
        lower case character)

PlØ     complement case
        (NCASE←NCASE)

        goto NCHAØ (to print
        upper case character)
```

P11 if the last character printed
was a quote (NQUOTE ≠ Ø), goto P13

if terminal's case is already
upper case, goto P1Ø

increment head
position counter

mark that last character sent
was quote (NQUOTE←1)

build a proper quote
character for model and type 2741

goto NCHAØ (to print quote)

PØ1 if character to be converted
is an ASCII carriage-return, linefeed,
or tab, goto PØ1A

if character to be converted is
an ASCII backspace, goto OIBS

build pointer to the auxiliary table
SCVTB based on the terminal
type and the character to be converted

goto PØ2

P03 if the character to be converted is a linefeed, goto P05

if the head position is not zero, goto P14

mark that last character sent not a carriage-return (MGOTR←0)

goto P04

P14 decrement terminal head position

goto NCHA0 (to print backspace)

P05 if enough nulls have been sent to terminal (NNULL = 0), goto P15

P17 decrement count of nulls to be sent to terminal (NNULL←NNULL - 1)

if enough nulls have not yet been sent to terminal, goto P06

mark that last character sent was not a carriage return

goto NCHA4 (to send a null to terminal)

P15 clear head position counter
after first noting its value

if head was so far out extra
nulls need to be sent, goto P15a

P15b goto NCHAØ (to print a new line)

P15a put large value in counter
of nulls to be sent

goto P15b

PØ6 goto NCHAØ (to send a
null terminal)

PØ1A if character to be printed
is a tab, goto OITAB

if character to be printed
is a linefeed, goto OILF

mark that last character printed
was a carriage return (MGOTR←1)

goto OIL4

OIBS — if head position counter ≠ Ø, goto OIBSa

OIBSb — goto NCHA4 (to print a backspace)

OIBSa — move head position counter back

goto OIBSb

```
OILF    if there is not a null character to be
        sent to the terminal, goto P17

        mark that a null character
        should be sent to terminal

        goto NCHAØ (to print a linefeed)
```

```
OITAB   if there is null character to
        be sent to the terminal, goto P17

        increment the head position
        counter by a bunch

        increment number of null characters
        to be sent to terminal by a bunch

        goto NCHAØ (to print a tab)
```

8.2.2.2.5.2 Slow Carriage Return

The SLOWCR code is optionally loaded into a TIP upon the request of the site.

SLOWCR	if a carriage return just sent, goto SLOWCRa
	if this character is a carriage-return, goto SLOWCRb
SLOWCRc	add 1 to head position
	goto NCHA4 (to print character)
SLOWCRb	mark that carriage-return just sent
	add a large constant to the head position
	goto SLOWCRc

```
SLOWCRa   build a constant which is
          function of device rate

          subtract this constant from
          head position

          if head is at
          left margin, goto SLOWCRd

SLOWCRe   update head position

          if this character is a
          linefeed, goto SLOWCRc

          goto NCHAØ (to print rubout)


SLOWCRd   zero head position

          clear just sent
          carriage-return bit

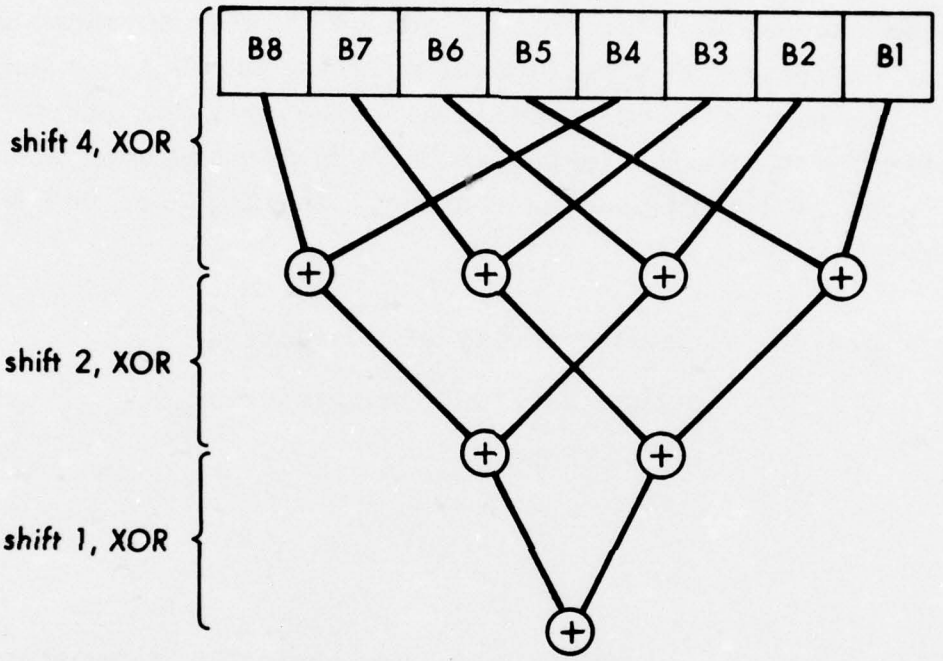
          goto SLOWCRe
```

8.2.2.2.5.3 Even Parity

The even parity calculation routine, GENPAR, calculates a character's parity with three successively smaller shifts followed by exclusive-or's. The effect of this is to successively "fold" the character on itself until only the character's parity is left. The parity is then appended to the original character and the character with parity is passed to NCH4 for dispatch to the correct carriage-return padding routine. The logic of GENPAR is then, simply:

```
DISPAR    compute parity of character
          append parity to character
          goto DISREG
```

For help in reading the GENPAR code, the character folding takes place as follows.


B8
B7
B6
B5
B4
B3
B2
B1
shift 4, XOR
shift 2, XOR
shift 1, XOR

8.2.2.2.5.4. Line Printers

Line printers connected to the TIP frequently require special padding characters after carriage-returns, form feeds, etc. The TIP possesses code to support two different brands of line printers, the ODEC printer and the Memorex printer, although a given TIP can only be configured with the code for one or the other but not both.

8.2.2.2.5.4.1 MEMORX

The MEMORX routine is the output dispatch that handles carriage control requirements of the Memorex 1240 Communications Terminal (and any other compatible device). The padding is tailored to a rate of 600 baud. The "rules" follow:

1) a line must be at least 43 characters long; if it is not, padding is added before sending the carriage return;

2) contiguous line feeds must be separated by 3 characters;

3) a vertical tab is followed by 27 padding characters;

4) a form feed is followed by 215 padding characters.

```
MEMRX     if not outputting character, goto MEMNUL

          if character is LF, goto MEMRXa

          if character is CR, goto MEMRXb

          if character is FF, goto MEM4

          if character is VT, goto MEM4

MEMSVI    decrement line count;
          output character

          goto NCHA4


MEMNUL    if finished with padding, goto MEMA1

          decrement count; output rubout

          goto NCHAØ


MEMRXa    if last character was a LF, goto MEM4

          mark it LF

          goto MEMSV1
```

```
MEM4        set up padding count

            goto MEMNUL


MEMRXb      if line is not already long
            enough, goto MEM4

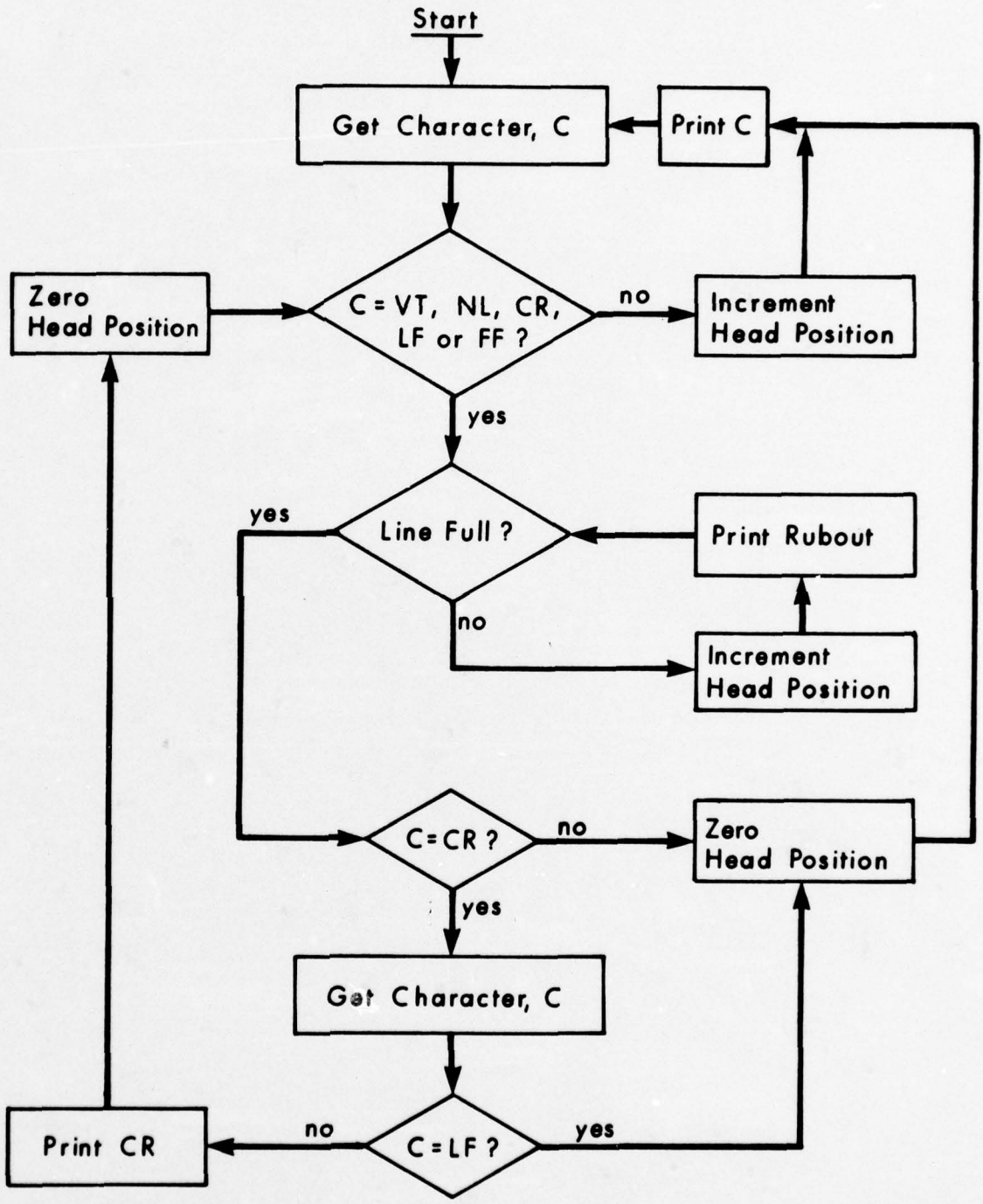
MEMRXc      set up count for new line

            goto MEMSV1


MEMA1       if character is not control character,
            goto MEMRXc

            set up appropriate count

            goto MEMSV1
```


Start
Get Character, C
Print C
Zero Head Position
C = VT, NL, CR, LF or FF ?
no
Increment Head Position
yes
yes
Line Full ?
Print Rubout
no
Increment Head Position
C = CR ?
no
Zero Head Position
yes
Get Character, C
Print CR
no
C = LF ?
yes

8.2.2.3 Background Timer (BTIME)

The background timer routine, BTIME, is called by the MLC clock interrupt routine to print nulls until the null counter is zero and then, if the device is a 2741 which is reverse breaking, to set the line turn-around flag.

```
BTIME       if one (2741) character time has not gone
            by, return from BTIME

            i←Ø

BTIME2      if output is in progress
            to device i, goto BTIMEa

            if there are
            sent to device i, goto BTIMEa

            decrement null counter

            if device i is not a 2741,
            goto BTIMEa

            if device i is not "reverse
            breaking", gcto BTIMEa

            clear "reverse breaking" flag

            set "line turn around" flag

            call IBECHO to print null

BTIMEa      i←i+1

            if i ≠ 64, goto BTIME2

            return from BTIME
```

8.2.3 Background

The TIP background loop, BACK, is quite straightforward as is shown below; BACK is called by the IMP background loop.

```
BACK    if TIP should be initialized
        (TIPFLG ≥ Ø), goto BACKa

        call SLURP

        call MODEMC

        call LOGGER

        if IMP is not accepting data from TIP,
        return from BACK

        exit through BCKSLP (initialized to BCKCHK)

BACKa   mark that initialization is no
        longer needed (TIPFLG←-1)

        call GOGGLE

        return from BACK
```

```
BCKSLP    return from BACK

BCKCHK    call PROBCK

          call PENDIN

          call SENDW

          call DMPSND

          if appropriate, call MTBACK

          call BCKSLP

          goto BCKCHK
```

8.2.3.1 Connection and Host Functions (PENDIN)

PENDIN is a subroutine called by the background loop. It checks all devices in round-robin fashion to see if they have pending any of the following: one, Telnet queue entries to be processed; two, data to go to the net; three, an allocate to be sent; four, miscellaneous messages to go, such as interrupts, RAR, RAS, etc.; five, a request for news; six, queued messages such as resets, echo replies, and replies to unsolicited messages. PENDIN serves as a wakeup for these jobs and takes appropriate action when it finds something to do. As a general comment, the TIP packs control messages going to the same Host into the same Host/Host protocol message. Using this feature, PENDIN continues to check all devices after it has set up a control message and appends any others going to the same Host.

```
PENDIN    if there is nothing to do,
          return from PENDIN

          init to the next device
          in round-robin order

PEND1     set up dispatch table
          based on flags; entry
          is NOP or JMP

PEND2     if entry is JMP, goto PENDT

PEND3     if entry is JMP, goto PENDA

PEND4     if entry is JMP, goto PENDB

PEND5     if entry is JMP, goto PENDC

PEND6     if entry is JMP, goto PENDE

PEND7     send off any messages to the
          IMP with a call to FIRE

          return from PENDIN

PEND8     inc. to next device

          if not finished cycling through all
          devices, goto PEND3

          goto PENRST
```

8.2.3.1.1 Device Related Functions

8.2.3.1.1.1 Process Telnet queue entries

Most New Telnet functions are performed in background, prompted by entries in a queue available to all devices in New Telnet mode. A queued item may be, for instance, a request to change echo mode initiated by the user, the program, or a remote Host. Any device may have none, one, or several items queued for it at any time. From an initial idle state, the current PENDIN device is checked for queue entries. If one is found, it is removed from the queue, all necessary information transferred to the device tables, and the function performed. Such processing may take several passes of PENDIN if, for example, the TIP must send a response and cannot immediately get buffer space. The TIP remembers how far it has gotten for each device, and finally returns to the idle state. The TIP understands the New Telnet options Timing Mark, Suppress Go Ahead, Echo, Remote Controlled Transmission and Echoing (RCTE), and Binary, plus the functions Break, Datamark (synch sequence), and Go Ahead; on user command the TIP will send the Telnet function codes for Break, Datamark (Synch Sequence), Abort Output, Are You There, Erase Character, Erase Line, and Interrupt Process.

PENDT	if the device is not actively doing Telnet, goto PEND3 (next PENDIN function)
	dispatch on table entry to idle state, or various stages of sending Telnet commands
TELSLP	put current state into dispatch table, then goto PEND3
TELIDL	set dispatch to idle state via TELSLP

TELI1Ø if nothing on queue for this device, mark it not busy, allow terminal input and goto TELIDL

TELI2Ø if connection is closed, remove item from queue and goto TELIDL

if connection is trying to open, goto TELIDL

if queue entry is unknown type, goto TEL1ØØ

save information from queue entry in table (TELDIS) entry

if item is a function code, goto TEL1Ø1

remove queue entry

if queue entry is initiate type, goto TEL2ØØ

if queue entry is respond type, goto TEL3ØØ

ignore sent and timing out types

if more entries, goto TELI2Ø

goto TELIDL

TEL1ØØ set up table (TELDIS) entry
to make negative reply

TEL1Ø1 remove queue entry

goto TEL31Ø

TEL2ØØ flag table entry as TIP
initiated type

decide if TIP will allow option
request based on current state and
user desires as stored in device tables

if no change is needed,
goto TELIDL

queue a sent type entry

goto TEL31Ø

TEL3ØØ determine proper response to option request
based on current state and user desires as
stored in device tables

perform any actions, if needed, to change
state of device with respect to option

if the current queue entry was response
to TIP's request, remove sent type entry
also and goto TELIDL

TEL31Ø inhibit terminal input

make all of buffer visible

send Telnet command encoded in device tables; debreak via TELSLP if buffer space is not available for any character, continuing as space becomes free

goto TELIDL

8.2.3.1.1.2 Transfer data to IMP

PENDA if device does not have data to go, goto PEND4

call SENDIT and SENDW to try to get it going

goto PEND4

8.2.3.1.1.3 Allocates

```
PENDB     if there is no need to send
          an allocate for device, goto PEND5

          if device is not being diverted
          to, goto PENDB1

          use capturing device's
          buffers for ALLOC

PENDB1    if connection is closing
          or closed, goto PEND5

          if connection not open
          yet, goto PENDB2

          call SENDAB to try get the interface
          and a "sublink" assigned

          if did not get interface,
          goto PENDB2

          set up an
          allocate message

          mark allocate
          as sent on the assigned sublink

          goto PEND5
```

```
PENDB2      set bit to try
            device later

            goto PEND5
```

8.2.3.1.1.4 News

```
PENDE       if not trying to
            get news, goto PEND8

            if tried all news
            Hosts, goto PEND8

            First time through PENDE?
            if not, goto PENDE1

            type "WAIT..."

PENDE1      call SENDAB to try
            to get interface

            if did not get interface
            goto PEND8

            send an RFC to
            a news Host

            set up next Host
            in list for next time

            goto PEND8
```

8.2.3.1.2 Host related functions

8.2.3.1.2.1 Send INS's

```
PENDC     if don't need to
          send interrupt, goto PENDI

          if connection not
          open, goto PENDI

          call SENDAB to try get
          the interface

          if did not get interface,
          goto PENDI

          set up an interrupt message

          goto PENDI
```

8.2.3.1.2.2 Send RASs and RARs

PENDI if there is no need to send a RAR for this device, goto PENDK

output to the device in progress? If so, goto PENDK

send a RAR

reset output pointers and mark to send a new allocate.

PENDJ port in limbo? If not, goto PENDK

mark transmit and receive connection open

type "connection restored"

PENDK if there is no need to send a RAS for this device, goto PEND6

data in transit through the network? If so, goto PEND6

send a RAS

mark that we are waiting for a RAR

clear transmit allocate to zero

goto PENDJ

8.2.3.1.2.3 Check Reset Table for New RST to go

PENRST for slot in reset table RSTTB
if slot is empty goto PENRSC

if slot has timing bits set, goto PENRS2

enter RST into special message queue,
if no room in queue, goto PENRSC

set timeout bits in reset slot

PENRSI cycle to next slot and if any,
goto PENRST

PENRSC if it is not time to timeout Reset table,
goto PENDUN

set Reset table clock

PENRS3 for slot in reset table
if slot is empty, goto PENDUN

if timing bits are not set,
goto PENRS4

increment timing bits by 1

if timing bits overflowed, delete entry
and shift up remaining slots by one

PENRS4 cycle to next slot and if any, goto PENRS3

goto PENDUN

8.2.3.1.2.4 Send Messages from Special Message Queue

PENDUN look at queue entry

if not empty, goto SKTT1

PENDU2 if not finished with queue, cycle to next entry and goto PENDUN

goto PEND7

SKTT1 call SENDAB to get interface

if not available, goto PEND7

send message: NXR, NXS, ERP, RRP, RST, or CLS to unsolicited RFC

if message was not RST or RRP, goto PENDU2

call CLSALL to break all connections

goto PENDU2

CLSALL set to start at port Ø

```
CLSAL9    search for next port talking to this Host

          if none, return from CLSALL

          if port is logging, free the logger
          and type out "Host not Responding"

          if connection is not open,
          goto CLSALA

          call SHUTDN

          putting connections in limbo?  If not,
          goto CLSAL8

          mark that port is in limbo

          type out "connection suspended"

          goto CLSAL9

CLSAL8    type out "Host broke the connection" and
          goto CLSAL9

CLSALA    putting connections in limbo?  If so,
          goto CLSAL9

          otherwise, clear flag that port has a
          limbo connection

          if flag was previously set,
          goto CLSAL8

          otherwise, goto CLSAL9
```

8.2.3.2 Connection Control (PROBCK)

PROBCK is a subroutine called by the background loop. It checks the status of all connections (active or not) and sends "open" or "close" messages to Hosts if required. PROBCK uses the two connection tables PSTATE and QSTATE, respectively send and receive, which indicate status as follows:

0 = try to open
1 = RFC sent
2 = RFC received, try to reply
3 = solid connection
4 = try to close
5 = CLS sent
6 = CLS received, try to reply
7 = no connection

```
PROBCK    if flag not set, return from PROBCK

          init counter; init index
          to 1st connection

          goto PROB2



PROB3     increment counter; increment index

          if not done all connection, goto PROB2

          return from PROBCK


PROB2     if this connection does not
          need doing, goto PROB3

          set flag to do PROBCK again

          if transmit connection and data
          link blocked, goto PROB3

          if receive connection and output
          active, goto PROB3

PROB1     try to get Host for
          control msg with call
          of SENDAB

          if did not get interface, goto PROB3

          if connection trying to close, goto PRO3

          connection trying to open

          if rcv. side, goto PRO3a
```

```
                transmit side, TRYPRT←10

                AC←code for STR

                goto PRO5


PRO3            TRYPRT←0
                AC←code for CLS

                goto PRO5


PRO3a           TRYPRT←dev + 2

                AC←code for RTS

PRO5            set up msg to Host

                if not trying to close, goto PRO5a

                call SHUTDN to close the connection

                return from PROBCK


PRO5a           call FIRE to send the message

                return from PROBCK
```

8.2.3.3 Initial Connection Protocol (Logger)

The TIP's Logger can do the Initial Connection Protocol for any number of devices simultaneously with the provision that only one ICP to a given Host may be occurring at any given time (this is a protocol constraint, not a TIP constraint). The logger will ICP to HOSTR over SOCKR1 and SOCKR2. While doing so, the logger uses HOSTS for its control variables.

The low order three bits of HOSTS are used as a "state counter" as follows:

state Ø) PORT IS WAITING TO BE SERVED BY THE LOGGER
state 1) LOGGER IS WAITING FOR RESET EXCHANGE
state 2) RESET REPLY HAS BEEN RECEIVED AND ICP CAN PROCEED
state 4) LOGGER IS WAITING FOR THE ICP CONNECTION TO CLOSE.

If the logger sees that a connection is currently or has recently been open to HOSTR, it goes directly from state Ø to state 2 and proceeds. Otherwise it enters a RST into the Reset table and goes into state 1.

The high order five bits of HOSTS are used as flags for the following events:

2ØØ bit: DATA TRANSMIT CONNECTION HAS BEEN OPENED BY HOST
4Ø bit: DATA RECEIVE CONNECTION HAS BEEN OPENED BY HOST
1Ø bit: THE 32 BITS OF SOCKET SPECIFICATION HAVE BEEN RECEIVED

The @N logic also uses the Logger by using HOSTS as memory for its RTS broadcasts, and then when some HOST responds, HOSTS is set to 1 and the main Logger logic is entered.

Called on the first background loop after the 40th MLC clock tick

LOGGER — step to next port

MDLOG for this port set?
if not, exit to TMCLS

dispatch on the low order 3 bits of HOSTS:
- Ø → LOGSØ
- 1 → LOGS1
- 2 → LOGS2
- 3 → LOGS3
- 4 → LOGS4

LOGSØ — if any HOSTR entry is desired Host, and the corresponding port is not in logger state Ø or 1, set HOSTS state 2 and goto LOGS2

if any HOSTR entry is desired Host, and the corresponding port is in logger state 1, step HOSTS to state 1 and exit to TIMCLS

call SETRST to queue a reset to destination Host
if no room in reset table, goto LOGGER

step HOSTS to state 1 and exit to TIMCLS

LOGS2 is any other port already actively logging to Host? if so, exit to TIMCLS

type "trying..."

step HOSTS to state 1

mark to have PROBCK open the ICP connection [Ø → low order three bits of QSTATE]

exit to TIMCLS

LOGS3 exit to TIMCLS

LOGS4 is ICP connection closed? if not, exit to TIMCLS

copy SOCKS1→SOCKR1; SOCKS2→SOCKR2

if data transmit connection has already been opened (by the Host), then set PSTATE to 2, type "OPEN T", and call FIXECH to begin in default echo mode

if data receive connection has already been opened, then set QSTATE to 2 and type "OPEN R"

copy HOSTR→HOSTS

clear MDLOG

exit to TIMCLS

SETRST look at slot in RSTTB

if not in use, place Host in slot and return 2 from SETRST

if destination Host is same as Host in current slot, return 2 from SETRST

if not finished with RSTTB, cycle to next slot and goto SETRST

return 1 from SETRST

The following routines are essentially parts of the Logger, but they receive control from other parts of the TIP.

Entered from SDIS (p. 8.2.3.5-5)

LOGDAT has either data connection been opened yet? if so, goto LOGDI

copy data into SOCKS1, SOCKS2

LOGD2 set "data received" in HOSTS [1Ø bit]

set QSTATE to 4 [to begin closing ICP connection]. goto FLUSH

LOG1 compare data with SOCKS1, SOCKS2 if it matches, goto LOGD2

type "ICP interfered with".

abort the login

goto FLUSH

Entered from GETCLS [p. 8.2.3.5.2.1.1-6]

LOGCLS if port is looking for an RSEXEC, goto TOOBAD

if CLS is not from correct HOST and not for ICP connection, goto TOOBAD

is "data received" set [1Ø bit of HOSTS]? if not, goto LOGRC2

mark QSTATE to close ICP connection

set logger state to 2

goto GETNOP

LOGRC2 type "refused"

abort the login

goto GETNOP

Entered from GETSTR, GETRTS [p. 8.2.3.5.2.1.1-2]

LOGRFC — if for receive direction (i.e., an STR), goto LOGRC1

if port is waiting for an RSEXEC, goto NWRFC1

from correct Host?
if not, goto TOOBAD

LOGRC6 — data receive connection opened or has socket data been received?
if not, goto LOGRC3

LOGRC4 — call USELNK

save supplied link in LINK

set "transmit connection opened" (2ØØ bit) in HOSTS

goto GETNOP

LOGRC3 — save supplied sockets in SOCKS1 and SOCKS2

goto LOGRC4

LOGRC1 — waiting for an RSEXEC?
if so, goto NWRFC2

from correct Host?
if not, goto TOOBAD

```
LOGRC9      STR for the ICP connection?
            if not, goto LOGRC5

            mark ICP connection open

            send an allocate for 64 bits

            goto GETNOP

LOGRC5      data transmit connection opened or has
            socket data been received?
            if not, goto LOGRC7

            do supplied sockets match SOCKS1 and SOCKS2?
            if not, goto TOOBAD

LOGRC8      set "data receive opened" (40 bit) in HOSTS

            goto GETNOP


LOGRC7      copy supplied sockets into SOCKS1, SOCKS2

            goto LOGRC8
```

```
NWRFC1      RFC from a valid RSEXEC server?
            if not, goto TOOBAD

            clear MDNEWS

            set MDLOG

            copy Host into HOSTR

            copy RSEXEC socket into SOCKR1, SOCKR2

            set logger to state 3

            goto LOGRC6


NWRFC2      RFC from a valid RSEXEC server?
            if not, goto TOOBAD

            clear MDNEWS

            set MDLOG

            copy Host into HOSTR

            copy RSEXEC socket into SOCKR1, SOCKR2

            set logger to state 3

            goto LOGRC9
```

8.2.3.3.1 Timeout Missing Events: TIMCLS

TIMCLS is the entry to a collection of routines which time out various theoretically transitional states of the TIP and perform other timing chores. It is linked to from LOGGER; hence, its basic rate is about 8 counts per second.

TIMCLS handles the following:

(1) Every 128th tick: sets TPOPEN to the number of open TIP connections to all devices. TPOPEN is displayed as bit 9 of the B-register.

(2) One transmit or receive connection for one device each tick (hence 128 ticks to make a complete circuit): times out waiting for a matching CLS if TIP has been waiting for two of these ticks.

(3) Each tick one device's OILATE timer is incremented. If a non-synchronous device stays busy for two of these ticks, it is freed up [MSKBSY cleared].

All the following routines run at a basic tick that is about a third of a second:

(4) At one port per tick (64 ticks to go around): Data link blocked and news [@N] is timed out. If the data link has been blocked for two consecutive counts, MDLNKB is cleared and "TIMEOUT" is printed on the user's terminal. If the news request has not been answered in two counts, MDNEWS is cleared and "TIMEOUT" is printed on the user's terminal.

At a basic tick of about a minute:

Loop over all ports and charge for a connected minute. If a port is diverting output it gets charged for two connected minutes. (This code has been left in place but does not assemble.)

Look through the New Telnet queue. Change sent type entries to be timing out type entries, all other information left as is for the entry. Remove timing out type entries from the queue.

At a basic tick of about 7.5 seconds:

Accounting timing is set up. If a port is attempting to logout [LGOFLG ≠ Ø], ACTFLG←≠Ø. If it has been 30 minutes since the last acknowledged accounting checkpoint, ACTFLG←≠Ø. If it has been 5 minutes since accounting data was first sent out and no acknowledge has been received, ACTKLG←≠Ø. (This code has been left in place but does not assemble.)

8.2.3.4 Send data to IMP

TILEAD store second leader word in IMP buffer

set up destination in first word
in IMP buffer

mark as not last
packet of message

fake interrupt of Host to IMP

return from TILEAD

8.2.3.4.1 Link Ø

Link Ø is used for Host-to-Host protocol control messages. All processes in the TIP needing to send such messages call the subroutine SENDAB to request the needed resources. Return 1 indicates failure. Return 2 means success, and SENDAB sets up a message on a "sublink" of link Ø, where allocates use non-zero sublinks 1 through 7, and other control messages use sublink Ø. SENDAB returns the sublink value in the AC. A calling process must request a non-zero sublink by setting the AC negative; a zero AC on entry requests sublink Ø. SENDAB allows a new request to piggyback on a going control message, so the send allocate process must check the sublink on return.

For further explanation of the sublink mechanism, see section 4.2, Deviations from Protocols.

SENDAB	if SENDAB does not already have interface (GOING <Ø) goto SENDA2
	if interface busy to correct destination, (GOING = Host number), return 2 with sublink in AC
	if to some other destination, return 1
SENDA2	if anything else (e.g., data) is already using interface (TIGOF ≠ Ø), return 1
	if caller requested a non-zero sublink, goto FNDSL
SENDA3	call TILEAD to send leader without last packet bit set

call PUT2 to fill in first three words of extended Header

return 2 from SENDAB with sublink in AC

FNDSL if a link Ø sublink is available to the destination Host, goto SENDA3

otherwise, return 1 from SENDAB

FIRE if GOING <Ø, return from FIRE

GOING ← -1

put padding in IMP buffer

fake interrupt to Host-to-IMP

return from FIRE

8.2.3.4.2 Not link Ø

Messages from terminals (i.e., messages not using link Ø) are sent from the TIP to the IMP by a set of routines. These are SENDIT, SENDW, SENDOF, MOVWRD, GETBUF, TOHOST, and TILEAD. The control structure of these routines is that SENDIT is called at clock level to lay claim to the TIP-to-IMP interface for a device. SENDW is called in the background loop and has a co-routine relationship to SENDOF. If a device has laid claim to the TIP-to-IMP interface, SENDW checks to see if the interface is not in use by the link Ø code, SENDAB. The interlock between the link Ø use of the interface and the data link use of the interface is the flag TIGOF. If SENDW can get the interface, it jumps into SENDOF. SENDOF does what it can about copying the data characters from the device input buffers to the IMP, debreaking via a call to SENDOF when it's necessary to wait for some reason. SENDOF calls TILEAD (also used by SENDAB) to give a leader to the IMP, calls GETBUF to get an IMP buffer to fill with data, calls MOVWRD to fill the buffer, and calls TOHOST to pass the filled buffer to the IMP.

```
SENDIT      if some device has not already
            got the TIP-to-IMP interface (DEVYCE ≠ Ø),
            capture interface for current device
            (DEVYCE←device)

            make all the buffer visible

            set the overrun flag

            return from SENDIT
```

SENDW

if no device is giving stuff to the IMP (DEVYCE ≠ Ø), return from SENDW

if IMP is expecting beginning of message (TIGOF < -1), goto SENDL1

return from SENDOF (restart SENDOF where it last left off)

```
SENDOF    return from SENDW


SENDL1    if link is blocked, return
          from SENDW

          if there is no message allocate,
          return from SENDW

          subtract 1 from the message
          allocate

          if there is no bit allocate,
          goto BFAIL2

          clear OVERRUN flag (MDOVER←Ø)

          call SHRINK to update character count

          if there is nothing in the
          input buffer to send, goto BFAIL2

          if bit allocate is big enough for all
          the characters to be sent,
          goto SENDL8


SENDL9    set overrun flag so rest
          will be sent later

          if characters to be sent do not
          fit in one message, goto SENDL1a
```

call TILEAD to send leader to IMP

build on extended header in IMP buffer

call MOVWRD repeatedly with successive characters from input buffer

call MOVWRD with padding

call TOHOST to close off message (last packet of message)

update bit allocate

mark that device no longer using SENDOF (DEVYCE←Ø)

charge the port for one message sent

return from SENDW

SENDL1a set number of characters to be sent to exactly 1 message

goto SENDL9

```
BFAIL2      add 1 back into message allocate

            device have an open XMIT connection?
            if not, clear MDOVER

BFAIL5      mark that device no longer using
            SENDOF (DEVYCE←Ø)

            call SENDOF

BFAIL       return from SENDW


MOVWRD      save the word

            if IMP buffer is full,
            goto MOVWRDa

MOVWRDb     store the saved word
            in the buffer

            increment the buffer fill pointer

            return from MOVWRD


MOVWRDa     call TOHOST to give buffer to IMP
            (not last packet in message)

            goto MOVWRDb
```

TOHOST set the last packet flag
as appropriate

set the blocked link bit

fake interrupt to Host-to-IMP

return from TOHOST

8.2.3.5 Accept Data From IMP (SLURP)

The routine SLURP is used to take messages from the IMP and to process them. The messages are of two main classes, irregular (IMP-to-Host control) messages which are processed by the routine IRREG, and regular messages. The regular messages in turn are of two classes, control messages on link Ø which are processed by the routine PROT, and data messages on links 3 through 65 which are processed by the routine DATABF. Of course various illegal messages may also arrive; these are discarded and reported as appropriate. The subroutine SDIS is used to sort out the message type and dispatch to the proper processing routine. The linkage between the rest of the background routines and the routines which will be described in this section is a co-routine linkage via the SLURP/SLURE co-routines.

SLURP — if the IMP has no data for the TIP, return from SLURP

return from SLURE (to restart SLURE where it last left off)

SLURE — save where left off for next call to SLURP

return from SLURP

```
FLUSHa    call NEXTBF to get next packet
          in message

FLUSH     if this is not last packet in message,
          goto FLUSHa

          call NEXTBF to get first
          packet in next message

          goto SDIS to dispatch on
          message type

          call SLURE to pass control to other
          background routines

          goto DATABF
```

```
NEXTBF      call H3OUIL to fake interrupt to
            IMP-to-Host routine

            if the IMP has something for the TIP,
            return from NEXTBF

            call SLURE to pass control to other
            background routines while waiting for something
            from IMP

            goto SDIS to dispatch on message type

            goto DATABF
```

SDIS save link (in L2)

save Host (in L1)

if packet is only leader, call NEXTBF to get first data packet

if message is not a regular message from a real Host, goto IRREG

save the Host/Host protocol byte count (in OUCNT1) and negative of byte count (in L4)

if link number = Ø, goto PROT

if link number = ACTLNK, goto GETACT (this operation is left in code, but not assembled)

if link number = DMPLNK, goto DMPASK

if link number = 1 or $\geq$65., goto BUGFA

otherwise, 3< link $\leq$ 65 (or, 1$\leq$ device $\leq$63)

if MDLOG is set, goto LOGDAT (p.8.2.3.3-4)

if connection to that Host is not open, report error and goto FLUSH

dispatch to magtape option if appropriate

goto DATABF

```
BUGFA     if link ≠ 66, report an error
          goto FLUSH
DMPASK    queue the dump request
          goto FLUSH
```

8.2.3.5.1 Irregular Messages

The irregular messages the TIP receives from the IMP are handled by the routine IRREG. The handling of the irregular messages is fairly straightforward, as shown by the following detailed logic descriptions. One complication is the handling of RFNMs and INCOMPLETE TRANSMISSIONs which gets involved with Host/Host protocol allocates and the TIP's terminal input buffering logic.

IRREG dispatch on message type

if type >10., =3, or =4 (undefined or NOP), goto FLUSH

if type =10. (INTERFACE RESET), report error and goto FLUSH

if type = 1 (ERROR WITHOUT ID), report error and goto FLUSH

if type =8 (ERROR WITH ID), report error and goto FLUSH

if type =2 (IMP GOING DOWN), goto IGD

if type =5 (RFNM), goto IRFNM

if type =6 (DEAD HOST STATUS), goto DEAD.Y

if type =7 (DESTINATION DEAD), goto DEAD

otherwise type =9 (INCOMPLETE TRANSMISSION)

report "INCOMPLETE TRANSMISSION"

call UNBLK to unblock link

if link is already unblocked, goto FLUSH

if link is control link, goto CLRLNK

IRINC3 restore allocates for Host to what they were before last message was sent

goto IRFNM3

CLRLNK scan through all devices looking for outstanding Allocate messages to this Host on this sublink and clear them

CLRL3 if an INC (vs. RFNM) also mark device to resend the allocate and poke allocate sender

goto FLUSH

IGD convert "in how long" and "for how long" fields to an ASCII string & insert into IMP-GOING-DOWN error message string

loop through all devices setting IMP-GOING-DOWN error message bit

goto FLUSH

IRFNM call UNBLK to unblock link

if link was already unblocked, goto FLUSH

if link is control link, goto CLRLNK

if device is in RCTE mode, adjust pointers so RFNMed characters indicate only characters which are RFNMed and have been processed for echoing (by OIECHO)

flush the RFNMed characters from the input buffer

if no characters are left in input buffer, goto IRFNM4

relocate them to bottom of the input buffer

IRFNM4 adjust input buffer pointer so it points the beginning of the available space

IRFNM3 call SENDIT and SENDW to attempt to send any new stuff to be sent

goto FLUSH

DEAD call UNBLK to unblock link

was there a link to unblock?

if not, goto FLUSH

if Host was dead
goto DEAD9

set for "NET TROUBLE"

goto DEADY8

DEAD9 "nonexistent Host" subcode?
if not, set for "HOST NOT RESPONDING"
and goto DEADY8

unlock input buffer

goto FLUSH

DEAD.Y check subcode. As appropriate, set for "HOST NOT RESPONDING", or "HOST SCHEDULED DOWN", or "UNSCHEDULED HOST SERVICE INTERRUPTION".

"HOW LONG" indicated? if not, goto DEADY8

set up time and date and append "UNTIL..." to error message.

DEADY8 call CLSALL with saved message.

goto FLUSH

```
UNBLK     if link = Ø,
          return 2 from UNBLK

UNBLKa    call FNDTRN to find a port using this
          link to this Host

          if none, goto UNBLKc

UNBLKd    if link is 77 and mag tape option is
          loaded, give "spurious" return from
          UNBLK

          if link not blocked, goto UNBLKb

          unblock link

          give success return from UNBLK

UNBLKb    report "extra RFNM"

          give "spurious" return from UNBLK

          did we find a connection but it was in limbo?
          if so, goto UNBLKd
```

UNBLKc report "no such data link"

give "spurious" return from UNBLK

FNDTRN set to start at port Ø

FNDTR3 find next port talking to this Host on this link

if none, give "no match" return from FNDTRN

connection open? If so, give success return from FNDTRN

connection in limbo? If not, goto FNDTR3

otherwise, give "no match" return from FNDTRN but with limbo connection flag set

8.2.3.5.2 Regular Messages

Regular messages come from the IMP in three forms, those on link zero (the Host/Host protocol control link), those on data links (data messages for TIP terminals), and those on ACTLNK. The link Ø messages are processed by the routine PROT, the data link messages are processed by the routine DATABF, and the ACTLNK messages are processed by GETACT.

8.2.3.5.2.1 Link Ø Messages

Link Ø is used for Host/Host protocol control messages. The handling of these Host/Host protocol control messages is done in six pieces:

- .handling RFCs and CLSs
- .handling ECOs, RSTs, and RRPs
- .handling ALLOCATES
- .counting INSs
- .handling RARs, RASs, and RAPs
- .ignoring illegal commands and commands not handled by the TIP.

The logic for dispatching on the various commands and ignoring the illegal and unhandled commands follows:

```
PROT      if no commands in the message, goto NOPROT

          set up GET1 co-routine to
          restart at GETNOP

PROTa     get next byte in message

          return from GET1 (restart GET1 where
          it last left off)


GET1      if there are any bytes left in
          message, goto PROTa

GETOUT    Poke PROBHO

          goto FLUSH


NOPROT    report reception of zero length
          protocol message

          goto FLUSH
```

```
GETNOPa   call GET2 and GET1 to flush bytes
          to the end of command

          call GET to get next character

GETNOP    dispatch on Host/Host protocol command type

          if ERR, Ret, GVB, INR, ERP
          or NOP, goto GETNOPA

          if RTS, goto GETRTS

          if STR, goto GETSTR

          if CLS, goto GETCLS

          if ALL, goto GETALL

          if INS, goto GETINS

          if ECO, goto GETECO

          if RST, goto GETRST

          if RAR, goto GETRAR

          if RAS, goto GETRAS

          if RAP, goto RASREQ
```

8.2.3.5.2.1.1 Handling RFCs and CLSs

RFCs (Request for Connections) are handled by the GETRTS/GETSTR routine. CLSs (Closes) are handled by the GETCLS routine. RFCs and CLSs are driven by an eight-state finite-state machine for each device. The states are:

Ø -- try to open
1 -- RFC sent
2 -- RFC received, try to reply
3 -- connection open
4 -- try to close
5 -- CLS sent
6 -- CLS received, try to reply
7 -- no connection

External events drive the finite-state machine into its even-numbered state. Every time the TIP finds a device in an even-numbered state, it immediately performs the function which allows it to move the device to the next sequential odd-numbered state.

GETRTS

GETSTR call SUCKS to get the socket numbers

if port is logging,
goto LOGRFC

save (in EXTRA) the link or byte size,
depending on whether RTS or STR

call TESTOK to see if it's ok to act on RTS
or STR; if not ok, goto TOOBAD

if the finite state machine for this
connection is in states 2,3,4,5, or c6,
report error and goto GETNOP

if state is Ø or 7, make state 1 and
goto GETSTRa

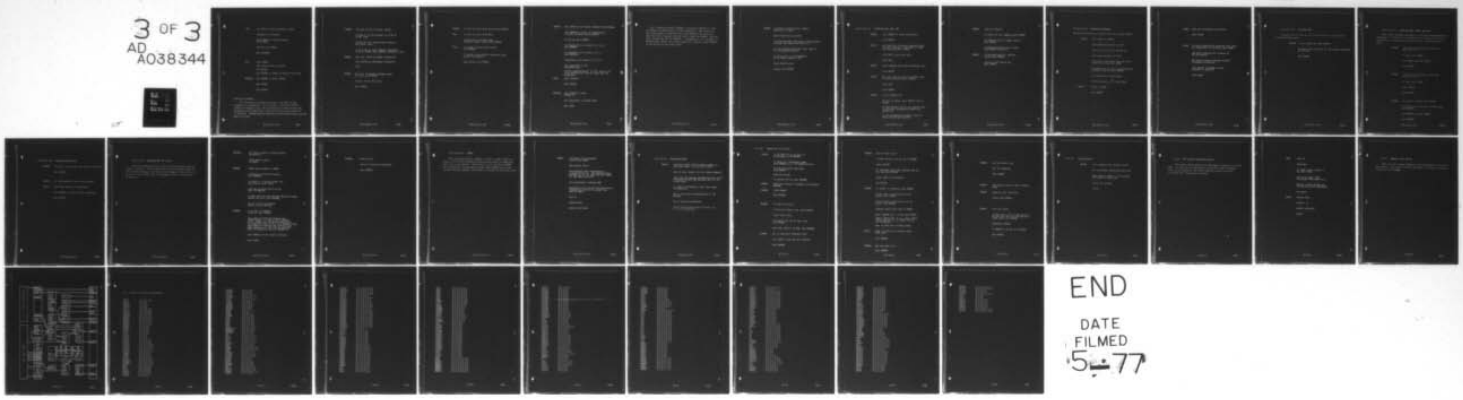
if state is 1, make state 2

GETSTRa save first socket number (in
SOCKS1 and SOCKS2)

if command is RTS, goto RTS

AD-A038 344 BOLT BERANEK AND NEWMAN INC CAMBRIDGE MASS F/G 9/2
THE TERMINAL INTERFACE MESSAGE PROCESSOR PROGRAM.(U)
FEB 77
UNCLASSIFIED TECHNICAL INFORMATION-91 DAHC15-69-C-0179
NI

3 OF 3
AD
A038344



END

DATE
FILMED
5-77

```
STR       set flag to send allocates to Host

          complete bit allocate

          poke PENALL to cause allocate
          to be sent

          save Host in HOSTR

          goto GETSTRb


RTS       call USELNK

          save link and Host in HOSTS

          set MDOVER

          call FIXECH to begin in default echo mode

GETSTRb   call ERRTEL to print "OPEN"

          poke PROBHO

          goto GETNOP
```

Subroutine USELNK:

If a Host has an internal problem in its NCP, it may "misplace" a connection. If it re-uses a link that the TIP thinks is already in use, the implication is that the Host has "forgotten" the former use of the link and the user should be so informed. USELNK handles checking for and annihilating previous uses of a link.

```
USELNK    for each of the 64 ports, repeat:

          if port is in the process of an @N or
          an @O, loop.

          if port is not issuing other links to
          this host, loop

          if port has an open transmit connection
          or is in limbo, goto USELK2; otherwise, loop

USELK2    tell port "HOST has RESET connection"

          call SHUTDN to annihilate connection

          loop

TOOBAD    get slot in special message queue;
          if none, goto GETNOP

          build a CLS in free slot

          goto GETEC1
```

TESTOK	if port is wild, good return from TESTOK
TE21	if Host is wild, goto TE21a
	if Host does not match what expected, error return from TESTOK
TE21a	if socket is wild, good return from TESTOK
	if socket is not what's expected, error return from TESTOK
	good return from TESTOK

```
GETCLS    call SUCKS to get socket numbers from message

          call TESTOK to check if everything ok
          to start closing this procedure

          if not ok, goto GETNOP

          if connection is in state 0, 6, or 7,
          goto GETNOP

          if connection is in state 5 or 1,
          goto GETCLSa

          connection is in state 2, 3, or 4

          put connection in CLS
          received state

          is port authenticated?  if not, log it out
          (this operation is left in code, but not
          assembled)

STRY      type "closed"

          goto GETNOP


GETCLSa   call SHUTDN to clean
          things up

          put connection in closed state

          goto STR4
```

The following routine, SHUTDN, is called by GETCLS to clear up things associated with a closed connection. The routine is also called when a connection is closed because the Host went dead (by DEAD), because the Host sent a Reset (by SNDRP), and the Host refuses the ICP (by PROBCK). Consequently, the inclusion of SHUTDN under GETCLS is a little unnatural; however, under GETCLS is as natural as any other place.

```
SHUTDN   if device is wild, reset sockets
         and Host to "<any>"

         mark connection as closed

         if shutting down send side, clear allocate
         counters and reset send link to Ø

         if not shutting down send link, mark to
         not retransmit allocate

         if device set to non-permanent
         @I N, reset escape to "@"

         clear INS/DM count

         return from SHUTDN
```

8.2.3.5.2.1.2 Handling ECO, RST, RRP

GETRRP — call RSTRST to clear pending RST

goto GETNOP

GETECO — get free slot in special message queue if none available, call GET1 to flush data byte and goto GETNOP

call GET1 to get data byte

build ERP

NEEDRP — queue message into special message slot

goto GETNOP

GETRST — get free slot in special message queue if none available, goto GETNOP

build RRP

goto NEEDRP

RSTRST — look at RSTTB slot

if slot is empty, goto GETRP3 (end of table)

if Host matches entry, set timeout bits to overflow (background PENRST will delete it)

if not finished with table, cycle to next slot and goto RSTRST

```
GETRP3      look at a device

            if device is lot logging, goto GETRP5

            if HOSTS is not in logger state 1,
            goto GETRP5

            if HOSTR matches source of RST,
            step HOSTS to state 2

GETRP5      if finished with all devices,
            return from RSTRST

            cycle to next device and
            goto GETRP3
```

8.2.3.5.2.1.3 Handling ALLOCATES

Received allocates are processed by the routine GETALL.

```
GETALL    save link (in ALINK)

          save message allocate (in AM)

          save bit allocate (in AH and AL)

          save word allocate (in ALL)

          if there is no device from this Host
          using this link, goto NOSND

          if connection is being resynchronized
          [SNTRAS nonzero], goto GETNOP

          call mag tape if appropriate

          if bit allocate ≤2^16, goto FIND5

NOSND     return a "NXS"

          goto GETNOP
```

FIND6 mark bit allocation as infinite

goto FIND8

FIND5 if newly received bit allocate will cause bit allocation to overflow, goto FIND6

add newly received bit allocate to bit allocation

add newly received message allocate to message allocation

call SENDIT and SENDW to send allocate if possible

goto FLUSH

8.2.3.5.2.1.4 Counting INSs

Counting INSs is done by the routine GETINS in a completely straightforward manner.

GETINS if not $3 \leq$ link ≤ 65, goto GETNOP

increment the INS count for the device indicated by the link

goto GETNOP

8.2.3.5.2.1.5 Handling RASs, RARs, and RAPs

These three commands implement an allocate resynchronization mechanism. In addition to returning the allocates to zero on a connection, a resynchronization will bring a connection out of limbo.

RASREQ look for port talking to this Host over this link

if none, goto NOSND

set SNDRAS and bump PENDTH

goto GETNOP

GETRAR look for port talking to this Host over this link

if none, goto NOSND

clear SNTRAS

goto GETNOP

GETRAS use link to compute port number

if port is not talking to correct Host, goto GETNOP

set SNDRAR and bump PENDTH

goto GETNOP

8.2.3.5.2.1.6 Handling NXR and NXS

GETNXR look for a port using this link to this Host

goto GETNX1

GETNXS use link number to set up port number

GETNX1 type "Host broke our connection"

call SHUTDN to clear out the connection

goto GETNOP

8.2.3.5.2.2 Messages Not On Link Ø

Arriving messages which are not on link Ø contain data for output to devices. After a few checks for the proper format of the received messages, the data in the message is copied into the device's output buffer and, if necessary, output to the device is started.

DATABFa — get device number of device being diverted to

store device number in OUDEV

DATABF — saved device number in OUDEV

if device is diverting output, goto DATABFa

if there is no output buffer for this device, goto FLUSH

save max allocate which can be sent (in MAXALL)

if more data has arrived than device's output buffer can hold, goto DATABFb

set up to send allocation equal to data received

DATABFc — if no data in message received, goto NODATA

Copy bytes of received message into output buffer for device until message end or packet end, whichever comes first. If packet end comes first, call NEXTBF to get next packet of message and continue copy. When message ends, set mark to indicate there is output for device (MIGOTO +1)

call OUNEW to start output to device

goto FLUSH

```
DATABFb     report error

            set up to send max allocation

            goto DATABFc
```

8.2.3.5.2.2.1 OUNEW

The following routine, OUNEW, is used to start output to a device, if it is not presently doing any and there is something new to go to the device. The routine is also called by MODEMC and consequently its description here under DATABF is not completely natural; however, here seems as natural as any other place.

```
OUNEW       if output is in progress,
            return from OUNEW

            mark device active

            build device table consisting of
            a single device #:  the terminal being
            started up, in the table set the "fake
            OI" bit (the sign bit).

            lock interrupts & change PRIM

            initialize # of "extra" OIs this device
            requires to achieve full speed [by
            calling IMAX]

            call OI

            restore PRIM

            return from OUNEW
```

8.2.3.5.2.3 Accounting Data

```
GETACT      sequence number match sequence number on
            last sent data?  if not, goto FLUSH

            save up Host number [for the acknowledgment]

            loop over the message documenting each port's
            connected minutes and messages sent by the
            echoed data.

            if logout confirmation, mark that logout
            is completed

            set to next send accounting data in 30
            minutes

            set to return acknowledge

            (this code has been left in place, but
            it is not assembled)
```

8.2.3.6 Modem and LIU Control

```
MODEMC      if we have set up no line to
            interrogate, goto MODEMa

            if this is a "lethargic" pass,
            goto MODEMg [i.e., if DSFLAG is set]

            if LIU is current loop type,
            goto MODEM1

            read LIU status

            if carrier not up, goto MODEMb

MODEM1      mark that carrier is present (by clearing
            MOCARR)

MODEM8      clear MOHANG

            goto MODEMa


MODEMb      re-read LIU status

            if data set ready is up, goto MODEMa

            reset delay timer

            if carrier was not up last time,
            goto MODEMa

            mark that carrier is down (set MOCARR)

MODEMh      set to drop data terminal ready

            call RESET to get the port unwound

            goto MODEM8
```

```
MODEMa    step to next device

          if next device is not Ø, go to MODEMi

          reset DSFLAG

          if lethargic pass timer [DSCLK] has not
          run out, goto MODEMi

          reset timer to one minute

          set DSFLAG

MODEMi    if output in progress, goto MODEMc

          if MLC input controller is to be
          reset, goto MODEMd

          if MLC output controller is to be
          reset, goto MODEMe

          restart output with call to OUNEW

          port logging out?  if not, goto MOD3b

          logout completed?  if so, clear LOGOUT
          (this code is left in place, but is
          not assembled)

          mark to drop data terminal ready

MOD3b     mark to raise data terminal ready
          next time

          goto MODEMc


MODEMd    get new input rate

          goto MODEMf
```

```
MODEMe      get new output rate

            set "OI expected"

            goto MODEMf


MODEMc      get current state of data terminal
            ready

MODEMf      output to MLC controller

            return from MODEMc


MODEMg      read LIU status

            if data set is not in the state of
            carrier data set low and data set
            ready high, goto MODEM8

            complement MOHANG

            if MOHANG is now Ø, goto MODEMh

            goto MODEMa
```

8.2.3.7 Dump Requests

```
DMPSND      if no requests are queued, return

            set up message requesting Host-link

            send version number of TIP systems
            and tables in dump list

            finish the message

            return
```

8.2.4 MLC Output Interrupt Routine

The routine which services the MLC Output Interrupt is called TOUT. In addition to being called by the MLC Output Interrupt, TOUT is also called by the Clock Interrupt routine (at OOPS) to restart MLC output when necessary.

```
TOUT    save AC

        save keys

        if there is not output to
        go, goto TOUTa

        set up an output from
        the buffer with stuff in it

        switch to other buffer for
        filling with more characters

        do output

TOUTa   restore keys

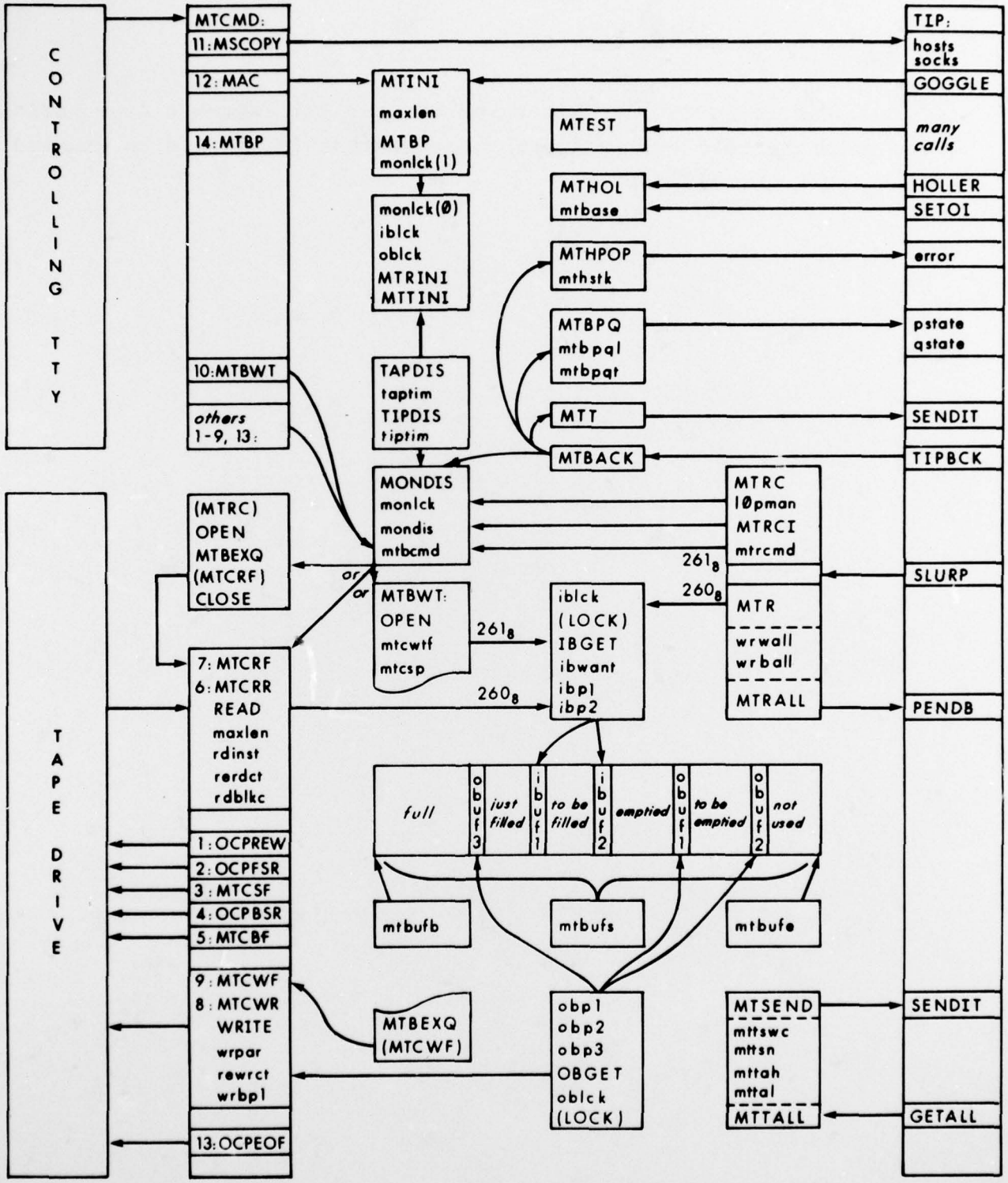
        restore  AC

        enable interrupts

        return
```

8.2.5 Magnetic tape option

There follows a block diagram for the TIP magnetic tape option. The magnetic tape option listing, attachment IV, should be studied along with the diagram.


CONTROLLING TTY
MTCMD:
11:MSCOPY
12:MAC
14:MTBP
10:MTBWT
others 1-9, 13:
MTINI
maxlen
MTBP
monlck(1)
monlck(Ø)
iblck
oblck
MTRINI
MTTINI
TAPDIS
taptim
TIPDIS
tiptim
MONDIS
monlck
mondis
mtbcmd
MTEST
MTHOL
mtbase
MTHPOP
mthstk
MTBPQ
mtbpql
mtbpqt
MTT
MTBACK
TIP:
hosts
socks
GOGGLE
many calls
HOLLER
SETOI
error
pstate
qstate
SENDIT
TIPBCK
MTRC
10pman
MTRCI
mtrcmd
SLURP
MTR
wrwall
wrball
MTRALL
PENDB
(MTRC)
OPEN
MTBEXQ
(MTCRF)
CLOSE
or
or
MTBWT:
OPEN
mtcwtf
mtcsp
261₈
260₈
iblck
(LOCK)
IBGET
ibwant
ibp1
ibp2
TAPE DRIVE
7:MTCRF
6:MTCRR
READ
maxlen
rdinst
rerdct
rdblkc
1:OCPREW
2:OCPFSR
3:MTCSF
4:OCPBSR
5:MTCBF
9:MTCWF
8:MTCWR
WRITE
wrpar
rewrct
wrbp1
13:OCPEOF
full
obuf3
just filled
ibuf1
to be filled
ibuf2
emptied
obuf1
to be emptied
obuf2
not used
mtbufb
mtbufs
mtbufe
MTBEXQ
(MTCWF)
obp1
obp2
obp3
OBGET
oblck
(LOCK)
MTSEND
mttswc
mttsn
mttah
mttal
MTTALL
SENDIT
GETALL

8.3 Index to Detailed Descriptions

ADDLF	8.2.2.1.1-5
BACK	8.2.3-2
BACKa	8.2.3-2
BCKCHK	8.2.3-3
BCKSLP	8.2.3-3
BFAIL	8.2.3.4.2-6
BFAIL2	8.2.3.4.2-6
BFAIL5	8.2.3.4.2-6
BINCON	8.2.2.1.1-4
BINECO	8.2.2.1.1-4
BREAK	8.2.2.1.1-3
BREAK1	8.2.2.1.1-3
BREAK2	8.2.2.1.1-3
BREAK3	8.2.2.1.1-3
BTIME	8.2.2.3-2
BTIME2	8.2.2.3-2
BTIMEa	8.2.2.3-2
BUGFA	8.2.3.5-6
CLKOI	8.2.2.2-1
CLOCK	8.2.2-1
CLOCK4	8.2.2-1
CLOCK4	8.2.2.1-2
CLRL3	8.2.3.5.1-3
CLRLNK	8.2.3.5.1-3
CLSALA	8.2.3.1.2.4-2
CLSALL	8.2.3.1.2.4-1
CLSAL8	8.2.3.1.2.4-2
CLSAL9	8.2.3.1.2.4-2
CONECO	8.2.2.1.1-4
CONEEE	8.2.2.1.1-7
CONESC	8.2.2.1.1-7
CONVT	8.2.2.1.1-4
DATABF	8.2.3.5.2.2-2
DATABFa	8.2.3.5.2.2-2
DATABFb	8.2.3.5.2.2-3
DATABFc	8.2.3.5.2.2-2
DEAD	8.2.3.5.1-6
DEAD9	8.2.3.5.1-6
DEAD.Y	8.2.3.5.1-6
DEADY8	8.2.3.5.1-6
DIRCHK	8.2.2.2.1-2
DISGET	8.2.2.2-5

DISG01	8.2.2.2-4
DISG02	8.2.2.2-4
DISP1	8.2.2.2-4
DISPAR	8.2.2.2.5.3-1
DISPAT	8.2.2.2-4
DISPC	8.2.2.1.3.1-7
DISR1	8.2.2.2-5
DISREG	8.2.2.2-5
DISTEL	8.2.2.2.2-1
DMPASK	8.2.3.5-6
DMPSND	8.2.3.7-1
DOTN	8.2.2.1.1-2
DOTN2	8.2.2-1
DOTN2	8.2.2.1.1-2
ECHBEL	8.2.2.1.1-6
ECHL	8.2.2.2.3-1
ECHO	8.2.2.1.2-2
ECHR	8.2.2.2.3-1
EOM	8.2.2.1.1-6
EOMa	8.2.2.1.1-6
ERR1	8.2.2.1.3.1-7
ERRTEL(X)	8.2.2.2.4-6
ERRTELa	8.2.2.2.4-6
ERRTELb	8.2.2.2.4-6
ESC	8.2.2.1.1-7
ESC1	8.2.2.1.3.1-2
ESC1a	8.2.2.1.3.1-3
ESC2	8.2.2.1.1-7
EXIT	8.2.2.1.3.1-4
FASTER	8.2.2.2.1-4
FEED	8.2.2.1.1-6
FIND5	8.2.3.5.2.1.3-2
FIND6	8.2.3.5.2.1.3-2
FIRE	8.2.3.4.1-2
FIXECH	8.2.2.2.2.1-2
FIXECO	8.2.2.2.2.1-2
FIXJMP	8.2.2.2.2.1-2
FLUSH	8.2.3.5-3
FLUSHa	8.2.3.5-3
FNDSL	8.2.3.4.1-2
FNDTRN	8.2.3.5.1-8
FNDTR3	8.2.3.5.1-8
GET1	8.2.3.5.2.1-2
GETACT	8.2.3.5.2.3-1
GETALL	8.2.3.5.2.1.3-1
GETBYT	8.2.2.2.2-3

GETCLS	8.2.3.5.2.1.1-6
GETCLSa	8.2.3.5.2.1.1-6
GETECH	8.2.3.5.2.1.2-1
GETECO	8.2.3.5.2.1.2-1
GETID	8.2.2.2.2-3
GETINS	8.2.3.5.2.1.4-1
GETNOP	8.2.3.5.2.1-3
GETNOPa	8.2.3.5.2.1-3
GETNX1	8.2.3.5.2.1.6-1
GETNXR	8.2.3.5.2.1.6-1
GETNXS	8.2.3.5.2.1.6-1
GETOUT	8.2.3.5.2.1-2
GETRAR	8.2.3.5.2.1.5-1
GETRAS	8.2.3.5.2.1.5-1
GETRP3	8.2.3.5.2.1.2-2
GETRP5	8.2.3.5.2.1.2-2
GETRRP	8.2.3.5.2.1.2-1
GETRST	8.2.3.5.2.1.2-1
GETRTS	8.2.3.5.2.1.1-2
GETSTR	8.2.3.5.2.1.1-2
GETSTRa	8.2.3.5.2.1.1-2
GETSTRb	8.2.3.5.2.1.1-3
GOGGLE	8.2.1-2
HOLL2	8.2.2.2.4-5
HOLLER	8.2.2.2.4-5
HUNT	8.2.2.1.4-2
HUNTa	8.2.2.1.4-2
IBMCON	8.2.2.1.1-4
IBMECO	8.2.2.1.1-4
IBMEEE	8.2.2.1.1-4
IBMESC	8.2.2.1.1-4
IBMIN	8.2.2.1.5.1-2
IBMOUT	8.2.2.2.5.1-2
IBMQ1	8.2.2.1.5.1-2
IBMQ2	8.2.2.1.5.1-2
IBMQ3	8.2.2.1.5.1-3
IBMQ4	8.2.2.1.5.1-3
IBMQ5	8.2.2.1.5.1-3
IBMQ6	8.2.2.1.5.1-4
IBMQ7	8.2.2.1.5.1-5
IBMQ8	8.2.2.1.5.1-5
IBMQ9	8.2.2.1.5.1-2
IBMQ10	8.2.2.1.5.1-3
IGD	8.2.3.5.1-4
INBS	8.2.2.1.5.1-8
INBS2	8.2.2.1.5.1-8

INCC	8.2.2.1.5.1-9
INC2	8.2.2.1.5.1-9
INC4	8.2.2.1.5.1-9
INDQ	8.2.2.1.5.1-7
INLC	8.2.2.1.5.1-5
INLF	8.2.2.1.5.1-6
INNL	8.2.2.1.5.1-9
INTAB	8.2.2.1.5.1-10
INUC	8.2.2.1.5.1-5
INUCa	8.2.2.1.5.1-5
IRFNM	8.2.3.5.1-5
IRFNM3	8.2.3.5.1-5
IRFNM4	8.2.3.5.1-5
IRINC3	8.2.3.5.1-3
IRREG	8.2.3.5.1-2
LFCHR	8.2.2.1.3.1-7
LFCHRa	8.2.2.1.3.1-7
LINBSY	8.2.2.2.1-3
LINBSYA	8.2.2.2.1-3
LOG1	8.2.3.3-4
LOGCLS	8.2.3.3-5
LOGDAT	8.2.3.3-4
LOGD2	8.2.3.3-4
LOGGER	8.2.3.3-2
LOGRC1	8.2.3.3-6
LOGRC2	8.2.3.3-5
LOGRC3	8.2.3.3-6
LOGRC4	8.2.3.3-6
LOGRC5	8.2.3.3-7
LOGRC6	8.2.3.3-6
LOGRC7	8.2.3.3-7
LOGRC8	8.2.3.3-7
LOGRC9	8.2.3.3-7
LOGRFC	8.2.3.3-6
LOGS0	8.2.3.3-2
LOGS2	8.2.3.3-3
LOGS3	8.2.3.3-3
LOGS4	8.2.3.3-3
MEM4	8.2.2.2.5.4.1-3
MEMA1	8.2.2.2.5.4.1-3
MEMNUL	8.2.2.2.5.4.1-2
MEMRX	8.2.2.2.5.4.1-2
MEMRXa	8.2.2.2.5.4.1-2
MEMRXb	8.2.2.2.5.4.1-3
MEMRXc	8.2.2.2.5.4.1-3
MEMSVI	8.2.2.2.5.4.1-2

MIFAST	8.2.2.2.1-4
MOD3b	8.2.3.6-2
MODEM1	8.2.3.6-1
MODEM8	8.2.3.6-1
MODEMa	8.2.3.6-2
MODEMb	8.2.3.6-1
MODEMC	8.2.3.6-1
MODEMc	8.2.3.6-3
MOdEMD	8.2.3.6-2
MODEMe	8.2.3.6-3
MODEMf	8.2.3.6-3
MODEMg	8.2.3.6-3
MODEMh	8.2.3.6-1
MODEMi	8.2.3.6-2
MOVWRD	8.2.3.4.2-6
MOVWRDa	8.2.3.4.2-6
MOVWRDb	8.2.3.4.2-6
NCH2	8.2.2.2-5
NCHA0	8.2.2.2-6
NCHA1	8.2.2.2-6
NCHA4	8.2.2.2-6
NEEDRP	8.2.3.5.2.1.2-1
NETCOM	8.2.2.2.2-3
NEWB3	8.2.2.2-3
NEWB3a	8.2.2.2-3
NEWBUF	8.2.2.2-3
NEWT0	8.2.2.2.2-4
NEWT30	8.2.2.2.2-5
NEWT40	8.2.2.2.2-5
NEWTDM	8.2.2.2.2-5
NEWTEL	8.2.2.2.2-4
NEWTSB	8.2.2.2.2-5
NEXTBF	8.2.3.5-4
NOPE	8.2.2.1.1-5
NOPE2	8.2.2.1.1-5
NOPROT	8.2.3.5.2.1-2
NOSND	8.2.3.5.2.1.3-1
NWRFC1	8.2.3.3-8
NWRFC2	8.2.3.3-8
OI	8.2.2.2-1
OIBS	8.2.2.2.5.1-7
OIBSa	8.2.2.2.5.1-7
OIBSb	8.2.2.2.5.1-7
OID204	8.2.2.2-2
OID210	8.2.2.2-2
OID220	8.2.2.2-2

OID300	8.2.2.2-2
OID310	8.2.2.2-2
OIDISP	8.2.2.2-2
OIECHO	8.2.2.2-4
OIL1	8.2.2.2-6
OIL2	8.2.2.2-1
OIL4	8.2.2.2.2-1
OIL5	8.2.2.2.2-3
OILF	8.2.2.2.5.1-8
OITAB	8.2.2.2.5.1-9
OLDTEL	8.2.2.2.2-1
OOPS	8.2.2-2
OUNEW	8.2.3.5.2.2.1-2
PO1	8.2.2.2.5.1-4
PO1a	8.2.2.2.5.1-6
PO2	8.2.2.2.5.1-2
PO3	8.2.2.2.5.1-5
PO4	8.2.2.2.5.1-2
PO5	8.2.2.2.5.1-5
PO6	8.2.2.2.5.1-6
PO9	8.2.2.2.5.1-3
P10	8.2.2.2.5.1-3
P11	8.2.2.2.5.1-4
P13	8.2.2.2.5.1-2
P14	8.2.2.2.5.1-5
P15	8.2.2.2.5.1-6
P15a	8.2.2.2.5.1-6
P15b	8.2.2.2.5.1-6
P17	8.2.2.2.5.1-5
PAR	8.2.2.1.3.1-6
PEND1	8.2.3.1-2
PEND2	8.2.3.1-2
PEND3	8.2.3.1-2
PEND4	8.2.3.1-2
PEND5	8.2.3.1-2
PEND6	8.2.3.1-2
PEND7	8.2.3.1-2
PEND8	8.2.3.1-2
PENDA	8.2.3.1.1.2-1
PENDB	8.2.3.1.1.3-1
PENDB1	8.2.3.1.1.3-1
PENDB2	8.2.3.1.1.3-2
PENDC	8.2.3.1.2.1-1
PENDE	8.2.3.1.1.4-1
PENDE1	8.2.3.1.1.4-1

PENDI	8.2.3.1.2.2-1
PENDIN	8.2.3.1-2
PENDJ	8.2.3.1.2.2-1
PENDK	8.2.3.1.2.2-1
PENDT	8.2.3.1.1.1-1
PENDUN	8.2.3.1.2.4-1
PENDU2	8.2.3.1.2.4-1
PENRS3	8.2.3.1.2.3-1
PENRS4	8.2.3.1.2.3-1
PENRSC	8.2.3.1.2.3-1
PENRSI	8.2.3.1.2.3-1
PENRST	8.2.3.1.2.3-1
PRO3	8.2.3.2-3
PRO3a	8.2.3.2-3
PRO5	8.2.3.2-3
PRO5a	8.2.3.2-3
PROB1	8.2.3.2-2
PROB2	8.2.3.2-2
PROB3	8.2.3.2-2
PROBCK	8.2.3.2-2
PROT	8.2.3.5.2.1-2
PROTa	8.2.3.5.2.1-2
Q2	8.2.2.1.3.1-4
Q3	8.2.2.1.3.1-6
Q3a	8.2.2.1.3.1-6
Q6	8.2.2.1.3.1-7
Q8	8.2.2.1.3.1-8
Q9	8.2.2.1.3.1-8
Qba	8.2.2.1.3.1-7
RASREQ	8.2.3.5.2.1.5-1
RCTEIN	8.2.2.1.1-6
REG	8.2.2.1.1-5
REGa	8.2.2.1.1-5
RESET	8.2.1-4
RESET2	8.2.1-5
RESET5	8.2.1-4
RESETA	8.2.1-4
RESMOR	8.2.2.2.1-4
RETURN	8.2.2.1.3.1-5
RETURNa	8.2.2.1.3.1-5
RSTRST	8.2.3.5.2.1.2-1
RTS	8.2.3.5.2.1.1-3
SDIS	8.2.3.5-5
SENDA2	8.2.3.4.1-1
SENDA3	8.2.3.4.1-1
SENDAB	8.2.3.4.1-1

SENDID	8.2.2.2.2-2
SENDIT	8.2.3.4.2-2
SENDL1	8.2.3.4.2-4
SENDL1a	8.2.3.4.2-5
SENDL9	8.2.3.4.2-4
SENDOF	8.2.3.4.2-4
SENDW	8.2.3.4.2-3
SET6	8.2.2.2.4-2
SETOI	8.2.2.2.4-2
SETOI2	8.2.2.2.4-3
SETOIa	8.2.2.2.4-2
SETOIb	8.2.2.2.4-2
SETOIc	8.2.2.2.4-3
SETRST	8.2.3.3-4
SHUTDN	8.2.3.5.2.1.1-8
SKTT1	8.2.3.1.2.4-1
SLOWCR	8.2.2.2.5.2-1
SLOWCRa	8.2.2.2.5.2-2
SLOWCRb	8.2.2.2.5.2-1
SLOWCRc	8.2.2.2.5.2-1
SLOWCRd	8.2.2.2.5.2-2
SLOWCRe	8.2.2.2.5.2-2
SLURE	8.2.3.5-2
SLURP	8.2.3.5-2
SPBYDM	8.2.2.2.2-1
SPBYTEa	8.2.2.2.2-1
SPBYTEc	8.2.2.2.2-1
SSYN1	8.2.2.1.3.1-6
STR	8.2.3.5.2.1.1-3
STRY	8.2.3.5.2.1.1-6
SYNTAX	8.2.2.1.3.1-7
TE21	8.2.3.5.2.1.1-5
TE21a	8.2.3.5.2.1.1-5
TELDON	8.2.2.2.2-4
TEL100	8.2.3.1.1.1-3
TEL101	8.2.3.1.1.1-3
TEL200	8.2.3.1.1.1-3
TEL300	8.2.3.1.1.1-3
TEL310	8.2.3.1.1.1-4
TELI10	8.2.3.1.1.1-2
TELI20	8.2.3.1.1.1-2
TELIDL	8.2.3.1.1.1-1
TELNOT	8.2.2.2.2-4
TELQ	8.2.2.2.2-5
TELQ1	8.2.2.2.2-5
TELQ2	8.2.2.2.2-5

TELSLP	8.2.3.1.1.1-1
TESTOK	8.2.3.5.2.1.1-5
TILEAD	8.2.3.4-1
TOHOST	8.2.3.4.2-7
TOOBAD	8.2.3.5.2.1.1-4
TOUT	8.2.4-2
TOUTa	8.2.4-2
UNBLK	8.2.3.5.1-7
UNBLKa	8.2.3.5.1-7
UNBLKb	8.2.3.5.1-7
UNBLKc	8.2.3.5.1-7
UNBLKd	8.2.3.5.1-7
USELK2	8.2.3.5.2.1.1-4
USELNK	8.2.3.5.2.1.1-4