60470000

CONTROL DATA CORPORATION

COMMUNICATIONS CONTROL PROGRAM VERSION 1 REFERENCE MANUAL

CONTROL DATA®
CYBER 170 SERIES
CYBER 70 SERIES MODELS 72, 73, 74
6000 SERIES COMPUTER SYSTEMS
CYBER 18 COMPUTER SYSTEMS
255X HOST COMMUNICATIONS PROCESSORS

LIST OF EFFECTIVE PAGES

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

PAGE	SFC†	REV
Cover		-
Title		-
ii		B
iii		B
v		B
vii		A
viii		A
ix		A
x		A
xi		B
1-1		B
1-2		B
1-3		A
1-4		B
2-1		B
2-2		A
3-1		B
3-2		B
3-3		B
3-4		A
3-5		A
3-6		A
3-7		A
3-8		A
3-9		A
3-10		A
3-11		A
3-12		B
3-13		B
3-14		A
3-15		B
3-16		B
3-17		A
3-18		A
3-19		A
3-20		B
3-21		B
3-22		A
3-23		A
3-24		A
3-25		A
3-26		A
3-27		B
3-28		A
3-29		A

PAGE	SFC†	REV
3-30		A
3-31		A
3-32		A
3-33		A
3-34		A
3-35		A
3-36		A
3-37		A
3-38		A
3-39		B
3-40		B
3-41		B
3-42		B
3-43		B
3-44		B
3-45		B
3-46		B
4-1		B
4-2		A
4-3		A
4-4		B
4-5		A
4-6		A
4-7		A
4-8		A
4-9		A
4-10		A
4-11		A
4-12		A
4-13		A
4-14		A
4-15		A
4-16		A
4-17		B
4-18		A
4-19		B
4-20		B
4-21		A
4-22		A
4-23		A
4-24		A
4-25		B
4-26		A
4-27		A
4-28		A

PAGE	SFC†	REV
4-29		A
4-30		A
4-31		A
4-32		A
4-33		A
4-34		A
4-35		A
4-36		A
4-37		A
4-38		A
4-39		A
4-40		A
4-41		A
4-42		A
4-43		A
4-44		A
4-45		A
4-46		A
4-47		B
4-48		A
4-49		A
4-50		A
4-51		A
4-52		A
4-53		A
4-54		A
4-55		A
4-56		A
5-1		A
5-2		A
5-3		A
5-4		A
5-5		B
5-6		A
5-7		A
5-8		A
5-9		A
5-10		A
5-11		A
5-12		A
5-13		A
5-14		A
5-15		A
5-16		A
5-17		A

PAGE	SFC†	REV
5-18		A
5-19		A
5-20		A
5-21		A
5-22		A
5-23		A
5-24		A
6-1		A
6-2		A
6-3		A
6-4		A
6-5		A
6-6		A
A-1		A
A-2		A
A-3		A
A-4		A
A-5		A
A-6		A
A-7		A
B-1		B
B-2		B
B-3		A
B-4		A
B-5		A
C-1		A
Index-1		B
Index-2		B
Comment		-
Address		-
Cover		-

SFC † Software Feature Change

PREFACE

This manual describes Version 1 of the Communications Control Program (CCP 1) that is used with the CONTROL DATA® 2550 Series Host Communications Processor (HCP).

The manual is intended to provide sufficient information for computer programmers to make minor modifications in adapting these programs to specific user functions. The manual is not intended to provide complete explanations of the programs such as would be required to make major modifications to the programs.

It is recommended that the user be familiar with the PASCAL programming language, the NOS/BE operating system, and the CROSS support software system.

Unless otherwise noted, all numeric values are decimal.

Additional information on both the hardware and software elements of the Control Data 2550 Series Computer Systems and other related equipment and systems can be found in the following documents:

Publication	Publication Number
NOS/BE System Programmer's Reference Manual	60494100
NOS/BE Reference Manual	60493800
NOS/BE Installation Handbook	60494300
NOS/BE Operator's Guide	60493900
UPDATE 1.2 Reference Manual	60342500
CCP Support Software 1 Reference Manual	88988400
PASCAL Reference Manual	96836100
CCP Support Software 1 General Information Manual	88988600
CCP Support Software 1 Diagnostic Handbook	88988700
Micro Assembler Reference Manual	88988800
Macro Assembler Reference Manual	88988900
Link Edit/Library Maintenance Reference Manual	60471200
CCP 1 Software Diagnostic Handbook	60470200

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

CONTENTS

APPENDICES

FIGURES

TABLES

INDEX

GENERAL DESCRIPTION 1

INTRODUCTION

Version 1 of the Communications Control Program (CCP 1) comprises all software residing in the 2550 HCP, including the base system software (with multiplex subsystem), network communications software, and interface programs. The CCP provides front-end communications functions for the CDC 6000, CYBER 70, and CYBER 170 Computer Systems utilizing the NOS/BE operating system.

INFORMATION REPRESENTATION

The 2550 HCP, CYBER 70, and other devices with which the system interfaces use a variety of bit numbering conventions and data format expressions. The convention used for information representation throughout this manual is defined below.

From the point of view of both the host (CYBER 70, 170, 6000) and the 2550 HCP, "input" refers to data flowing upline from 2550 HCP to host. Similarly, "output" means data flowing downline from host to 2550 HCP.

To represent a field within a word or byte, the least significant bit of the smallest addressable unit is labeled bit zero. According to ASCII standard, this is the bit transmitted or received first. Pictorially, the least significant bit is shown at the right of all byte or word layouts.

Bytes are numbered in the sequence in which they are transmitted or received over an interface, with the first byte labeled byte zero.

Consecutive bytes of a data stream (e.g., as received from a communications line) are pictorially represented as one or more rows of bytes with byte zero at the upper left, independent of the manner in which the same data might be represented in a storage layout. In general, a contiguous data stream received from a communications line is not written into a single block of consecutive storage locations, but rather is split into scattered storage buffers that contain control and chaining information in addition to the stored data values.

Consecutive words of a control block or table entry are labeled with the address of the lowest-addressed word of the group. The lowest address is relative address zero and appears at the top of the pictorial.

SYSTEM HARDWARE

The CDC 2550 Series Computer Systems are designed for integration into a network communications system in which a large-scale computer (such as the CDC 6000 or CYBER 70/170) is the host computer for which the CDC 2550 Series Computer provides front-end communications services. Figure 1-1 illustrates a typical configuration for such a system.

The CDC 2550 Series Computers are currently offered in two models (2550-1 and 2550-2) in which, for the most part, hardware and software components are identical with the major differences occurring in the memory and line termination capacities. In this manual, the CDC 2550 HCP and its associated software are referred to as the Network Processing Unit (NPU).

In general, system hardware components are described in detail in separate documents (see preface); and, therefore, such descriptions are not repeated here.

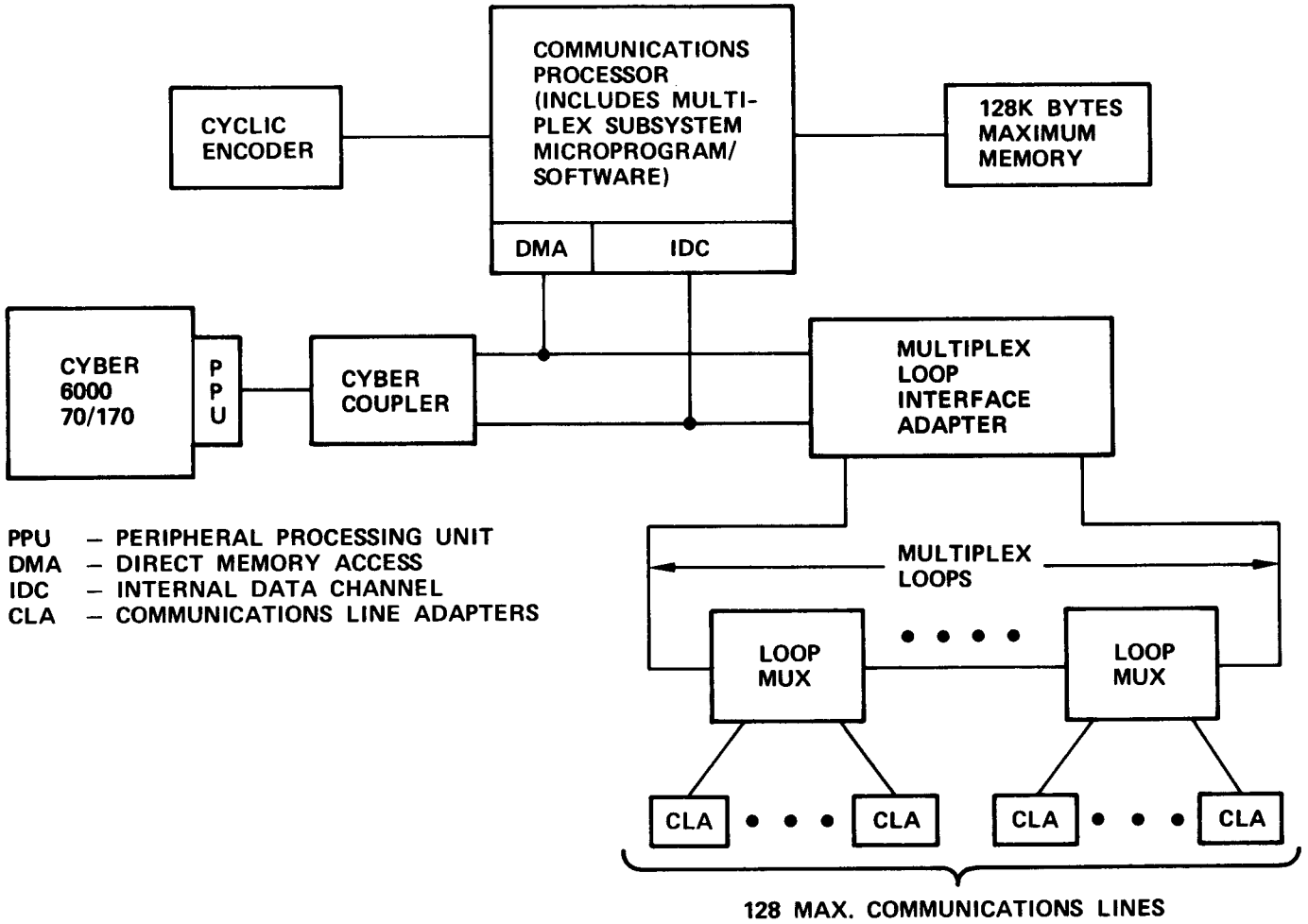


Figure 1-1. Typical System Configuration

Total system software falls into three major classifications: host software, the communications control program (CCP), and support software. Only the CCP is described in this manual.

HOST SOFTWARE

This manual describes only that version of the CCP which interfaces with the CDC NOS/BE operating system. In conjunction with the NOS/BE operating system, the NPU acts as a front-end and multiplexer to connect the host computer to varied communications terminals. The NOS/BE operating system access method is described in detail in other documents (see preface); and, therefore, such descriptions are not repeated here.

COMMUNICATIONS CONTROL PROGRAM

The CCP is the software package contained within the NPU. It is divided into three major parts: base system software, network communications software, and interface programs.

The base system software, which includes the multiplex subsystem, comprises those elements of the CCP that are required for all system applications. These include system subroutines that provide NPU resource allocation, memory space management, and control of peripheral devices directly associated with the NPU (local peripheral devices). Facilities to add line-dependent or terminal-dependent logic at various software levels are also provided. Basic line and terminal data structures support combinations of switched, dedicated, point-to-point, multipoint, synchronous, or asynchronous lines at a full range of standard line transmission speeds between 50 and 9600 bits-per-second (bps). A terminal connection may contain clustered devices with traffic addressability to the device level. The maximum number of devices associated with each line and the maximum total number of devices is not limited by the base system software; but, rather, is a function of memory space and the requirements of the given application.

The multiplex subsystem, which forms a major portion of the base system software, contains both hardware and software elements that establish data and control paths for information interchange between the communications lines and the appropriate protocol handlers within the NPU. The multiplex subsystem employs a "multiplex loop" concept in which data is received from communications lines and stored in line-oriented input buffers and data from line-oriented output buffers is distributed to the communications lines, all on a real-time demand-driven basis.

The network communications software provides network routing and service message processing to establish initial line and terminal configurations and to report error and traffic statistics.

The interface programs provide logical connections and interaction necessary to transmit information to and from the NPU. They include the host interface program (HIP) and terminal interface programs (TIPs). The HIP accomplishes block data transfers between the host computer and the NPU, monitors the host for failure and recovery, and reports host status to the NPU. The TIPs provide the logic to control the line and terminal protocol and to ensure orderly data transmission (in either direction) between the NPU and the connected terminal.

SUPPORT SOFTWARE

The CCP support software includes compiler, assembler, and utility programs for the development and maintenance of CCP software. The CCP support software runs on the CYBER 70/170 computers and may be used to enhance the CCP for adaption to particular user applications

The CCP support software system, described in detail in a separate document (see preface) consists of the following programs:

- Pre-Compiler
- 2550 Series PASCAL Compiler
- Format Program
- 2550 Series Micro Assembler
- 2550 Series Macro Assembler
- Library Maintenance Program
- Link Editor

Another specially designed support software program, the link editor provides additional off-line table initialization capabilities (see preface).

PROGRAMMING LANGUAGES

The 2550 HCP is a microprogrammable processor for which programs can be written at three different levels. For maximum efficiency, the micro assembler generates programs operating at the basic machine level, a macro assembler frees the programmers from much of the details of the microlevel, and the PASCAL compiler language may be used to generate the majority of functions.

Each programming language is described in detail in separate documents (see preface).

PROGRAM ADAPTION 2

INTRODUCTION

The CCP is designed to facilitate easy user-adaption to meet a wide range of applications. This is accomplished by permitting the user to select only those elements of the basic programs that meet the particular needs of his applications and allowing him to easily configure the selected elements to the types of terminals and the data rates and formats expected. Further, the CCP is designed to permit easy modification of certain basic program interfaces to accomplish special user functions; and, when needed, additions may be added to the software for applications not within the immediate contemplation of the manufacturer.

PROGRAM ELEMENT SELECTION

The selection of programs that are built into the CCP is controlled by UPDATE DEFINE, directives described in the UPDATE 1.2 Reference Manual. These are directives input to the UPDATE program that control the programs to be compiled and built into the CCP. These directives can be modified at the time of system installation so that certain programs and associated data structures are either included or excluded from the CCP, as necessary.

The features defined by these directives, which are optional and dependent upon specific installation requirements, include the following:

- TTY Terminal Driver - If an installation has teletype-compatible terminals, include the DEFINE TTY card in the UPDATE directive card deck. Causes all logic necessary to service teletype terminals to be built into the CCP.

- Mode 4 Terminal Driver - If an installation must service Mode 4 terminals, include the DEFINE MODE4 card in the UPDATE directives card deck. This causes all logic necessary to service Mode 4 terminals to be built into the CCP.

- Debug Aids - To include available debug aids into CCP generation, include the DEFINE DBUGALL card in the UPDATE directives card deck. This causes debug aids including the test utility program (TUP), breakpoint, and traps to be included in the CCP.

- Buffer Stamping - To invoke buffer stamping, include the DEFINE STAMPING card in the UPDATE directives card deck. Although classified as a debug aid, buffer stamping must be separately defined to be included in the CCP.

PROGRAM MODIFICATION

Program modifications can be categorized as either installation parameter changes or logic changes to source coding.

INSTALLATION PARAMETER CHANGES

Installation parameters provide a method to easily change CCP variables to comply with installation requirements. For expample, the 2550 memory size might vary for each installation. By appropriately changing the CCP variables

pertaining to memory size during system installation, such system variations may be easily accommodated.

Installation parameters may be changed either through changes to the Post Link Editor (LINKZAP) initialization directives or by changes to the PASCAL source global variables. LINKZAP initialization directive changes affect the initialization of CCP variables at the time of load module generation. PASCAL source global variables are changed by performing source code updates to the variables and recompiling.

Extreme care must be employed when updating these parameters as any errors can critically affect system operation. All installation parameters and acceptable value ranges for each parameter are identified in the CCP 1.0 installation procedures. All changes to these parameters must be within their specified ranges.

SOURCE CODING LOGIC CHANGES

To make modifications to logic in existing source programs, the programmer must be extremely familiar with all aspects of the system so that the changes do not negatively affect the CCP. Before any such changes are incorporated they should be verified as correct and compatible with the remainder of the system. This includes retaining consistency with the system global variables and the basic system philosophy of structural programming.

SOFTWARE ADDITIONS

CCP design permits easy addition of new software in several areas. Additions are made using PASCAL source programs and LINKZAP initialization directives. However, before incorporating any additions to the system, it is extremely important that the programmer be thoroughly familiar with the system to prevent deterioration of the system. It is also important to note that additions to the software may also entail additions to the system globals and LINKZAP initialization directives.

The following software areas are accessible for software addition:

- TIPs - To service terminals not usually supported by CCP 1.0, add terminal interface logic for the new terminals. Currently, CCP 1.0 supports only teletype and Mode 4 terminal types.
- POIs - Additional logic sequences can be added to the five predefined points-of-interface (POIs), such as those necessary to provide logic for functions such as auditing or statistics.
- Timing Procedures - To incorporate additional time-dependent functions, code the procedure and add its name to the table (CBTIMTBL) that drives all timing procedures.
- Worklist and OPS Level Program Expansion - To receive program control, new OPS level procedures must be added to the program table (B0PGMS) scanned by the system monitor. Associated with each entry in the program table is a worklist queue. Therefore, each new OPS level procedure must also have a worklist queue (B0WKLSTS) added to the system. All OPS level procedure additions entail adding the procedure source code, adding any system globals and LINKZAP directives, adding the procedure to the OPSMON program table, and adding an associated worklist queue.

INTRODUCTION

This section provides an overall view of the CCP software functions performed and the general manner in which these functions are accomplished.

LOAD PROCESS

Typically, NPU operating programs and tables are formatted into a load file that is resident in the mass storage of the host. To start NPU operation, that load file containing both the main memory-resident programs and writable micromemory-resident programs must be transferred (loaded) into the NPU.

The CCP load file contains all programs and tables except line control blocks and terminal control blocks. For line control blocks, the load file defines a contiguous space in main memory and, thus, fixes the maximum number of such blocks that can be configured. Terminal control blocks, however, are built in dynamically acquired space and, therefore, are not limited in the number that can be configured.

The format of a load file record includes a prefix made up of 15 60-bit words, a header that is a single 60-bit word, one or more blocks (each having a maximum of 120 16-bit words), and an end-of-record.

LOAD FILE PREFIX FORMAT

The load file prefix includes 15 60-bit words in the following format:

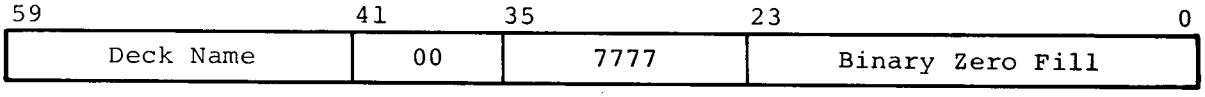
WORD	SIGNIFICANT BIT POSITIONS 59–47	41–35	29–23	17–0
0	7700	0016	Binary Zero Fill	
1	Deck Name		Binary Zero Fill	
2	Date			
3	Time			
4	Operating System Name		Oper. System Version	
5	Language Processor Name			Lang. Proc. Ver.
6	Lang. Proc. Mod. Level		Binary Zero Fill	
7	Binary Zero Fill			
8	Language Processor Information			
9	or			
10	Binary Zero Fill			
11				
12	User Comments			
13	or			
14	Binary Zero Fill			

Field values are in octal notation; words 2 through 14 are in display code. The prefix fields are described below:

Deck Name	Three-character deck (file) name in display code
Date, Time	Compile-time values returned by operating system DATE and CLOCK requests (display code)
Operating System Name, Language Processor Name	1 to 6 left-justified characters with binary zero fill (e.g. NOS/BE, CROSS, etc.)
Language Processor Modification Level	1 to 6 left-justified characters with binary zero fill giving language processor modification level such as PSR summary number or Julian type date of latest mod (e.g. 143 or 74190)
Language Processor Information	Options that affect the object code that are supplied by the language processor (e.g., DEBUG MODE or RELEASE MODE)
User Comments	Information supplied by the user, such as deck modification level (e.g., PSR LEVEL 53)

LOAD FILE HEADER FORMAT

The single 60-bit header word format is as follows:



The deck name in this word is the same as that appearing in the prefix.

LOAD FILE PACKET FORMAT

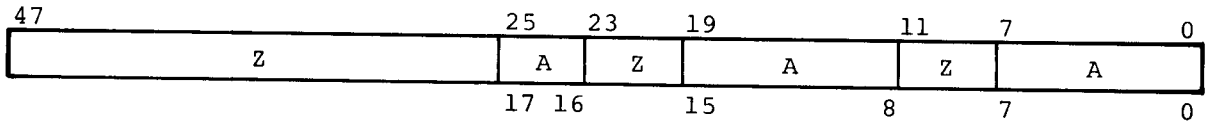
Each block contains a maximum of 120 contiguously located 16-bit NPU main memory words. The NPU main memory address of the first word of the group appears first, followed by the number of words (length) of the block in 16-bit words, followed by the word contents. These fields are formatted as illustrated below.

Successive blocks need not be contiguous in the NPU main memory nor in ascending memory address sequence. A block may contain any number of words between 1 and 120 (decimal).

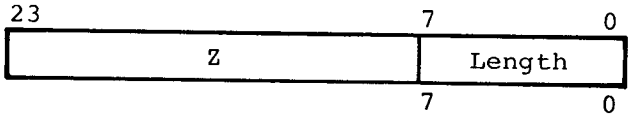
When the NPU is operating normally, it periodically sends status information to the host. If the NPU fails, the host detects the lack of status transmission and may initiate a dump process to transfer the contents of the NPU memory to the host.

To transfer (dump) information from the NPU to the host, the host executes the following procedure:

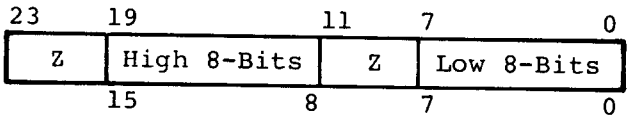
1. The host reads the three coupler registers (coupler status register, NPU status register, and order word register) and retains these values for incorporation into the register dump record (Record 4).



First Word (Address Field)



Second Word (Length Field)



NPU Main Memory Word Contents

A = NPU main memory address
Z = Binary zero Fill
Length = 1 to 170 (octal)

2. The host builds the dump header record (Record 1) containing the channel and equipment number of the coupler, the date, and the time.

3. The host reads the entire NPU main memory (starting at address zero) and formats the data into blocks containing up to 120 16-bit words each, with the entire group of blocks thus constructed comprising the main memory dump record (Record 2) of the dump file.

4. The host loads a "dump bootstrap" program into the NPU main memory starting at address zero and causes the program to be executed. This program overwrites a portion of the micromemory with a "micromemory dump" routine that copies micromemory into main memory and generates a 16-bit checksum of the micromemory dump. The "dump bootstrap" program then copies the 2550 file registers, writes the value 8 (decimal) into the NPU status register of the coupler (to indicate ready for dump), and halts.

5. The host then reads the NPU main memory and formats the micromemory dump record (Record 3) and the register dump record (Record 4).

Upon completion of the dump procedure the operator can initiate a reload of the NPU. In such a reload operation, the host fetches and writes the NPU load into the NPU main memory in an attempt to restart NPU operation. After writing each packet into the NPU main memory, the host reads back the packet and compares it to the load record. Any mismatch causes the host to note the failure.

DUMP HEADER RECORD FORMAT

The dump header record format consists of two 60-bit words followed by a blank fill character in bit position 59 of a third word, formatted as follows:

BITS	59	53	47	41	35	29	23	17	11	5	0
	CN	EN	bl	H	H	·	N	N	·	S	
	S	bl	M	M	/	D	D	/	Y	Y	
	bl										

where:
- CN = Channel number (binary)
- EN = Equipment number of NPU coupler (binary)
- HH = Hour
- NN = Minute
- SS = Second
- MM = Month
- DD = Day
- YY = Year
- · = Period character
- / = Slash character
- bl = Blank fill

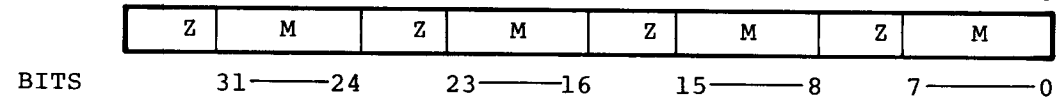
} Display Code

MAIN MEMORY DUMP RECORD FORMAT

The main memory dump record consists of multiple packets of data in the format previously described for the load file packet. All such blocks contain up to 120 NPU memory words. The first word address of the first packet is zero, and succeeding blocks contain the contents of successively higher addressed NPU memory words.

MICROMEMORY DUMP RECORD FORMAT

The micromemory dump record also consists of multiple packets in the format previously for the load file packet, with the first word address the micromemory address, and no block containing more than 60 words of 32 bits each in the word format shown below:

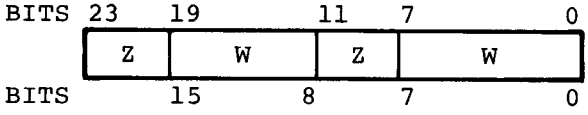


where: Z = Binary zero fill
M = Micromemory word contents (bits 0-7, 8-15, 16-23, and 24-31)

REGISTER DUMP RECORD FORMAT

The format of the register dump record (Record 4) consists of file register group 1 (256 16-bit words), file register group 2 (32 16-bit words), and three or more 16-bit register words. The first three register words are the coupler status register, NPU status register, and the order word obtained from the coupler prior to the main memory dump operation.

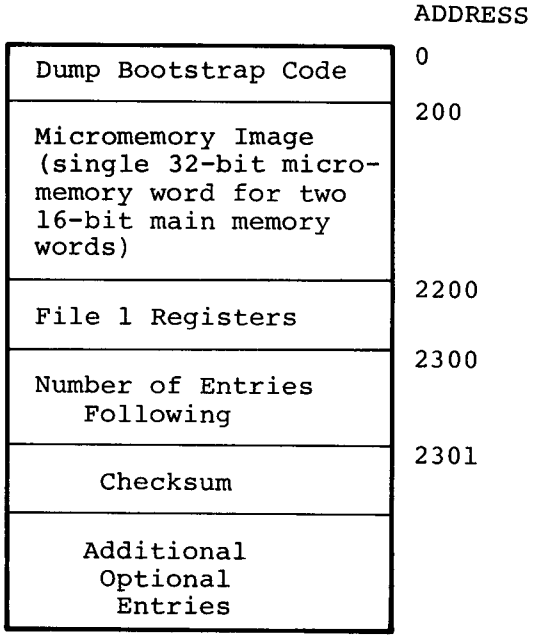
The file and register words are transferred in a 24-bit format as follows:



where: Z = Binary zero fill
W = Word contents

DUMP BOOTSTRAP NPU MEMORY FORMAT

The following is a memory map illustrating the dump bootstrap NPU memory format:



INITIALIZE PROCESS

Following NPU main memory load, the host issues a function code to start NPU operation. The NPU performs its initialization operations and begins normal coupler idle protocol by loading the NPU status register in the coupler with the IDLE code. The host may then begin configuration.

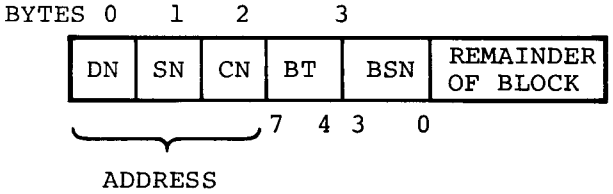
BLOCK PROTOCOL

The block is the basic unit of transmission through the NPU for the transfer of information in either direction between the host computer and the communications terminals. Blocks come in three general types: data, command, and control. A block, including header, is never more than 2048 bytes in length, but may be shorter.

For teletype or teletype-compatible terminals, a block usually contains a particular message or, when the message exceeds the maximum block length, the message is divided into two or more blocks. For Mode 4 terminals, the block corresponds to the unit of transmission to or from the terminal.

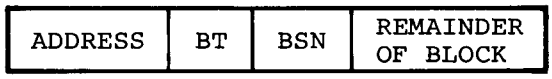
BLOCK FORMAT

Every block contains a header that, as a minimum, includes a three-byte address, a four-bit block type field, and a four-bit block serial number. The header may or may not contain other information, and its format along with the remainder of the block is a function of the block type (data, command, or control). The basic block header format is as follows:



where: DN = Destination Node
SN = Source Node
CN = Connection Number
BT = Block Type (Byte 3, bits 4-7)
BSN = Block Serial Number (Byte 3, bits 0-3)

As the address portion of all blocks is identical, block formats will hereafter be shown in the general form:



DESTINATION/SOURCE NODES

Within the block header, the destination node (DN) field identifies the receiving facility (node), and the source node (SN) field identifies the transmitting facility for the block in which they are contained. As each node field is a single byte, the permissible range of values is 0 through 255.

Generally, the host computer is assigned a single node identification (ID) number and each terminal connected to a particular NPU is assigned a unique ID. When the number of logical connections between a host computer and a particular NPU exceeds 255, however, additional node ID numbers (as many as four) are assigned to the host computer. Host interfaces are assigned the lower ID values and NPU (terminal) nodes are assigned the higher.

In a single-host, single-NPU system the host interface is assigned node ID zero and the terminal node is assigned node ID one. Thus, traffic going upline (NPU to host) has a destination node of zero and a source node of one. Traffic going downline (host to NPU) has a destination node of one and a source node of zero.

CONNECTION NUMBER

A logical connection is the association between a terminal control block (TCB) in an NPU and an application process (AP) in a host computer. By this logical connection, traffic is communicated between the terminal and the application process. The TCB contains all status information relative to a particular terminal and also contains a host-assigned connection number. The connection number is a single byte, yielding a permissible range of 1 to 255. Each block traveling downline to the TCB or upline from the TCB bears the connection number assigned to the TCB. Connection numbers assigned to all TCBs associated with a particular host-NPU pair must be unique.

SERVICE CHANNEL

A block with a connection number equal to zero is called a "service message" and the logical connection by which it is communicated is called the "service channel". Unlike logical connections that can be dynamically created or destroyed, the service channel always exists. Service messages are always commands and are used to establish logical connections and to communicate control, status, and error data in support of the common equipment and software servicing the logical connections.

BLOCK TYPES

There are three general block types: data, command, and control. However, there are two types of data blocks (BLK and MSG). Command blocks include the general command block (CMD) and the specially formatted service message block. Control blocks include four different control types (BACK, BREAK, RESET, and TERMINATE). Each is separately described in the following paragraphs. Table 3-1 lists and defines the block type (BT) codes placed in byte 3, bits 4-7, of the header.

TABLE 3-1. BLOCK TYPES

Block Type Code	Name	Function
1	BACK	Block Acknowledgement - indicates that the block with the same serial number which had been passed in the opposite direction via the same logical connection has been processed.
2	BLK	A block containing a portion, but not the last segment, of a data message.
3	MSG	A block containing the last segment, or all, of a data message.
4	CMD	Command
5	BREAK	A block indicating a discontinuity in the data stream traveling in the opposite direction.
6	RESET	An element which resets a data stream following BREAK.
7	TERMINATE	An element which terminates the logical connection.

DATA BLOCKS

A data block contains between zero and 2043 bytes of data immediately following a five-byte header. The general format for such blocks is as follows:

BYTES 0-2 3 4 5-2

ADDRESS	BLK or MSG	BSN	BFC	DATA

where: BLK
or
MSG = Block type code 2 or 3
BSN = Block serial number
BFC = Block format code

A data message is a self-contained data stream unit of communications terminated by an end-of-message indicator. In a half-duplex two-party communication, the end-of-message indicator signals that the transmitter is ready to receive.

If a message contains 2043 or fewer bytes, it may be transmitted by a single MSG-type block. If longer (or if for any other reason it is desired to segment the message), all segments except the last are transmitted via BLK-type blocks and the final segment is transmitted as a MSG-type block.

The block format code appears in every data block, both upline and downline. For all except downline blocks to a MODE 4 TIP, the block format code value is zero. For MODE 4 output data blocks, the block format code (BFC) is as follows:

BFC Code	Interpretation
0	Clear Write
1	Reset Write
2	Write

Data blocks are maintained in dynamically allocated data buffers and conform to the format shown in Appendix C of this manual.

COMMAND BLOCKS

The format for a command block is as follows:

ADDRESS	4	BSN	CC	OPTIONAL PARAMETERS

where: CC = Command code

Typically, a command block is imbedded within a stream of data blocks and is executed as it is encountered. That is, all data blocks received prior to the command must be transmitted to the terminal before the command is executed.

Because of the wide variety of commands and because certain commands only have significance to certain TIPs, command functions are described in conjunction with the descriptions of the TIPs which follow later in this section.

SERVICE MESSAGE BLOCKS

A service message is a command with a connection number (part of the address) equal to zero. The format for a service message block is as follows:

BYTES 0 1 2 3 4 5 6

DN	SN	0	4	BSN	DP	SP	SC	OPTIONAL PARAMETERS

ADDRESS — CONNECTION NUMBER

where: DP = Destination process
SP = Source process
SC = Service code

The three bytes DP, SP, and SC can be considered together as the command code.

Service messages generally include those employed in configuring lines and terminals and acquiring status information from the NPU.

CONTROL BLOCKS

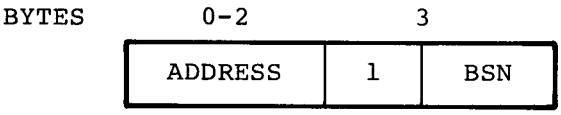
A logical connection consists of four logical channels: upline forward, upline reverse, downline forward, and downline reverse. Upline defines a block traveling from NPU to host, downline is from host to NPU. Forward identifies blocks transmitted by the generator of the block serial number (types BLK, MSG, or CMD) and reverse identifies blocks carrying response block serial numbers (types BACK or BREAK). The RESET and TERMINATE types always have a block serial number of zero.

Data and commands traveling downline are communicated via the downline forward channel and upline block acknowledgment (BACK) is sent to the host via the upline reverse channel. If the output stream is interrupted, the NPU sends a BREAK (instead of BACK) via the upline reverse channel. The NPU then accepts no further output data or commands on the logical connection until a RESET is received via the downline forward channel.

All blocks carry a block serial number (BSN). The highest number allowable for the BSN is set during program build by the block serial number limit (BSNL) parameter and can be any value between 1 and 15. Blocks traveling on forward channels are sequentially numbered beginning with zero and recycling to zero each time the BSNL is reached.

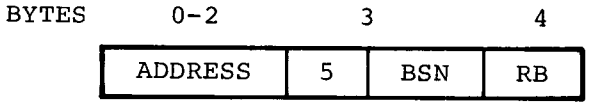
When BSNL is non-zero, a logical connection can only support that pre-established number of outstanding blocks (transmitted but not acknowledged) on a forward channel.

When an output data block is delivered to a terminal and receipt is verified by the terminal, the NPU returns a BACK control block to the host via the upline reverse channel. The format for the BACK is as follows:



The BACK will have a BSN equal to that of the block being acknowledged.

When it becomes necessary to discontinue output, the NPU sends a BREAK control block to the host via the upline reverse channel. That BREAK will have a BSN one greater than the previous BACK (modulo BSNL+1). The format for the BREAK is as follows:



where: RB = Reason for BREAK

The reason for BREAK field is zero for all except those generated by the Mode 4 TIP (see Mode 4 TIP description later in this section).

After sending a BREAK, the NPU discards all data and commands for that logical connection for which it has not sent a BACK and continues to do so until a RESET is received.

The process sending a BREAK must not send another BREAK until after it receives a RESET. The process receiving a BREAK must perform recovery from the BREAK condition. A TIP always recovers from a downline BREAK by retrans-

mitting to the host all data and command blocks sent to the host but not acknowledged when the BREAK is received. The way in which the host recovers is determined by the RB field.

The format for the RESET control block is as follows:

BYTES	0-2	3	
	ADDRESS	6	0

The RESET is communicated via the forward channel and data or commands following the RESET are queued for the receiving process. The block serial number (BSN) on the first data or command block following a RESET is the same as that which appeared in the BREAK that solicited the RESET.

The format for the TERMINATE control block is as follows:

BYTES	0-2	3	
	ADDRESS	7	0

No blocks are sent following a TERMINATE and the NPU processes a TERMINATE even if a BREAK is outstanding. After a host sends a TERMINATE, it may receive any block type before the upline TERMINATE response is received.

The NPU does not necessarily forward all outstanding BACKs before it sends an upline TERMINATE response. Blocks in the process of being transmitted to or from the terminal at the time the TERMINATE is received from the host are normally completed, but no new transmission is started or solicited. If receipt of input in response to a poll has not commenced when the downline TERMINATE is processed, such input is ignored if received. For TIPs operating non-buffered terminals, partial input blocks, assembled when the TERMINATE is processed, are discarded.

CONFIGURATION PROCESS

The host dynamically configures both lines and terminals for the NPU by sending downline messages during normal system operation. The downline service message to configure a line is sent for each line and defines the operating parameters for the line type. The NPU responds with a "configure line response" upline service message and the host sends an "enable line" message. When the line is enabled, the NPU returns a "line operational" service message to the host. The host then configures terminals for the line by sending one or more "configure terminal" messages. The NPU returns a "configure terminal response" message for each terminal configured.

Only the "configure line", "configure line response", "configure terminal", and "configure terminal response" service messages are described here. The "enable line" and "line operational" service messages are described in this section under the heading LINE AND TERMINAL STATUS.

CONFIGURE LINE SERVICE MESSAGE

Format for the "configure line" service message is as follows:

Byte Numbers											
0	1	2	3	4	5	6	7	8	9	10	11
DN	SN	0	4	BSN	1	0	0	PORT	SUB-PORT	LINE TYPE	TERM TYPE	MASK

where: DN = Destination Node Addr
SN = Source Node Address
BSN = Block Serial Number
PORT = 1 through 127
SUB-PORT = 0
LINE TYPE = (see table 3-2)
TERM. TYPE = (see table 3-3)
MASK = 0 (unless TERM. TYPE is 8, then see "Automatic Recognition")

LINE TYPE CODE

The line type code in the "configure line" service message is acquired by combining the CLA, modem, and circuit type information according to the specifications given in table 3-2.

TABLE 3-2. LINE TYPE CODES

Line Type Code	CLA Type	Modem Type	Circuit Type	Transmission Facility	Controlled Carrier
1	2560-1	RS232-201A Comp.	Switched	Half Duplex	Yes
2	2560-1	RS232-201B Comp.	Dedicated	Full Duplex	Yes
3	2560-1	RS232-201B Comp.	Dedicated	Full Duplex	No
4	2561-1	RS232-103E/113 Comp.	Switched	Full Duplex	No
5	2561-1	RS232-103 Comp.	Dedicated	Full Duplex	No

CLA types 2560-1 and 2561-1 are both general-purpose communications line adapters. Type 2560-1 operates synchronously and type 2561-1 operates asynchronously. Both types feature half- or full-duplex operation; even, odd, or no parity generation and checking; and self-test (loop back) mode. In addition, type 2560-1 has the following general features:

1. Code length 6, 7, 8, or 9 (8+1 parity) bits
2. Software established frame synchronization on character
3. All of the above features (including those in the paragraph above) can be selected by program command
4. Speeds to 9600 bps (determined by modem)
5. Provisions for external clock source
6. Full RS-232C/CCITT V24 interface
7. Data transfer overrun/underrun detection

Type 2561-1 has the following general features in addition to those given in the foregoing paragraph:

1. Code length 5, 6, 7, or 8 bits (exclusive of parity bit, if any)
2. All standard speeds to 9600 baud
3. Input and output speeds may be different
4. Stop bit length of 1, 1.5, or 2 bit times

Mask Value	Terminal Type Supported
1	5
2	6
3	5 and 6
4	7
5	5 and 7
6	6 and 7
7	5, 6 and 7

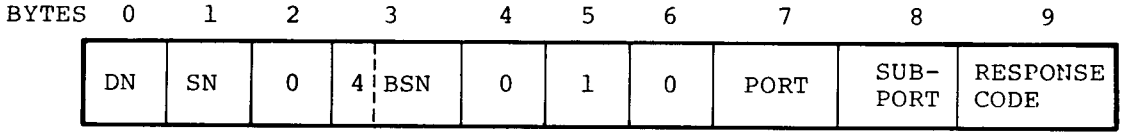
For automatic recognition of Mode 4 terminals, a convention is established that relates the address of the terminal controller to the terminal type as follows:

Terminal Controller Address	Terminal Type
70	5
71	6
72	7

Only one controller can be connected to a Mode 4 line for which automatic recognition is specified. When the line becomes operational, the TIP repeatedly polls those controller addresses permitted by the mask value. When a response is obtained to a poll, polling stops and the terminal type inferred from the responding controller is reported in the "line operational" service message.

CONFIGURE LINE RESPONSE SERVICE MESSAGE

In response to the "configure line" service message from the host, the NPU always returns the "configure line response" upline service message which is formatted as follows:



where: DN, SN, BSN, and SUB-PORT are as defined in the "configure line" service message and the RESPONSE CODE is interpreted as follows:

RESPONSE CODE	INTERPRETATION
0	Line configured
1	Line number too large or zero
2	Line already configured
3	Invalid line type
4	Invalid terminal type
5	Invalid mask

CONFIGURE TERMINAL SERVICE MESSAGE

After a line has been configured and the NPU has properly responded to the configuration, the line still must be enabled by an "enable line" service message to which the NPU responds with a "line operational" or "line inoperative" message.

When the line is operational, the host then proceeds to configure terminals for the line by issuing one or more "configure terminal" service messages. The format for the "configure terminal" message is as follows:

TABLE 3-3. TERMINAL TYPE CODES

Term Type Code	TIP Type	Char. Transmission Characteristics Key	General Description	Specific Terminals Supported
1	TTY	1	10 cps, 110 baud	Teletype M33, M35, and M38 CDC 713-10
2	TTY	2	15 cps, 150 baud	Teletype M37 CDC 713-10
3	TTY	3	30 cps, 300 baud	CDC 713-10
4	TTY	1-3	Automatic recognition of above line speeds	Any teletype compatible terminals specified above
5	MD4	4	Mode 4A BCD	214, 217 (200 UT), 713-12, 732-12, 714-1
6	MD4	4	Mode 4A ASCII	217, 731-12, 732-12, 734-1
7	MD4	4	Mode 4C	711-10, 714-10/20
8	MD4	4	Automatic recognition of above by repeated poll of Controllers 70, 71, and 72	Any Mode 4 terminals specified above, after address strapping which implies terminal type from controller address
9	TTY	5	60 cps, 600 baud	Teletype M40
A	TTY	6	120 cps, 1200 baud	Teletype M40

TABLE 3-4. CHARACTER TRANSMISSION CHARACTERISTICS KEY CODES

Key Code	Bits per Second*	Character Length	Stop Bit Length	Character Parity	Sync Character
1	110	8	2	none	-
2	150	8	1	none	-
3	300	8	1	none	-
4	-	7	-	odd	16 (hexadecimal)
5	600	8	1	none	-
6	1200	8	1	none	-

*Input and Output, Asynchronous Only

5. All of the above features (including those in the foregoing paragraph) can be selected by program command

6. Full RS-232C/CCITT V24 interface, including reverse channel detection and control, terminal busy, and originate mode

7. Break detection and generation

8. Data transfer overrun detection

The modem type code specifies an interface standard (e.g., EIA RS-232C) and one or more AT&T Data Sets for which the defined control procedures are compatible. Modems produced by other manufacturers may be used if they are compatible with the listed AT&T Data Sets.

For switched lines, the modem is conditioned by the "data terminal ready" interface signal and answers incoming calls upon receipt of the "ring indicator" signal from the modem.

Communications lines must be identified as either half-duplex (HDX) or full-duplex (FDX), representing the characteristics of the communications facility and not the mode of data transfer over the line. Thus, it is important not to assume that a 2-wire circuit is necessarily a half-duplex facility as some modems operate full-duplex with 2-wire circuits.

The NPU can operate full-duplex facilities with a constant carrier or controlled carrier. With constant carrier, the transmit carrier remains on continuously; and line failure is reported if the received carrier remains off for a period equaling or exceeding the failure verification period. With controlled carrier, the transmit carrier is raised and lowered with each transmission block and the received carrier is expected to behave similarly.

TERMINAL TYPE CODE

The terminal type code contained in the "configure line" service message is acquired as specified by table 3-3. Table 3-4 defines the character transmission characteristic key codes employed in forming the terminal type code.

AUTOMATIC RECOGNITION

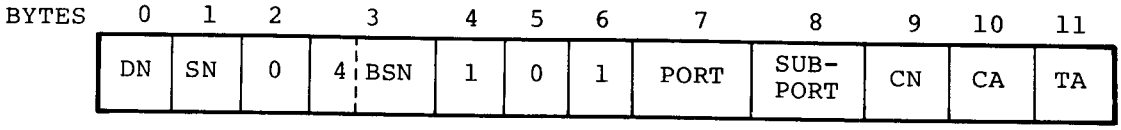
Automatic recognition is an option available only on switched lines. If selected (by specifying terminal type 4 or 8), the TIP identifies the terminal type by analysis of input after the line becomes operational.

The user of a teletype-compatible terminal on an automatic recognition line must press the carriage return key after line connection is established. The TIP programs the CLA to sample the line at 300 baud, and the first character received by the TIP indicates the speed of the terminal as follows:

Hexadecimal Character Received	Terminal Speed (in baud)
9C or 8C	110
E6	150
8D	300

The TIP then reports the current terminal type in the "line operational" service message and resets the CLA to sample and transmit at the indicated rate.

For Mode 4 automatic recognition (terminal type 8), a 3-bit mask value specifies which of the Mode 4 category terminals are to be supported on the line, as follows:

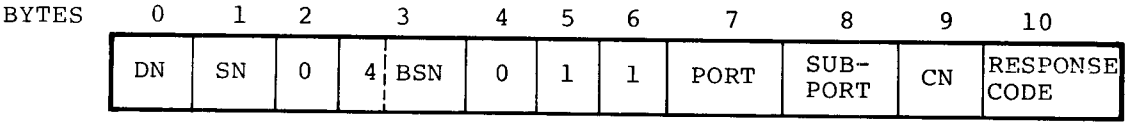


where: DN, SN, BSN, PORT, and SUB-PORT are as defined in the "configure line" service message
CN = Connection Number
CA = Cluster address
TA = Terminal address

A terminal control block (TCB) can be built only when a line is enabled and operational and such a block remains in existence until a TERMINATE is received or a "disable line" or disconnect line" service message is processed.

CONFIGURE TERMINAL RESPONSE SERVICE MESSAGE

In response to the"configure terminal" message, the NPU always returns a "configure terminal response" service message formatted as follows:



where: DN, SN, BSN, PORT, and SUB-PORT are as defined in the "configure line" service message
CN = Connection number and RESPONSE CODE is interpreted as follows:

RESPONSE CODE	INTERPRETATION
0	Terminal control block (TCB) built
1	Line inoperative*
2	TCB already exists
3	Line not enabled
4	Invalid line no.
5	CN already assigned
6	Invalid CA
7	Invalid TA

*If a line becomes inoperative prior to receipt of the "configure terminal" message, the NPU first reports "line inoperative", then responds to the "configure terminal" message with a "configure terminal response" message in which the response code indicates line inoperative (code 1).

LINE AND TERMINAL STATUS

The following service messages are employed to establish line and terminal status. In general, lines can be enabled, disabled, disconnected, and ruled operational or inoperative. Terminal status is normally reported by a TIP when terminal failure is detected via an upline command.

ENABLE LINE SERVICE MESSAGE

In the normal sequence of events, the host responds to a "configure line response" service message from the NPU by issuing an "enable line" service message in the following format:

BYTES	0	1	2	3	4	5	6	7	8
	DN	SN	0	4 BSN	2	0	0	PORT	SUB-PORT

DN, SN, BSN, PORT, and SUB-PORT are as defined in the "configure line" service message

LINE OPERATIONAL/LINE INOPERATIVE SERVICE MESSAGE

The response to a downline "enable line" service message is either a "line operational" or "line inoperative" service message, as appropriate. The format for the "line operational" service message is as follows:

BYTES	0	1	2	3	4	5	6	7	8	9
	DN	SN	0	4 BSN	0	3	0	PORT	SUB-PORT	CTT

DN, SN, BSN, PORT, and SUB-PORT are as defined in the "configure line" service message and CTT is the current terminal type.

Upon receiving the "line operational" service message, the host normally configures the terminals for the line by sending one or more "configure terminal" service messages.

The "line inoperative" service message is sent from the NPU to the host only when an attempt is made to enable an inoperative line or when line or modem conditions cause the line to become inoperative. This service message is not sent when the line is disabled, disconnected, or terminated by the host. The format for the "line inoperative" service message is as follows:

BYTES	0	1	2	3	4	5	6	7	8	9
	DN	SN	0	4 BSN	0	3	1	PORT	SUB-PORT	LEC

DN, SN, BSN, PORT, and SUB-PORT are as defined in the "configure line" service message and the line error code (LEC) is interpreted as follows:

LEC	INTERPRETATION
0	Re-enable line at first opportunity.
1	Re-enable line after predetermined timeout.
2	Do not re-enable line until next initialization of NPU.

LEC equal to zero is the normal response for a switched line. LEC equal to one indicates an abnormal modem signal or diagnostic test to prevent line enables from flooding the NPU. LEC equal to two indicates a bad CLA or other long-term outage.

Terminal control blocks (TCBs) are not automatically deleted when a line becomes inoperative. The host must explicitly TERMINATE each logical connection or command a "disable line" or "disconnect line".

The following modem conditions cause the line to be reported as inoperative. Timeouts used in conjunction with conditions ensure that lines are not declared inoperative because of transient conditions.

Data-Set-Ready - If data-set-ready (DSR) drops, data-terminal-ready (DTR) is immediately turned off and "line inoperative" is reported with LEC = 0 for switched lines (types 1 or 4) and LEC = 1 for dedicated lines (types 2, 3, or 5).

Clear-to-Send (Type 201 modems only, line types 1, 2, or 3) - If clear-to-send (CTS) does not rise within one second after request-to-send (RTS) and fall within one second of the fall of RTS, DTR is turned off and "line inoperative" is reported with LEC = 1. CTS is not monitored for 103/113 modems (line types 4 or 5).

Data-Carrier-Detect (FDX constant carrier, line types 1, 2, or 3) - Once the line is operational, if data-carrier-detect (DCD) drops and remains off for 10 seconds, DTR is turned off and "line inoperative" is reported with LEC = 1. Abnormal operation of DCD on HDX lines or controlled carrier lines (line types 1 or 2) does not influence line status.

TERMINAL STATUS

Terminal status is reported to the host via an upline command message from the TIPs when terminal failure is detected. The format of these command messages is described later in this section. Such messages do not change the state of the terminal control block (TCB) with regard to the logical connection nor is the the line state (as recorded in the line control block) modified. The host may attempt to recover from terminal failure or may TERMINATE the associated logical connection.

HOST INTERFACE PACKAGE

CYBER COUPLER HARDWARE PROGRAMMING

The CYBER coupler is the hardware interface between the CYBER 70/170 host and the NPU. The host may interface with one or two couplers on the same channel, but the NPU interfaces with only one coupler.

The coupler, essentially, has three transmission circuits:

1. A half-duplex data circuit for transmission of programs or data between the host memory and the NPU memory.

2. A full-duplex control circuit over which the host and NPU perform necessary "handshaking" protocol.

3. A supervisory circuit that monitors transaction status. The coupler also provides an execution control circuit used by the host to start or stop NPU microprogram execution or to reset the microinstruction address counter.

REGISTERS

The coupler registers directly accessed by the host for normal data transmission and status reporting are illustrated in figure 3-1 and are briefly described as follows.

1. Coupler Status Register - A 16-bit register of which only the 12 low-order bits can be read by the host. This register identifies the reason for an interrupt or alarm and indicates occurrence of a transaction or change in register status to both the host and the NPU. See table 3-5 for register specifications.

2. NPU Status Word - A 16-bit register of which only the 12 low-

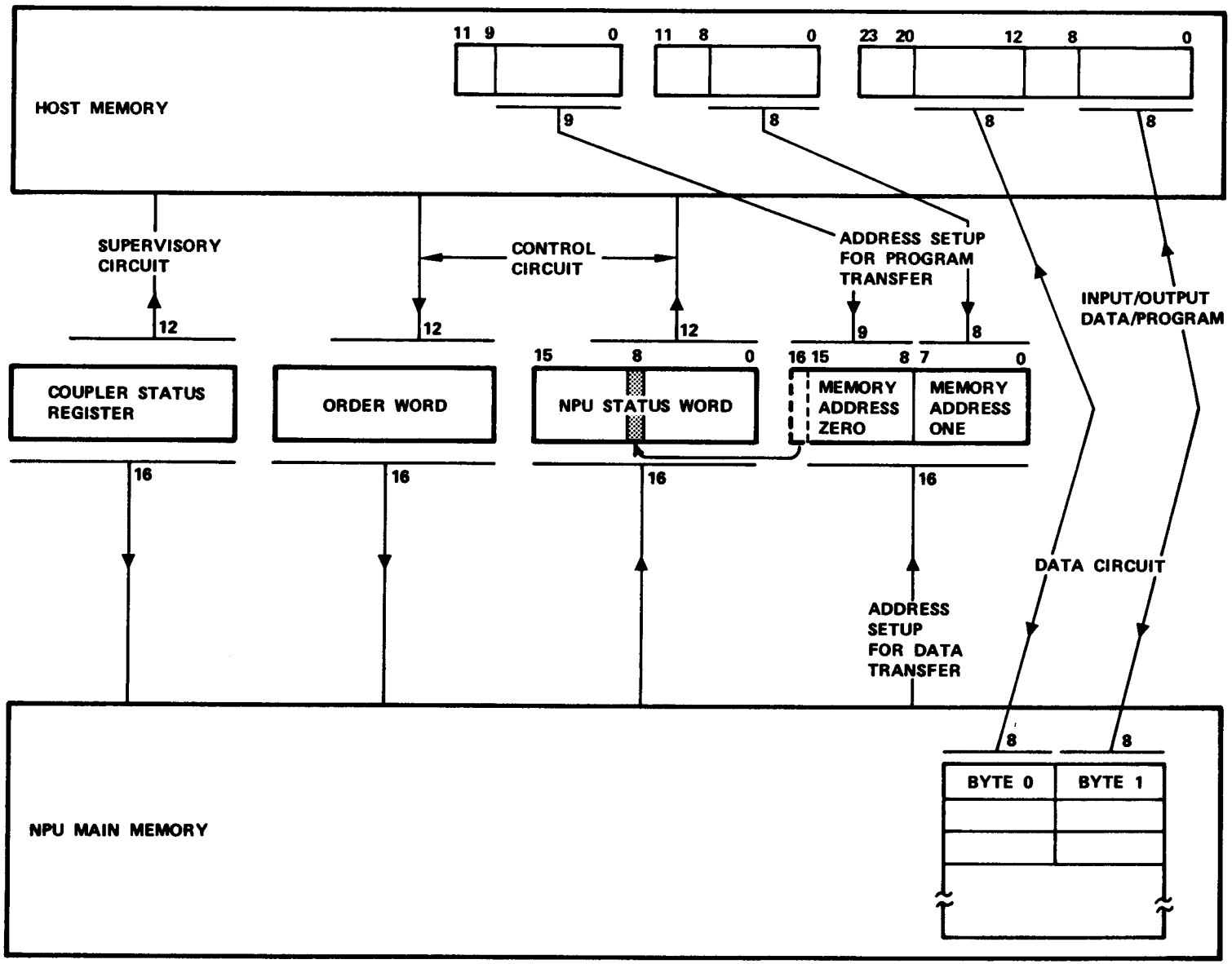


Figure 3-1. CYBER Coupler Registers

TABLE 3-5. COUPLER STATUS REGISTER

Bit Number	Flag Name	Set Condition	Interrupt Alarm*	Reset** Condition
0	Memory Parity	NPU memory parity error	A	1
1	Memory Protect Fault	NPU memory protect fault	A	1
2	NPU Status Word Loaded	NPU writes status word	-	2
3	Memory Address Register Loaded	Host or NPU writes memory address one	-	3
4	External Cab. Alarm	Power failure	I	4
5	Transmission Complete	Host completes any input or output operation	I	4
6	Transfer Terminated by NPU	NPU terminates transfer (not used)	I	4
7	Transfer Terminated by Host	Host sets channel inactive during data I/O	I	4
8	Orderword Register Loaded	Host writes orderword	I	5
9	NPU Status Read	Host reads NPU status word	-	4
10	Timeout	Coupler selected and active 3+ seconds in host data I/O operation	I	1
11	CYBER 170 Channel	Enable parity switch on and data channel 12-bit word (plus parity) not odd	A	6
12-13	Not Used			
14	Chain Address Zero	Coupler finds zero in last word of NPU buffer	-	4
15	Alarm	Positive transition of any flag causing alarm	A	4

NOTES:

*Raising associated flag causes alarm (A), interrupt (I) or neither alarm or interrupt (-).

**All flags are reset (cleared) by Master Clear. All except bit 2 are reset when NPU or host clears the coupler. Other reset conditions are as follows:

1. Reset when coupler status register is cleared.
2. Reset when host reads NPU status word.
3. Reset on first direct memory access (DMA).
4. Reset when NPU reads coupler status register.
5. Reset when NPU reads orderword.
6. Reset when NPU reads coupler status register or by enable parity switch positive transition.

order bits can be read by the host. This register is used by the NPU to communicate software-defined status codes to the host. These status codes are interpreted as follows:

Code Value	Interpretation
0	Ignore value and read again
1	Idle
4	Ready for output
7	Not ready for output
8	Ready for dump
11	Input available - other than BLK or MSG
13	Input available - BLK or MSG with no more than 248 characters
14	Input available - BLK or MSG with more than 248 characters.

3. NPU Order Word - A 16-bit register of which only the 12 low-order bits can be written into by the host. This register permits the host to communicate a software-defined order code to the NPU. Output ready is the only order currently used. The code identifies the type of output available as follows:

Code Value	Interpretation
1	BACK block
2	BLK block
3	MSG block
4	CMD block
5	BREAK block
6	RESET block
7	TERMINATE block

4. NPU Address Register - A 17-bit register, all of which can be written into by the host, is used in loading or dumping NPU memory. The high-order 9 bits (address register bits 16-8) are designated as "memory address zero" and the low-order 8 bits (address reigster bits 7-0) are designated "memory address one". The host writes addresses into the register by first writing "memory address zero", then "memory address one". Address register bit 16 is actually implemented as NPU status word bit 8 and, therefore, cannot be used for other purposes.

COUPLER OPERATING MODES

The coupler employs four operating modes: NPU control, load/dump, single word transfer, and block transfer. Two of these modes (load/dump and block transfer) transfer information between the coupler and the NPU memory via a direct memory access (DMA) port. The DMA port transfers 16-bit words, but the host transfers only 12-bit words to or from the coupler. Therefore, to form a full 16-bit word in either direction, the host performs two transfers, the first transferring bits 15-8 (byte 0) of the 16-bit NPU word and the second transferring bits 7-0 (byte 1). The four high-order bits of each host word are not transferred to the NPU and when transferring from NPU to host, the coupler sets the four high-order bits of each host word to zero. As an even number of 12-bit words must be transferred in each transaction, the coupler adds a word of all zeros when an uneven number of bytes is encountered.

When the coupler operates in the block transfer mode, data is transferred as 8-bit characters occupying bit positions 7-0 in the 12-bit host word and either bit positions 15-8 or 7-0 in the 16-bit NPU word. During both input and output, bit 11 of the host word is set to indicate the last character of a transmission. Bits 10-8 of the host word are not normally used. If, however, they are set to a non-zero value during an output transaction, the coupler executes an immediate chain to the next buffer sequence. This forced chaining prior to an end-of-buffer condition should be used sparingly to prevent excessive buffer usage.

Characters are transferred to and from contiguous host memory locations, but locations within the

NPU memory buffers and the buffer chaining mechanism are of no consequence to host operation.

When the coupler operates in the NPU control mode, it allows the host to stop and restart execution of the NPU microprogram and to set the microinstruction counter to zero.

The single-word transfer mode is used for NPU status word and order word transfers. The NPU can write into the NPU status word at any time, but the host can read it only after it has been loaded by the NPU and must not read it again until a new word is loaded. The host determines when the NPU status word is loaded by testing bit 2 of the coupler status register. This bit is set when the NPU loads the word and reset when the host reads the word. Similarly, the host can load the order word at any time and the NPU can read it only after it has been loaded. Bit 8 of the coupler status register is set when the host loads the order word and is reset when the NPU reads the word.

For block (multiple character data) transfer operation, the coupler requires cooperation of both the host and NPU. Either the host or NPU may initiate the operation and, when both have completed setup, the transfer can take place. For output, the host cannot directly determine if the NPU has completed setup and, therefore, must rely upon cooperation by the NPU. For input, the host can test the channel to determine whether the channel is full or empty. Unless the channel is filled by the NPU within 12 microseconds after the input request to the coupler, a failure is signaled.

The NPU sets up its side of the coupler for a data transfer by loading the buffer length register (not used by the host) and storing the address of the first buffer of a buffer chain into the NPU memory address register. Output transfer is terminated by the presence of bit 11 in any character in the host output data stream. The host must disconnect the channel following transfer of this word. Input transfer is terminated when the last valid character of the last buffer of a buffer chain is transferred. The last character is stored in the host with bit 11 on and the coupler automatically disconnects the channel after the word is transferred. If bit 15 of the second word is set to one, it indicates the last buffer.

HOST INTERFACE

The coupler is programmed from the host side by setting a function code and executing an I/O instruction. Table 3-6 lists and describes these function codes which occupy the nine low-order bits of the 12-bit host word. The equipment code (coupler address on the channel) is contained in the three high-order bits. The equipment code is determined by switch settings on the coupler.

Load, dump, and multiple character transfers occur at a maximum rate of one 12-bit host word per microsecond. DMA contention may, however, cause a somewhat lower overall rate; but such delays should be infrequent and of relatively short duration.

NPU INTERFACE

The NPU issues commands using the SIO instruction to program the coupler. The SIO is also used to read the order word and to write the status word. Data block input/output takes place via direct memory access (DMA) which is transparent to the software. Table 3-7 lists and describes the NPU commands.

DATA TRANSFER PHYSICAL PROTOCAL

The data transfer physical protocol performs data transfer and error checking. The physical protocol is described by a pair of flow diagrams, one showing operation of the host (figure 3-2) and the

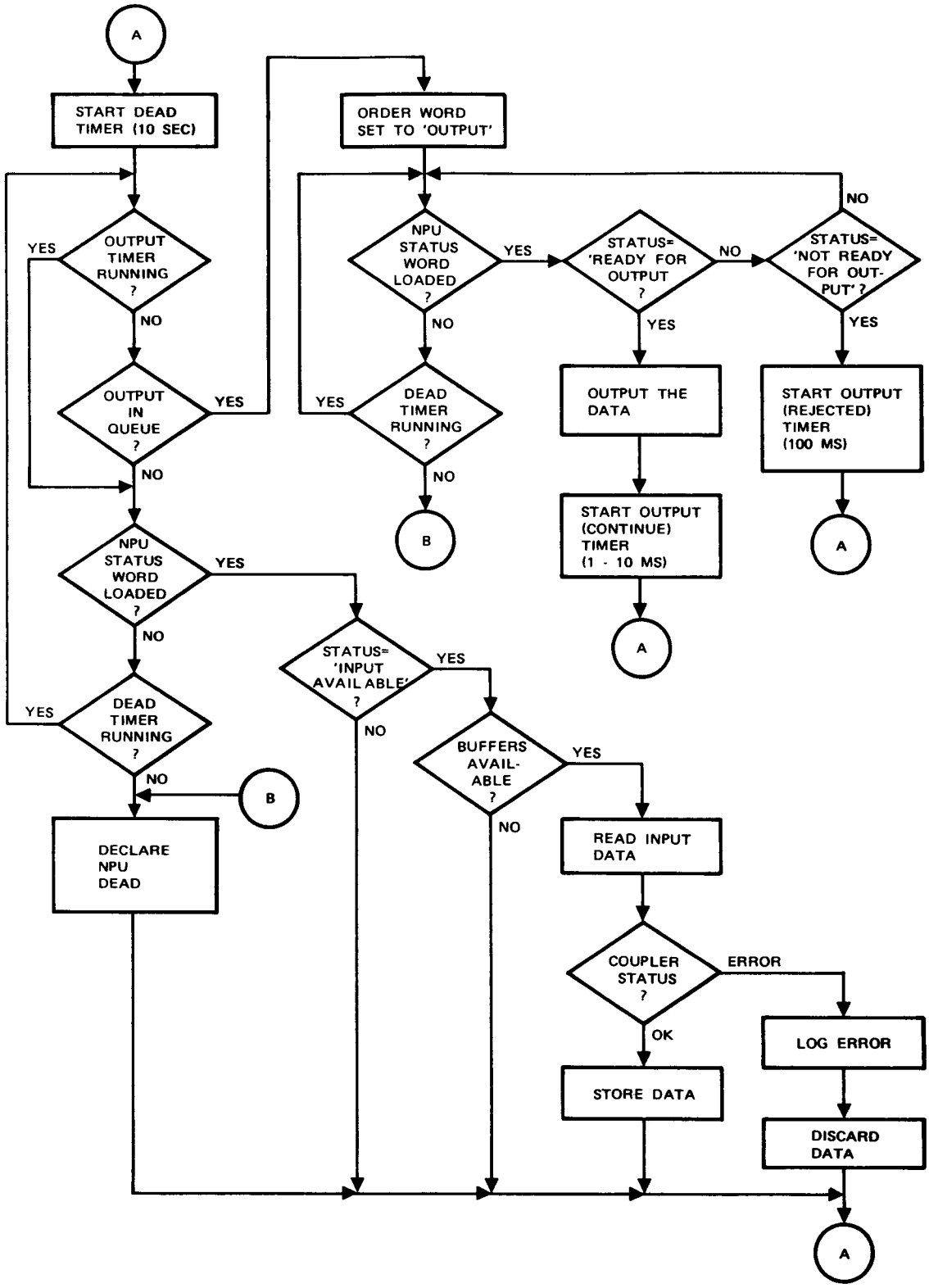


Figure 3-2. Data Transfer Protocol - Host Sequence, Flow Diagram

TABLE 3-6. HOST FUNCTION CODES

Host Function Code	Octal Value	Description
Clear NPU	200	Used prior to loading or dumping the NPU, stops NPU operation and sets the micromemory address register to location zero.
Start NPU	040	Starts the NPU emulator (microcode) at the address in the micromemory address register. Emulator must always be started at location zero; therefore NPU must be cleared before issuing this function code.
Input Program	007	Used to dump NPU main memory.
Output Program	015	Used to load NPU main memory.
Clear Coupler	400	Resets coupler control logic and most registers.
Output Memory Address Zero	010	This pair of function codes is used to set NPU main memory accessing for load and dump operations.
Output Memory Address One	011	
Output Order Word	016	Loads the coupler orderword register. Causes an NPU interrupt.
Input Coupler Status	005	Checks the state of various registers and flags in the coupler. Tests whether NPU has loaded the NPU status word.
Input NPU Status	004	Inputs NPU status word previously loaded by the NPU.
Input Order Word	006	Allows host to read back order word it has written. Used only prior to NPU dump operation.
Input Data	003	Allows characters to be input to the host. Coupler must have been previously set up by the NPU.
Output Data	014	Allows characters to be output from the host. The coupler must have been previously set up by the NPU.

TABLE 3-7. NPU COMMAND CODES

NPU Command	Hexadecimal Value	Description
Input Switch Status	0654	Allows NPU to check host data channel device address, online/ offline switch setting, alarm override switch setting, etc. Used during initialization.
Output Buffer Length	0658	Sets coupler to follow NPU buffer chains for current buffer length in use. Used during initialization and each data transfer.
Clear Coupler	060C	Resets coupler control logic and most registers. Used during protocol error processing. NPU status word contents, except for bit 8, are not affected.
Input Coupler Status	0650	Used in NPU interrupt handler to determine reason for interrupt.
Input Order Word	0660	Used in NPU interrupt handler to input order word previously loaded by the host.
Output NPU Status	0648	Used to send control codes to the host.
Output Memory Address	066C	Sets up coupler for data transfer by pointing coupler to the start of an NPU buffer chain.

other showing operation of the NPU (figure 3-3).

In figure 3-3, a large arrowhead is used in some locations to indicate a point at which the NPU waits for the next coupler interrupt. While waiting, the coupler program reentry point is saved and the "dead" timer runs while the NPU services other processes. When the interrupt occurs, the NPU resumes service of the coupler at the location saved. If the reason for the interrupt is one of those listed below the arrow, the service proceeds as shown. If an interrupt occurs for some other reason, an error has occurred. In such case, the error is logged and the protocol is restarted at point A. If the "dead" timer timeout occurs before the interrupt, the host is classified by the terminals as being down, but the transaction remains pending. When the transaction is completed, the terminals again consider the host to be operational.

As detailed in figures 3-2 and 3-3, the principal features of the protocol are:

- At any time, the host can order output or the NPU can specify input available and set the coupler appropriately.

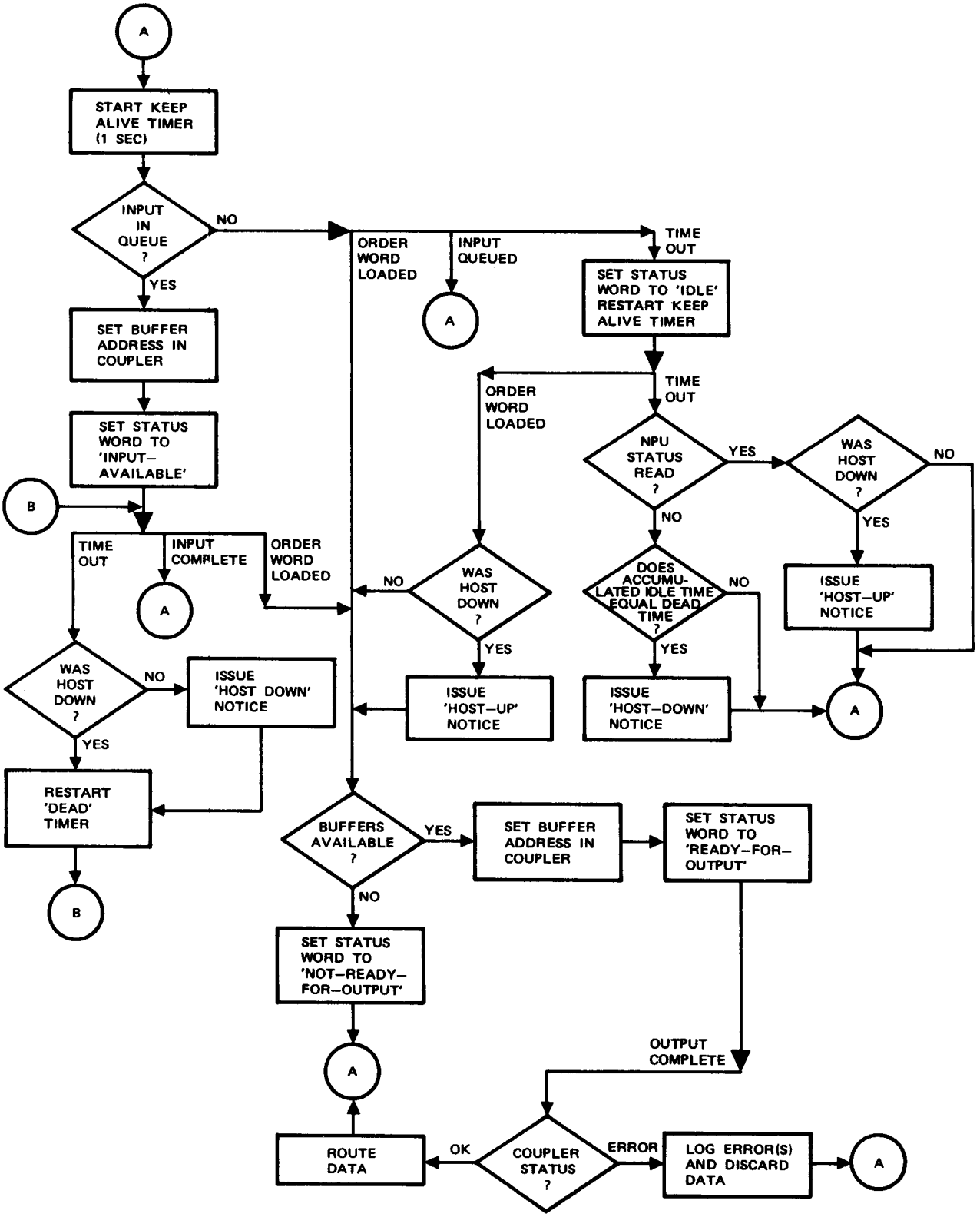


Figure 3-3. Data Transfer Protocol - NPU Sequence, Flow Diagram

- If a conflict occurs, the NPU normally allows output, but the NPU can reject output when buffer space is lacking.
- The host can refuse input by simply ignoring the input available signal from the NPU.
- If the dead timer in the host times out, indicating a response failure, the NPU is reloaded. If the dead timer in the NPU times out, indicating a host busy or failure condition, the protocol remains in its present state.
- If the NPU rejects output, the host performs a short timeout before again requesting output to prevent swamping the NPU with interrupts.
- If the NPU accepts output, the host allows the NPU to indicate if input data is available before again ordering output.
- Once a data transfer is initiated, the transaction must be completed or the entire transaction is discarded.
- The receiver (host or NPU) performs error checking and, if detected, errors are logged in the CE error file, the data received is discarded, and the protocol is reset. A retry is not automatically attempted.

DATA TRANSFER

The coupler can perform block transfers in only one direction at a time. Therefore, the data transfer protocol is half-duplex. The host and NPU independently bid for the channel. The host bids for the channel by issuing an output order word with the OUTPUT function code. The NPU bids by commanding the output memory address to point to the start of the input block buffer chain and further commanding output NPU status with the input-available status bit set. If both host and NPU bid for the channel at approximately the same time, the NPU normally allows output and rebids for input at completion of the output operation.

The NPU receives an interrupt when the host writes an order word or completes a data transfer. The coupler status word indicates the reason for interrupt. Therefore, the host need not separately indicate via the control circuit that a transaction is complete, as that information is automatically available via the supervisory circuit.

ERROR CHECKING

Errors are of three types: contaminated data, incomplete transactions, or failure of the interface to respond. The first two types are handled by the physical protocol. Only good and complete data blocks are accepted and bad blocks are discarded. Physical level protocol does not perform transmission retry or attempt recovery of discarded or lost blocks. Interface failure causes the interface to be declared inoperative, but the protocol returns to the initial state and waits for interface response.

TIMERS

Five timers (three in the host and two in the NPU) are used by this protocol. The host contains the "dead", "output continue", and "output rejected" timers and the NPU includes "keep alive" and "dead" timers.

Three timers accomplish failure detection. The 1-second "keep alive" timer in the NPU provides periodic idle status signals to the host when no traffic is in progress. The 10-second "dead" timer in the host times out if the host fails to receive either an idle or an input request signal

within its 10-second period. Upon such "dead" timer timeout, the host declares the NPU to be dead and initiates the NPU dump/reload sequence. Similarly, the NPU 30-second "dead" timer senses coupler interrupts and, if none arrive within the 30-second period, the NPU declares the host dead, but continues the transaction.

When the host and the NPU both bid for the channel at approximately the same time, the contention is resolved in favor of the host by permitting output. When the output transaction ends, the host starts a brief (1 to 10-millisecond) "output continue" timer to allow the NPU to request input if the NPU has data queued for the host. This timer prevents the host from monopolizing the channel with output and flooding the NPU with data.

If the NPU encounters a scarcity of buffers, it rejects the host's request for output. To regulate the rate at which the host bids for the channel, a 100-millisecond "output rejected" timer limits the frequency of coupler interrupts to the NPU when the NPU has a scarcity of buffers.

HOST FAILURE AND RECOVERY

When the NPU detects failure of the coupler to provide an interrupt before "dead" timer timeout, it causes a message (HOST DOWN) to be placed in the output queue of all terminals except those in the batch mode. The status of the terminals or the logical connections is not changed in any way. However, while the host is down, no new input messages are accepted from the terminals as input regulation is forced. As required, service messages continue to be generated by the NPU and are queued for input to the host. In this way, when the host again becomes operational, these messages will be transferred to the host. The NPU stops generating service messages if too few buffers are available for their storage.

When the host again begins responding, a message (HOST UP) is sent to those terminals previously notified of the HOST DOWN condition if they are still connected.

A terminal interface package (TIP) interfaces the terminal control blocks (TCBs) and line control blocks (LCBs) of the communications network to the terminal. A TIP includes both hardware and software elements. In interfacing with the communications network, the principal concerns of the TIP are mode control and error control with most of the software elements devoted to "exception processing". The interface between the TIP and the terminal is a function of the design of the terminal, with several broad classes of design and several variations within each class. Each class of terminal is supported by a TIP designed for the class.

NETWORK INTERFACE

OUTPUT QUEUING

A common routine in the NPU queues downline forward blocks output data and commands to the TCB. This routine discards the block if the accept output (AO) flag in the TCB is zero. An output queued (OQ) flag indicates the presence of output information for the TIP to process (see table 3-8).

UPLINE BREAK

The common "send break" subroutine (figure 3-4) indicates a discontinuity in the output stream. This routine purges the output queue described above, sets AO to zero to prevent further queuing of output information, and sends an upline BREAK message with a code indicating the reason for the break.

TABLE 3-8. TIP FLAG INTERPRETATIONS

Flag Code	Initial Value	Name	Set	Reset	Tested
OQ	0	Output Queued	Output Data or Command Received From Host	1. Upline Break 2. Purge Last Block in Queue	1. Quiescent Loop 2. Idle Loop
AO	1	Accept Output	Reset Received	Upline Break	Output Queueing
A1	TTY: 1 MD4: 0	Accept Input	Poll Command	1. Downline Break 2. Batch or Interactive command 3. Batch Interrupt 4. End of Cards 5. Upline Break	1. Quiescent Loop While in Batch Mode 2. Idle Loop 3. After Print Output 4. Any Input Received from Terminal
BT	0	Batch Mode	Batch Command	Interactive Command	1. Quiescent Loop 2. Initiating Poll 3. Input Received When Polling for Cards
TCK	0	Toggle Currently Known	Poll for Toggle Determines Value	Poll for Toggle Can't Determine Value	1. Poll for E-Code/ Data 2. Before Output
TSE	0 or 1	Toggle State Expected	1. Toggled When Any Write Performed 2. Set to Received Toggle When Response Received to Poll for Toggle		When Polling for Toggle, to Determine if Write Needs to be Retransmitted
BFQ	0	Blank Fill Queued	At Creation of Input Blank Fill	Terminal Verifies That Write E1 Accepted	1. Output Data 2. Poll Command
E3Q	0	E3 Queued	At Creation of Write E3 (Read Cards)	Terminal Verifies that Write E3 Accepted	1. Output Data 2. Poll Command 3. Interactive Command
WFC	0	Waiting for Cards	Starting Polling for Cards	Data Received in Response to Card Read Poll	1. Poll Command 2. Interactive Command 3. Before Output
BNB	0	Block Not 'Back'ed	1. Write Performed But Toggle can't be Determined 2. Write to Printer Completed, but E-Code Response Not Obtained	Write Transaction Properly Completed	1. Output Data 2. Poll for Toggle Obtains Response to Update State of Another TCB
TWK	0 or 1	Toggle was Known	Write to Printer Completed but E-Code Response not Obtained	Write Performed, but Toggle can't be Determined	1. Output Data but BNB is set 2. Poll for Toggle Obtains Response to Update State of Another TCB
TSD	0 or 1	Toggle Subsequently Determined	Poll for Toggle Obtains Response to Update State of Another TCB	When BNB is Set	1. Restart Output After Failure 2. Poll for Toggle Obtains Response to Update State of Another TCB
TWC	0 or 1	Toggle was Correct	Received Toggle Equals TSE when Response Obtained to Poll for Toggle Which Updates State of Another TCB	Received Toggle Does not Equal TSE When Response Obtained to Poll for Toggle Which Updates State of Another TCB	Restart Output After Failure

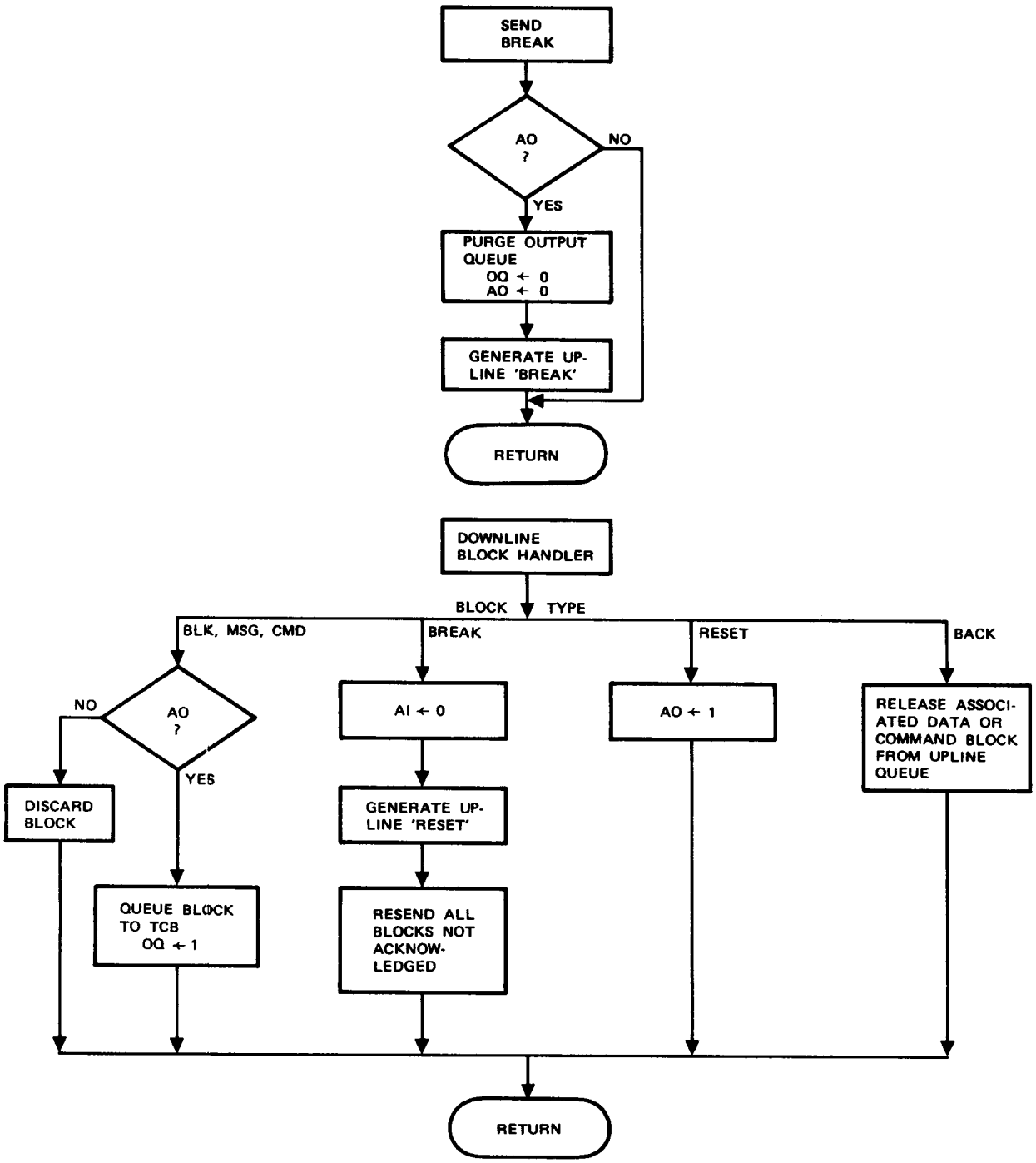


Figure 3-4. Common TIP Subroutines Flow Diagram

DOWNLINE BREAK

The host commands the TIP to stop input by sending a downline BREAK message. This block is acted upon when received, without being output queued. This message causes the accept input (AI) flag to be set to zero. Blocks not acknowledged by the host will be resent by the TIP following an upline RESET. The method by which input is again accepted (AI set to one) is a function of the particular TIP design.

TELETYPE (TTY) TIP

OPERATING MODES

The TTY TIP operates in either the interactive or tape mode. In the interactive mode, the TIP interfaces the network to a teletype device for either input or output. In the tape mode, the TIP interfaces the network to a tape reader.

In the interactive mode, the TIP operates in half-duplex fashion with three states (idle, input, and output). The TIP is driven from the idle state to the input state by arrival of a character from the terminal and is driven from the idle state to the output state by arrival of an output block from the host.

Once in the input state, the TIP remains in that state until an end-of-input message is sensed. If no output is in queue, the TIP returns to idle. If there is output in queue, the TIP goes directly to the output state. The TIP remains in the output state until the output queue is exhausted or until the terminal operator presses the BREAK key or inputs a character. While in the input state, the break (input framing error) is ignored.

When the TTY TIP is commanded to enter the tape mode, it sends X-ON to start the tape reader. While the TIP is in the tape mode, the host should not send output.

DATA FORMATS

Characters transmitted between the host and terminal during both input and output are passed in full 8-bit form, without code translation or parity generation and checking. Messages or characters sent by the TTY TIP to the terminal, however, always have even parity.

An input message is sent in one or more blocks. The maximum size of an input BLK or MSG block generated by the TIP is controlled by a program build parameter and is in the range 1-2043 bytes. When the message length exceeds the maximum block size parameter, all but the last block will be type 2 (BLK) blocks and the last block will be a type 3 (MSG) block. Where the message length is less than the maximum block size, the single block sent is of the type 3 (MSG) type. If input is terminated by other than normal input delimiting (as defined in paragraph entitled Mode Set), the message is never completed and the last block of the incomplete message is of type BLK.

MODE SET

The host sends the mode set command to the TIP. Its format is as follows:

BYTES	0-2	3	4	5
	ADDRESS	4 BSN	1	MODE

where:

Mode	Name	Description
0	Inter-active	On input, echo carriage return to line feed and line feed to carriage return. Carriage return defines end-of-input message
1	Tape	An X-ON is sent to the terminal to start the tape reader when the command is executed. End-of-input message is defined by the input line remaining in the marking state for 200 milliseconds or longer.

The mode set command is not executed and the mode set response is not sent unless the mode can be changed. If the terminal is already in the mode commanded, no response is sent. When the TIP changes mode, the command is returned to the host as a mode set response. Since the command is part of the output queue, it is not processed while the TIP is in the input state.

Comparison of input characters to carriage return and line feed considers bits 0-6, but not bit 7.

INPUT REGULATION

Before soliciting or accepting input messages, the TTY TIP checks buffer thresholds to ensure a sufficient supply of the size required. A higher threshold value is used for new messages than for messages already in progress. This gives higher priority to messages in progress when contending for available buffers.

If a message once started cannot acquire a buffer to maintain continuity, the partial block is sent to the host as a BLK type, followed by an upline command indicating that the message is not complete (i.e., no MSG block follows).

INPUT STOPPED SEQUENCE

The following sequence occurs when either a downline break (host failure) or input regulation (insufficient buffers) occurs.

1. The partial input block is placed into the upline data queue as a BLK type block.
2. An input stopped command with the following format is created:

BYTES 0-2 3 4

ADDRESS	4	BSN	0

3. A message consisting of an output break (250-millisecond spacing condition) followed by the text INPUT STOPPED is immediately sent to the terminal.
4. The above message places the TIP in the output mode and any input characters received are discarded.
5. The TIP remains in the output state if output is in the queue.

UPLINE BREAK

If the operator presses the BREAK key or inputs a character during output, the TIP calls the SEND BREAK subroutine and causes output to stop. The break condition is not timed out. If the break condition continues too long, the modem disconnects the line if that option is installed in the system.

MODE 4 TIP

The Mode 4 TIP interfaces the communications network to the card

reader, printer, keyboard, and CRT display of a CDC 200 user terminal (UT). Mode 4 terminals are connected to the NPU via synchronous lines operating at speeds to 9600 bits per second. The TIP is insensitive to line speed, with the CLA performing bit sampling and transmission under control of modem-supplied clock signals.

Operation of point-to-point or multi-drop Mode 4 lines is half-duplex (i.e., the TIP either transmits or receives, but does not simultaneously do both on the same line). Dedicated (non-switched) lines may support multiple clusters and multiple terminals per cluster, but all clusters on a line must be of the same terminal type. A switched line may connect to a single cluster only, but the cluster may support multiple terminals. Where multiple terminals are on a line, terminals are serviced in a periodically repeating sequence without priority.

OPERATING MODES

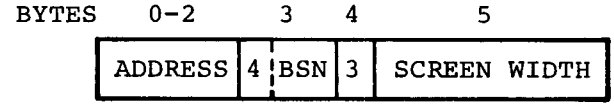
The card reader, printer, keyboard, and CRT display of a 200 UT all have the same terminal address. The 200 UT operates with the card reader and/or printer in the batch mode and with the keyboard or CRT display in an interactive mode. Mode selection is performed by the terminal operator via operator commands interpreted by the application process or by downline commands from the host. Device selection is performed by E-codes appended to the output by the TIP as a function of the current mode. E-codes coming from the terminal are stripped from the input before the data is forwarded to the host.

Interactive Mode

In the interactive mode, output assumes priority over input. Solicitation of input is accomplished by polling, initiated by the host sending a downline start polling command or after the host has delivered output to the terminal. Thereafter, polling continues until receipt of a mode set command, a downline break, or a terminal error that generates an upline break. Data is sent to the host as MSG blocks.

If there is data in the output queue, the TIP stops polling the terminal and delivers the block. If a BLK block, the TIP unconditionally awaits more output and does not resume polling. If a MSG block and there is no more data in the queue, the TIP resumes polling if input is being accepted (A1 flag on) or returns to a quiescent loop to await more output. The terminal verifies receipt of output data by sending BACK to the host.

The interactive mode is typically used with CRT terminals. When the TIP receives interactive input, it generates blank fill response to force the cursor to the first position of the next lower line. To accomplish this, the TCB must be supplied with screen width information. Normally, when the TCB is built, screen width is initialized to 80 characters. However, this parameter may be changed by a downline command with format as follows:



Batch Mode

In the batch mode, output and input have equal priority. This mode is typically used with card readers or line printers, with card reading and printing interleaved on a one-block-in then one-block-out basis as long as output data is queued.

Card reading is only initiated by the host sending a start polling

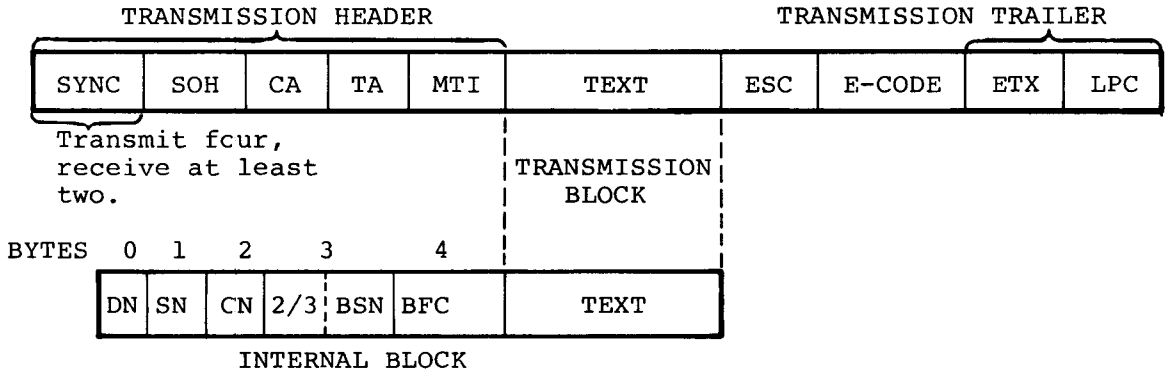
downline command. The TIP generates the write E3 output to the terminal to enable card buffer transfer. The TIP then polls for card data. Upon receiving data, the process continues until receipt of a mode set command, downline break, terminal error, the last card is read, or until the terminal interrupts card reading. If the input has an E3 code, input data is sent as a BLK block. If E2 code or a CYBER EOF (3E-56, hexadecimal), input data is sent as a MSG block. If an E1 is received when polling for card input or the printer E code, the input data is discarded, the send break subroutine is called, and a batch interrupt is indicated.

Output data may be either BLK or MSG blocks, without effect upon input solicitation. The BACK for an output block is sent to the host when an E3 or E1 is obtained from the printer. If an E2 is obtained from the printer, the TIP calls the send break subroutine and a printer failure is indicated. If an E1 is received when polling for the printer E code, the output block BACK is sent before the break.

TRANSMISSION BLOCK AND TERMINAL BLOCK FORMATS

Data blocks are always coded in ASCII. The TIP does character-for-character code conversion for BCD terminals (terminal type = 5). Those BCD terminals with switch selection of internal or external BCD code must have the switch in the external position. Certain control sequences imbedded within the data stream consist of an escape code followed by a second byte. The escape code is converted, but the second byte is not.

The format of a Mode 4 data transmission block is as follows:



The TIP inserts 14 SYN characters between the MTI and TEXT on all output where MTI is coded for clear write or reset write. Each byte, including the longitudinal parity check (LPC) character, has odd parity in bit 7. The LPC is coded to create odd parity on bits 0-6 of all characters except SYN characters. The TIP generates output parity and passes the received parity bit (entire received 8-bit character) to the host. Input converted from BCD to ASCII coding will have correct ASCII parity.

Poll, reject, acknowledge (ACK), and error transmission blocks are non-data blocks and have the following format:

SYNC	SOH	CA	TA	MTI	ETX	LPC

TERMINAL ADDRESSING

Terms CA and TA in the transmission header illustrated above are the cluster address and terminal address, respectively. The range of permissible values (in hexadecimal notation exclusive of parity) for the cluster and terminal addresses is a function of terminal type, as indicated below:

	200 UT	711	714
CA (Cluster Address)	70-7F	20-7F	20-7F
TA (Terminal Address)	60		61-6F
Cluster Controller		60	
CRT/Keyboard		61	

Bit 4 of the terminal address is the toggle bit. As shown in the foregoing table, the bit value is zero. However, it may be one. When the NPU transmits to the cluster, it changes this bit with each succeeding output. The input response carries the same value in the same bit position if the output was correctly received by the cluster controller or the opposite value if the output was not correctly received.

E-CODES

Device selection is performed by E-codes appended to the output by the TIP. Similarly, E-codes incoming from the terminal indicate the responding device and also report status. Received E-codes are stripped from the input data by the TIP. The codes below are in hexadecimal notation, exclusive of parity.

E-Code	WRITE (Output)	READ (Input)
E1	to CRT (text)	from CRT (text)
E2	to printer (text)	from printer (no text), indicates possible error in printing last block
		from card reader (text), indicates that card reading has stopped
E3	to card reader (no text), enables transfer of card buffer to CRT buffer	from printer (no text), indicates that last block correctly printed
		from card reader (text), normal card data
E4	to CRT (text), position the start index	Not possible

ERROR CORRECTION AND LOAD REGULATION

The Mode 4 TIP performs short-term recovery for both input and output. The TIP retains three error counters as follows:

Error Counter	Type of Error
1	No response---after transmitting to the terminal a response timeout occurs. SOH is never received.
2	Bad response---CA or TA does not correspond to terminal addressed by transmit block, invalid MTI, invalid or missing E-code, ETX missing (over-length block or premature drop of data carrier detect, character parity or LPC error, or text in a block that should not have text.
3	ERROR response---indicates transmit error

Whenever any error occurs, the TIP increments the appropriate counter and retries the output-input sequence. If any counter reaches a pre-established (at program build) threshold in an attempt to complete a single transaction with the terminal, the TIP calls the SEND BREAK routine and specifies the reason for break (RB) as one of the error counter numbers defined above.

If the TIP is unable to acquire sufficient buffers for an input block or if the host is down, it discards the partial block and again polls the terminal at a later time when the condition is cleared. No error counter is incremented by this operation. However, a counter in the NPU statistics block is incremented to indicate the number of times such regulation has taken place.

LONG-TERM ERROR RECOVERY

Rather than continue error retries for an extended period, the TIP sends an upline BREAK and discontinues service to an error-producing terminal, but retains status information to allow orderly recovery. To accomplish such recovery, the host must send a downline RESET followed by output blocks and commands not previously block acknowledged. These output blocks and commands are sent in precisely the same sequence in which they were originally sent. A permissible alternative is to TERMINATE the logical connection. Any other action (such as initiating a mode change) may result in loss or duplication of data at any terminal serviced by the same controller.

MODE 4 TIP FLOW DIAGRAM

Figure 3-5, sheets 1 through 3, is a flow diagram of the Mode 4 TIP showing the interface of the TIP with both the host and the terminal. Shown are:

- Mode of the terminal (batch/interactive)
- Input state (polling or not polling, ACPINP flag)

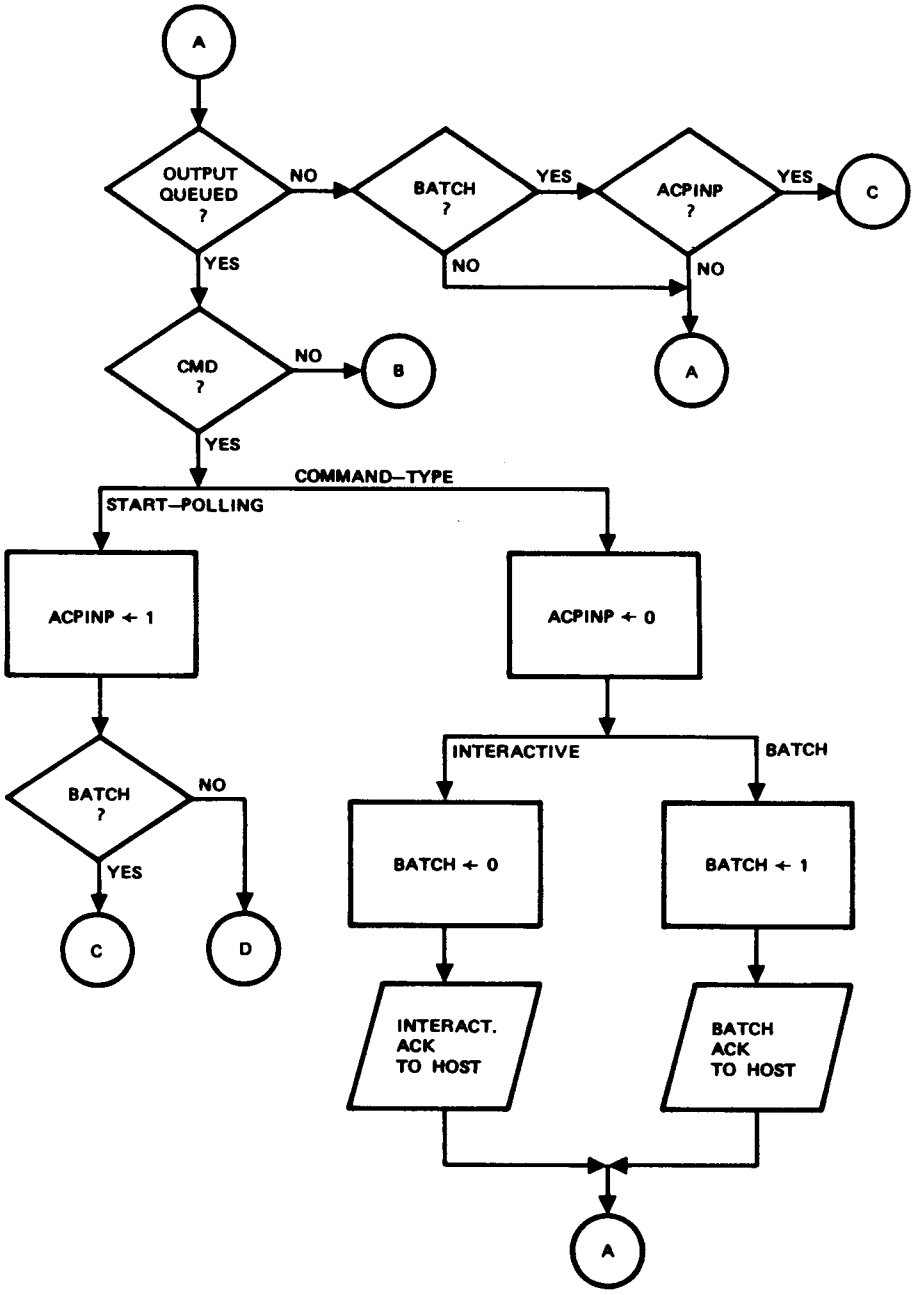


Figure 3-5. Mode 4 TIP Flow Diagram (Sheet 1 of 3)

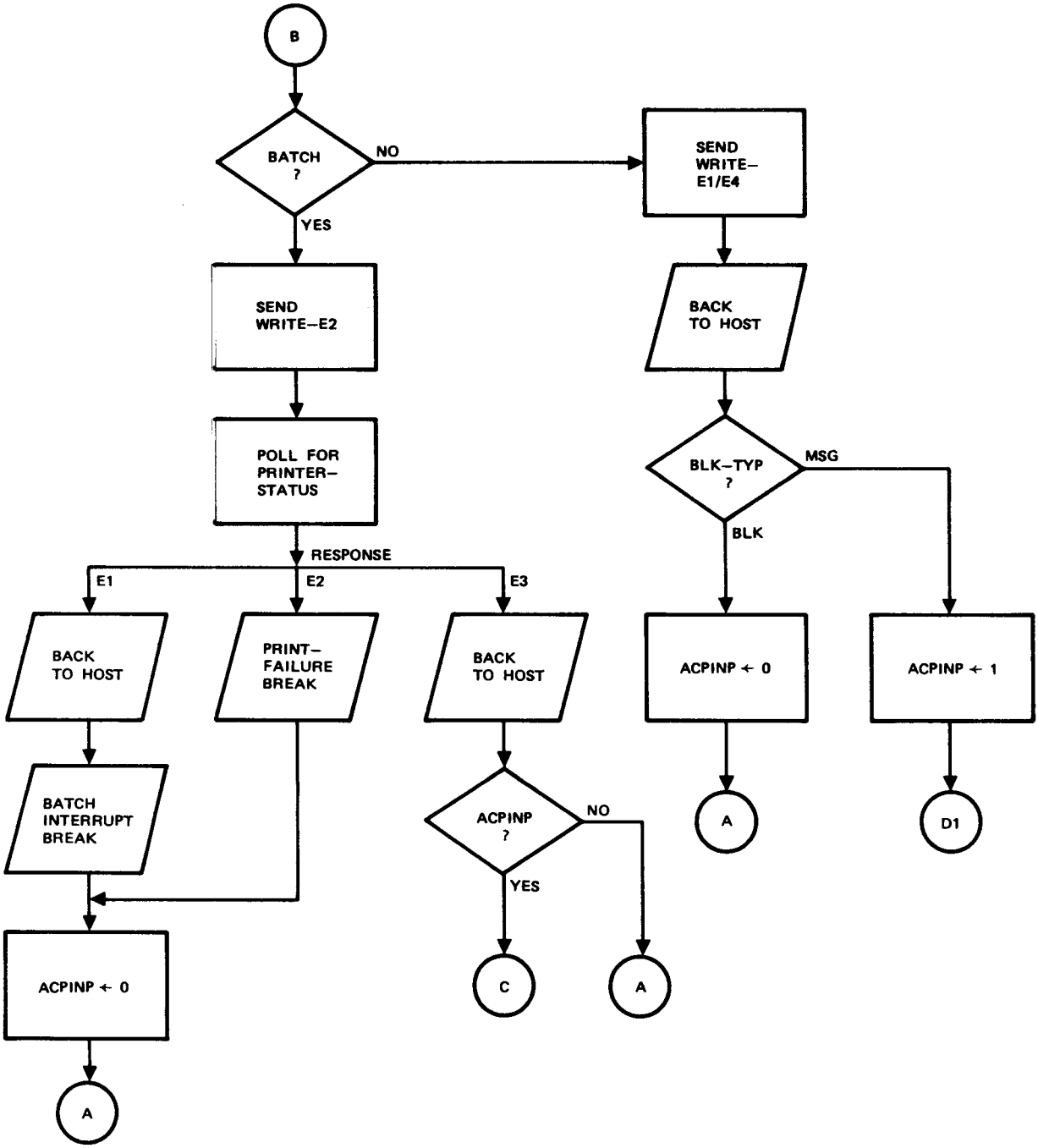


Figure 3-5. Mode 4 TIP Flow Diagram (Sheet 2 of 3)

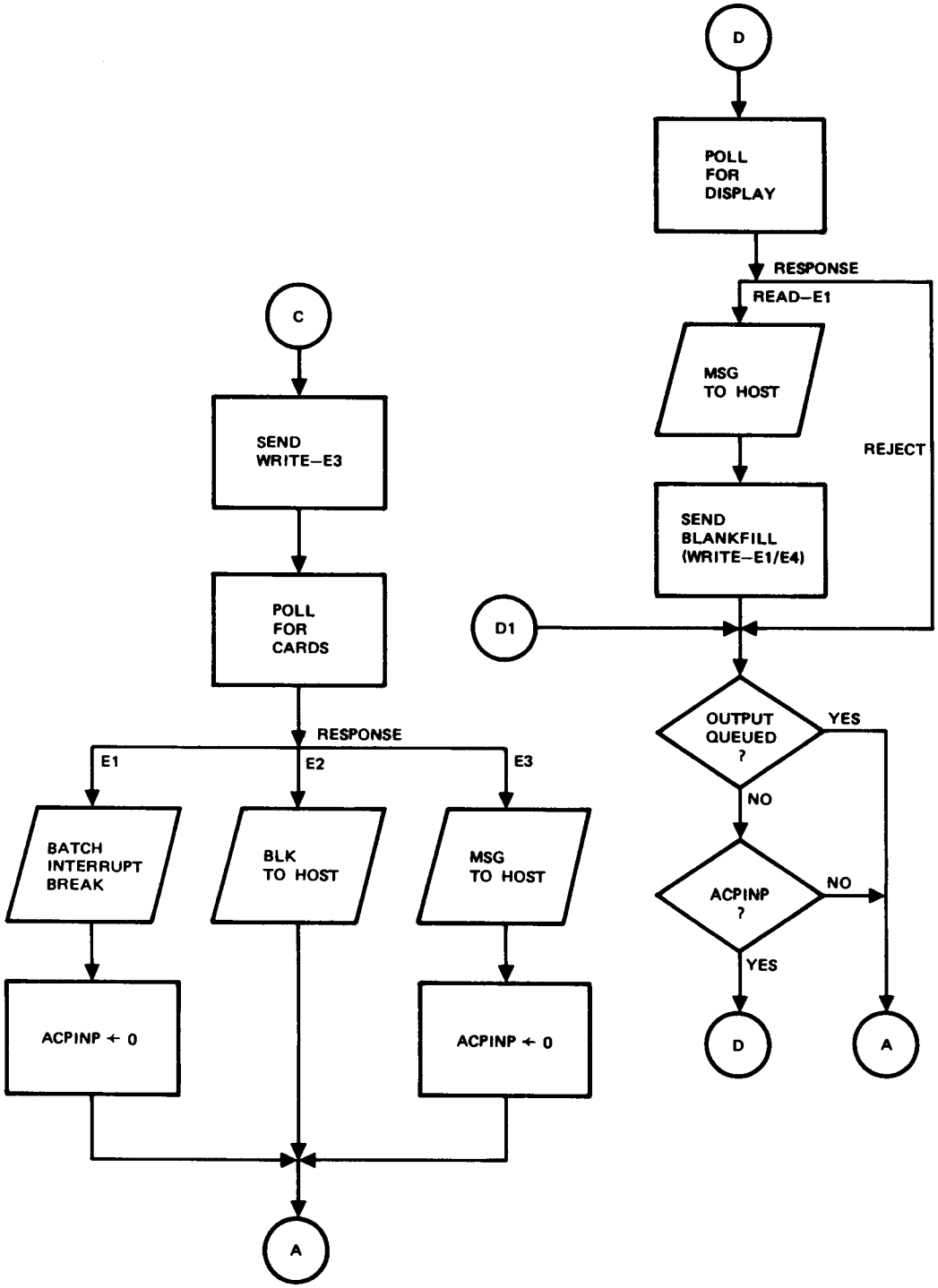


Figure 3-5. Mode 4 TIP Flow Diagram (Sheet 3 of 3)

- Reason for poll (cards, display, print status)
- Output queue checking-processing
- Batch interrupts
- Printer failure sequence
- Command processing
- Block acknowledgement (BACKs)

Not shown are:

- Error recovery
- Multiplexing multiple terminals on the same line
- Protocol details (such as toggle bit maintenance)

The following are summaries of the Mode 4 commands and upline break code interpretations:

Command Code	Direction	Description
1	Downline	Set Mode
1	Upline	Mode Set Response
2	Downline	Start Polling
3	Downline	SET Screen Width

Reason for Break (RB)	Description
1	No Response
2	Bad Response
3	Error Response
4	Printer Failure
5	Batch Interrupt

SET MODE COMMAND AND RESPONSE

The following is the format for the Mode 4 TIP downline set mode command and upline mode set response command:

BYTES	0-2	3	4	5
	ADDRESS	4 BSN	1	IB

where: IB = 0 = Interactive Mode
IB = 1 = Batch Mode

The host sends the mode set command to the TIP and, when the TIP changes mode, it returns the command to the host as a mode set response. A TCB is initialized to interactive mode when built. Downline data following an interactive mode set command is sent to a CRT by appending an E1 or E4 as a function of the terminal type. Terminal type 5 gets E1, terminal type 6 or 7 gets E4. Downline data following batch mode set command is sent to the printer by appending E2.

START POLLING COMMAND

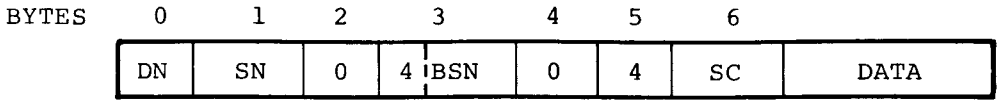
The format for the start polling command is as follows:

BYTES	0-2	3	4
	ADDRESS	4 BSN	2

No response is made to this command. In the interactive mode, receipt starts polling of the keyboard. In batch mode, receipt causes the TIP to send "clear write E3" to transfer the card reader buffer to the CRT buffer. The TIP then polls the terminal until the block of card records is received, then repeats the process.

ERROR REPORTING AND STATISTICS

All error reports and statistics are upline service messages with the following format:



where: SC = Service code as follows:

SC	Message
0	Report to CE error file
1	NPU statistics
2	Line statistics
3	Mode 4 terminal statistics

All service messages with a source process of 4 are limited to 28 bytes of data in the data field. All such messages received by the host are time-stamped and recorded on a file.

REPORTS TO CE ERROR FILE

A service message is created for every detected hardware-related abnormality. This includes all NPU-related hardware such as the coupler, MLIA, loop multiplexers, and CLAs and also all connected hardware such as modems, lines, and terminals. The creation of a CE error file report is separate from and in addition to statistics accumulated in the NPU and periodically supplied to the host as described in later paragraphs.

To prevent swamping the NPU or host with error messages when an oscillatory condition arises, an error counter is incremented with each error message generated and, when the counter reaches a pre-established (at program build) threshold, the error event is discarded rather than recorded. The counter is periodically reset to zero by timeout of another counter whose threshold is also a pre-established parameter.

The first byte of the data portion of a CE error report (byte 7) contains an error-type code. As many as 27 bytes of identification data may follow this code byte.

STATISTICS

The NPU contains a statistics reporting timer (interval selected at program build time) that causes statistics messages to be generated at pre-established intervals. Statistics blocks are maintained for the NPU, for each line control block, and for each Mode 4 terminal control block. One such block is dumped and cleared at each timer timeout. Therefore, the accumulation period for a statistic is equal to the number of statistics blocks multiplied by the timer interval in seconds.

In addition to the normal statistics message generated by timer timeout, disabling a line causes the associated statistics block to be dumped and cleared. A Mode 4 terminal statistics block is also dumped and cleared when the logical connection to the terminal is broken. If a counter contained within a statistics block overflows, that counter is set to all ones and the statistics block is dumped and cleared.

The data field of a statistics message contains up to 12 counters, each consisting of a single 16-bit word. These counters begin at byte 10. For an NPU statistics message, bytes 7 through 9 are zero. For a line statistics message, byte 7 is the port number and bytes 8 and 9 are zero. For a Mode 4 terminal statistics message, byte 7 is the port number, byte 8 is the cluster address (CA), and byte 9 is the terminal address (TA).

Each of the counters within a particular statistics message is incremented upon occurrence of a particular event. The events monitored by each counter in each message are as follows:

NPU Statistics Message	
Counter Word	Event Monitored
0	Service message generated
1	Service message processed
2	Block discarded due to bad address
3	Block discarded due to bad format
4	Mode 4 input regulation started
5	Teletype input regulation started
6	Host failure
7	Service message received out of sequence

Line Statistics Message	
Counter Word	Event Monitored
0	Block transmitted
1	Block received
2	Characters transmitted*
3	Characters received*

*For good blocks, length of block is added to the total at end-of-block transmission or reception.

Mode 4 Terminal Statistics Message	
Counter Word	Event Monitored
0	Block transmitted*
1	Block received**
2	Block retransmitted
3	Block received, but not accepted due to error
4	Upline BREAK due to error (not batch interrupt)

*Does not include blocks retransmitted.
**Does not include blocks not accepted due to errors.

ERROR/STATISTICS MESSAGE REROUTE

The NPU local teletype console can be designated as the message delivery point for all upline error and statistics service messages. The command that performs this function is entered at the NPU console and specifies a parameter as follows:

Parameter	Interpretation
0	Discard all upline error and statistics service messages.
1	Print all upline error and statistics service messages (in hexadecimal) at the NPU console.
2	Send all upline error and statistics messages to the host.
3	Send all upline error and statistics messages to the host and also print at NPU console.

ONLINE DIAGNOSTICS

The NPU online software includes basic CLA and modem loopback tests. These tests execute basic data and control functions to determine if all received data and status is normal. The online diagnostics can be invoked from the local NPU console to test one or more communications lines without impacting services on other lines in the system. Local modem loopback tests can also be initiated from the NPU console for modems possessing the loopback feature.

Use of a suspected bad line is halted via a command entered at the NPU console which removes the line from service. The NPU console is then used to initiate the resident online diagnostics to loop data at various points in the communications system. A loopback jumper plug for both synchronous and asynchronous CLAs is furnished as a special tool with every NPU system. Using the online diagnostics and the loopback plug, problems can be isolated to a CLA. Methods and procedures for further isolation of problems in the external communications system, including loopback modems and communications lines, are described in the software diagnostic handbook (see preface).

The host is notified that a line has been taken out of service by the "line inoperative" service message. Line error code 0 is used and causes the host software to attempt to enable the line (either dedicated or switched) after a predetermined time delay. When the host sends the "enable line" message to the NPU, a BACK message acknowledges receipt of the "line enable" message, but no "line inoperative" message is returned to the host. After corrective action has been taken and the line is returned to service from the NPU console, the NPU software responds to the "enable line" message from the host by enabling the line and sending the upline "line operational" service message.

ONLINE DIAGNOSTIC COMMANDS/RESPONSES

The following commands and responses establish the interface between the operator and the online diagnostic program. Commands are entered and responses are received through the local NPU console in the standard service message format. The terms used in the commands are interpreted as follows:

DN = Destination node address = 2 hexadecimal characters specifying the ID for the NPU.

SN = Source node address = 2 hexadecimal characters specifying the ID for the host.

PORT = Port number = 2 hexadecimal characters specifying the port associated with the line to be affected by the command.

SUBPORT = Subport number = 2 hexadecimal characters specifying the subport associated with the line to be affected by the command.

CLA TYPE = 00 if 2560-1 CLA
= 01 if 2561-1 CLA
= 02 if 2560-2 CLA

MDCL = Modem Class (table 3-9)

PLACE LINE OUT OF SERVICE

This command, entered at the NPU console, causes the TIP to terminate all activity on the specified line. The host is notified that the line is not in service by the "line inoperative" service message with line error code = 0, which specifies that the line is to be re-enabled after a predetermined time delay. The format of this command is as follows:

BYTES	0	1	2	3	4	5	6	7	8
	DN	SN	0	4	BSN	3	0	0	PORT	SUBPORT

PLACE LINE IN SERVICE

This command, entered at the NPU console, allows the line to be returned to operational service by an "enable line" service message that may be either currently outstanding or subsequently issued by the host. The format for this command is:

BYTES	0	1	2	3	4	5	6	7	8
	DN	SN	0	4	BSN	3	1	0	PORT	SUBPORT

START CLA INTERNAL LOOPBACK TEST

This command initiates the CLA internal loopback test which consists of a CLA command test and a data verification test. The CLA command test verifies operation of the CLA as it relates to command functions. System servicing of other lines is not affected by this command. The command has the following format:

DN	SN	00	40	03	02	00	PORT	SUBPORT	CLA TYPE	00

Any errors detected during the CLA test result in printout of a response service message with an appropriate error code at the local NPU console and termination of the test. To restart the test, reenter the "start CLA internal loopback test" command at the local console.

START MODEM LOOPBACK TEST

If modem loopback is available, this command isolates problems occurring further out in the communication system. The test consists of a data verification test with limited analysis of modem control signals. System servicing

of other lines is not affected by this command. The command has the following format:

DN	SN	00	40	03	02	01	PORT	SUBPORT	CLA TYPE	MDCL

START EXTERNAL LOOPBACK TEST

This command provides for loopback of data external to the CLA. The test consists of a command and data verification test with the primary purpose of verifying operation of the line drivers and receivers. The loopback jumper plug (2560-1 External Test Connector for synchronous CLA or 2561-1 External Test Connector for asynchronous CLA) must be connected to the CLA to be tested before this command is entered at the console. System servicing of other lines is not affected by this command. The command has the following format:

DN	SN	00	40	03	02	02	PORT	SUBPORT	CLA TYPE	00

TERMINATE TEST

This command, entered while a test is in progress, causes the test to terminate at the end of the normal test cycle currently being executed. System servicing of other lines is not affected by this command. The command format is as follows.

DN	SN	00	40	03	03	00	PORT	SUBPORT

The diagnostic test responses are output to the local console in the following standard format:

DN	SN	00	40	00	04	00	RCEC	PORT	SUBPORT

where: RCEC = Response Code or Error Code (see below)

Response codes are interpreted as shown in tables 3-10 through 3-12.

TABLE 3-9. MODEM CLASS

Test Type	CLA Type	Max Modem Speed	MOD Class	MODEMS
INTERNAL LOOPBACK	ALL	N/A	0	N/A
EXTERNAL LOOPBACK	ALL	N/A	0	N/A
	2560-1 2560-2 2560-3			All Synchronous Modems with Loopback Capabilities i.e., 201B, 203
MODEM LOOPBACK	2561-1	100	2	103, 113, 202E, 202C, D, R, VADIC
		110	3	
		120	4	
		133.3	5	
		150	6	
		300	7	
		600	8	
		800	9	
		1,050	A	
		1,200	B	
		1,600	D	
		2,400	F	
		4,800	10	
		9,600	12	

TABLE 3-10. RESPONSE CODE INTERPRETATION

Response Code (hex)	Meaning	Remarks
A0	Line is out of service	Normal response to "place line out of service" command.
A1	Command rejected	System temporarily low on buffers.
A2	Line in service	Normal response to "place line in service" command.
A3	Diagnostics in process	Response to "place line in service" command if diagnostics still in process.
A4	Diagnostics started	Normal response to "diagnostic function" command.
A5	Invalid line number or bad command	Invalid line number issued in command or command code (byte 5) is not valid.
A6	Invalid CLA type	Invalid CLA type issued in command.
A7	Invalid test mode	Invalid diagnostic test mode (byte 6) issued with command.
A8	Line not out of service	Response to "start diagnostics" command if line specified was not out of service when command issued.
A9	Test already in process	Response to a "start diagnostic" command if any test is already in process.
AA	Invalid modem class	Invalid modem class issued in command.
DD	Test completed, no errors	Normal response to a "terminate test" command.
DE	Diagnostic not in progress	Response to "terminate test" command if not preceded by diagnostic command.
Error codes indicate detection of an abnormal condition during text execution. Error detection causes the		on-line diagnostic program to terminate the test. Error codes are interpreted in tables 3-11 and 3-12.

TABLE 3-11. ERROR CODE INTERPRETATION

Error Code (hex)	Meaning
AB	Unsolicited input detected
AC	Unsolicited output data demand detected
AD	Input loop error
AE	Output loop error
AF	Parity error
B0	Framing error
B1	Data transfer overrun
B2	Next character not available
B3	No CLA status after CLA status was requested
B4	Unsolicited CLA status
B5	CLA status not cleared after input supervision on (ISON) was sent
B6	No status after request to send (RTS) or input status request (ISR) was sent
B7	No clear to send (CTS) after RTS
B8	No status after data terminal ready (DTR)
B9	No data set ready (DSR) after DTR
BA	No signal quality detect (SQD) after DTR
BB	No ring (RI) after DTR
BC	No status after secondary request to send (SRTS)
BD	No secondary received line signal detector (SRLSD) after SRTS
BE	No CLA status after local mode (LM)
BF	No data carrier detect (DCD) after LM
C0	Unsolicited status after originate mode (OM)
C1	No status or improper operation of RI after terminal busy (TB)
C2	No status after new sync (NSYN)
C3	Improper operation of DCD, RI, quality monitor (QM) after NSYN
C4	No RI after RTS
C5	Input data timeout during data verification test
DF	Unsolicited status after LM

TABLE 3-12. DATA COMPARE ERROR RESPONSE CODE

Error Code	CLA Type	Parity	Baud	Stop Bit
C6	SYNC	Even	-	-
C7	SYNC	Odd	-	-
C8	SYNC	No	-	-
C9	ASYNC	Even	40	1
CA	ASYNC	Odd	85.4	2
CB	ASYNC	No	100	1
CC	ASYNC	Even	110	2
CD	ASYNC	Odd	120	1
CE	ASYNC	No	133.3	2
CF	ASYNC	Even	150	1
C0	ASYNC	Odd	300	2
D1	ASYNC	No	600	1
D2	ASYNC	Even	800	2
D3	ASYNC	Odd	1,050	1
D4	ASYNC	No	1,200	2
D5	ASYNC	Even	1,600	1
D6	ASYNC	Odd	1,600	2
D7	ASYNC	No	2,400	1
D8	ASYNC	Even	2,400	2
D9	ASYNC	Odd	4,800	1
DA	ASYNC	No	9,600	2
DB	ASYNC	Even	9,600	1

BASE SYSTEM SOFTWARE

INTRODUCTION

The base system software comprises those elements of the total CCP that are required for all 2550 system applications. It includes the standard operating system components for program control and allocation of system resources, commonly used subroutines, and the program logic to control the multiplex subsystem and peripheral equipment. It is designed for easy incorporation of user options and program modification for adaptation to particular applications.

The multiplex subsystem is included as a part of the base system software. However, because of its importance within the system, the multiplex subsystem is separately described in section 5 of this manual.

BUFFER MAINTENANCE

All main memory space not allocated to program and permanent data structures is assigned as dynamic work area. Such areas are assignable as buffers in word sizes of 8, 16, 64, or 128 words. A system option allows sequentially ordered selection of from one to four of these sizes, of which two sizes can be designated as data buffers (see Installation Handbook). The base system assumes 16-word and 32-word buffers are always available. The buffer maintenance procedures control allocation of these areas among the various size buffers and regulate obtaining and releasing of these buffers for required purposes.

Buffer maintenance functions provided include:

1. Obtain a single buffer of a specified size,
2. Release a single buffer of a specified size,
3. Obtain several buffers of a specified size,
4. Release several buffers of a specified size,
5. Release a chain of mixed buffers in two sizes, and
6. Test buffer availability against a specified threshold.

Functions 1, 2, and 6 are available at any interrupt level and 3, 4, and 5 are restricted to use at the OPS level only.

In conjunction with testing buffer availability against a specified threshold, buffer maintenance periodically adjusts buffer distribution by size by using buffer mating facilities to, where possible, replenish buffer pools that are below threshold.

As an option, buffer stamping is available (see Installation Handbook). With this option a separate word outside of the buffer is used to keep diagnostic status of each buffer. A separate stamp word location must be available for the greatest number of possible buffers in the system. Therefore, the buffer stamp area set aside in the memory must be sufficient to accommodate a separate stamp word for the maximum number of the smallest size buffer that can be fitted into the allotted buffer area. This option extends the diagnostic ability of buffer maintenance at the expense of greater use of processor time.

OBTAINING BUFFERS

The buffer maintenance can provide either a single buffer or several buffers of a specified size.

OBTAINING A SINGLE BUFFER

The calling sequence to obtain a single buffer of a specified size is:

```
PBGET1BF (parm)
```

where parm is a constant of the enumeration type B0BUFSIZES that specifies buffer size. PBGET1BF is a PASCAL function and returns the value of B0BUFPTR that points to the base of the buffer obtained. BBGET1BF also uses the buffer control block for the specified size buffer.

Interrupts are inhibited during execution. A system halt is evoked if the buffer pool is down to the last buffer, the next free buffer has a bad chain address, or (optional) buffer stamping indicates the buffer is already in use. The chain word of the buffer is cleared to zero.

OBTAINING ONE OR MORE BUFFERS

The calling sequence to obtain several buffers of the same size is:

```
PBBUFGET (parm1, parm2)
```

where parm1 specifies the number of buffers to be obtained and is of the type integer and parm2 specifies buffer size and is of the enumeration type B0BUFSIZES. PBBUFGET is a PASCAL function and returns a value of the type B0BUFPTR that points to the first (and possibly only) buffer obtained.

PBBUFGET is used only at the OPS interrupt level and no threshold checks are performed.

RELEASING BUFFERS

The following calling sequences are used, respectively, to release a single buffer of a specified size, release one or more buffers of a specified size, or release a chain of mixed buffers of two different sizes. When released, a buffer returns to the free pool of other buffers of that size. Checks are made to ensure that the address is a valid buffer address and to determine if the buffer has already been released to the free buffer pool. If the stamping option is selected, special checks are performed to verify that the buffer has not already been released and is of the correct size.

RELEASING A SINGLE BUFFER

The calling sequence to release a single buffer is:

```
PBREL1BF (parm1, parm2)
```

where parm1 is a pointer to any address within the range of the buffer to be released and parm2 is a constant of the enumeration type B0BUFSIZES specifying buffer size. Parm1 is a PASCAL VAR parameter that is altered by the procedure so that, upon completion, parm1 contains the chain value of the last buffer released.

RELEASING ONE OR MORE BUFFERS

The calling sequence to release several buffers of the same size is:

```
PBBUFREL (parm1, parm2, parm3)
```

where parm1 specifies the number of buffers to be released and is of type integer, parm2 is a constant of the enumeration type B0BUFSIZES that specifies buffer size, and parm3 is a pointer to the first (and possibly only) buffer being released and is of type B0BUFPTR. Parm3 is altered by the procedure to contain the chain value of the last buffer released.

If parm1 is set to zero, all buffers in the chain are released. If parm1 is greater than the number of buffers

in the chain, only those buffers in the chain are released and the excess value of parm1 is ignored.

This calling sequence can only be used at the OPS interrupt level.

RELEASING A MIXED CHAIN

The calling sequence to release a chain of mixed buffers of two different sizes is:

```
PBDBREL (parm1, parm2)
```

where parm1 specifies the number of buffers to be released and is of type integer, and parm2 is a pointer to the first buffer to be released and is of the type B0BUFPTR. Parm2 is altered to contain the chain value of the last buffer released and, if parm1 is either zero or of a greater value than the number of buffers in the chain, results are the same as for PBBUFREL.

This calling sequence can only be used at the OPS interrupt level.

TESTING BUFFER AVAILABILITY

The calling sequence to test buffer availability is:

```
PBBFAVAIL (parm1, parm2, parm3)
```

where parm1 specifies the number of buffers required and is of the type integer, parm 2 specifies the size of buffers required and is of type B0BUFSIZES, and parm3 specifies the total free space threshold and is of type B0BUFLEVELS. PBBFAVAIL is a PASCAL function and returns a "true" value if the test indicates sufficient buffers available.

This procedure permits testing for buffer available space. Availability of total free space is recorded each time the buffer adjustment procedure (PBADJUST) is called. Sufficient overall buffer space and buffers of the specified size must be available. To force input regulation, the system causes PBBFAVAIL to refuse buffers to all except the service module.

This calling sequence may be used at any interrupt level.

BUFFER ADJUSTING, MATING, AND STAMPING

Buffer maintenance includes facilities automatically called by the timing services to periodically replenish, where possible, the pools of particular buffer sizes that are below their established thresholds. This is done by employing the combined adjusting (PBADJUST) and mating (PBMATE) procedures to take buffers from pools that are above their thresholds and pair them to form a buffer of the next larger size or break them to form two of the next smaller size. These calling sequences are not available to the user.

A buffer stamping facility is also available as a system option to increase the diagnostic capability of buffer maintenance at the expense of greater execution time. When used, a word outside of the buffer area is assigned to each potential buffer of the smallest allocated size. This word records whether the buffer is free or in use, buffer size, and the address of the last procedure that either requested or released the buffer. The buffer stamping calling sequence is not available to the user, but operates automatically when the option is selected. Figures 4-1 through 4-4 illustrate buffer stamping formats for common operations.

LIST SERVICES

Lists provide a convenient method to handle communications between software modules that do not use direct calls. Figure 4-5 depicts worklist organization. The list services

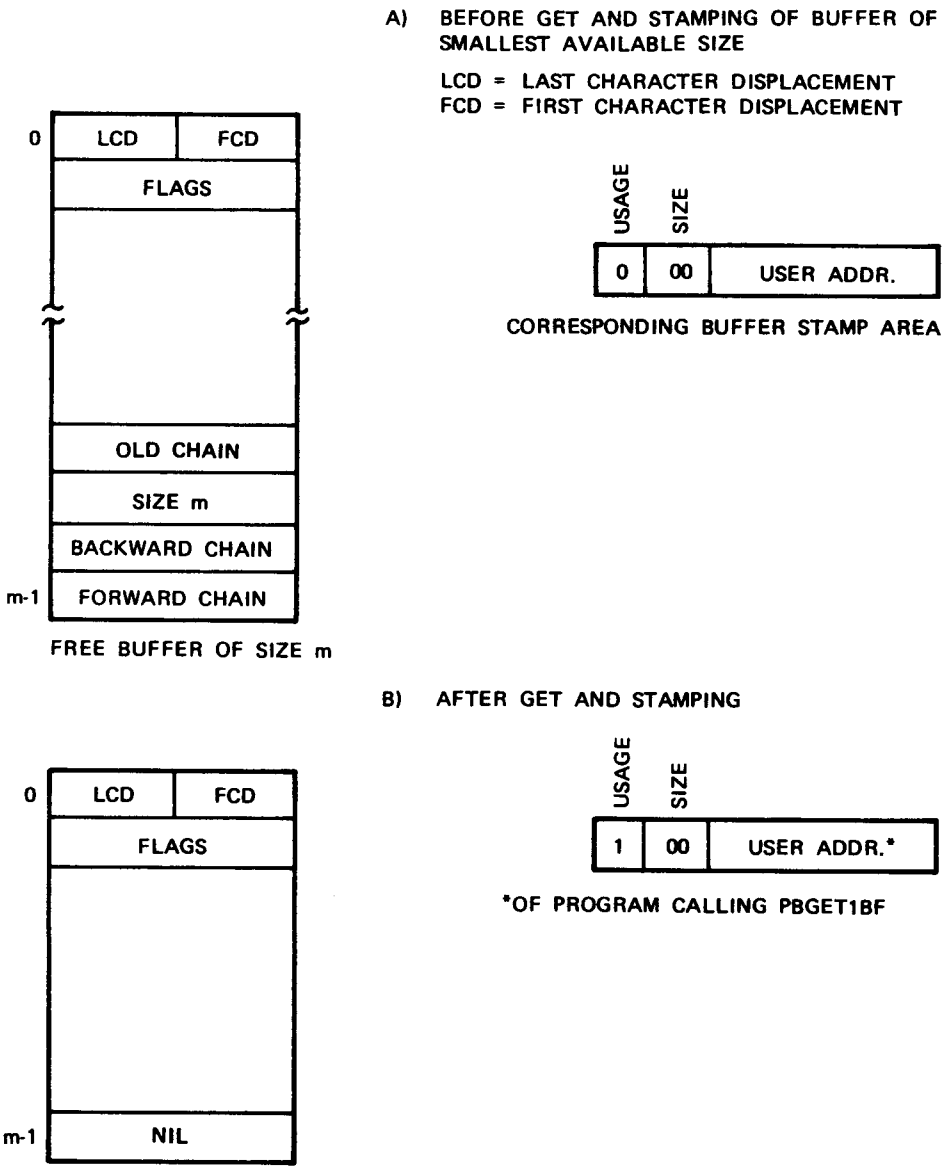


Figure 4-1. Buffer GET and Stamping

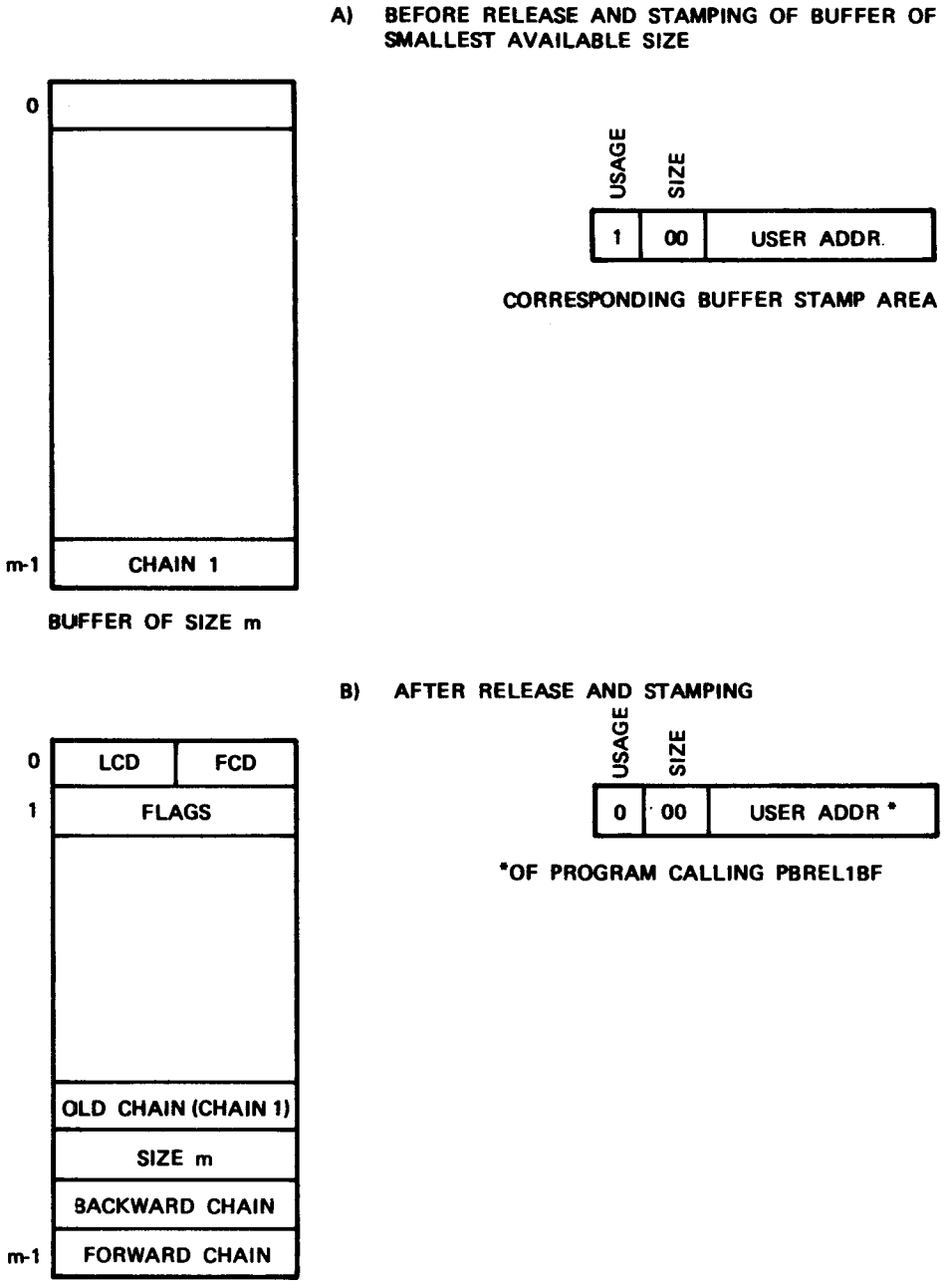


Figure 4-2. Buffer Release and Stamping

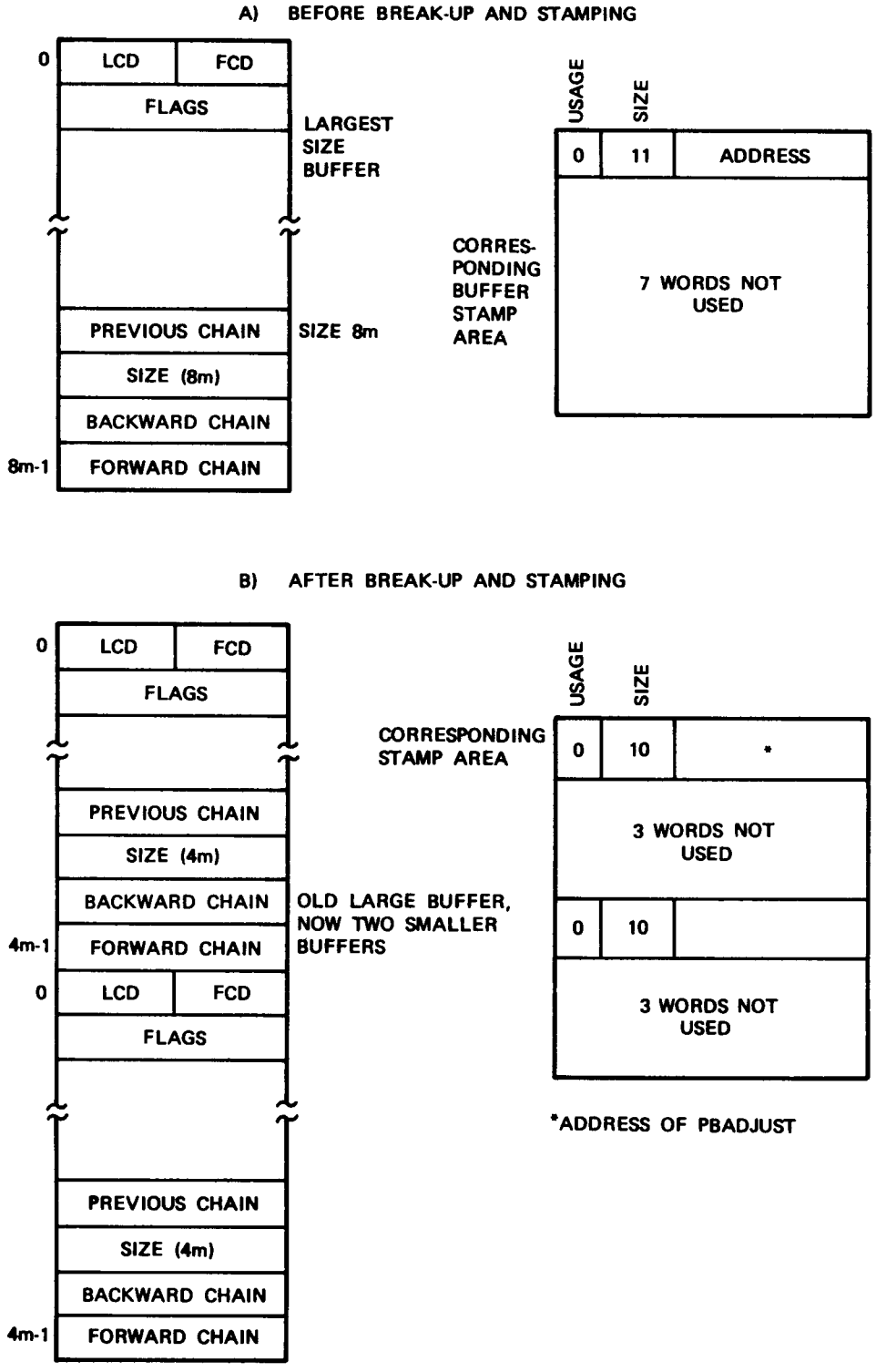


Figure 4-3. Buffer Break-up and Stamping

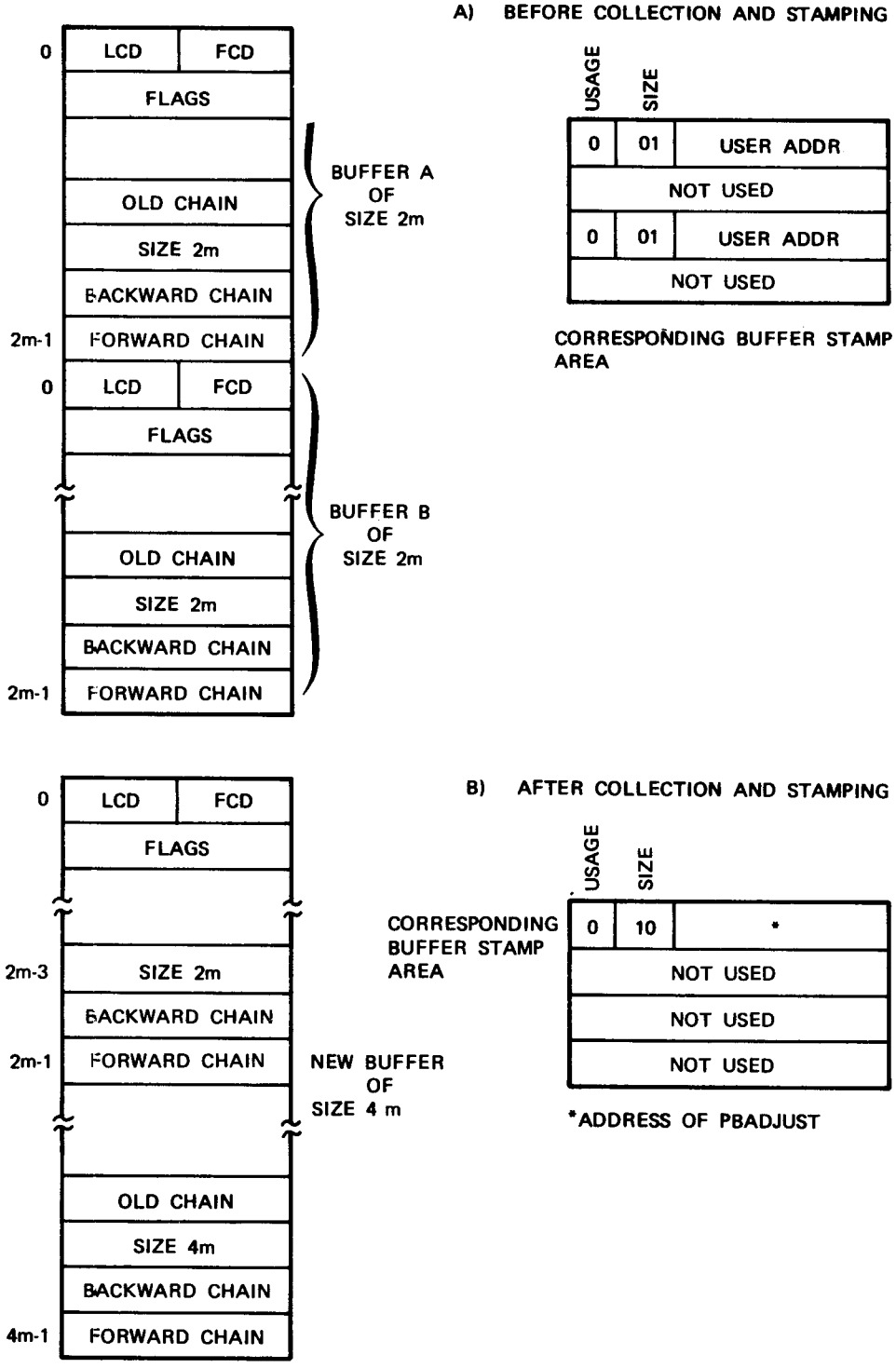
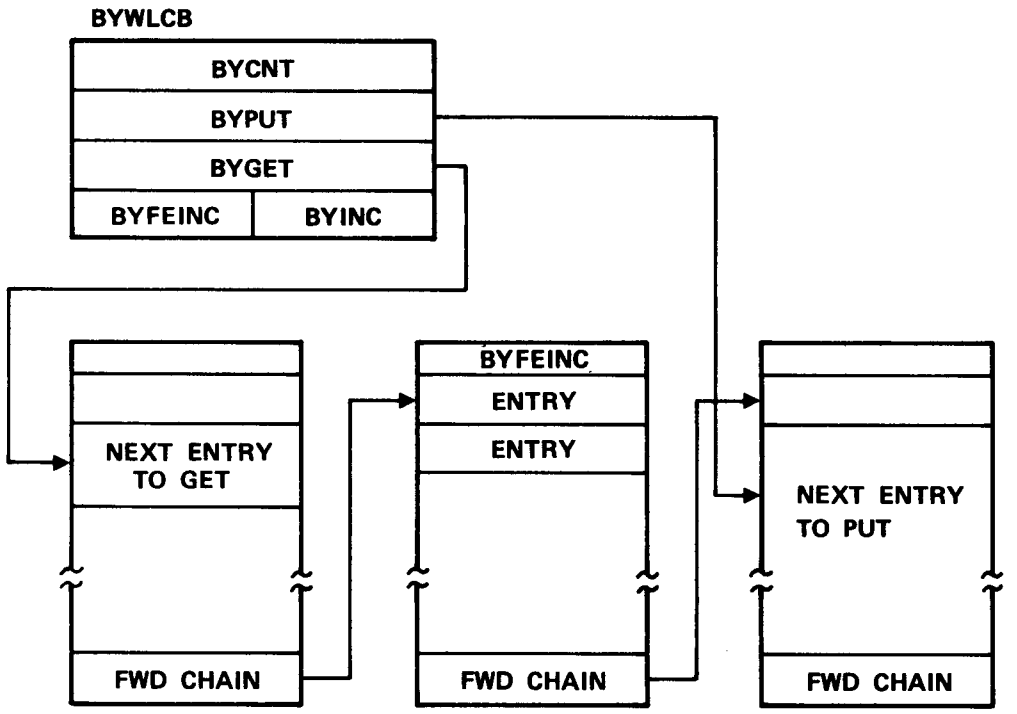


Figure 4-4. Buffer Collection and Stamping



BYCNT - ENTRY COUNT

BYINC - ENTRY SIZE

BYFEINC - DISPLACEMENT IN BUFFER TO FIRST ENTRY

Figure 4-5. Worklist Organization

function manipulates worklists with variable entry sizes. Functions provided by list services include:

1. Make (PUT) worklist entries from any priority level (including OPS level), by terminal type, or without disturbing the intermediate array.

2. Extract (GET) an entry from a list.

Characteristics of lists managed by list services are:

1. First in, first out.

2. Entries may be from one to six words in length, but all entries in a particular list must be the same length.

3. Lists are maintained in dynamically assigned space.

4. There is no maximum on the number of entries in a list or on the number of lists serviced.

Contention between priority interrupt levels is resolved by defining an intermediate worklist array (BWWLENTRY) with 6-word entries for each possible system interrupt level. Worklist entry parameters are assembled and extracted in the intermediate worklist area corresponding to their interrupt level.

A worklist entry is passed to PBLSPUT and data is obtained from PBLSGET through a global array named BWWLENTRY. Each element of the array has a variant record structure consisting of one case for each logical entry structure. When each new worklist-driven program is created, the format of the new worklist may be added as another case to the PASCAL-type definition B0WKLSTS. Thus, each worklist may have unique fields and names.

There are 17 elements to the array BWWLENTRY, one for each priority interrupt level. To access the proper interrupt level, the global variable LEVELNO is used. For example, to access a field of a particular worklist entry at the proper interrupt level, the following expression is used:

```
BWWLENTRY [LEVELNO] . FIELDNAME
```

Access the fields of the worklist entry to store information before calling PBLSPUT or to obtain information after calling PBLSGET. For programs always run at a specific interrupt level (e.g., OPS, CPL, RTC, etc.), constants may be used to increase efficiency.

If a program using PBLSPUT or PBLSGET calls a program also using PBLSPUT or PBLSGET, information in the worklist entry BWWLENTRY may be changed upon return. In such cases, one of the following techniques must be used to ensure proper data integrity:

1. Put all information in worklist entry and call PBLSPUT before calling the second program.

2. Call PBLSGET and access all pertinent information from worklist entry before calling the second program.

3. Save and restore the worklist entry from BWWLENTRY.

MAKE A WORKLIST ENTRY

GENERAL

PBLSPUT puts an entry into a worklist from any priority interrupt level. The calling sequence is as follows:

```
PBLSPUT (parm)
```

where parm is a symbolic constant of the enumeration type B0WKLSTS and is an index to the proper worklist control block.

OPS LEVEL

PBPUTLIST is for use at OPS level only and is the exact equivalent of

PBLSPUT except that more efficient code is generated. This procedure should be used by all OPS level users.

BY TERMINAL TYPE

PBPUTYP is intended for use only by those procedures where the worklist in which the entry is made is determined by the type of terminal being serviced. PBPUTYP takes as input an integer that is treated as the address of a worklist entry. The second word of the worklist must be the number of the line being serviced. PBPUTYP then resolves the worklist name and calls the list services firmware.

NOTE

If the address given to PBPUTYP is zero, it is assumed that the worklist entry is located in the appropriate element of the intermediate array BWWLENTRY [LEVELNO].

The calling sequence to make a worklist entry by terminal type is as follows:

```
PBPUTYP (parm)
```

where parm is a PASCAL variable or constant of type INTEGER. For example, the user building a worklist entry in BWWLENTRY [LEVELNO] uses PBPUTYP (0) and the user building a worklist entry in a local data structure uses:

```
ADDR (local structure, var)
PBPUTYP (var)
```

WITHOUT DISTURBING INTERMEDIATE ARRAY

To make a worklist entry in a specified worklist without destroying the contents of the intermediate array (BWWLENTRY [LEVELNO]), use the following calling sequence:

```
PBDLPUT (parm1, parm2)
```

where parm1 is a record variable containing the worklist entry to be put and parm2 is a variable or constant of type B0WKLSTS indicating which worklist is to receive the entry.

EXTRACT A WORKLIST ENTRY

GENERAL

PBLSGET obtains an entry from a worklist for a program at any priority interrupt level and signals if the list is empty. The calling sequence is as follows:

```
PBLSGET (parm)
```

where parm is a symbolic constant of the enumeration type B0WKLSTS and is an index to the proper worklist control block. PBLSGET is a PASCAL function that returns a 'true' value only when the list is empty.

OPS LEVEL

PBGETLIST is used only at OPS level and is the exact equivalent of PBLSGET except that more efficient code is generated. This calling sequence should be used by all OPS level users.

SYSTEM MONITOR

The 2550 series computer is a multiple-interrupt-level processor. Interrupts are serviced in a priority scheme in which all lower priority interrupts are disabled during execution of a program operating at a higher priority level. When no interrupt is in effect, the processor runs at its lowest priority, known as the operations monitor (OPS) level.

The system monitor controls allocation of time to programs running at the OPS level and gives control to a program by scanning tables that define programs and the worklist serviced by the program. Control is

released to the first program encountered with work in queue.

SCAN SEQUENCING

By appropriate structuring of the driver tables, both a priority and a program dependency scheme are incorporated into the system monitor. First, one or more programs are associated to become a segment. Then segments are defined as either priority or nonpriority segments. Scanning is perfcrmed as follows:

1. Each priority segment is scanned in turn.

2. After each complete scan of the priority segments, scan the nonpriority segments.

3. Return to the priority scan after one nonpriority program is serviced or after no work is found for any nonpriority program.

4. In scanning the programs within a segment, start at the next program after the last program to receive control or at the end of the previous inconclusive scan.

5. Release control to the first program encountered having work in queue or, if no such program is found, return to the segment scan.

NOTE

A program may cause the segment to which it belongs to wait (PBWAIT). In such cases, control returns to the waiting program each time that segment is scanned. The program monitors for the end of the WAIT condition and all programs in the waiting segment are locked out for the period of the WAIT.

USER INTERFACES

Figure 4-6 illustrates the organization of the OPS monitor tables. Note that the table setup exists for both priority and nonpriority programs and that a pointer indicates which of the two structures is currently being accessed. To add an OPS level program, add a new priority or nonpriority segment table and establish a worklist to drive the new program.

To provide a multiprogramming feature, OPSMON allows a program to WAIT for completion of a particular event. The WAIT function provides a means of contention resolution between programs and increases overall program response time since only programs in other segments may be scheduled while a program is waiting.

When OPSMON scans a WAITing segment, control is immediately returned to the waiting program. It is left to the individual programs to monitor for completion of the event or events for which the WAIT has been issued. Only programs given control by OPSMON may call the WAIT calling sequence, the format of which is as follows:

```
WAIT
```

OPSMON exits by transferring control to the OPS level program that has been selected for execution.

Each time a program completes, OPSMON initializes a timer (BTTIMER). This timer is advanced and checked by the interrupt level timer routine (PBTIMER) at specific system-defined intervals and, if the timer expires, it indicates that some OPS level program (perhaps OPSMON) has been abnormally delayed. OPSMON execution is then terminated by a call to to PBHALT.

TIMING SERVICES

Any system program that requires time regulation and which must be run at regular intervals is referred to as a time-dependent program (TDP). Timing services provides the means of running these programs at regular intervals. Also included in timing services are procedures used to main-

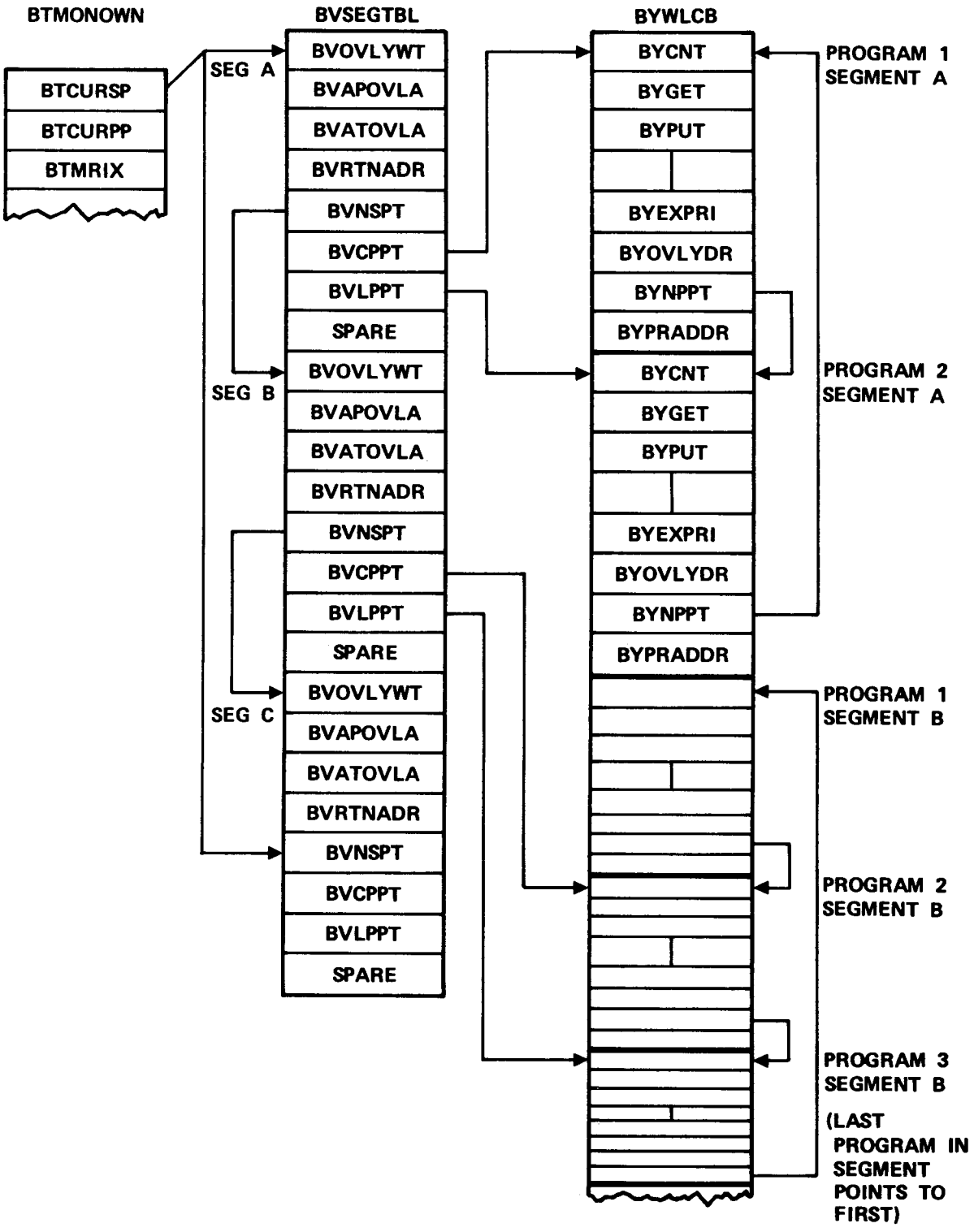


Figure 4-6. OPS Monitor Table Organization

tain the list of active line control blocks (LCBs) and a system option for maintenance of month, day, hour, minute, and second timers.

TIME-DEPENDENT PROGRAM (TDP) TYPES

TDPs are divided into two groups, those run at an interrupt level and those run at the OPS level. Typically, interrupt level TDPs run many times each second and OPS level programs have periods that are multiples of half seconds.

OPS level TDPs are controlled by PBTIMAL, an OPS level program that provides timing for TIPs and other programs. Timing for TIPs is provided via the LCB timers. Each base system LCB has an 8-bit field that is used by the TIP as a timer. PBTIMAL advances these LCB timers at regular intervals. If a TIP requires additional timing, the program must provide a procedure (referred to as a timal appendage) to handle such additional timing. TIMAL gives control to timal appendages at the same rate that it advances LCB timers.

Interrupt level TDPs are managed by the interrupt level timer routine, PBTIMER. This routine is driven by the MP17 real-time clock and is responsible for controlling the rate at which the clock interrupts. A counter and limit indicator control the real-time clock interrupt rate. The counter is advanced by the real-time clock each 3.33 milliseconds and a macro interrupt occurs when the count reaches the limit. The count is globally available as a general-purpose binary clock, but must never be altered by any software.

TIMER MAINTENANCE

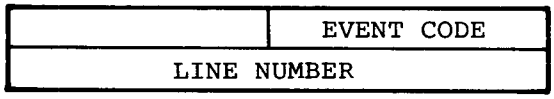
PBTIMAL is a worklist-driven OPS level program that is given control at half-second intervals. PBTIMAL services TDPs that are completely independent of each other and the only restriction on programs controlled by PBTIMAL is that the intervals at which they are run must be multiples of one-half second.

Any TDPs controlled by PBTIMAL may be put on skip by setting the time-remaining indicator for that program to zero. The program is then disabled until the time-remaining indicator is set to a nonzero value.

TDPs defined in the TDP table receive control by procedure call at the specified interval. No parameters are passed. Timing services are provided to TIPs via the base timer field (BZLTIMER) in each LCB and via timal appendages provided by the TIP itself.

When BZLTIMER is not in use, its value is zero. When a TIP must start a timer for a particular active line, it sets BZLTIMER to the interval to be measured (in units of one-half second) to a maximum of 127.5 seconds. PBLCBTMSCN decrements BZLTIMER each half second and, when BZLTIMER expires (becomes zero), PBLCBTMSCN makes a worklist entry for the TIP controlling that line. The LCB timer can be deactivated or restarted at any time by setting the timer field to the appropriate value. The LCB timer may only be modified by OPS level programs.

The format of the TIP worklist entry is as follows:



where EVENT CODE is AOTIMEOUT.

INTERRUPT LEVEL TIMER

The interrupt level timer (PBTIMER) is given control each time a real-time clock interrupt is acknowledged. Functions of PBTIMER include control of the interval between real-time clock interrupts, providing control to the multiplex subsystem timer (PMT200M) at specified intervals, making a worklist entry for PBTIMAL at half-second intervals, and (optional) incrementing CDCOUNT at a specified interval. The calling sequence for this routine is:

```
PBTIMER
```

The interrupt level timer requires no parameters and returns no values. PBTIMER is called only from the first level real-time-clock interrupt handler (PBLNO8).

DATE-TIME MAINTENANCE

PBTIMOFDAY is a time-dependent program that may optionally be included with timing services at system build time. Its function is to maintain the current date and time, with time being kept as the standard 24-hour clock with hours, minutes, and seconds in binary (as opposed to internal ASCII) The date is kept as month and day, also in binary. Once each second, PBTIMOFDAY is called by PBTIMAL (via CBTIM) to update the time and date. For proper operation of PBTIMOFDAY, there must be an entry in CBTIMTBL as follows:

CBTIMTBL [n].CBTIMER has a value of 2.

CBTIMTBL [n].CBINTVAL has a value of 2.

CBTIMTBL [n].CBADDR has the entry address of PBTIMOFDAY.

where n is a constant of the enumeration type COTDPGMS.

Output of PBTIMOFDAY is the advancement of the time of day and advancement of the date (at midnight).

ACTIVE LINE CONTROL BLOCK (LCB) LIST MAINTENANCE

Timing services includes programs to maintain the active LCB list. Included are routines to enter an LCB into the active list, remove an LCB from the active list, and to search the active list to determine if a particular LCB is included. The active LCB list is maintained in zero (when empty) or more chained buffers.

To enter an LCB into the list of active LCBs, execute the following calling sequence:

```
PBLLENTR (parm)
```

where parm is a variable of the INTEGER type representing the 16-bit line number of the LCB to be added to the list CELCBACT↑. When the last buffer of the active LCB chain is full, PBLLENTR obtains another buffer and establishes appropriate chaining links. New entries are made at the end of the active list.

To remove an entry from the active LCB list, use the following calling sequence:

```
PBLLRMOV (parm)
```

where parm is a variable of the INTEGER type that specifies the line number of the LCB to be removed. The entry to be removed is located and replaced by the last entry in the list (except when the last entry is specified for removal). If the last buffer in the chain becomes empty, that buffer is released. If an attempt is made to remove an entry not in the list, SYSTEM HALT is called.

The calling sequence:

```
PBLLSRCH (parm)
```

where parm is a variable of the INTEGER type that specifies the line number of the searched-for LCB. If the searched-for LCB is found in the active list, a true Boolean value is returned. If not, a false Boolean value is returned. In addition to the Boolean value, PLLLSRCH provides the exact location of the last entry found. A pointer to the buffer and index within the buffer is saved in the global data structure CFLLPARM. By using a global structure, the information is made available to all users without requiring passing of the parameters if they are not needed.

INTERRUPT HANDLER

The computer can recognize 16 different macro interrupts, each with its own address to which control

is transferred when the interrupt is recognized. When the computer is processing a particular interrupt, it is defined as being in that interrupt state (state 00 through 15). However, before the computer can recognize an interrupt, the corresponding mask bit in the interrupt mask register must be set and the interrupt system must be activated.

BASIC INTERRUPT PROCESSING

The interrupt mask register is set by an inter-register command and the interrupt system is activated by the enable interrupt command. Upon recognizing an interrupt, the hardware automatically stores the appropriate program-return address in a storage location reserved for the activated interrupt state. This ensures that, after interrupt processing, the software can return to the interrupted program.

With the return address stored, the hardware deactivates the interrupt system and transfers control to an interrupt handler program that begins at an address specified for that interrupt state. The program thus entered stores all registers (including the interrupt mask register and overflow) in addresses reserved for the interrupt state. The interrupt mask register is then loaded with a mask to be used while in this interrupt state, with one-values in this mask indicating interrupts with higher priority than the interrupt state being processed. The program then saves the current software priority level, sets the new software level, activates the interrupt system, and processes the interrupt.

During such interrupt processing, an interrupt request with higher priority may interrupt. However, such interrupts also cause storage of return address links to permit sequential interrupt processing according to priority level with eventual return through the return addresses to the main-stream computer program.

The computer exits from an interrupt state when processing is completed at that level by inhibiting interrupts, restoring registers to their states prior to interrupt, and executing the exit interrupt command. This command retrieves the return address stored when the interrupt state was entered. Control is transferred to the return address and the interrupt system is again activated.

MASK REGISTER

The priority of interrupts is under control of the computer program. Such priority is established by an interrupt mask for each interrupt state that enables all higher priority interrupts and disables all lower priority interrupts. When an interrupt state is entered, the mask for that state is placed in the mask register. Bit 0 of the mask register corresponds to interrupt state 00, bit 1 corresponds to interrupt state 01, etc. If a bit is set, it indicates that the corresponding interrupt state has a higher priority than the interrupt state to which the mask belongs. Thus, there may be as many as 16 levels of priority.

NOTE

Priority of an interrupt state can be changed during program execution.

USER INTERFACES

Because each interrupt handler is an independent program, there are no specific user interfaces. However, pertinent information is necessary to enable modification of and additions to the interrupt handlers.

XJKMASK is an array containing interrupt masks for the 16 interrupt states. To access a particular interrupt mask, use the interrupt state number as an index. LEVELNO is the core location where the current software priority level is saved.

Table 4-1 lists the 16 interrupt states, gives the value for the delta field for its exit instruction, the storage location for its return address, and the location of the first instruction of the interrupt handler program. Current interrupt assignments and their associated software priority are listed in table 4-2.

INITIALIZATION

The initialization module gains control immediately after the NPU is loaded and prepares the system for on-line operation. After these initialization functions are performed, the initialization program is no longer needed. Just prior to releasing control, the space occupied by this program is added to the dynamic buffer pool.

Initialization consists of a series of procedures. The base system supplies those procedures common to all systems and customized initialization procedures may be added as required. As each initialization procedure is completed, an appropriate flag bit is set in the NPISFL word of the NPINTAB table. When procedures are added, the call that returns the initialization space to the buffer pool must remain in the last position, and the procedure that effects this return must remain as the last physical procedure.

BUFFER INITIALIZATION (FIRST PHASE)

Each buffer size defined for a system is represented by a buffer control block that contains GET and PUT pointers, etc., and a threshold indicating need for replenishment by the buffer adjustment program. The buffer initialization program starts at the base of the area assigned to buffers and releases appropriate increments of space to the buffer pool. Starting with the smallest size buffer pool and moving up through the defined pools, sufficient space is released up to the base of the buffer area holding the initialization program itself. That area is not released at this time.

LIST CONTROL BLOCK INITIALIZATION

List service assumes that all list control blocks are in a condition that allows an entry to be made to an empty list without first having to obtain a buffer. Thus, initialization obtains a buffer for each list control block and primes the GET and PUT pointers.

MULTIPLEX LOOP INTERFACE ADAPTER (MLIA) INITIALIZATION

The MLIA constitutes the major communications interface for the NPU. MLIA initialization consists of a series of commands that place the MLIA in the proper operating mode. Parameters are read back to ensure that the MLIA took correct action. If the MLIA is not ready or cannot be set up correctly, initialization is aborted.

MICROPROGRAM LINKAGE INITIALIZATION

The emulation and multiplex subsystem microprograms use volatile storage elements such as the interrupt mask registers and file registers which must be initialized prior to operation. In addition, the microprograms may be partially resident in non-volatile ROM and partially resident in reloadable RAM. Thus, initialization is required to perform the following:

1. Load RAM with multiplex subsystem firmware.
2. Set interrupt vector table and other constants in file registers 1 and 2.
3. Enable internal and external interrupts.

TABLE 4-1. INTERRUPT STATE DEFINITIONS (PBINTRAPS)

Interrupt State	Exit Instruction Delta Field Value	Location of Return Address	Location of First Instruction of Interrupt Handler Program
00	00	0100	0101
01	04	0104	0105
02	08	0108	0109
03	0C	010C	010D
04	10	0110	0111
05	14	0114	0115
06	18	0118	0119
07	1C	011C	011D
08	20	0120	0121
09	24	0124	0125
10	28	0128	0129
11	2C	012C	012D
12	30	0130	0131
13	34	0134	0135
14	38	0138	0139
15	3C	013C	013D

TABLE 4-2. INTERRUPT ASSIGNMENTS

Interrupt State	Software Priority	Interrupt Description
0	P1	Memory parity, program protect, power fail, software breakpoint
1	P6	Communications console (TTY or CRT)
2	P2	Multiplex loop error (MLIA)
3	P3	Multiplex subsystem - level 2
4	P16	Line Printer (2571/2570)
5	P7	Coupler No. 2 (2558-1)
6	P7	Coupler No. 1
7	P8	Tape cassette
8	P9	Real-time clock
9	P10	
10	P11	
11	P12	Card Reader (2571/2572)
12	P13	Output data demand received (MLIA)
13	P14	Input line frame received (MLIA)
14	P15	Spare
15	---	Macro breakpoint
	P17	OPS level programs

Prior to operation, an equipment configuration test is performed to ensure that specific hardware is present, powered, and in the correct operating mode. The tests primarily check that: 1) two or more CLAs are not assigned the same port number, 2) proper status is received from the coupler, and 3) the MLIA is operating properly.

PROTECT SYSTEM SET-UP

The NPU has a program protect feature using a protect bit associated with each storage location to prevent inadvertent corruption of programs in transfer of control. During initialization, all core storage except the buffer area is set to the protect mode. This prevents a hardware malfunction from storing into program areas.

BUFFER INITIALIZATION (SECOND PHASE)

After all initialization procedures have been executed, the buffer area containing the initialization program is released to the free buffer pool for the largest size buffer defined.

STANDARD SUBROUTINES

OVERVIEW

The standard subroutines included as part of CCP 1.0 are self-contained programs available to the user for performance of useful and commonly needed functions. Table 4-3 lists these subroutines in alphabetical order by mnemonic subroutine name. This same sequence is used in describing the subroutines in following paragraphs. As shown in the table, most can be called from any software priority level.

CALLING ASSEMBLY LANGUAGE PROGRAMS FROM PASCAL

Procedure calls to assembly language programs from PASCAL are the same as calls to other PASCAL programs. The same calling sequence code is generated:

```
RTJ     program
ADC     parm1
 .        .
 .        .
 .        .
ADC     parmn
```

An assembly language program handles parameters as PASCAL. To treat a parameter as a value parameter, load the contents of the parameter and store locally. To treat a parameter as a variable parameter, load the address of the parameter and use as a pointer. Parameters that are fields less than a word in length in a packed record are unpacked into a temporary word and the address of the temporary word is passed to the called program.

Function calls to assembly language programs differ in that a PASCAL forward reference describing the calling sequence must appear before all function calls in the source code so that type-checking on the function return value can be performed.

CALLS TO PASCAL PROGRAMS FROM ASSEMBLY LANGUAGE

All calls to PASCAL programs from assembly language must conform to the calling sequence code described in the preceding paragraph. The caller, of course, must be familiar with the calling sequence of the PASCAL program being called.

TABLE 4-3. ALPHABETIC LIST OF STANDARD SUBROUTINES

Subroutine Name	Description	Type*	Language†	Type Checking Defeated
PBAEXIT	Save R1 and R2	NI	AP	X
PBAMASK	AND Interrupt Mask	NI	AP	X
PBBEXIT	Restore R1 and R2	NI	AP	X
PBCALL	Call Program by Address	NI	AP	X
PBCLRPRCT	Clear Protect Bit	NI	AP	X
PBCOPYBFRS	Copy a Chain of Buffers	O	PF	
PBDISPLAY	Display Msg on Console	O	PP	X
PBDLTXT	Delete Text	O	PP	
PBDUMP	On-line Dump	NI	PP	X
PBFILE	Load/Display File 1	O	AP	X
PBFMAD	Convert from ASCII to Decimal	R	PF	
PBFMAH	Convert from ASCII to Hex	R	PF	
PBHALT	System Halt	NI	PP	X
PBLMASK	Reload Interrupt Mask	NI	AP	X
PBLOAD	Load a Canned Message	R	PP	X
PBMAX	Get Max of 2 Numbers	NI	PF	
PBMEMBER	Test ASCII Set Membership	NI	PF	

TABLE 4-3. ALPHABETIC LIST OF STANDARD SUBROUTINES (Continued)

Subroutine Name	Description	Type*	Language†	Type Checking Defeated
PBMIN	Get Min. of 2 Numbers	NI	PF	
PBOMASK	OR Interrupt Mask	NI	AP	X
PBQUICKIO	Quick Output	R	PP	
PBSETPROT	Set Protect Bit	O	AP	X
PBSMASK	Set Interrupt Mask	NI	AP	X
PBSLJ	Test Selective Jump Switch	NI	AF	X
PBSTRIP	Strip Empty Buffers	O	PP	
PBTIPDBG	Execute User Code	O	PP	
PBTOAD	Convert to ASCII Decimal	R	PP	
PBTOAH	Convert to ASCII Hex	R	PP	
PIPRINT	Print Structure Addresses	--	PP	
PTCTCHR	Count Characters	NI	PF	
QDEBUG	PASCAL Debug Option Handler	NI	PP	
QENTRY	Recursive Procedure Entry Code	--	AP	
QEXIT	Recursive Procedure Exit Code	--	AP	
QULOCK	Non-Int. Function Exit Code	--	AP	

*NI = Non-interruptable
O = OPS Level only
R = Re-entrant

†PP = PASCAL procedure
PF = PASCAL function
AP = Macro Assembler procedure
AF = Macro Assembled function

DEFEATING TYPE-CHECKING IN PASCAL PROCEDURE CALLS

As the PASCAL compiler is a one-pass compiler, when it encounters a procedure call in source code it may or may not have processed the calling sequence of the called program.

If the calling sequence has been processed, all parameters of the user's procedure are error checked. The type of each parameter must correspond to the type specified in the calling sequence and the number of parameters must be the same. No expressions and no fields of less than a word in length in a packed record can be variable parameters.

If the calling sequence of a program has not been processed when a call to it is encountered, a subroutine jump to an external symbol is generated, the standard calling sequence is generated, and no error checking is done on the parameters. This situation is said to defeat type-checking in the procedure call.

If used carefully, defeating type-checking is a useful technique. For example, arrays with the same element types but of different lengths are treated as different types by PASCAL. Therefore, any program needing variable length array input as a variable parameter must defeat type-checking. Ramifications of defeating type-checking are:

1. All calls from PASCAL programs to assembly language procedures automatically defeat type-checking.

2. PASCAL and assembly language functions cannot defeat type-checking.

PBAEXIT - SAVE R1 AND R2

PBAEXIT is used at the label of a GOTO(EXIT) statement when that statement occurs within one or more executable WITH statements. PBAEXIT restores R1 and R2 from a specified save area so that they may be used as base addresses of the structures associated with the first two executable WITH statements in the PASCAL program. The calling sequence is:

```
PBAEXIT (parm)
```

where parm is the name of the two-word save area for R1 and R2.

PBBEXIT is used to save R1 and R2 before executing the GOTO(EXIT). Other user information is:

Program Type:	Noninterruptable
Language:	Macro Assembler
Procedure or Function:	Procedure
Function Type:	---
Type-Checking Defeated:	Yes

NOTE

A GOTO(EXIT) from within a non-interruptable program does not perform an UNLOCK before exiting.

PBAMASK - AND INTERRUPT MASK

PBAMASK forms a logic AND function between a given interrupt mask and the current interrupt mask in the M register. The old value is stored in the global array JKTMASK [LEVELNO]. The resultant mask becomes the new mask value in the M register. PBAMASK, in conjunction with PBLMASK, is used to selectively disable and enable one or more software interrupt levels. The calling sequence is:

```
PBAMASK (parm)
```

where parm is a value parameter specifying the value to be logically ANDed with the current interrupt mask. Other user information is:

Program Type:	Noninterruptable
Language:	Macro Assembler
Procedure or Function:	Procedure
Function Type:	---

Type-Checking Defeated:	Yes

The following restriction also applies:

PBAMASK and PBLMASK pairs cannot be nested on the same software priority level. See PBLMASK.

PBBEXIT - RESTORE R1 AND R2

PBBEXIT is used before a GOTO(EXIT) is executed from within one or more executable WITH statements. PBBEXIT saves R1 and R2 in a specified save area which may be used as base addresses of the structures associated with the first two executable WITH statements. The calling sequence is:

```
PBBEXIT (parm)
```

where parm is the name of the two-word save area for R1 and R2. Other user information is:

Program Type:	Noninterruptable
Language:	Macro Assembler
Procedure or Function:	Procedure
Function Type:	---
Type-Checking Defeated:	Yes

PBCALL - CALL PROGRAM BY ADDRESS

PBCALL calls a procedure from PASCAL by address, rather than by name. Unlike other procedure calls, PBCALL can pass a variable number of parameters, corresponding to the number of parameters expected by the calling procedure. Example:

```
type pgms = (pgm1,...pgmn);
var  table: array [pgms] of integer:
     index: pgms;
addr ({program1}, table [pgm1];
.
.
.
addr ({programn}, table [pgmn]);
.
.
.
{set up index}
PBCALL (table [index]); {call progrm,
                         no parameters}
```

The PBCALL calling sequence is:

```
PBCALL (addr, parm1,...parmn)
```

where addr is the address of the program to be called and parm1 through parmn are optional and are the parameters passed to the called program. Other user information is:

Program Type:	Noninterruptable
Language:	Macro Assembler
Procedure or Function:	Procedure
Function Type:	---
Type Checking Defeated:	Yes

Structured flow is as follows:

```
procedure PCBALL;
begin
  {store return address in called
   procedures entry point}
  {jump to procedure}
end;
```

PBCLRPOT - CLEAR PROTECT BIT

PBCLRPOT clears the protect bit at the specified address. Its calling sequence is as follows:

```
PBCLRPOT (parm)
```

where parm is the address at which the protect bit is to be cleared. Other user information is:

Program Type:	Noninterruptable
Language:	Macro Assembler
Procedure or Function:	Procedure

Function Type:	---
Type Checking Defeated:	Yes

PBDISPLAY - DISPLAY MESSAGE ON CONSOLE

This sequence queues a message for output to the local console. Its calling sequence is:

```
PBDISPLAY (parm)
```

where parm is a variable parameter specifying the message to display. Example:

```
const crlf = $D0A
      bell = $707;
var   msg : packed array [0..10]
            of char;
value msg = (crlf, bell,=ABCDEF]=);
.
.
.
PBDISPLAY (msg);
```

NOTE

Every canned message must have a right bracket (]) as an end of text delimiter.

Other user information is:

Program Type:	OPS Level Only
Language:	PASCAL
Procedure or Function:	Procedure
Function Type:	---
Type Checking Defeated:	Yes

PBDISPLAY uses the PBLOAD and PBIOSERV subroutines to, respectively, load a canned message and to provide input/output services. PBDISPLAY also uses system structure JCOPSLRP (OPS level console LRP).

PBDLTXT - DELETE TEXT

PBDLTXT deletes text in a data buffer chain by advancing the FCD. The calling sequence is:

```
PBDLTXT (parm1, parm2, parm3)
```

where parm1 is a variable parameter of type B0BUFPTR containing the address of the buffer where text deletion is to begin. The FCD in this buffer must point to the first character to delete. Parm1 is updated if deletion crosses buffer boundaries. The FCD in the returned buffer points to the next character to process.

Parm 2 is a Boolean value parameter specifying whether to release source buffers if buffer boundaries are crossed during deletion. If true, source buffers are released.

Parm3 is an integer value parameter specifying the number of text characters to delete.

NOTE

PBDLTXT does not advance beyond the end of text. Deletion stops when end of text is reached. The OPS level error flag B6OSERR is returned false if deletion beyond end of text is attempted.

Other user information is:

Program Type:	OPS Level Only
Language:	PASCAL
Procedure or Function:	Procedure
Function Type:	---
Type Checking Defeated:	No

Structured flow is as follows:

```
procedure PBDLTXT;
begin {set B6OSERR}
10: if {buffer boundary crossed} then
      if {chain nil} then
        begin {set FCD = LCD in buffer}
              {reset B6OSERR}
        end
      else
        begin {chain to next buffer}
              {decrement deletion
               count by LCD-FCD of
               previous buffer}
```

```
          if {buffer release
              requested}
            then {release previous
                  buffer}
          GOTO 10 .
        end
    else {set FCD to FCD + deletion
          count}
end;
```

PBDUMP - ON-LINE DUMP

PBDUMP transfers information from within specified limits of the core memory to a specified local peripheral device. The calling sequence is:

```
PBDUMP (parm1, parm2, parm3)
```

where parm1 and parm2 are value parameters specifying the start and stop dump addresses respectively, and parm3 is a value parameter specifying the local peripheral device as follows:

0	Null device
1	Teletype
2	Line Printer

Other user information is:

Program Type:	Noninterruptable
Language:	PASCAL
Procedure or Function:	Procedure
Function Type:	---
Type Checking Defeated:	Yes

External subroutines used by PBDUMP are:

PBTUPDUMP	TUB Dump Formatter
PBQUICKIO	Quick Output
PBTESTIORDY	Test I/O Device Ready
PBWRITE	Write Character or Function

The output is formatted in the dedicated TUP buffer JUTUPOUT. The structured flow for PBDUMP is as follows:

```
procedure  PBDUMP;
begin
  if {output to teletype}
    then {set teletype to write mode}
         {reset EOT flag in JUTUPOUT}
  repeat {format one buffer of output}
         PBQUICKIO
  until {EOT flag set in JUTUPOUT}
end;
```

PBFILE - LOAD/DISPLAY FILE 1

PBFILE actually consists of two programs: PBEF (Load File 1) and PBDF (Display File 1). Both programs execute special firmware sequences to perform the load or display operations. Because of firmware timing constraints, a maximum of 12 transfers per call can be specified during on-line operation. During off-line operation, as many as 256 transfers can be specified.

PBEF transfers the contents of memory to File 1 starting at a specified register. The PBEF calling sequence is:

```
PBEF (parm1, parm2)
```

where parm1 is a value parameter formatted as follows:

Bits 15 8 7 0

Number of words to load	First File 1 register to load

To load all 256 registers, set parm1 to 0. Parm2 is a value parameter specifying the address of the first memory location to transfer.

PBDF transfers the contents of File 1, starting at register n, to memory. The PBDF calling sequence is:

```
PBDF (parm1, parm2)
```

where parm1 is a value parameter formatted as follows:

Bits 15	8 7	0
Number of words to move	First File 1 register to transfer	

To display all 256 registers, set parm1 to $0. Parm2 is a value parameter specifying the memory address to receive the first register transfer.

Other user information for both PBEF and PBDF is as follows:

Program Type:	OPS Level Only
Language:	Macro Assembler
Procedure or Function:	Procedure
Function Type:	---
Type Checking Defeated:	Yes

PBFMAD - CONVERT FROM ASCII DECIMAL

PBFMAD converts up to five ASCII decimal digits in a buffer into one 16-bit word. The calling sequence is:

```
PBFMAD (parm1, parm2, parm3)
```

where parm1 is a variable parameter of type INTEGER. The converted word is returned in parm1. Parm2 is a value parameter of type B0BUFPTR specifying the buffer address where the decimal digits to be converted are located. Parm3 is a variable parameter of type integer specifying the index where the first decimal digit to be converted is located within the buffer.

PBFMAD is a Boolean function in which, if PBFMAD is true, the conversion is successful and, if false, indicates bad data or a bad start/stop index. Other user information is:

Program Type:	Re-entrant
Language:	PASCAL
Procedure or Function:	Function
Function Type:	Boolean
Type Checking Defeated:	No

PBFMAD uses external subroutine PBMEMBER to test ASCII set membership. System table JNCNVTFROM (convert from ASCII) is used.

PBFMAH — CONVERT FROM ASCII HEXADECIMAL

PBFMAH converts ASCII hexadecimal characters in a buffer to a 16-bit word. The calling sequence is:

```
PBFMAH (parm1, parm2, parm3)
```

where parm1 is a variable parameter of type B0OVERLAY. The converted word is returned in parm1. Parm2 is a value parameter of type B0BUFPTR that points to the buffer address where the hexadecimal characters to be converted are located. Parm3 is a variable parameter of type integer specifying the index where the first hexadecimal character to be converted is located within the buffer.

Like PBFMAD, PBFMAH is a Boolean function which, if true, indicates the conversion is successful and, if false, indicates either bad data or a bad start/stop index. Other user information is:

Program Type:	Re-entrant
Language:	PASCAL
Procedure or Function:	Function
Function Type:	Boolean
Type Checking Defeated:	No

PBFMAH, like PBFMAD, uses external subroutine PBMEMBER and system table JNCNVTFROM. Structured flow for PBFMAD is as follows:

```
function PBFMAH;
begin
  if {start/stop index in range}
    then
    repeat
      if {character a delimiter
          (blank, slash, comma
          or (D)}
      then GOTO 10
(D) is controlled D on console
    keyboard
```

```
  else
    begin {test for valid hex
           character}
          {convert character}
          {bump to next character}
    end
    until {bad character} v {end
           of buffer reached}
10: {bump buffer index}
    {PBFMAH : = NOT [error
                     indicator]}
end;
```

PBHALT halts the system after a recognizable but irrevocable condition has occurred. System halt always locks interrupts, saves the software registers, and clears the MLIA and coupler. The user then has the option to either branch directly to post mortem dump or to print the halt message at the local console, enable the local console and line printer interrupts, and enter the TUP mode. This option is controlled by the global Boolean JXHALTFLAG. If true, post mortem dump is called. The default value is false.

The calling sequence for system halt is:

```
PBHALT (parm)
```

where parm is an integer value parameter specifying the halt code. The halt message printed at the local console is:

```
*HALT XXXX YYYY
```

where XXXX is the return address of the program calling PBHALT and YYYY is the hexadecimal halt code. The halt code is also stored at location $30, the halt return address is stored at $31, and the registers are stored beginning at location $32.

Other user information is:

Program Type:	Noninterruptable
Language:	PASCAL
Procedure or Function:	Procedure
Function Type:	---
Type Checking Defeated:	No

External subroutines used by PBHALT are:

PBMDUMP	Post Mortem Dump
PBTESTIORDY	Test I/O Device Ready
PBWRITE	Write to I/O Device
PBLOAD	Load a Canned Message
PBTOAH	Convert to ASCII Hexadecimal
PBQUICKIO	Quick I/O
PBSMASK	Set Interrupt Mask
PBTUP	Test Utility Program

System tables used are:

JUTUPTABLE	TUP Table
JACT[TTY]	Local Console Controller Table
J1QUICKPTR	Global Buffer Pointer for PBQUICKIO

The structured flow for PBHALT is as follows:

```
procedure PBHALT;
begin
  {save registers}
  {clear coupler, MLIA}
  if JXHALTFLG then {post-mortem
                     dump}
  {clear all local peripherals}
  {set up *HALT message}
  {print *HALT message}
  {enable teletype and line
   printer interrupts}
  repeat
    {enable interrupts}
    {call TUP}
  until false
end;
```

PBLMASK - RELOAD INTERRUPT MASK

This sequence loads the interrupt mask stored in the global array JKTMASK into the mask (M) register. JKTMASK is indexed by LEVELNO, the current system priority level. PBLMASK, in conjunction with PBAMASK, selectively disables and enables one or more software priority levels.

Example: Assume the user has a segment of code where the local console interrupt must be locked out. Reference type definition J8HDWLINE in SDS.

```
var   ALLINT : SETWORD;
value ALLINT = ($FFFF);
      .
      .
      .
PBAMASK (ALLINT - [J8TTY]);
{code to protect from local
 console interrupt}
PBLMASK;
```

The calling sequence is:

```
PBLMASK
```

Other user information is:

Program Type:	Noninterruptable
Language:	Macro Assembler
Procedure or Function:	Procedure
Function Type:	---
Type Checking Defeated:	Yes

NOTE

To ensure that once an interrupt is disabled by PBMASK that it is not inadvertently re-enabled by an intermediate interrupt level, all first level interrupt handlers logically AND the current value of the interrupt mask (M) register with the new interrupt mask value for that software priority level.

PBLOAD - LOAD A CANNED MESSAGE

The PBLOAD sequence loads a user-defined message into a user-supplied buffer at a user-specified start position. The calling sequence is:

```
PBLOAD (parm1, parm2, parm3,
        parm4)
```

where parm1 is a value parameter of type B0BUFPTR and points to the location where the canned message is to be loaded, parm2 is a variable parameter specifying the message to be loaded, parm3 is an integer value parameter that is the index of the start position in the buffer for the first character of the message, and parm4 is the index to the last data position in the buffer. Parm4 overrides the message length. Example:

```
var   Buffer: B0BUFPTR; {assume a
                         32-word
                         buffer}
      MSG : J0ML10:
value MSG = (=0123456789]=);
      .
      .
      .
PBLOAD (BUFFER, MSG, J1FRSTCHAR,
        J1LST32);
```

NOTE

All canned messages must have a right-bracket (]) as end of message delimiter unless parm4 - parm3 is less than the message length.

Other user information is:

Program Type:	Re-entrant
Language:	PASCAL
Procedure or Function:	Procedure
Function Type:	---
Type Checking Defeated:	Yes

PBMAX - GET MAXIMUM OF TWO NUMBERS

PBMAX is a function that returns the maximum of two given numbers. The calling sequence is:

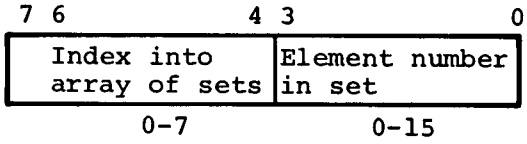
```
PBMAX (parm1, parm2)
```

where parm1 and parm2 and integer value parameters. The maximum of parm1 and parm2 is returned by PBMAX. Other user information is:

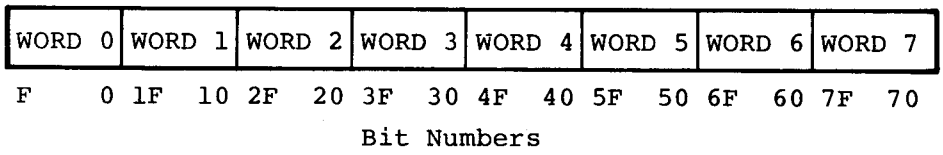
Program Type:	Noninterruptable
Language:	PASCAL
Procedure or Function:	Function
Function Type:	Integer
Type Checking Defeated:	No

PBMEMBER - TEST ASCII SET MEMBERSHIP

PBMEMBER is used to determine whether or not a given ASCII character is a member of a user-defined set of ASCII characters. PBMEMBER overcomes the 2550 PASCAL restriction of having one-word, 16-element sets by accessing an array of one-word sets. A character is broken up for testing as follows:



128 bits are reserved (one for each possible ASCII character) in an array of type JSASCIISET, where JSASCIISET = array [0..7] of SETWORD. Characters are located in the set by bit number; e.g., a blank ($20) is bit number $20. Bits of the JSASCIISET array are numbered as follows:



Therefore, the value initialization for testing hexadecimal characters is:

```
var   JSHEXSET : JSASCIISET;
value JSHEXSET = (0, 0, 0, $3FF,
                  DIGITS 0-9
                  $7E, 0, 0, 0);
                  CHARACTERS A-F
```

The calling sequence is:

```
PBMEMBER (parm1, parm2)
```

where parm1 is a value parameter of type BOOVERLAY containing the character to test and parm2 is a variable parameter of type JSASCIISET and is the set to test parm1 for membership. PBMEMBER is a Boolean function and returns a true if the character is in the set and false if it is not. Other user information is:

Program Type:	Noninterruptable
Language:	PASCAL
Procedure or Function:	Function
Function Type:	Boolean
Type Checking Defeated:	No

PBMIN - GET MINIMUM OF TWO NUMBERS

PBMIN is a function that returns the minimum of two given numbers. The calling sequence is:

```
PBMIN (parm1, parm2)
```

where parm1 and parm2 and integer value parameters. The minimum of parm1 and parm2 is returned by PBMIN. Other user information is:

Program Type:	Noninterruptable
Language:	PASCAL
Procedure or Function:	Function
Function Type:	Integer
Type Checking Defeated:	No

PBOMASK - OR INTERRUPT MASK

PBOMASK employs a logical OR function to combine a given interrupt mask with the current mask in the M register, the result becoming the new interrupt mask value in the M register. The calling sequence is:

```
PBOMASK (parm)
```

where parm is a value parameter specifying the mask value to OR with the current interrupt mask. Other user information is:

Program Type:	Noninterruptable
Language:	Macro Assembler
Procedure or Function:	Procedure
Function Type:	---
Type Checking Defeated:	Yes

PBQUICKIO - QUICK OUTPUT

PBQUICKIO writes one buffer of ASCII information to a user-specified peripheral device. PBQUICKIO is a noninterruptable program and is not intended for use while the system is on-line. The calling sequence is:

```
PBQUICKIO (parm1, parm2)
```

where parm1 is an integer value parameter specifying the local peripheral device as follows:

0 Null device

1 Teletype

2 Line Printer

Parm2 is a value parameter of type B0BUFPTR containing the address of the buffer of ASCII output that is to be output. Other user information is:

Program Type:	Noninterruptable
Language:	PASCAL
Procedure or Function:	Procedure
Function Type:	---
Type Checking Defeated:	No

External subroutines used are:

PBTESTIORDY	Test I/O Device Ready
PBWRITE	Write to I/O Device

System table used is:

JACT	Peripheral I/O Controller Tables

PBSETPROT - SET PROTECT BIT

PBSETPROT sets the protect bit at a specified address. The calling sequence is:

```
PBSETPROT (parm)
```

where parm is the address where the protect bit is to be set. Other user information is:

Program Type:	Noninterruptable
Language:	Macro Assembler
Procedure or Function:	Procedure
Function Type:	---
Type Checking Defeated:	Yes

PBSMASK - SET INTERRUPT MASK

This calling sequence loads a specified interrupt mask value into the M register to become the new interrupt mask. The calling sequence is:

```
PBSMASK (parm)
```

where parm is a value parameter specifying the new interrupt mask value to be loaded into the M register. Other user information is:

Program Type:	Noninterruptable
Language:	PASCAL
Procedure or Function:	Procedure
Function Type:	---

Type Checking Defeated:	Yes

PBTIPDBG - EXECUTE USER CODE

PBTIPDBG is an OPS program available as a debug aid. It enables the user to execute special code via a worklist entry into PBTIPDBG. The code is added to an already existing case statement in PBTIPDBG. The case statement label is the first word of a five-word worklist entry made to engage the code. The remaining four words are optionally used to pass information to drive the user code. In current implementation, case statement label 0 is a 20-word patch area.

The worklist entry to PBTIPDBG can be made from anywhere in the system or from the local console via the TUP LP command. As PBTIPDBG is an OPS program, there is no calling sequence. Other user information is:

Program Type:	Entered from OPS Monitor only
Language:	PASCAL
Procedure or Function:	Procedure
Function Type:	---
Type Checking Defeated:	No

Structured flow is:

```
procedure  PBTIPDBG;
begin
  if  {worklist non-empty} then
    case {first word of WL entry} of
      0: {20-word patch area}
      2: {print date-time}
      3: {modify POI and/or block
          length for TIP testing}
      4: {set date and time}
    end
end;
```

PBTOAD - CONVERT TO ASCII DECIMAL

PBTOAD converts one 16-bit word to as many as five ASCII decimal digits with leading zeros suppressed. The converted digits are stored in a specified position in a buffer, followed by a blank. The calling sequence is:

```
PBTOAD (parm1, parm2, parm3,
        parm4)
```

where parm1 is an integer value parameter containing the word to be converted, parm2 is a value parameter of type B0BUFPTR pointing to the buffer that will store the converted ASCII digits, and parm3 and parm4 are integer value parameters respectively specifying the start and stop indices for storing the converted ASCII digits in the buffer. Other user information is:

Program Type:	Re-entrant
Language:	PASCAL
Procedure or Function:	Procedure
Function Type:	---
Type Checking Defeated:	No

The JMCNVTO (convert to ASCII) system table is also used. Structure flow is as follows:

```
procedure  PBTOAD;
begin
  {adjust start/stop index if too
   small/large}
  {reset flag}
  for I : = {stop index} down to
            {start index} do
    if {flag} ∧ {parm1 = 0}
      then {store blank in position
            I in buffer}
      else
      begin
        {convert one decimal digit}
        {divide parm1 by 10}
        {set flag}
      end
  {store blank after last character
   converted}
end;
```

PBTOAH - CONVERT TO ASCII HEXADECIMAL

PBTOAH converts one 16-bit word into four hexadecimal characters. The

converted characters are stored in a specified position in a buffer, followed by a blank. The calling sequence is:

```
PBTOAH (parm1, parm2, parm3,
        parm4)
```

where parm1 is a value parameter of type B0HEX and contains the word to be converted, parm2 is a value parameter of type B0BUFFPTR that points to the buffer where the converted hexadecimal characters are to be stored, and parm3 and parm4 are integer value parameters, respectively, specifying the start and stop indices for storing the characters within the buffer. Other user information is:

Program Type:	Re-entrant
Language:	PASCAL
Procedure or Function:	Procedure
Function Type:	---
Type Checking Defeated:	No

The JMCNVTO (convert to ASCII) system is used.

PIPRINT - PRINT STRUCTURE ADDRESSES

This is an initialization program that prints the names and addresses of commonly used system structures. PIPRINT is engaged by setting the global Boolean JXPRINT to true before entering initialization. The default value of JXPRINT is false. PIPRINT obtains addresses from a fixed area of core memory starting at location $150. Thus, all addresses available are present in the post-mortem dump. The format of the address area is given in table 4-4. Additional structure addresses can be easily added by:

1. Enter an EXT and an ADC for the new structure into the MACRO assembler program ADDRESSES.

2. In PIPRINT, increment the constant NUM by the number of new structure addresses added.

3. Add the first six letters of the structure names to the value statement in PIPRINT, being sure to add these statements in the same position as the ADC in ADDRESSES.

PIPRINT is an initialization program and, therefore, cannot be called by an on-line user.

PTCTCHR - COUNT CHARACTERS

This calling sequence is a function that returns the total number of characters in a mixed chain of data buffers. The calling sequence is:

```
PTCTCHR (parm)
```

where parm is a value parameter of type B0BOFPTR pointing to the first buffer in the chain to be counted. PTCTCHR is an integer function returning the number of characters in the buffer chain. Other user information is:

Program Type:	Noninterruptable
Language:	PASCAL
Procedure or Function:	Function
Function Type:	Integer
Type Checking Defeated:	No

PBSTRIP - STRIP EMPTY DATA BUFFERS

This calling sequence strips empty buffers from a mixed chain of data buffers. An empty data buffer is defined as a buffer with the LCD less than the FCD. Any empty buffers in the chain are released to the free buffer pool. The chain to the first nonempty buffer in the chain is returned. If all buffers in the chain are empty, nil is returned. The calling sequence is:

```
PBSTRIP (parm)
```

where parm is a variable parameter of type B0BUFPTR pointing to the

TABLE 4-4. PIPRINT ADDRESS AREA FORMAT

$150	0	BYWLCB	WORKLIST CONTROL BLOCK	BASE
	1	JSWLADDR	WL ENTRY BY LEVELNO	
	2	B1TCB	INTERNAL PROCESSING TCB	
	3	B1BUFF	INTERNAL PROCESSING BLOCK	
	4	JKMASK	INTERRUPT MASKS	
	5	JKTMASK	PBAMASK SAVE AREA	
	6	BGPLIST	OPS PROGRAM LIST	
	7	CBTIMTBL	TIMAL TABLE	
	8	JACT	PD CONTROLLER TABLE	
	9	BECTLBK	BUFFER CONTROL BLOCK	
	A	BEBSA	BUFFER STAMP AREA	
	B	CLBFSPACE	BFR SPACE IN NO. SMALL BFRS	
	C	BKPIKT	POI TABLE	
	D	J1UERRSTAT	UPLINE ERR-STAT ROUTING	
	E	J1USVM	UPLINE SERVICE MSG ROUTING	
	F	J1DSVM	DOWNLINE SERVICE MSG ROUTING	
	10	0		
	11	NAPORT	PORT TABLE	MUX SUBSYSTEM
	12	BQCIB	CIRCULAR INPUT BUFFER	
	13	NECCST	CLA CMD STATUS TABLE	
	14	MLSTABLE	CLA CURRENT STATUS TABLE	
	15	0		
	16	CGLCBS	LINE CONTROL BLOCKS	LINES - TIPS
	17	CHSUBLCB	SUB LCBS	
	18	BJTIPTYPT	TIP TYPE TABLE	
	19	NJTECT	TERMINAL CHARACTERISTICS TABLE	
	1A	0		
	1B	JF1SNPTBLE	SNAPSHOT CORE TABLE	DEBUG AIDS
	1C	JFWRAP	WRAP-SNAP TABLE	
	1D	J1STRT	START ADDR FOR PBDUMP	
	1E	J1QUICKPTR	BFR PTR FOR PBQUICKIO	
	1F	0		

first buffer of the chain to be checked for empty buffers. Other user information is:

Program Type:	OPS Level only
Language:	PASCAL
Procedure or Function:	Procedure
Function Type:	---
Type Checking Defeated:	No

PBCOPYBFRS - COPY A CHAIN OF BUFFERS

This calling sequence copies the data portion of a chain or part of a chain of any size buffers (all the same size or mixed) into data buffers. The input of PBCOPYBFRS is the first buffer of the chain to copy and a one-word packed record of type JTCOPYB containing:

1. The number of source buffers to copy. Zero causes PBCOPYBFRS to copy to nil source chain.
2. The source buffer size.
3. The destination buffer size (large or small data buffers only).
4. A Boolean indicating whether or not the source is a mixed data buffer chain. This parameter overrides number 2 above.
5. A Boolean specifying whether or not to release the source buffers after copying.

The calling sequence is:

```
PBCOPYBFRS (parm1, parm2)
```

where parm1 is a value parameter of type JTCOPYB as follows:

```
JTCOPYB = packed record
            JTNUM     : B08BITS;     {no. of buffers to copy}
            JTSSIZE   : B0BUFSIZES   {source buffer size}
            JTDSIZE,                 {dest. buffer size}
            JTSMIXED,                {mixed data buffer source chain}
            JTRLS     : BOOLEAN      {release source buffers}
          end;
```

Parm2 is a value parameter of type B0BUFPTR pointing to the first source buffer to copy. PBCOPYBFRS is a function that returns a buffer pointer to the first buffer of the copied buffer chain. Other information is:

Program Type:	OPS Level only
Language:	PASCAL
Procedure or Function:	Function
Function Type:	B0BUFPTR
Type Checking Defeated:	No

The following external subroutines are used by PBCOPYBFRS:

PBREL1BF	Release 1 Buffer
PBGET1BF	Get 1 Buffer

Structured flow is as follows:

```
function  PBCOPYBFRS;
begin
  if {source chain not nil} then
  begin
    {get first destination buffer}
    {set FCD in first destination
     buffer for Q-chaining}
    repeat
      if {source buffer not empty}
      then {copy source buffer}
      {chain to next source buffer}
      {decrement JTNUM}
    until {source buffer nil} v
          {JTNUM = 0}
  end;
end;
```

MISCELLANEOUS USER AIDS

Useful information is recorded in various locations of core as follows:

Location	Information
$10F	Build Release Number
$11F	Associated SCOPE build level (i.e., 410)
$12F	Source Cycle Number and Patch Level Indicator
$13F	Terminal Support Indicator (OF3, OF4, OF7)
$140	Jump to Post-Mortem Dump
$142	Jump to UTOPIA
$144	Jump to initialization of system (MAIN$)
$150	Table of commonly used structure addresses (see PIPRINT). The table name is ADDRESSES.

After downline loading of the system, execution begins at location 0. The Macro Assembler program BEGIN is linked at location 0 to set up PASCAL run time information in registers and transfer control to initialization. BEGIN loads registers R1 through R4 as follows:

- R1 Last word of dynamic stack area (DSTKLW)
- R2 Last word of dynamic variable area (DVARLW)
- R3 First word of dynamic stack area (DSTKFW)
- R4 First word of dynamic variable area (DVARFW)

DSTKLW, DVARLW, DSTKFW and DVARFW are initialized as entry points at Link Edit time. After these registers are loaded, BEGIN jumps to the start of system initialization (MAIN$).

PASCAL COMPILER SUBROUTINES

The PASCAL compiler can generate calls to four special subroutines:

QDEBUG	PASCAL Debug Option Handler
QENTRY	Recursive Procedure Entry Code
QEXIT	Recursive Procedure Exit Code
QULOCK	Noninterruptable Program Exit Code

These four routines are not available to the user but must be present in the system.

The PASCAL compiler generates calls to QDEBUG when errors are detected by code generated by turning on certain compile time PASCAL debug options. Refer to the PASCAL Reference Manual for details concerning these options. QDEBUG prints a message on the local console identifying the error:

```
*XXXX   Y   ZZZZ
```

where XXXX is the return address to the program calling QDEBUG (hexadecimal)

Y is the error number (decimal) (see table 4-5)

ZZZZ is the error parameter (hexadecimal)

QENTRY is the recursive procedure entry code that updates the dynamic stack pointers and saves off the necessary values when a recursive procedure is called.

QEXIT is the recursive procedure exit code that unstacks the latest entry in the dynamic stack when a recursive

TABLE 4-5. QDEBUG ERROR IDENTIFICATION

Error Number	Cause	Error Parameter
1	Assignment Out-of-Range	Out-of-Range Value
2	Array Index Out-of-Range	Out-of-Range Value
3	Divide by 0	---
4	Dynamic Variable Overflow	Next Available Address
5	Dynamic Stack Overflow	Next Available Address
6	Global Interrupt Count Negative	---

procedure exits (either normally or via a GOTO EXIT).

QULOCK is called upon exit from a noninterruptable PASCAL program. QULOCK performs an UNLOCK and returns control to the caller without destroying a function return value in the A register.

QUEUE SERVICES

The queue services subroutines provide for the first-in, first-out queuing of linked lists of data buffers (referred to as segments in the following). One or more of the four available queue services routines (PBPTlSEG, PBPTNSEG, PBGTlSEG, and PBGTNSEG) may be included in a point-of-interface (POI) sequence of subroutine calls.

Each queue is associated with a particular terminal control block (TCB) that contains either a pointer to the first segment in the queue or a pointer to a queue control block (QCB). The QCB contains a count of the number of segments in the queue and pointers to the first and last segments in the queue.

Figure 4-7 illustrates queue structure for both input and output queues. Each data buffer in a segment may be either of two different sizes. Segments are linked in the order in which they are to be removed from the queue. The chain address is contained in the last word of the data buffer and the last data buffer is identified by a NIL pointer in the chain address location. The first segment in the list (that which is not pointed to by any segment in the list) is the first segment to be removed from the queue.

PUT ONE SEGMENT IN QUEUE

To put a single segment into a queue, execute the following calling sequence:

```
PBPTLSEG (R3SEGPTR)
```

where R3SEGPTR contains a pointer to the segment to be placed in the queue and is of the type B0BUFPTR. B6OSERR is true if this segment is the first to be placed in queue. For parameter setup, BlTCB is a global variable that contains a pointer to the TCB associated with the queue.

PBPTlSEG is the name of the function, is of the type integer, and contains zero if the queue was busy and a segment was not placed in queue, or contains one if the queue was not busy and a segment was placed in queue.

In the foregoing, it is assumed that R3SEGPTR and BlTCB do not contain NIL pointers.

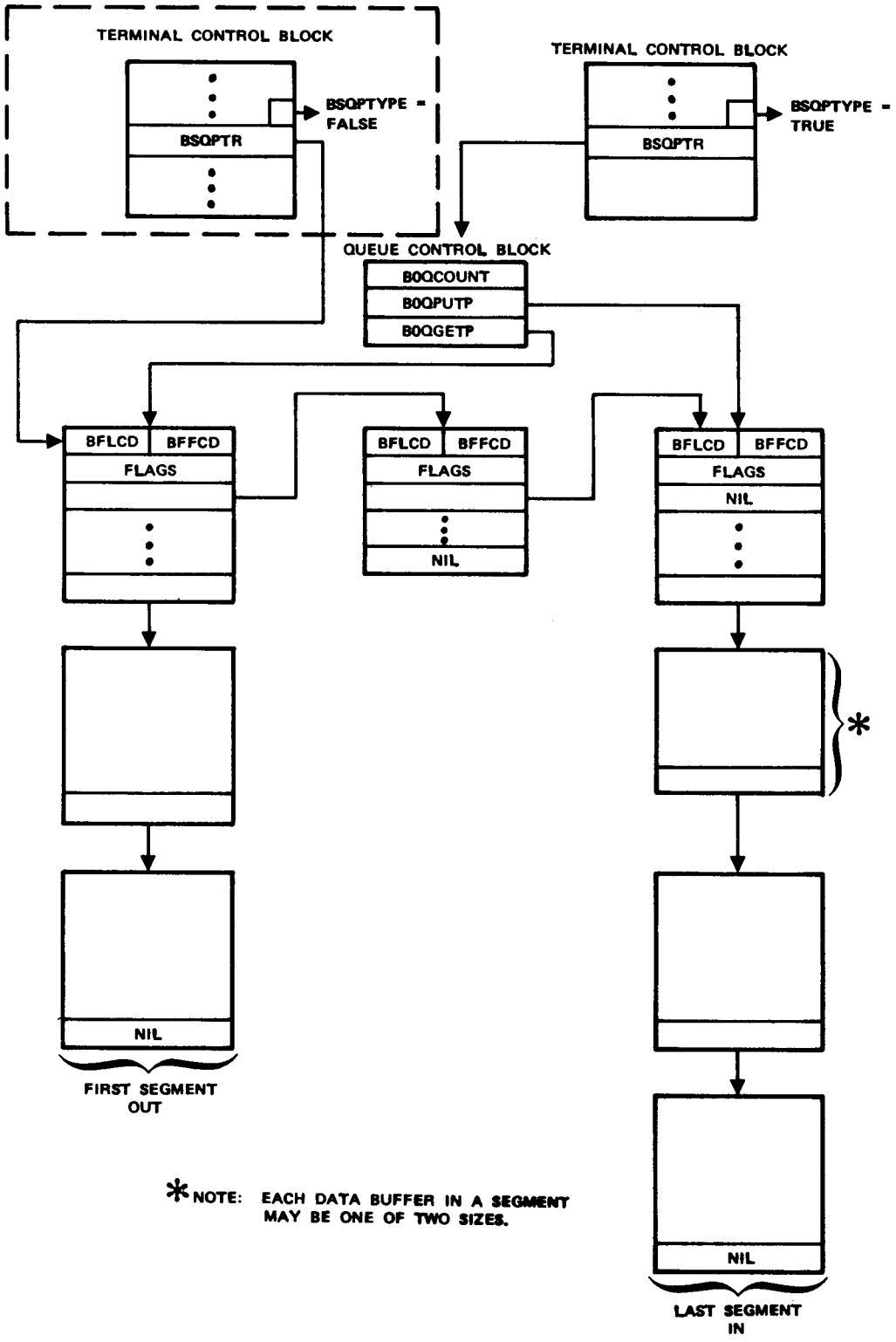


Figure 4-7. Structure of a Queue

GET ONE SEGMENT FROM QUEUE

The following calling sequence gets (removes) one segment from queue:

```
PBGT1SEG (R3SEGPTR)
```

where R3SEGPTR contains a pointer to the segment to be removed from queue and is of the type B0BUFPTR. For parameter setup, B1TCB is a global variable containing a pointer to the TCB associated with the queue.

PBGT1SEG is a PASCAL function which returns a value of type integer: zero if the queue was busy and no segment was removed from the queue; one if the queue was not busy, but was empty, and no segment was removed from the queue; and two if the queue was not busy and not empty and a segment was removed from the queue.

It is assumed in the foregoing that B1TCB does not contain a NIL pointer. It should further be noted that the chain word of a returned segment will not necessarily be NIL.

PROCESS DRIVER

The process driver subroutine selects application points-of-interface (POI). There are five such POIs, two related to the source of a transaction and three related to the destination. These POIs are post input (B1IPSIN), internal input (B1IPINT), internal output (B1PROQ), pre-output (B1IPROP), and post-output (B1IPSOP). The calling sequence for the process driver is as follows:

```
PBPOI (parm)
```

where parm is one of the five POI values given above.

Prior to calling the process driver, the user program (TIP or internal process) must set parameters in a globally defined area. Those parameters include:

B1TCB = Address of terminal control block (TCB)

B1BUFF = Address of first buffer

B1SEGS = Number of segments

B1PRI = Priority of queue from which selection is to be made (null = select highest priority available)

B1KEY = POI key (if zero, key is obtained from TCT)

The process driver is called with the appropriate POI key value between 1 and 5, extracted either from B1KEY or from the input or output TCB. Except for the POI key, all parameters are passed by priming global variables to allow a common interface to the process driver, to facilitate passing of parameters to the POI sequences, and to provide debug assistance. The inputs for the various sequences are shown in table 4-6.

Figure 4-8 illustrates process driver system relationships. This procedure operates only at the OPS level.

CONSOLE SERVICES

For certain applications, a local console is used as a communications supervisory position. Three console functions can be selectively activated or deactivated by the console operator (or at build time). These functions are: communications supervisor, orderwire, and diagnostics. When one or more of these functions are transferred to a remote console, the corresponding functions must be deactivated at the local console.

The communications supervisor (COMSUP) function is employed for input of console control messages. The orderwire function is employed for both input and output traffic messages. The diagnostic function is used for output of hardware diagnostic messages.

TABLE 4-6. PROCESS DRIVER SEQUENCE INPUTS

	TCB Address	Segment Address	Queue Priority	Number of Segments
Post Input	X	X		
Internal Input	X	X		
Internal Output	X	X		
Pre-Output	X		X	
Post Output	X	X		X

CONSOLE WORKLIST ENTRY

A type B0CHWL worklist entry is made by the internal process output procedure for every message placed in an empty console queue. Such entry contains the console TCB address.

CONSOLE CONTROL MESSAGES

All console control messages begin with a slash (/) and end with an end-of-transmission code (D). Table 4-7 contains console control messages and the results of each.

TEXT PROCESSOR

The TIP programmer is responsible for preparing all output in line-compatible form before dispatching it via the multiplex subsystem. Preparing such output may require code conversion, enveloping, CRC or LRC calculation, and other protocol-defined conventions. Since this requirement can be very time consuming, text processing is accomplished by firmware.

The text processor (TP) provides the basic features needed to perform common text processing functions. For specialized protocols, the facility to execute user-written state programs is also available within the TP to alter text processing on a character-by-character basis. The microprogram decodes and executes the state program instructions.

The basic features of the text processor are:

1. CRC/LRC accumulation
2. Code conversion
3. Exception character recognition
4. Character insertion
5. Character counting

The following options are also available for text preparation:

1. Overwrite source text during processing or create a new text string in destination buffers.
2. Select release of source buffers after processing into destination buffers.
3. Select the size (large or small) of the destination data buffers.

The text processor accepts input chains of mixed data buffers. Empty buffers (LCD less than FCD) are allowed anywhere in the source chain. No empty destination buffers are generated.

The user can write any of four different state program types:

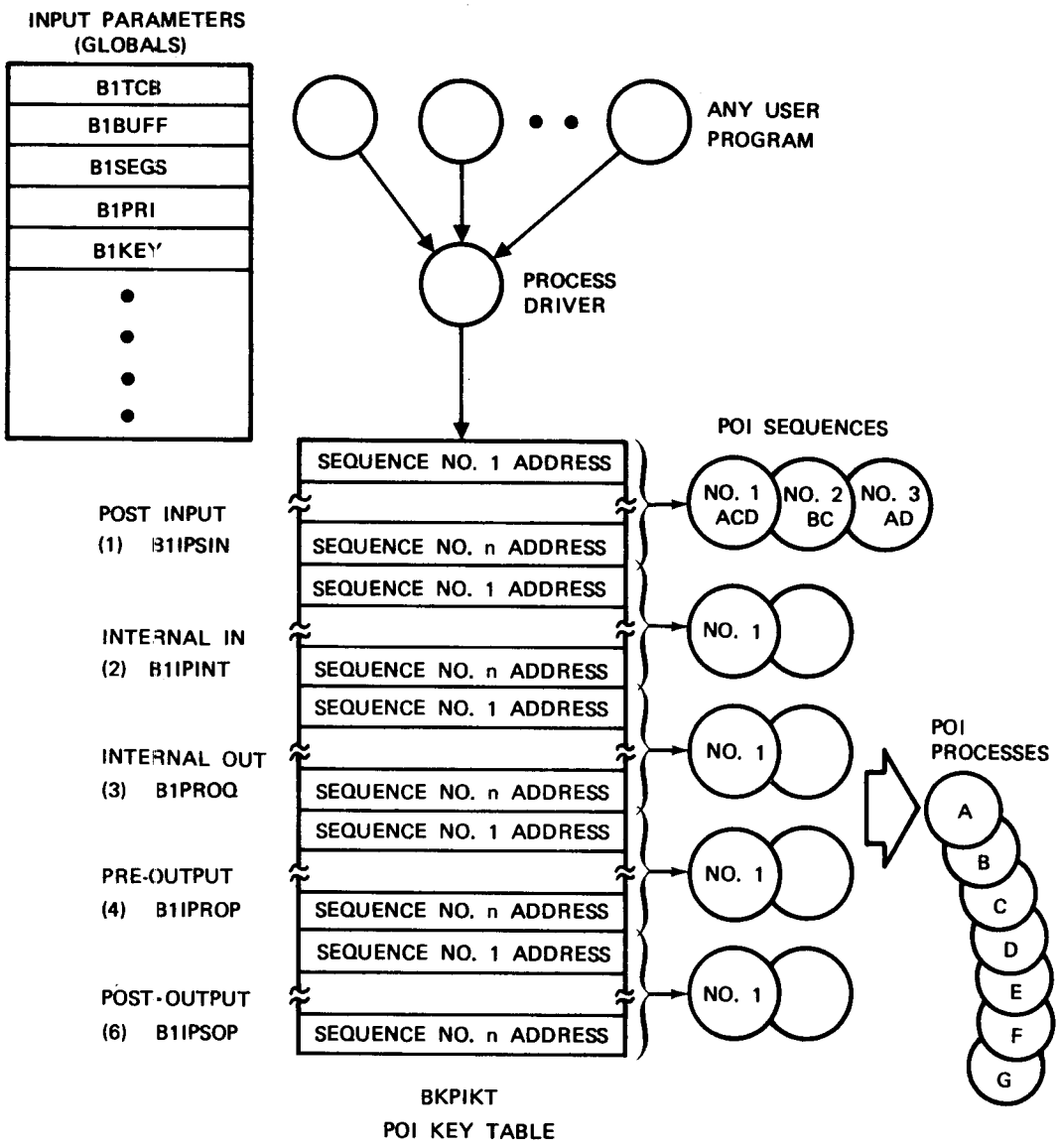


Figure 4-8. Process Driver System Relationship

TABLE 4-7. CONSOLE CONTROL MESSAGES

Message	Results
/SUP	Selects COMSUP function
/ORD	Selects ORDERWIRE function
/DIA	Selects DIAGNOSTIC function
	NOTE Only one of the above functions can be active at any given time. The last function selected before the operator goes to write mode is the active function.
/QIS	Current function printed
/ACT aaa*	Function activated
/DEA aaa*	Function deactivated
/REQ	Interrupted output message requeued
/CAN	Interrupted output message cancelled
/CPR	Printer and paper tape reader connected to controller (hardware echo)
/DPR	Printer and paper tape reader disconnected from controller (no hardware echo - for non-ASCII input)

*aaa = SUP, ORD, or DIA

1. Header build
2. Trailer build
3. Exception character processing
4. Processing when a character count limit is reached

Header and trailer build programs are always performed by the text processor. State programs can be nested.

The state program instructions are the building blocks for more complex protocol logic. The currently defined state program instructions are:

1. Load TP command word
2. Process next source character
3. Process character given in instruction
4. Process character from user packet
5. Bump to next source character
6. Process left 8 bits of CRC
7. Process right 8 bits of CRC
8. Load TP parameter
9. Save TP parameter
10. Exit TP
11. Continue TP

12. Jump relative

Parameters are passed to the microcode TP through the File 1 registers. These registers are accessible to the user state programs via the load and save TP parameter state instructions (table 4-8).

MACRO TEXT PROCESSOR

The interface between the user and the firmware text processor is the PASCAL program PBTP which sets a few of the TP parameters in the File 1 registers, calls the firmware TP, and handles returns from the firmware TP to enable macro interrupts and to obtain destination buffers. The calling sequence of the macro text processor is:

```
PBTP (parm)
```

where parm is a variable parameter of type JPTPACKET, the text processor parameter packet (table 4-9).

MICROPROGRAM TEXT PROCESSOR

The microprogram TP contains an internal counter that is decremented for each character processed. The initial value set into the counter is a build-time parameter loaded into File 1 register 1B during initialization. As the firmware TP executes, macro and IDP interrupts are inhibited. Therefore, the initial value is selected to control the length of time such interrupts are to be inhibited. When the counter becomes zero, the firmware returns to PBTP to enable macro and IDP interrupts tc occur. PBTP does nothing but return to the firmware TP to continue processing.

The end-of-buffer check for both source and destination buffers is performed after the last character has been processed and stored, but before the next character is obtained, to save processing time.

When an exception character is encountered, control is transferred to the designated exception state program if the state program address is greater than $FF (hexadecimal). If less, the state program address is returned to the user as a result code.

When the character count equals the character count limit, the character counter is cleared and control is transferred to the character count limit state program if the state program address is greater than $FF (hexadecimal). Otherwise, the state program address is returned to the user as a result code.

TEXT PROCESSOR DATA STRUCTURES

The text processor parameter packet (table 4-9) contains nine words, specified as words 0 through 8. Words 1 through 5 are parameters that may be initialized by the user. Words 6 and 7, the header and trailer build state program addresses, must be provided by the user. Words 0 and 8, respectively the user return code and destination buffer address, are parameters that return values from the text processor to the user.

Other parameters that the user can set up (TP command word, exception table address, CRC/LRC initial value, etc.) should be set in the header build state program using the load TP parameter state instruction. The user parameter packet structure is open to the user and, typically, contains characters for insertion, character counts, TP parameter load values, TP parameter save areas, and state program addresses. The source buffer address is assumed by PBTP to be in the global variable B1BUFF.

The TP command word controls the character counting, CRC calculation, code translation, and exception character processing functions. The format of the TP command word is as follows:

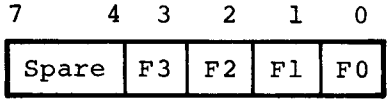


TABLE 4-8. TEXT PROCESSOR FILE 1 REGISTER ASSIGNMENTS

File 1 Register	User Mnemonic	Description
D0	JOUSERPKT	User parameter packet address
D1	JOHBSPA	Header build state program address
D2	JOTBSPA	Trailer build state program address
D3	JODBFR	Destination buffer address
D4	JODLFCD	Destination LCD, FCD
D5	JOSBFR	Source buffer address
D6	JOSLFCD	Source LCD, FCD
D7	JOSLNTH	Source buffer length-1
D8	JOCRCWD	CRC polynomial
D9	JOCRC	CRC result
DA	JOCHRLMT	Character count limit
DB	JOCHRCNT	Character count
DC	JOCMDWD	TP command word
DD	JOEXCP	Exception table address
DE	JOXLATE	Code translate table address
DF	JOE1SPA	Exception 1 state program address
EO	JOE2SPA	Exception 2 state program address
E1	JOE3SPA	Exception 3 state program address
E2	JOCLSPA	Character count limit state program address
E3	JOCSPA	Current state program address
E5	------	Internal character count
E6	JORTNSPA	Return state program address
1B	------	Internal character count limit

TABLE 4-9. TEXT PROCESSOR PARAMETER PACKET

JPTPCODE	0	User Return Code	Integer
JPBRLS	1	Source Buffer Release Flag	Boolean
JPSRCETOSRCE	2	Source to Source Flag	Boolean
JPDBSZE	3	Destination Buffer Size (large or small)	Boolean
JPCRCWD	4	CRC/LRC Polynomial (see figure 4-11)	Integer
JPUSERPRT	5	User Parameter Packet Address	Integer
JPHDR	6	Header Build State Program Address	Integer
JPTRLR	7	Trailer Build State Program Address	Integer
JPDBFR	8	Destination Buffer Address	B0BUFPTR

where:

F0 = Exception character processing

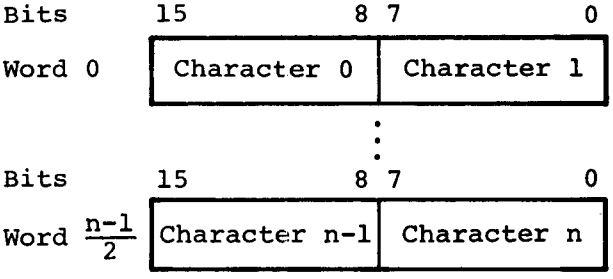
F1 = Character counting

F2 = Code translation

F3 = CRC calculation

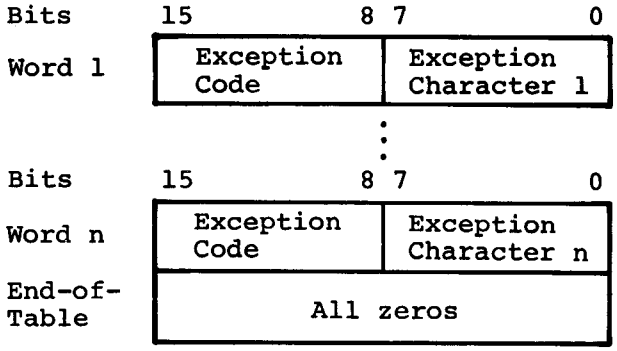
To enable one or more of the functions, set the desired bit or bits. These bits are processed from right to left. If the code translate/exception character bits are set, but the translate table/exception table addresses are zero, no code translation/exception character processing will be done.

The code translation table format is as follows:



The character to translate is used as the index into the code translation table.

The format of an exception character table is as follows:



The exception codes are JOE1SPA, JOE2SPA, and JOE3SPA (table 4-8). Thus, each exception table can contain any number of characters. Encountering any such character in the source text results in control being transferred to one of three exception state programs specified by the exception code.

RETURNING TO THE USER

Any state program address less than $100 (hexadecimal) causes control to be returned to the user. Values 3-$FF in the state program address are returned to the user in JPTPCODE in the TP parameter packet. Return codes 0-2 are reserved. Code 0 means that text processing is com-

plete (end-of-text or exit TP state program instruction encountered). Return codes 1 and 2 are used internally by PBTP. This enables the user to gain control to perform logic the text processor cannot perform. To return to the firmware TP without calling PBTP, the user must employ the following code:

```
INST ($C400,JOFRTN,   {LDQ return}
     $BA2);           {EMS Q      }
```

Control is returned to the firmware routine that last called the main firmware TP sequence.

CRC/LRC POLYNOMIALS

Table 4-10 illustrates the polynomials used to generate CRC/LRC.

DEBUG AIDS

There are four types of debug aids available:

1. Test Utility Program (TUP)
2. Traps
3. Maintenance/Programmer Panel Interface
4. Post-Mortem Dump

Each is separately described in the following paragraphs. Figure 4-9 is an overview diagram of the debug aids system.

TEST UTILITY PROGRAM (TUP)

TUP is an interactive, machine-language-oriented debug aid that allows the user to perform a wide variety of functions. Table 4-11 lists the available TUP commands. Rules for operating TUP from the communications console include:

1. Maximum of eight parameters (each consisting of one to four hexadecimal characters) per TUP command.
2. Control Ⓐ to enter TUP mode, control Ⓓ to leave TUP mode.

TABLE 4-10. CRC POLYNOMIALS

Polynomial	Initialization Value	Polynomial Value
$X^{16} + X^{15} + X^2 + 1$	0	0
$X^{16} + X^{12} + X^5 + 1$	0	1
$X^{12} + X^{11} + X^3 + X^2 + X + 1$	0	2
$X^6 + X^5 + 1$	0	3
$(X^8 + 1)$	0/$FF00 (See note)	4
$X^8 + X^7 + X^6 + X + 1$	0	5
$X^8 + X^7 + X^6 + X + 1$	0	6

NOTE

Used also to generate LRC polynomials. 0 for even LRC, $FF00 for odd LRC.

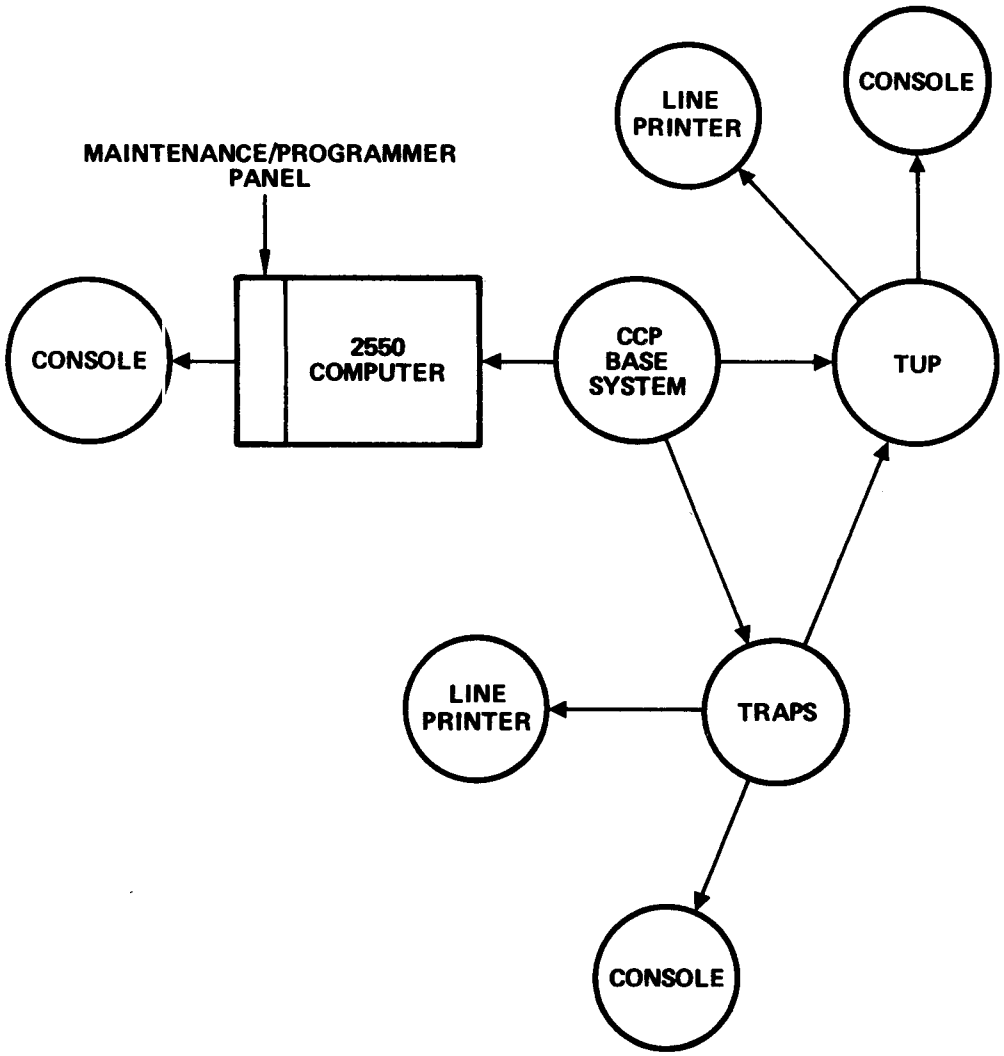


Figure 4-9. Overview Diagram of the Debug Aids System

TABLE 4-11. TUP COMMANDS

Command	Mode*	Syntax
System Halt	NH	SH/
System Restart	H	SR/
Dump Hex	U	{DPC / DPL}, start addr, stop addr, base addr/
Load Hex	U	LHX, start addr, base addr/C word 1,...word 8/
Enter Halt	U	EH, halt loc 1,...halt loc 4/
Halt Restart	H	RS/
Display Registers	H	DR/
Load Registers	H	E{R}, load value/ R=1,2,3,4,Q,A,I,or M
Display File 1	U	DF, file 1 register (0..$FF)/
Load File 1	U	EF, file 1 register (0..$FF), load value/
Get a Buffer	U	BG, buffer size (0..3)/
Release a Buffer	U	BR, buffer addr, buffer size (0..3)/
Get a Worklist Entry	U	LG, WL number/
Put a Worklist Entry	U	LP, WL number, word 1,...word 6/
Make BP Table Entry	U	EB, start addr, stop addr, breakpoint code/
Remove BP Table Entry	U	RB, start addr, stop addr, breakpoint code/
Enable Software BP	U	BL, priority level (0..$11)/
Disable Software BP	U	DL, priority level (0..$11)/
Device Assignment	U	DA,LIO,PD/

*H = halt mode only
NH = not in halt mode
U = unrestricted

3. Slash (/) is TUP end-of-message delimiter.

4. Question mark (?) or control © cancels TUP input message.

5. Manual interrupt cancels TUP output.

6. Every TUP input message has response (minimum of carriage return and line feed).

7. Commas and blanks are interchangeable TUP parameter delimiters.

8. Backspace (represented by ← or underline) and CONTROL-H erase previous characters.

9. *ERR prints if input is invalid (e.g., bad command, nonhex parameters, etc.).

In the following paragraphs that describe the available TUP command, the terms restricted mode and halt mode are synonymous and both refer to the condition in which the OPS level is locked out. When the characters XXXX are shown in the input format, they indicate that zero to four hexadecimal characters can be entered. XXXX in the output response indicates that four hexadecimal characters are printed. Where the symbol Δ is shown in the input format, either a blank space or a comma may be entered. In the response formats, commas and spaces are normally printed as shown.

SYSTEM HALT

The system halt (SH) command places the system in the restricted mode with the OPS level locked out. Certain TUP commands can only be executed while the system is in the restricted mode (see table 4-5).

Format: SH/
Response: *SYS HLT
(or *ERR SYS HLT if already in restricted mode)

SYSTEM RESTART

System restart (SR) returns the system to the unrestricted mode (OPS level) after a system halt.

Format: SR/

Response: *
(or *ERR SYS HLT if not in restricted mode)

LOAD HEX

The load hex (LHX) command sets up the load address for subsequent C commands to use in loading hexadecimal information from the local console to the processor core memory. The formats associated with the LHX and subsequent C commands are as shown in figure 4-10.

The base address value in the LHX command is optional and is used for relative addressing.

The C command can load from 1 to 8 words in core memory. For each word loaded, the load address is incremented. Thus, multiple C commands load contiguous core locations. Any other TUP command can be executed between C commands without destroying the load address. A new load address is set by executing a subsequent LHX command.

The response to the C command is the previous contents of the word locations being loaded with new words. If an attempt is made to load an out-of-range location, dashes are printed following the last word successfully loaded.

DUMP HEX

Two commands (DPC and DPL) are provided to control dumping of hexadecimal information. DPC dumps the core memory contents to the local console. DPL dumps the core memory contents to the assigned TUP dump device. The DPC and DPL command and

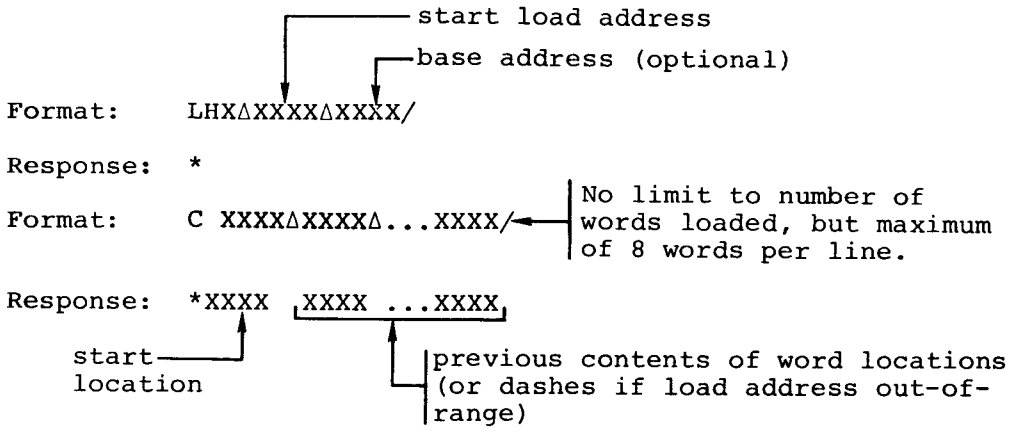


Figure 4-10. C Command and Response Format

response formats are similar and are as shown in figure 4-11.

In either the DPC or DPL command, if only a start address is specified (no stop address), a single word is dumped from core. If the user attempts to dump from an out-of-range address, the following error response is printed:

Response: *ERR

ENTER HALT

The enter halt (EH) command patches a 2-word return jump into the TUP breakpoint handler at a specified location. The command establishes memory address traps that cause the system to enter the halt (restricted) mode. See figure 4-12.

The return jump is a 2-word macro assembler instruction. The halt address contents printed in response, therefore, are also two words. If the halt address attempts to access an out-of-range address, dashes are printed as a response. When the return jump is executed, the TUP breakpoint procedure is entered and the following response is printed before the system is placed in the halt (restricted) mode:

Response: *HLT XXXX ← halt location

RESTART FROM HALT

The restart from halt (RS) command returns control to the system from the TUP breakpoint handler at the instruction after the return jump patched in by the enter halt (EH) command or from a software break breakpoint:

Format: RS/

Response: *
(or *ERR SYS HLT if the user attempts to restart from the system halt (SH) command)

DISPLAY REGISTERS

The display registers (DR) command causes the contents of registers R1, R2, R3, R4, Q, A, I, and M to be displayed in the sequence stated. Valid information is displayed only when the system is in the halt (restricted) mode.

Format: DR/

Response: *1 = XXXX 2 = XXXX
3 = XXXX ...M = XXXX

LOAD REGISTER

The load register command enters a specified value into a specified

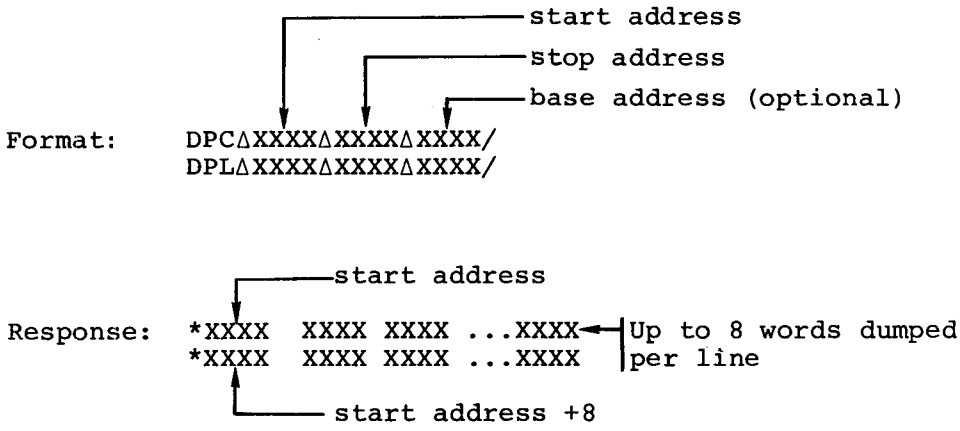


Figure 4-11. DPC or DPL Command and Response Format

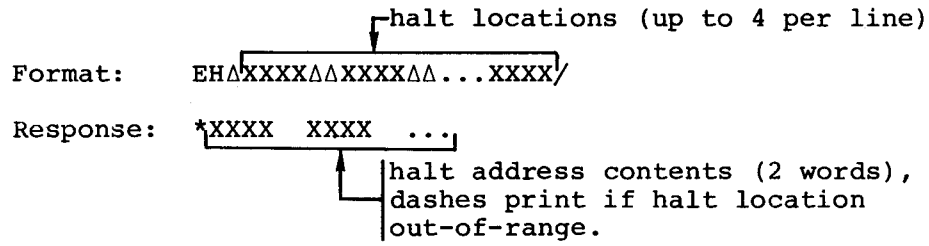
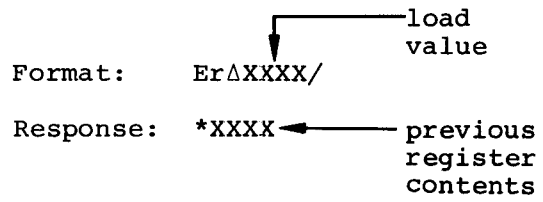


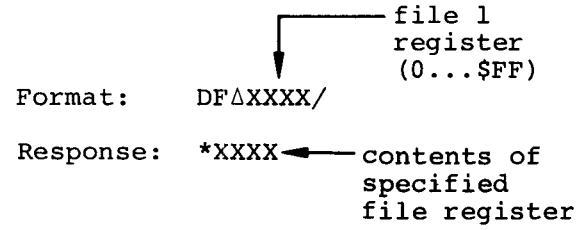
Figure 4-12. Enter Halt Command and Response Format

register. The response is the old register contents.



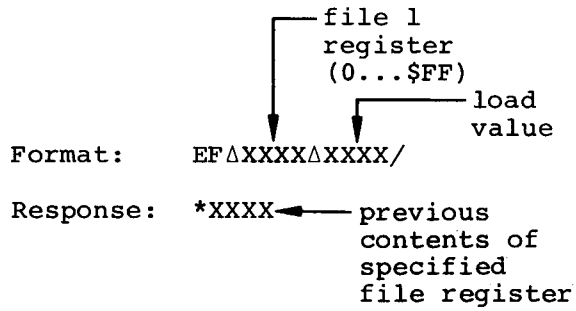
where r equals 1, 2, 3, 4, Q, A, I, or M to specify the register to be loaded.

DISPLAY FILE 1



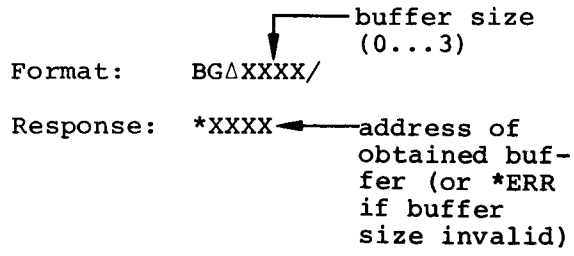
An error response prints if the user attempts to display an invalid file 1 register.

LOAD FILE 1

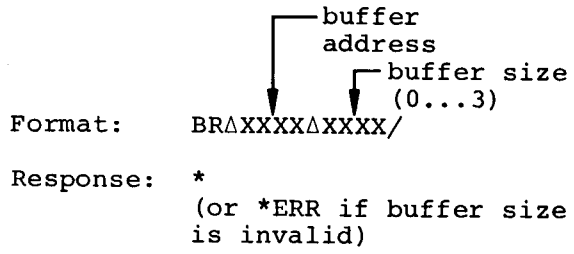


An error response prints if the user attempts to load an invalid file 1 register.

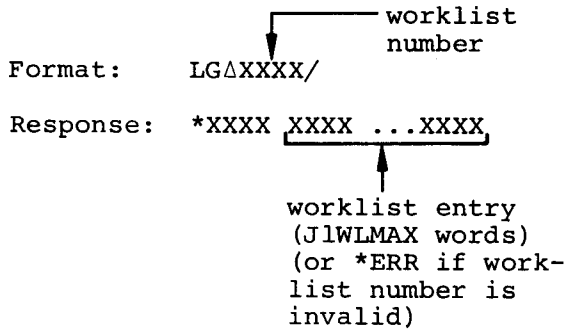
GET A BUFFER



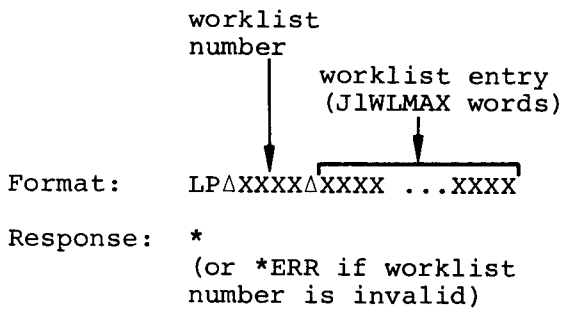
RELEASE A BUFFER



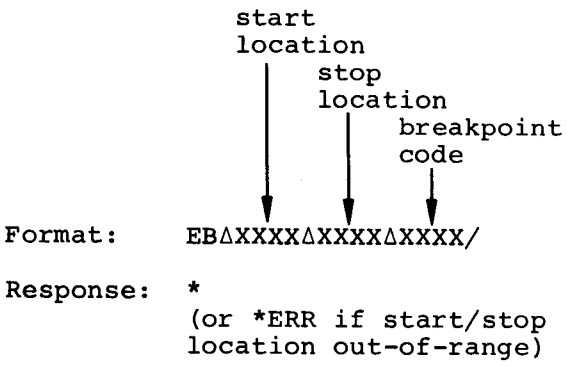
GET A WORKLIST ENTRY



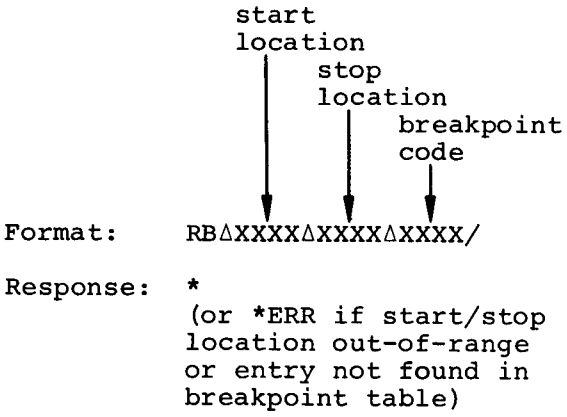
PUT A WORKLIST ENTRY



PLACE ENTRY INTO BREAKPOINT TABLE

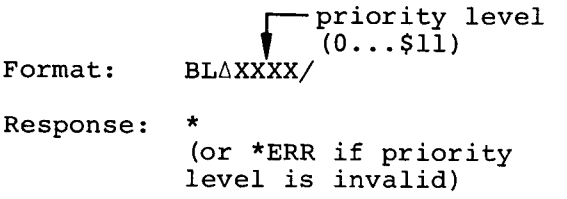


REMOVE ENTRY FROM BREAKPOINT TABLE

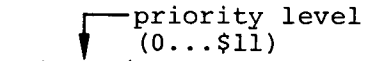


ENABLE SOFTWARE BREAKPOINT

This command allows software breakpoints to occur on a specified software priority level to allow re-entrant code to be breakpointed at desired priority levels only.



DISABLE SOFTWARE BREAKPOINT

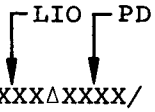


Format: DLΔXXXX/

Response: *
(or *ERR if priority level is invalid)

DEVICE ASSIGNMENT

The device assigrment (DA) command allows the user to dynamically assign logical input/output (LIO) to physical devices (PD).



Format: DAΔXXXXΔXXXX/

Response: *
(or *ERR if either LIO or PD is out-of-range)

The available PD codes are:

0 = null device
1 = local console

The available LIO codes are:

1 = supervisory output
8 = TUP dump to assigned device
9 = snapshot core
$A = snapshot registers
$B = online dump
$D = quick output

TRAPS

Software traps (breakpoints) are created by generating program protect faults using the maintenance/programmer panel program protect system. At initialization, all of core memory is protected except for the buffer and global areas. When the enter breakpoint (EB) TUP command is issued, the specified instruction is left unprotected. When the protected instruction following the unprotected instruction is executed, a program protect fault occurs. The fault instruction executes as a selective stop and a level 1 interrupt is generated. The interrupt handler then passes control to the desired debug routine if the interrupt occurred at a software priority level for which software breakpoints have been enabled.

The following general information concerning breakpoints should be noted:

1. Software breakpoints are machine-language-oriented, not PASCAL-oriented.

2. Routines in which interrupts are locked out cannot be breakpointed.

3. The following instruction types cannot be breakpointed:

   a. Instructions that write into non-buffer or non-global memory.

   b. Jump, return jump, or skip instructions.

   c. Instructions that are privileged (EIN, IIN, SPB, CPB, and inter-register instructions with destination register M).

4. Two consecutive instructions cannot be breakpointed.

5. The instruction length must be entered with the EB instruction. For example, to breakpoint the 3-word instruction at locations $100, $101, and $102, the EB format is EB, 100, 102, {BP code} /.

TRAP PROCEDURE ENTRY

To enter a trap procedure from a software priority level via a software breakpoint, perform the following steps:

1. Using the enter breakpoint (EB) TUP command, make an entry in the breakpoint table.

2. To enable the trap to occur, use the enable breakpoint (BL) TUP command.

3. Leave the TUP mode (optional).

4. Set the console to write mode (optional).

5. Switch on the program protect system.

TRAP PROCEDURE DISABLE

To turn off (disable) all breakpoints, switch off the program protect system. To turn off all breakpoints on specifined priority levels, use the DL TU TUP command. To remove an entry from the breakpoint table, use the RB TUP command.

AVAILABLE TRAPS

The currently available traps are listed in table 4-12.

The trap calling sequences are as follows:

```
PBTUPBREAKPT

PB1SNAP (parm)

PB2SNAP (parm)

PB3SNAP (parm)
```

where parm is a variable parameter of type JFSNAPTABLE.

```
PBDUMP (parm1, parm2, parm3)
```

where parm1 and parm2 contain the start dump and stop dump addresses, respectively, and parm3 is the output device mnemonic (type integer).

```
PBQUICKIO (parm1, parm2)
```

where parm1 is the output device mnemonic (type integer) and parm2 is the pointer to the output buffer (type B0BUFPTR).

```
PBWRAPSNAP (parm1, parm2)
```

where parm1 is a variable parameter of type JFSNAPTABLE and parm2 is the index into the snapshot table JFWRAP.

MAINTENANCE/PROGRAMMER PANEL INTERFACE

The maintenance/programmer panel interface accepts seven different control characters. These are: H, I, J, K, L, colon (:), and question mark (?). H through L identify the type of data or operation entered or returned, the colon terminates all entries except master clear, and the question mark accompanied by correct parity generates a master clear to the computer, memory, and peripheral devices. There is no response to the master clear entry.

TABLE 4-12. AVAILABLE TRAP LISTING

Trap Name	Breakpoint Code	Procedure Called	Queues Output	Available as System Debug Aid
TUP	7	PBTUPBREAKPT	No	Yes
Snapshot Core	9	PB1SNAP	Yes	Yes
Snapshot Registers	$A	PB2SNAP	Yes	Yes
Snapshot Return Address	$F	PB3SNAP	Yes	Yes
On-Line Dump	$B	PBDUMP	No	Yes
Quick I/O	$D	PBQUICKIO	No	Yes
Wrap Around Snap	$E	PBWRAPSNAP	No	Yes

A normal entry consists of one control character H through L; zero, two, four, or eight hexadecimal digits (0 through F), and colon. A normal response consists of one control character that identifies the data that follows, four or eight hexadecimal digits, and colon. If a transmission or operator error occurred on entry, the control character is replaced by an asterisk (*) and the function control register (see table 4-13) is displayed. All entries except question mark cause a response unless bit 16 of the function control register is set.

CONTROL CHARACTER H FUNCTIONS

Control character H clears the function control register (FCR) bit position specified by the two hexadecimal digits following the control character. For example, H1B clears hexadecimal position 1B (decimal position 27) of the FCR, thereby resetting the Interrupt System Active bit.

See also Stop/Go functions.

CONTROL CHARACTER I FUNCTIONS

Control character I functions identical to control character H except that the specified bit position is set rather than cleared.

See also Stop/Go functions.

CONTROL CHARACTER J FUNCTIONS

Control character J replaces the contents of specified digit positions in the FCR. While it may be used to change the value of any FCR digit (0-7), it is generally used to change display digits 0 and 1. The value of display digits 0 and 1 specify the MP17 parameter to be displayed or entered (see table 4-14).

J functions generally consist of the control character J, followed by two hexadecimal digits, followed by a colon. The first hexadecimal digit specifies the FCR digit (0-7) and the second specifies the value that digit is to assume (0-F). Thus, to set display digit 1 to specify the A register (register 4), enter:

```
J14:
```

The J code is also used to alternately display the upper and lower 16-bit portions of a 32-bit register. For example, entry of:

```
J:
```

complements the upper/lower (U/L) indicator on the maintenance panel and causes the opposite halt of the specified register to be displayed.

CONTROL CHARACTER K FUNCTIONS

Control character K is used to either display or enter data values into the facility specified by display digit 1. Therefore, before using this control function, the value of display digit 1 must be properly established. Thereafter, enter K followed by a colon to display the value or enter K followed by four or eight hexadecimal digits that are to be entered, followed by a colon.

For example, to display the P register, enter the following sequence:

```
J11:  (Sets display digit 1 to
       value 1 to specify P
       register)

K:    (Causes display of
       facility specified
       by display digit 1)
```

To enter the hexadecimal value 14FE into the breakpoint register, enter the following sequence:

```
J16:  (Sets display digit 1 to
       value 6 to specify BP
       register)
```

TABLE 4-13. FUNCTION CONTROL REGISTER (FCR)

Bit		Digit	Bit Definition	
31	1F	(LSB)	Overflow Status	Status Only
30	1E	7	Protected Instruction Status	
29	1D		Protect Fault Status	
28	1C		Parity Error Status	
27	1B	6	Interrupt System Active Status	
26	1A		Auto-Restart Enabled Status	
25	19		Micro Running Status	
24	18		Macro Running Status	
23	17	5		
22	16			
21	15		Auto Display	
20	14		Enable Console Echo	
19	13	4	Enable Micromemory Write	
18	12		Multilevel Ind Add Mode	
17	11			
16	10		Suppress Console Transmit	
15	0F	3		00=BP off 01=Inst ref BP 10=Store op BP 11=All ref BP (including READ operand)
14	0E			
13	0D		BP Int (BP Stop if Clr)	
12	0C		Micro BP, Step, Go, Stop (Macro if Clr)	
11	0B	2	Step	
10	0A		Selective Stop	
09	09		Selective Skip	
08	08		Protect Switch	
07	07	1	Display 1	
06	06			
05	05			
04	04			
03	03	0	Display 0	
02	02			
01	01			
00	00	(MSB)		

TABLE 4-14. DISPLAY CODE DEFINITIONS

Code					Display 1	Display 0
0	0	0	0	0	Function Control Register	F2 (File 2)
1	0	0	0	1	P Register (IAC)	N Register
2	0	0	1	0	I Register (current inst.)	K Register
3	0	0	1	1		X Register
4	0	1	0	0	A Register	Q Register
5	0	1	0	1	Micro-Instruction Register	F Register
6	0	1	1	0	Breakpoint	F1 (File 1)
7	0	1	1	1	P-MA (Micro Page Address)	Main Memory
8	1	0	0	0	SM1 Status/Mode 1	
9	1	0	0	1	M1 (Interrupt Mask)	Micro Return Jump Register
A	1	0	1	0	SM2 Status/Mode 2	
B	1	0	1	1	M2 (Interrupt Mask)	
C	1	1	0	0		Micromemory
D	1	1	0	1	A*	
E	1	1	1	0	X*	
F	1	1	1	1	Q*	

K14FE: (Enters 14FE value into facility specified by display digit 1)

CONTROL CHARACTER L FUNCTIONS

The function of control character L is identical to the function of control character K except that it is associated with display digit 0, rather than display digit 1.

Note that when main memory is displayed or entered, the display digit 1 selection specifies the main memory address to be displayed or entered and, therefore, display digit 1 must be set for either the P register (1) or A register (4). After display or entry, the register specified by display digit 1 is incremented by 1.

When micromemory is displayed or entered, the K register is the least significant eight bits of the address and the N register provides the remaining bits. After display or entry, the K register is incremented by 1.

STOP/GO FUNCTIONS

Control characters H and I provide stop/go control functions when used without following hexadecimal characters. Control character H provides the Go function and control character I provides the Stop function. If bit 12 (decimal) of the FCR is set, a micro stop/go function is specified. If bit 12 is not set, a macro stop/go function is specified. The response to a start/stop function is display of the FCR.

MASTER CLEAR FUNCTION

A master clear can be generated in any of several different ways:

1. Entry of a question mark (?) character from the console.

2. Pressing the MC (master clear) button on the maintenance panel.

3. Pressing the autoload button.

4. A signal from a peripheral controller.

5. By the power-on master clear.

BREAKPOINT FUNCTIONS

There are two types of breakpoints, macro and micro. If bit 12 of the FCR is clear, macro breakpoint (BP) is selected. If bit 12 is set, micro BP is selected.

For macro BP, bits 14 and 15 of the FCR are used together to select one of four macro BP modes:

00 = Breakpoint off

01 = Instruction reference BP

10 = Store operand BP

1 = All references BP (includes READ operand)

A macro BP occurs if the breakpoint register is equal to the main memory address and the select conditions are met. If bit 13 (decimal) of the FCR is set, an interrupt (rather than a stop) occurs when the breakpoint conditions are met. Currently, there is no software supporting a hardware breakpoint interrupt.

For micro BP, the micro page address (P/MA) is compared to the lower 12 bits of the breakpoint register and the upper/lower (U/L) indicator is compared to bit 13 of the breakpoint register. If all bits are equal and the combination of FCR bits 14 and 15 is not zero, a micro stop occurs.

MULTIPLEX SUBSYSTEM INTERFACES 5

INTRODUCTION

The multiplex subsystem contains the hardware, microprograms, and software elements necessary to provide data and control paths for information interchange between the various protocol handlers in the communications system software and the many communications lines. The design of the subsystem is based on the "multiplex loop" concept which is a demand-driven mechanism for gathering input data and status from the communications lines and distributing output data and control information to the communications lines on a real-time basis. Figure 5-1 depicts the basic elements of the multiplex subsystem.

Line oriented input and output buffers provide temporary storage for data in the multiplex subsystem. The subsystem also provides special table-driven and dynamically controlled processing of characters received from communications lines. Generally, the subsystem processes data on a character-by-character basis while the user programs (e.g., terminal interface programs) process data on a message or block basis. Circuit, modem, and subsystem status is detected and transferred to the user programs in the form of work demands. Control information is received from the user programs in the form of commands and these commands are decoded and executed by one or more of the subsystem elements.

HARDWARE COMPONENTS

The multiplex subsystem, as illustrated in figure 5-1, includes the multiplex loop interface adapter (MLIA), loop multiplexers, and communications line adapters (CLAs).

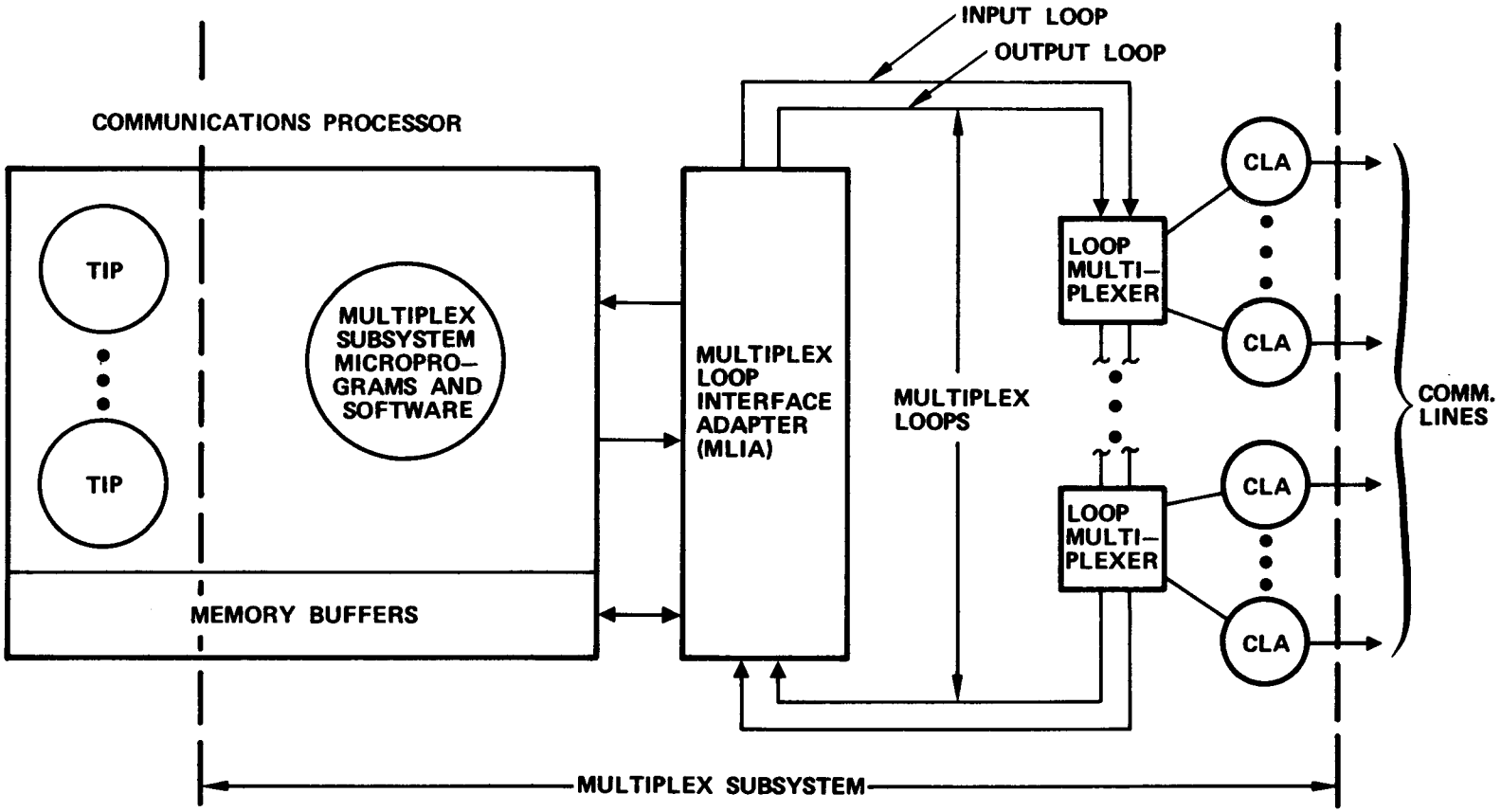
MULTIPLEX LOOP INTERFACE ADAPTER (MLIA)

The MLIA provides hardware interface between the multiplex input/output loops and the multiplex subsystem software. Major functions provided by the MLIA are:

1. Management of the input/output loops.
2. Input data buffering to compensate for the difference in rate at which characters are removed from the input loops and the rate at which they are stored in the main memory.
3. Output data demand (ODD) detection and buffering.
4. Multiplex loop error detection.
5. Generation of interrupts for the multiplex subsystem microprograms and software for functions such as:
   a. Output data demand received.
   b. Line frame received.
   c. Loop error conditions.

LOOP MULTIPLEXERS

The loop multiplexers provide interface between a group of as many as 32 CLAs and the demand-driven multiplex loop. Its primary function is to receive parallel data from the CLAs and present it to the serial input loop in the loop cell format. Conversely,



CLA – COMMUNICATIONS LINE ADAPTER
TIP – TERMINAL INTERFACE PROGRAM

Figure 5-1. Basic Elements of the Multiplex Subsystem

it assembles serial data in the loop cell format from the output loop and presents it to the CLAs in parallel form.

COMMUNICATIONS LINE ADAPTERS

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

SUBSYSTEM INTERFACES

User interfaces to the multiplex subsystem can be divided into three categories:

1. Command Driver Interface - Command communication to the multiplex subsystem to control data flow to and from the communications lines.

2. TIP Subroutines - Techniques employed by the multiplex subsystem to communicate input events to the user.

3. State Program Tables - Techniques employed by the user to direct the multiplex subsystem to provide specific input processing functions on a character-by-character basis via the STATE program tables.

COMMAND DRIVER INTERFACE

The command driver calling sequence from the OPS level is:

```
PBCOIN (parm)
```

and the command driver calling sequence from level 2 is:

```
PMCDRV (parm)
```

where parm is the name of the request packet. The general format of a request packet is as follows:

Bits	15	0
WORD 0	Command	Parameter
WORD 1	Line Number	
WORD 2	Parameters	
WORD 3	Parameters	
WORD 4	Parameters	

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

```
NKCLRL    - Clear Line
NKINIL    - Initialize Line
NKCONTR   - Control
NKENBL    - Enable Line
NKINPT    - Input
NKOUTPUT  - Output
NKENDIN   - Terminate Input
NKENDOUT  - Terminate Output
NKDISL    - Disable Line
```

CLEAR LINE COMMAND

The clear line command causes the subsystem to clear (reset) all line-oriented software and hardware (CLA) functions associated with the line specified by the line number. The command format is as follows:

Bits	15 8	7 0
WORD 0	NKCMD	
WORD 1	NKLINO	
WORD 2		NKLTYP

where:

```
NKCMD   =  Command code (NKCLRL)
NKLINO  =  Line number (identifies port and subport)
```

NKLTYP = Line type. Specifies line-type table entry. Defines the physical characteristics of the port, modem, and circuit type.

INITIALIZE LINE COMMAND

The initialize line command establishes the line type of the specified port and places the line in a mode in which the subsystem monitors and processes modem and circuit related status. Other line related functions (e.g., processing of input and output characters) are inhibited while the line is in the initialize mode. The command format is as follows:

Bits	15 8	7 0
WORD 0	NKCMD	
WORD 1	NKLINO	
WORD 2		NKLTYP

where:

NKCMD = Command Code (NKINIL)
NKLINO = Line number
NKLTYP = Line type. Specifies the line type table entry.

CONTROL COMMAND

The control command serves a twofold purpose. It may define the character transmission characteristics of a given line according to the transmission characteristics key (NKTCKY) for input/output signaling rate, character length, parity type, stop bit duration, and sync character. The command may also specify up to five modem/circuit control functions such as: echo, break, terminal busy, resync, etc. Such control functions are specified in the optional fields of the command packet.

Generally, the command is used to initialize or alter the character transmission characteristics of the line or to generate circuit control functions. This command must not be issued prior to the initialize command. The command format is as follows, with optional modem/circuit functions as defined in table 5-1.

Bits	15 14 8	7 6 0		
WORD 0	NKCMD		NKTCKY	
WORD 1	NKLINO			
WORD 2	F1	NKFUN1	F2	NKFUN2
WORD 3	F3	NKFUN3	F4	NKFUN4
WORD 4	F5	NKFUN5	ZEROS	

where:

NKCMD = Command code (NKCONTR)
NKTCKY = Optional character transmission key. If non-zero, references the character transmission characteristics table to specify the input/output speed, character length, parity, stop bit duration, and sync character.
NKLINO = Line number
NKFUN1 = Optional modem/circuit function.
thru If F1-F5 equals one, the function is set (turned on).
NKFUN5 If F1-F5 equals zero, the function is reset (turned off).

ZEROS = Delimit end of options and must be placed in the byte following the last requested modem/circuit function. A maximum of five functions may be specified.

TABLE 5-1. OPTIONAL MODEM/CIRCUIT FUNCTIONS

Function Mnemonic	Function Provided	Description
NOISR	STATUS*	CLA Status Request
NORTS	RTS	Request to Send
NOSRTS	SRTS	Secondary Request to Send (Supervisory Channel)
NOOM	OM	Originate Mode/Auxiliary Modem Control
NOLM	LM	Local Mode/Auxiliary Modem Control
NODTR	DTR	Data Terminal Ready
NOTB	TB	Terminal Busy (line busy out)
NORSYN	RSYN*	Resynchronize
NONSYN	NSYN	New Sync
NOBREAK	BREAK	Send Break
NODLM	DLM*	Data Line Monitor
NOECHO	ECHO	Echoplex Mode
NOLBT	LBT	Loop Back Test
NOPARY	Even/Odd Parity	Select even or odd parity**
NOPI	Parity Inhibit	Disable parity option

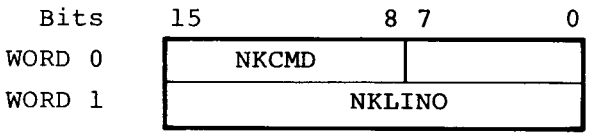
*Pulsed functions. Provide momentary signal and need not be reset.

**If reset flag is set in request packet (F1-F5), even parity is selected. If reset, odd parity is selected.

NOTE: Reference the applicable CLA manuals for descriptions of the above functions.

ENABLE LINE COMMAND

This command directs the subsystem to activate, as a function of line type, the necessary modem signals to allow the local modem to connect to the specified communications line. The command also conditions the subsystem to monitor and analyze any changes in the modem status for signals indicating that a "line connect" has occurred. As a further function, this command dynamically allocates memory space for the line control block (LCB). Character processing functions are inhibited during the time the line is in the enable mode. The format for the enable line command is as follows:



where:

NKCMD = Command code (NKENBL)
NKLINO = Line number

INPUT COMMAND

This command directs the subsystem to initiate the processing of data on the specified input line (i.e., turn on the input side of the CLA). The processing functions provided by the subsystem are determined by the following parameters:

1. Terminal Type (NKTTYP) - Indexes a terminal characteristics table that defines the terminal characteristics in terms of the code set, character transmission characteristics, CRC polynomial type, packet size, etc. The subsystem initializes the line control block (LCB) with the above parameters.

2. Initial Input Processing State (NKISTAI) - Indexes a state process to define the initial input processing functions provided by the subsystem.

3. Special Character (NKSCHR) - Optional field. If called in a state instruction, directs the subsystem to compare incoming characters against a special character. Instruction may be included in any state process.

4. Block Length (NKBLKL) - Optional field. The TIP may override the maximum block length specified in the terminal characteristics table with the new block length value in this field. The subsystem stores the block length count in the character count 2 field of the LCB. The TIP may then define a state process to decrement character count 2 when characters are received and to provide processing functions when the count reaches zero.

5. Buffer Options - The subsystem allows the TIP to specify either of two different data buffer sizes for each terminal type. Another option directs the subsystem to store the first character of a new data block (or packet) into a new data buffer at the address specified by the first character displacement (FCD). These options are implemented as follows:

   a. If WORD 3 of the command packet contains a buffer pointer (NKIBP), the subsystem derives the FCD from WORD 0 of the data buffer. The buffer size defined in this first data buffer specifies the size of subsequent data buffers.

   b. If WORD 3 of the command packet contains a first character displacement (NKIFCD), the subsystem obtains a buffer from the available buffer pool and uses NKIFCD as a displacement to store the first character. Before obtain-

ing the buffer, the subsystem selects the buffer size according to two parameters stored in the terminal characteristics table. Parameter 1 (packet size) has an implied buffer size (established at program build time) or, if parameter 1 is zero, the subsystem selects the buffer size from parameter 2 (buffer size).

Once executed, an input command remains in effect until the subsystem receives either a terminate or disable command. The character transmission characteristics must be defined by the control command before the input command is issued. The format for the input command is as follows:

Bits	15 12	8 7	0
WORD 0	NKCMD	NKTTYP	
WORD 1	NKLINO		
WORD 2	NKSCHR	NKISTAI	
WORD 3	NKIBP or NKIFCD		
WORD 4	F1 F2 F3	NKBLKL	

where:

NKCMD = Command code (NKINPT)

NKTTYP = Index to terminal characteristics table

NKLINO = Line number

NKSCHR = Special character (optional)

NKISTAI = Initial input processing state. Index to initial state process executed when first character is received.

NKIBP or NKIFCD = Input buffer pointer or first character displacement. If NKIBP, contains address of first input buffer. If NKIFCD, the subsystem obtains the next available buffer from the pool and stores the first character received in the byte specified by NKIFCD. If bits 0-15 are zero, the subsystem obtains the next available buffer from the pool and uses the NKIFCD stored in the terminal characteristics table as displacement for the first data character.

F1 = Code Translation Required Flag (NFNOXL). One equals translate, zero equals no translation.

F2 = Parity Strip Flag (NKRPRT). One causes high order bit of 8-bit character (7 bits plus parity) to be stripped, zero causes no stripping.

F3 = Input after output flag (NKINOUT). Must be set by HDX protocols which must operate with full duplex lines.

NKBLKL = Maximum Block Length (optional). An entry in this field overrides the block length specified in the terminal characteristics table.

OUTPUT COMMAND

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

The output buffer pointer and FCD in the first word of the output buffer specifies the first character to be output. An option in the command packet (NKOFCD) allows the TIP to specify the first output character regardless of the FCD in the output buffer.

NOTE

Before selecting the NKOFCD option, the TIP must ascertain that the output line is idle. The command driver extends control to the system error routine if this option is selected while an output message is in process.

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

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

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

Receipt of an output command for a line on which an output message is in process causes the subsystem to chain the new message to the last buffer of the message currently being output. If the possibility exists that the new message may be chained to the current message, the user must prime NKLBPMP.

The format of the output command is as follows:

Bits	15 — 8	7 — 0
WORD 0	NKCMD	
WORD 1	NKLINO	
WORD 2	NKOBP	
WORD 3	NKLBPMP	
WORD 4	F1	NKOFCD

where:

NKCMD = Command code (NKOUTPUT)

NKLINO = Line number

NKOPB = Output buffer pointer

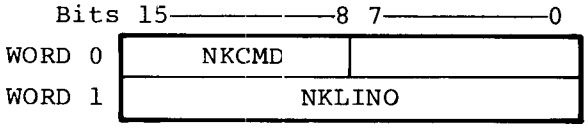
NKLBPMP = Address of last buffer of previous message. Chains new message to last buffer of the message being output.

F1 = End of Transmission Flag (NKENDX). On controlled carrier lines, directs the subsystem to turn the line around after the last buffer of the block is transmitted.

NKOFCD = Optional first character displacement.

TERMINATE INPUT COMMAND

Enables the TIP program to direct the subsystem to immediately terminate input processing functions on the specified line. Input characters received after execution of this command are discarded. The TIP program may, by issuing an input command, direct the subsystem to resume input on the line. Transmission line characteristics are not altered by the terminate command and, therefore, the TIP need not generate a control command.



where:

NKCMD = Command code (NKENDIN)

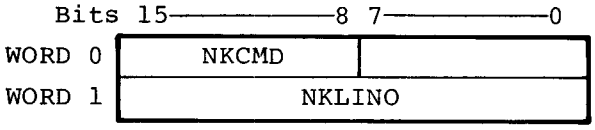
NKLINO = Line number

After processing the terminate input command, the subsystem generates a worklist entry containing the first and last buffer addresses of a message, assuming a message was in the process of being input.

TERMINATE OUTPUT COMMAND

This command enables the TIP program to direct the subsystem to immediately terminate output processing functions on the specified line. After processing the terminate command, a worklist entry is generated containing the memory address of the current output buffer and the position of the last character transmitted to allow the TIP to resume output of an interrupted message at the point of interrupt. This command is used when the TIP wishes to interrupt an outgoing message for a higher priority message or because of an abnormal line condition.

The format of the terminate output command is as follows:



where:

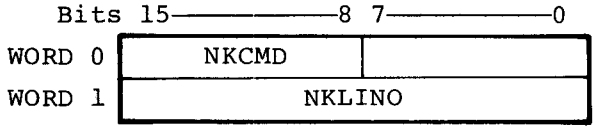
NKCMD = Command code (NKENDOUT)

NKLINO = Line number

DISABLE LINE COMMAND

This command directs the subsystem to terminate all processing functions of the specified line with the exception of monitoring changes in modem status. Proper modem control signals are generated to inhibit further communication exchange between the local modem and the communications line. The subsystem also releases all data structures defining the character processing functions for the line. Character transmission characteristics are maintained. To reactivate, an enable command followed by either an input or output command must be issued.

Format for the disable line command is as follows:



where:

NKCMD = Command code (NKDISL)

NKLINO = Line number

TERMINAL INTERFACE PROGRAM (TIP) SUBROUTINES

The common TIP subroutines module consists of two major subroutines (PTWAIT and PTTER) supported by a number of control (PBCONT) subroutines. These routines provide the interface between the multiplex

subsystem and the TIP handlers, removing from the TIP the necessity of handling all possible multiplex-generated event work lists. This is accomplished through a program control block (PCB) defined as part of each base line control block (LCB).

As event worklists are received by the multiplex level handler (PTTER), the worklist code is translated into a 1-bit flag whose position within the PCB defines the event. A corresponding user event word specifies under which events the user requires control. Therefore, when correspondence exists between the user word and the event word, the user receives control.

The supporting subroutines allow the user to selectively manipulate fields in the PCB.

PTWAIT – TIP EVENT WAIT

This subroutine allows the user to specify any combination of 16 pre-defined events occurring at the multiplex level for which control will be returned to the user if the specified correspondence occurs. The user can also specify the length of time the system will wait for such an occurrence and the level at which control will return when either the events or timeout occurs.

Table 5-2 lists and defines PTWAIT field names. The calling sequence for this subroutine is as follows:

```
PTWAIT (parm1, parm2, parm3)
```

where parm1 specifies the level at which control will return (HLOPS for OPS level or HLMUX for multiplex level), parm2 is the user event mask of type WTSET, and parm3 indicates the length of wait in half-second increments (type B08BITS).

PTTER – TIP EVENT PROCESSOR

This subroutine is called by the work list processor to process TIP-related events. The routine translates the work code defining the event into one of 16 possible event flags, then checks to determine if the event is one for which the TIP is waiting. If such is the case, the routine prepares to return to the specified level. If not, the event is noted and the routine exits back to the work list processor. The calling sequence for this subroutine is:

```
PTTER
```

The routine requires a multiplex level work list entry containing the work code, line number, and work code dependent parameters. It further requires that global word SBLEVEL be set to HLMUX prior to the call of PTTER and reset to HLOPS at return from PTTER.

PROGRAM CONTROL BLOCK (PCB) DEFINITION

The program control block (PCB) is included within each line control block (LCB) as words 3 through E (hexadecimal). The PCB format is as illustrated in figure 5-2 with word definitions as detailed in tables 5-3, 5-4, and 5-5.

SUPPORTING TIP SUBROUTINES

The following is a list of the supporting control subroutines (PBCONT) associated with PTWAIT and PTTER:

Routine	Action
PTSTWM (P : WTSET);	Inclusive OR P with BZWAITMASK
PTRSWM (P : WTSET);	Selective Clear BZWAITMASK by P

WORD	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	NAME	TYPE
3	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15	F16	BZWSTAT	WTSET
4	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15	F16	BZWAITMASK	WTSET
5	F17	F18	F19	F20	F21	F22	F23	F24	F25	F26	F27	F28	F29	F30	F31	F32	BZERRORS	INTEGER
6																	BZRETADDR	HTERROR
7	F33	F34	F35	F36	F37	F38	F39	F40	F41								BZLINSTA	HTLINSTA
8	F42								F43								BZWTCOUNT	B08BITS
9																	BZERAC	HTERROR
A																	BZR1SAVE	INTEGER
B																	BZR2SAVE	INTEGER
C																	BZITBA	B0BUFPTR
D																	BZOTBA	B0BUFPTR
E																	BZ1FBA	B0BUFPTR

Figure 5-2. Program Control Block (PCB) Format

TABLE 5-2. PTWAIT LINE CONTROL BLOCK FIELD NAMES AND DEFINITIONS

Word	Field	Set Name	Definition
3	F1	HWWK8	Work code 8 detected by PTTER
	F2	HWWK7	Work code 7 detected by PTTER
	F3	HWWK6	Work code 6 detected by PTTER
	F4	HWWK5	Work code 5 detected by PTTER
	F5	HWWK4	Work code 4 detected by PTTER
	F6	HWWK3	Work code 3 detected by PTTER
	F7	HWWK2	Work code 2 detected by PTTER
	F8	HWWK1	Work code 1 detected by PTTER
	F9	HWSTAT	CLA status detected by PTTER
	F10	HWOTERM	Output terminated detected
	F11	HWITERM	Input terminated detected
	F12	HWOBT	Output buffer transmitted
	F13	HWRING	Ring indicator detected
	F14	HWCON	Connect indicator detected
	F15	HWRFO	Ready for output detected
	F16	HWRFI	Ready for input detected
4	Identical to Word 3 - User requests wait mask for specified event		

Routine	Action
PTSTSW (P : WTSET);	Inclusive OR P with BZWSTAT
PTRSSW (P : WTSET);	Selective Clear BZWSTAT by P
PTSTLS (P : PCSET);	Inclusive OR P with BZLINSTA
PTRSLS (P : PCSET);	Selective Clear BZLINSTA by P
PTTERM;	Set flush, clear wait and timer run bits
PTCLRPCB;	Clear BZWSTAT, BZWAITMASK, BZERRORS, BZLINSTA (left byte) BZRETADDR, BZERAC, BZIFBA, BZITBA, and BZOTBA

Routine	Action
PTGOOPS;	This routine immediately returns control to the TIP except that execution continues at the OPS level.
PTRETOPS;	TIP must call this routine (when at OPS level) at the completion of its tasks to properly return control to the calling routine.
PTRETMUX;	Same as PTRETOPS except that this routine is only called to relinquish control at the MUX level.

TABLE 5-3. PROGRAM CONTROL BLOCK (PCB) WORD DEFINITIONS

Word	Field	Set By	Reset By	Description
3	F1-F16	PTTER	TIP	Event Status Word
4	F1-F16	PTWAIT	PTWAIT	User Wait Mask
5	F17-F32	PTTER	PTWAIT	Errors Detected Since Last PTWAIT
6		PTWAIT	PTWAIT	User's Return Address
7	F33	PTWAIT	PTWAIT	Timer Active
	F34	PTWAIT	PTWAIT	Return Level
	F35	PTWAIT	PTWAIT	Wait Outstanding
	F36			-Spare-
	F37			-Spare-
	F38	TIP	TIP	Input Active
	F39	TIP	TIP	Output Active
	F40	TIP	TIP	Line Active
	F41	PTCLAS	PTCLAS	CLA Status Analyzer Control Field
8	F42	PTWAIT	PTWAIT	Count of the No. of times PTWAIT is Called
	F43	PTCLAS	PTCLAS	CLA Status Analyzer Control Field
9		PTTER	TIP	Accumulated Error Event History
A		PTWAIT	PTWAIT	User's Register 1 at Time of PTWAIT Call
B		PTWAIT	PTWAIT	User's Register 2 at Time of PTWAIT Call
C		PTTER	TIP	Buffer Address Received when Input Began
D		PTTER	TIP	Buffer Address Received when Input Ended
E		PTTER	TIP	Buffer Address Received when Output Ended

Subroutines PTGOOPS, PTRETOPS, and PTRETMUX are further described in the following paragraphs as is the subroutine PBCONTINUE which is also associated with PTWAIT and PTTER.

PTGOOPS

This subroutine, available only at the MUX level, allows the caller to continue execution at the OPS level. The user must then refer-

TABLE 5-4. ERROR FIELD (WORDS) NAMES AND DEFINITIONS

Field	Set Name	Boolean Name	Definition
F17	HE200	HB200MSTO	200 Millisecond timeout detected
F18	HEDCDNOT	HBDCDNOT	Data Carrier Detected not on
F19	HEDTO	HBDTO	Data transfer overrun
F20	HEBUFTNR	HBBUFTHR	Buffer threshold exceeded
F21	HEFES	HBFES	Framing error status detected
F22	HEMRTO	HBMRTO	Modem response timeout
F23	HEMXLOOP	HEMXLOOP	MUX loop error detected
F24	HECLASTOV	HBCLASTOV	CLA Status overflow
F25	HEDSRNOT	HBDSRNOT	Data Set Ready not on
F26	HEUNSOLIO	HBUNSOLIO	Unsolicited input/output detected
F27	HEODDTO	HBODDTO	ODD Timeout
F28	HETIMEOUT	HBTIMEOUT	Event timeout detected
F29	HESDSRNOT	HBSDSRNOT	Switched-line Data Set Ready not on
F30	HEBREAK	HBBREAK	Break character detected
F31	HENCNA	HBNCNA	Next character not available
F32	HESPI	HBSPI	Spare

Field Name	Fields	Definition	Type
HIERROR	F17-31	All errors	B0 OVERLAY
HISFTINP	F17-21	Soft input errors	B05 BITS
HIHARDER	F22-29	Hard errors	B08 BITS
HISFTOUT	F30-32	Soft output errors	B03 BITS
HIINPERR	F17-29	All input errors	B013 BITS
HIOUTPERR	F22-32	All output errors	B011 BITS

ence level-dependent parameters at the OPS level as control has been returned to the instruction following the procedure call at the OPS level.

The user calls PTGOOPS at the MUX level and the routine builds a continue worklist, saving in the worklist the following information:

1. The current call count (BZWTCOUNT).
2. The line number (HALINO [HLMUX]).
3. The address of the instruction following the procedure call.
4. The error status (HAERR[HLMUX]).

The worklist entry is then placed in the continue worklist processor's queue. When the entry is processed, the caller receives control at the OPS level.

TABLE 5-5. LINE STATUS FIELD (WORD 7) NAMES AND DEFINITIONS

Field	Set Name	Boolean Name	Definition
F33	HSTRUN	HTTRUN	Timer active
F34	HSLEVEL	HTLEVEL	Return level: 0 = OPS, 1 = MUX
F35	HSWAIT	HTWAIT	Wait outstanding
F36	HSSP9	HTSP9	Spare
F37	HSFLSH	HTFLSH	Not used
F38	HSIACT	HTIACT	Input active
F39	HSOACT	HTOACT	Output active
F40	HSLACT	HTLACT	Line active

PTRETOPS

As a TIP may receive control from a variety of sources (continue worklists, timeouts, direct monitor call, etc.), PTRETOPS provides a means by which the TIP will return control to its caller. PTRETOPS also preserves the continuity of control flow. For example, if the TIP receives control as a result of a timeout, the timeout program has extended control. Consequently, to properly relinguish control back to the timeout program, the TIP must call PTRETOPS to prevent return of control to the monitor and bypassing of the remainder of the timeout program functions.

PTRETOPS may only be called at the OPS level. Prior to extending control to the TIP, the continuation address SRETOPS is primed by the program extending control. The value of SRETOPS depends upon the program extending control. SRETOPS is used by PTRETOPS to extend control back to the caller of the TIP to preserve continuity.

PTRETMUX

PTRETMUX is functionally identical to PTRETOPS, except that continuity is preserved at the MUX level.

PTRETMUX may only be called by the TIP at the MUX level and uses SRETMUX to preserve continuity. The TIP at the MUX level must call PTRETMUX whenever the TIP wishes to relinguish control.

PBCONTINUE

This procedure receives control from the monitor program as a result of a continue worklist entry in its queue. PBCONTINUE provides interface between MUX-related events and the TIP at the OPS level. As events are detected at the MUX level by PTTER, control may be returned to the TIP at the OPS level through the enqueuing of the continue worklist entry for PBCONTINUE.

Continue worklist entries are processed as follows:

1. The line number is used to set up array entries HALCBP [HLOPS] and HALINO[HLOPS].

2. The error field is used to set up array HAERR[HLOPS].

3. The return address specifies the address at which the TIP will receive control.

4. The count determines if control should be returned. The count in the worklist must match the count in the PCB or the worklist entry will not be processed and the TIP will not receive control.

The continue worklist entry format is as follows:

Word			
0	F1	F2	F3
1	F4		
2	F5		
3	F6		

Word	Field	Field Name	Definition
0	F1	MMWTCOUNT	Call count in PCB at time entry was queued
	F2	MMBAV	Buffer Address Valid flag. Always zero
	F3	MMWKCOD	Worklist work code
1	F4	MMLINO	Line Number
2	F5	MMRETADDR	TIP return address
3	F6	MMWTERR	Error status at time entry was queued

GLOBAL INTERFACES

HALCBP [HLOPS, HLMUX] of BZLCBP

This array contains pointers to the current LCB. PTWAIT accesses the appropriate entry depending upon the current level in order to perform the required wait functions.

HALINO [HLOPS, HLMUX] of BZLINO

In similar fashion as above, the array contains the line number of the current line.

HAERR [HLOPS, HLMUX] of HTERROR

The user may reference an entry in this array upon return from PTWAIT to determine the errors for the line since the previous call to PTWAIT.

SBLEVEL

This parameter contains the values HLOPS or HLMUX. It is value initialized to HLOPS but must be set to HLMUX prior to call of PTTER by the Worklist Processor. It is used by the supporting subroutines to access the proper entry in HALCBP.

SRETOPS

The variable contains the address of the instruction in the monitor following the monitor's call to the TIP.

SRETMUX

This variable contains the MUX level address to which control will return at the MUX level when the TIP has completed its functions. It is set to cause a return to the Worklist Processor.

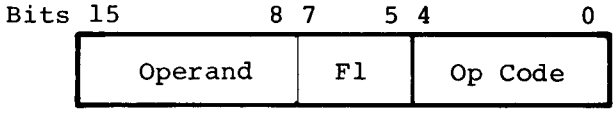
S5TEMP

This variable is used by the TIP subroutine PBCONTINUE and PBLCBT to pass parameters to subroutine SRET, a routine to set up the required return parameters to the TIP at the OPS level.

STATE PROGRAM TABLES

A state program consists of main memory tables controlling input processing functions for a specified line based upon the current input state (field ISTAI in the MUX LCB) and the current input character. Such a program may consist of one or more state processes, with each state process containing one or more state process instructions. The number of state processes and state process instructions is a function of the processing requirements for the particular terminal protocol. Figure 5-3 is an overview of the state program.

A state process instruction contains an operand, function (Fl) code, and operation (Op) code in the following format:



where:

Operand - Contains parameters used during execution of the Op code.

Fl - Specifies one of the following actions:

1. If Fl is zero, execute next state process instruction.
2. On skip instructions, skip-relative to the current state process instruction, with Fl specifying the relative skip count.
3. Exit and relinquish control to the input data processor microprogram at one of seven entry points specified by Fl as follows:

Entry Point	Fl Value	Entry Point Function
1	001	Discard character
2	010	Store character
3	011	Accumulate CRC and store character*
4	100	Accumulate CRC and discard character
5	101	Undefined
6	110	Undefined
7	111	Undefined

*CRC accumulation is performed on the untranslated character. The character stored is always the translated character if such translation is required.

Op Code - Defines specific processing task, such as replace character, set state counter, build worklist entry. Also defines interpretation of Fl field.

STATE PROCESS INSTRUCTIONS

Subject to expansion as new requirements become evident during implementation of the multiplex subsystem, the following process instructions represent a list of those instructions now defined:

1. Set/Reset Input Message-In-Process Flag
2. Replace Character
3. Build Event Worklist
4. Terminate Input Buffer
5. Skip If CRCS Equal
6. Decrement Character Count
7. Initialize Character Count

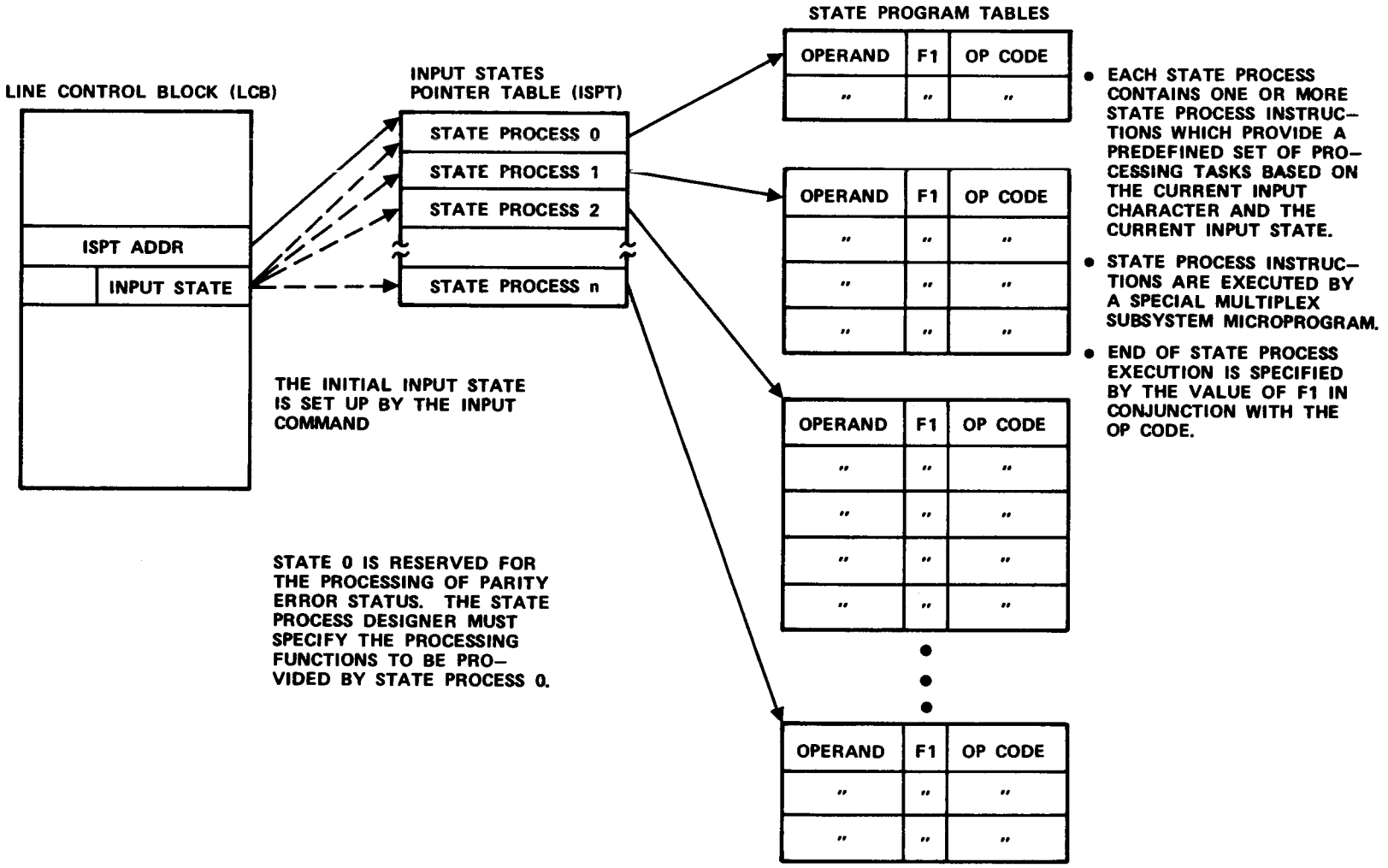
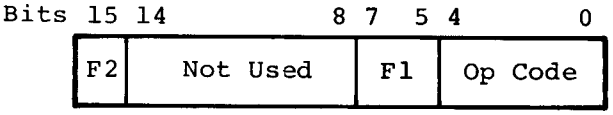


Figure 5-3. State Program Overview

8. Set/Execute Input State
9. Store Block Length Character
10. Skip If Character Less Than Operand
11. Skip If Input State
12. Skip If Character Not Equal
13. Skip If Special Character Equal
14. Resync
15. Set/Reset Translate Mode
16. Reset Cyclic Checksum Storage
17. No Operation (NOP)

SET/RESET INPUT MESSAGE-IN-PROCESS FLAG INSTRUCTION

This command directs the subsystem to set or reset the input message-in-process flag maintained in the multiplexer line control block. Format of this instruction is as follows:



where:

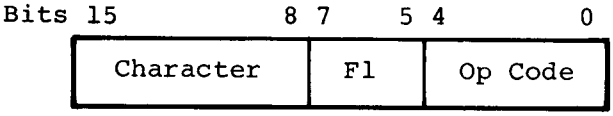
F2 = If zero, reset message-in-process flag. If one, set message-in-process flag.

F1 = See basic format in paragraph entitled State Process Table.

Op Code = 01 (hexadecimal)

REPLACE CHARACTER INSTRUCTION

This instruction provides for the substitution of a special input character by the character specified in the operand field. The format of this instruction is as follows:



where:

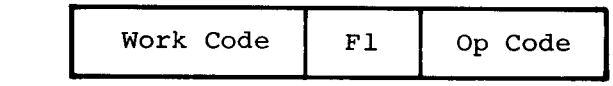
Character = Substitute character

F1 = See basic format in paragraph entitled State Process Table.

Op Code = 02 (hexadecimal)

BUILD EVENT WORKLIST INSTRUCTION

Directs the subsystem to generate an event worklist entry using the work code specified by the contents of the operand field. The subsystem also places the current value of the input buffer pointer and line number in the worklist entry. The format of this instruction is as follows:



where:

Work Code = Work code to be placed in the worklist entry.

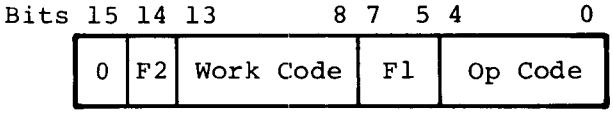
F1 = See basic format in paragraph entitled State Process Table.

Op Code = 03 (hexadecimal)

TERMINATE INPUT BUFFER INSTRUCTION

This instruction terminates the current input buffer and generates a worklist entry containing the current buffer address and the first buffer address of the input data block. Upon execution, this instruction causes the IDP to

obtain a new buffer upon receipt of the next input character. The FCD for this buffer is obtained from field IBFCD in the LCB, initialized by the command driver at the time an input command is issued. IBFCD must always specify the left byte of a data buffer word. The format for the terminate input buffer instructions is as follows:



where:

F2 = Flag which, when set, indicates that the level 2 worklist processor is to repoint the current input buffer (stored in word +2 of the worklist entry) to top of buffer, calculate the LCD and store the LCD in word 0 of the current input buffer.

F1 = See basic format in paragraph entitled State Process Table.

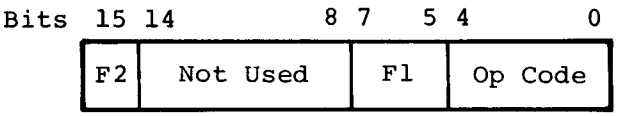
OP Code = 04 (hexadecimal)

NOTE

When designing a state process using this instruction, this instruction should be the last instruction executed in the state process table.

SKIP IF CRC EQUAL INSTRUCTION

Directs the subsystem to test a 1-character block check character (BCC) against an accumulated CRC. An equal condition causes a relative skip to the state process instruction specified by F1. An unequal condition causes the next state process instruction to be executed. The format of the instruction is as follows:



where:

F2 = If zero, compare 8-bit result. If one, compare 7-bit result (provides for accumulation and testing of a longitudinal encoded BCC, including a vertical parity bit).

F1 = If the incoming CRC is equal to the accumulated CRC, specifies the relative skip address within the state process.

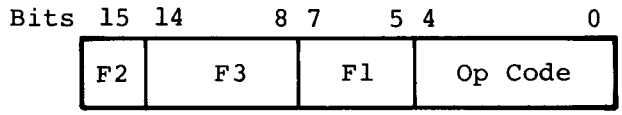
Op Code = 05 (hexadecimal)

NOTE

For a hexadecimal CRC polynomial, the first BCC character should be accumulated by a state process that relinguishes control to the input data processor at entry point 4 (accumulate CRC and discard character).

DECREMENT CHARACTER COUNT INSTRUCTION

This instruction directs the subsystem to decrement and test either of two character counters maintained in the LCB. F2 specifies the counter to be decremented. When the character counter reaches zero, the contents of F3 are used (in lieu of F1) as a relative skip address. The instruction format is as follows:



where:

F2 = If zero, decrement character counter 1. If one, decrement character counter 2.

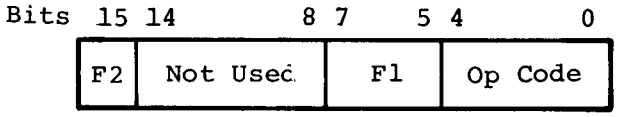
F3 = When the specified character counter reaches zero, F3 specifies relative skip address (in lieu of F1).

F1 = See basic format in paragraph entitled State Process Table.

Op Code = 06 (hexadecimal)

INITIALIZE CHARACTER COUNT INSTRUCTION

Initializes either of two character counters maintained in the LCB. F2 specifies the character counter to be initialized. Character counter 1 is initialized from the packet size maintained in the TCT table. Character counter 2 is initialized from the maximum block length maintained in the LCB. The instruction format is as follows:



where:

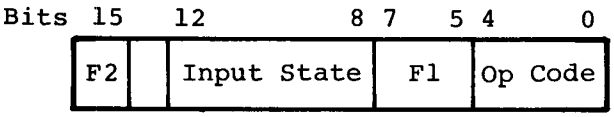
F2 = If zero, initialize character counter 1. If one, initialize character counter 2.

F1 = See basic format in paragraph entitled State Process Table.

Op Code = 07 (hexadecimal)

SET/EXECUTE INPUT STATE INSTRUCTION

This instruction changes the current input state (field ISTA in the LCB) to the value specified by bits 8-12 of the operand field. The input state selects the state process to be executed as a function of external stimuli such as the input data. An option is also provided to immediately execute the state process specified by field ISTA in the LCB, allowing the current state process to select and execute a new state process. The instruction format is as follows:



where:

F2 = If zero, the state process interpreter executes the function specified by F1. If one, ignores F1 field and executes the state process specified by field ISTA in LCB. In either case, assuming the input state field is non-zero, the input state field is transferred to ISTA field in LCB.

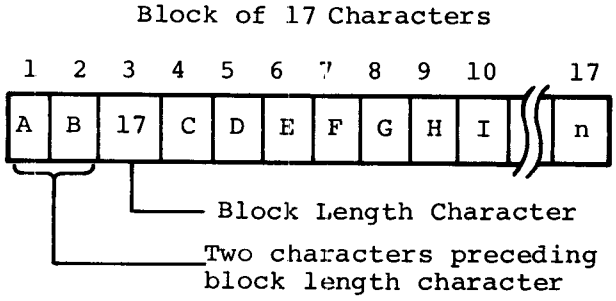
Input State = Input state code transferred to ISTA field in LCB.

F1 = See basic format in paragraph entitled State Process Table.

Op Code = 08 (hexadecimal)

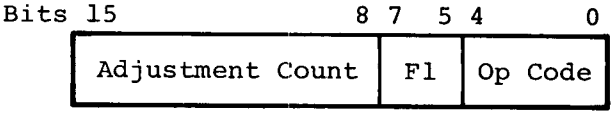
STORE BLOCK LENGTH CHARACTER INSTRUCTION

Directs the subsystem to use the current input character as the initial block length value and to store this value in the character count 1 (CNT1) field of the LCB. The operand field must specify whether the count in the block length character includes characters preceding the block length character. An example of the block length character is illustrated below:



For the above example, the operand must contain an adjustment count of three to compensate for characters A and B and the block length character itself.

The instruction format is as follows:



where:

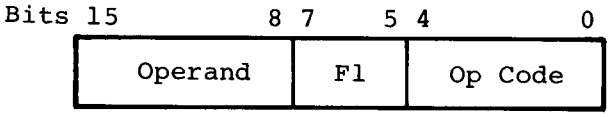
The adjustment count field specifies the number of characters preceding the block length characters, as indicated above.

F1 = See basic format in paragraph entitled State Process Table.

Op Code = 09 (hexadecimal)

SKIP IF CHARACTER LESS THAN OPERAND INSTRUCTION

Compares the untranslated input character with the value contained in the operand field. If the input character is less than the operand, a relative skip is executed to the state process instruction specified by the value of F1. If the input character is equal to or greater than the operand, the next state process instruction is executed. The instruction format is as follows:



where:

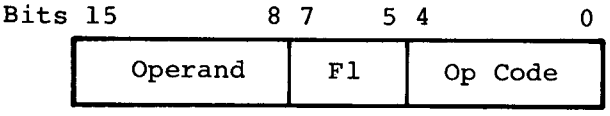
Operand = Comparison character

F1 = See basic format in paragraph entitled State Process Table.

Op Code = 0A (hexadecimal)

SKIP IF INPUT STATE LESS THAN OPERAND INSTRUCTION

Compares the current input state (field ISTAI in the LCB) with the operand. If the input state is less than the operand, a relative skip is executed to the state process instruction specified by the value of F1. If the input state is equal to or greater than the operand, the next state process instruction is executed. The instruction format is as follows:



where:

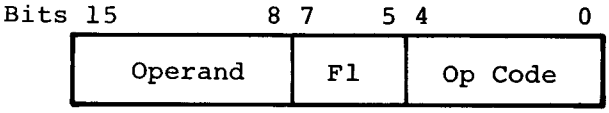
Operand = Comparison character

F1 = See basic format in paragraph entitled State Process Table.

Op Code = 0B (hexadecimal)

SKIP IF CHARACTER NOT EQUAL INSTRUCTION

Compares the untranslated input character with the operand value, and an unequal condition results in a relative skip to the state process instruction specified by the value of F1. An equal comparison results in execution of the next sequential state process instruction. The instruction format is as follows:



where:

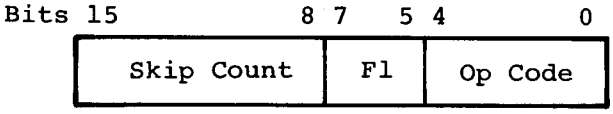
Operand = Comparison character

F1 = See basic format in paragraph entitled State Process Table.

Op Code = 0C (hexadecimal)

SKIP IF SPECIAL CHARACTER EQUAL INSTRUCTION

Compares the special character (SCHR) stored in the LCB with the input character. If equal, the state process interpreter uses the value in the skip count field as a relative skip address to a state process instruction. Upon a non-compare condition, the state process interpreter tests F1 and reacts according to the basic format. The instruction format is as follows:



where:

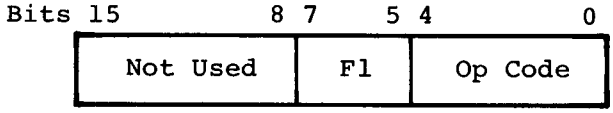
Skip Count = Relative skip address employed if special character and input character are equal.

F1 = See basic format in paragraph entitled State Process Table.

Op Code = 0D (hexadecimal)

RESYNC INSTRUCTION

This instruction directs the state process interpreter to generate a resynchronization command to the CLA on which the current character was input. The resynchronization command places the CLA in the sync acquisition mode and the CLA discards all characters until the next SYNC character is sensed. The instruction format is as follows:



where:

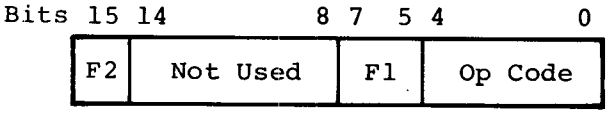
F1 = See basic format in paragraph entitled State Process Table.

Op Code = 0E (hexadecimal)

SET/RESET TRANSLATE MODE INSTRUCTION

Sets or resets the input translate logic in the input data processor microprogram. The current input character is stored according to the previous mode of

translation. This instruction is normally used on communication lines on which data can be switched from character mode to binary (transparent) mode. The instruction format is as follows:



where:

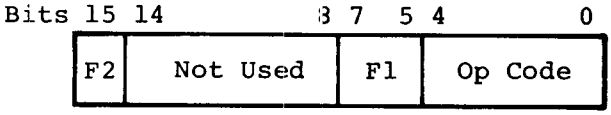
F1 = If zero, set translate mode. If one, reset translate mode.

F1 = See basic format in paragraph entitled State Process Table.

Op Code = 0F (hexadecimal)

RESET CYCLIC CHECKSUM STORAGE INSTRUCTION

This instruction resets the cyclic checksum word (field CRCS in the LCB). This word contains the partially accumulated checksum for communication lines requiring cyclic encoding and decoding. The instruction is normally used in the state process that detects the first character to be checksummed. The instruction format is as follows:



where:

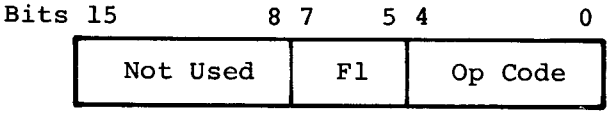
F2 = If zero, reset cyclic checksum storage to all zeros. If one, reset cyclic checksum storage to all ones.

F1 = See basic format in paragraph entitled State Process Table.

Op Code = 10 (hexadecimal)

NO OPERATION (NOP) INSTRUCTION

This instruction directs the state process interpreter to execute a no operation (NOP) instruction. The format of the instruction is as follows:



where:

F1 = See basic format in paragraph entitled State Process Table.

Op Code = 00 (hexadecimal)

INTRODUCTION

The network communications software comprises all programs that are not included in the base system (including the multiplex subsystem) and are not part of the interface programs. These procedures include directory services programs, a service module, and header build functions.

DIRECTORY SERVICES

Directory services consist of the routing process and a group of procedures that support directories required by the routing process. In general, the routing process is divided into internode and intranode routing. Directory services also perform directory updates.

INTERNODE ROUTING

Internode routing is the process of determining to which node a block is addressed. The destination node directory is indexed by the destination node (DN) value contained in the block header to obtain the address of a link control block or, when a block arrives at its destination node, to obtain the address of a source node directory. The NPU may contain only one logical node, with its identifying number maintained in a global data structure. For routing, host nodes need not be distinguished from terminal nodes.

INTRANODE ROUTING

Intranode routing determines to which terminal or internal process a block is addressed once the block has reached its destination node. Two directory lookups are performed using the connection number (CN) and source node (SN) values within the block header to provide addressing information.

In intranode routing, CN is first tested and, if zero, the block is addressed to internal process and routing is finished. If CN is nonzero, SN indexes the source node directory to select the connection directory associated with the SN. Then, CN indexes that connection directory to provide the TCB address. Either of these directory lookups may indicate an invalid block address, in which case an indication to that effect is given to the caller.

ADDING OR DELETING DIRECTORY ENTRIES

Directory entries may be added or deleted at the request of the service module. Entries in the connection directory are referred to as "connection definitions".

OPERATING LEVEL

All procedures within directory services run at OPS level.

DESTINATION NODE (DN) DIRECTORY

The destination node directory contains an integer value associated with each valid DN address (0-255). For the local node (meaning within the same physical node), the directory provides the address of the source node directory associated with the logical node. For all other logical nodes, the directory entry provides a link control block address. A zero entry indicates a nonexistent node (an unassigned value of DN).

The destination node directory is located in a fixed memory location.

SOURCE NODE (SN) DIRECTORY

The local logical node has a source node directory used to select the connection directory associated with the pair of nodes indicated by DN and SN. Nonzero entries indicate the address of the connection directory.

The source node directory is located in dynamic buffer space.

CONNECTION (CN) DIRECTORY

In each logical node there is a connection directory for all other logical nodes with which there is at least one connection defined. An entry in the connection directory provides the address of a terminal control block (TCB).

The connection directory is located in dynamic buffer space.

ROUTING PROCESS

The routing process (PNROUTE) performs both internode and intranode routing using the addressing information (DN, SN, CN) found in each block header and the associated routing directories. The calling sequence is as follows:

```
PNROUTE (parm)
```

where parm is a PASCAL variable of type B0BUFPTR that selects the input block. PNROUTE is a PASCAL function that returns an integer value interpreted as follows:

0 = Block addressed to internal process (service module)

1 = Invalid address

2 through 65,535 = Terminal control block (TCB) address

The input to PNROUTE is a block containing DN, SN, and CN in its header. This block is accessed via the pointer-type variable parm. If DN is local and CN is zero, the block is addressed to internal processing and value SN is ignored.

DN indexes the destination node directory to obtain an address. If the address obtained is zero, the destination of the block is undefined and PNROUTE returns an indication to that effect.

If the destination is not the local logical node, the address obtained is a link control block address that is returned to the caller. If the destination is the local logical node, the address obtained is a pointer to a source node directory.

If the DN directory lookup yields a pointer to a source node directory, SN provides an index within that directory to obtain either a pointer to a connection directory or an indication that SN is invalid. If SN is invalid, that information is returned to the caller. If not, CN is used as an index within the connection directory. If CN is valid and nonzero, a TCB address is obtained that is returned to the caller.

ADD DIRECTORY ENTRY

The procedure that adds directory entries (PNDIRADD) operates at the request of internal processes. Buffers are added to directories as necessary. The calling sequence for this procedure is as follows:

```
PNDIRADD (parm1, parm2, parm3,
          parm4)
```

where parm1 and parm2 are PASCAL variables whose values must fall between 0 and 255 and respectively represent the DN and SN values; parm3 is a PASCAL variable that

must fall between 1 and 255 and represents CN; and parm4 is a PASCAL variable of type B0BUFPTR that is a pointer to the TCB to be inserted into the routing directory.

PNDIRADD output is the addition of DN directory entries and, in some cases, primary or secondary SN and CN directory segments.

DELETE DIRECTORY ENTRY

This procedure (PNDIRDLT) removes entries from the routing directories at the request of internal processes. If the removal causes a directory buffer to become empty, the buffer is released to the pool of available buffers of its size. The calling sequence for this procedure is as follows:

```
PNDIRDLT (parm1, parm2, parm3)
```

where parm1 and parm2 are PASCAL variables whose values must fall between 0 and 255 and respectively represent DN and SN. Parm 3 is also a PASCAL variable that must fall between 1 and 255 and represents CN.

NOTE

If parm2 is negative 1, all entries associated with DN = parm1 will be removed from the directories. If parm3 is negative 1, all entries associated with DN = parm1 and SN = parm2 will be removed from the directories.

Output from PNDIRDLT is the removal of an entry (connection definition) from the routing directory and the release of any buffers that become empty.

SERVICE MODULE

The service module is a group of procedures that handle communications with the host via the service channel. They provide system configuration, statistics reporting, user notification on host failure/recovery, and control of on-line diagnostics. The service module consists of the following procedures:

PNSMWL	Service Module Worklist Handler
PNSMES	Downline Service Message Handler
PNSMTIP	TIP Response Handler
PNTCBREL	Terminal Control Block (TCB) Release
PNSMDISP	Service Message Dispatcher
PNPSTAT	Periodic Statistics Dump
PNDSTAT	Immediate Statistics Dump
PNHOSTSTATUS	Host Status Notification
PNDIAGIF	On-line Diagnostics Control

A build-time parameter specifies the maximum number of lines that may be configured for the NPU. By means of service messages, the host specifies which lines are to be configured and their parameters. When each line becomes operational, the host then specifies the terminals to be configured for that line. The service module provides handling of such configuration messages between the host, the NPU, and the TIPs.

Within each LCB and TCB, use and error counters are retained. Periodically the NPU sends line, terminal, and NPU statistics to the host. The service module also handles the sending of such statistics messages to the host when a statistics counter overflows. When errors are detected, the service module sends appropriate error messages to the host.

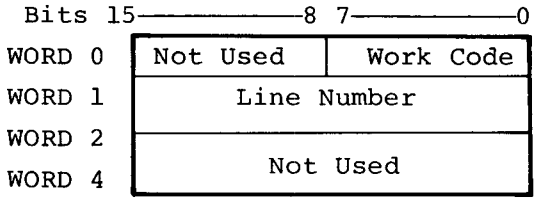
SYSTEM CONFIGURATION FUNCTIONS

The service module performs system configuration under the direction of the host. Some configuration tasks are delegated to the individual TIPs via worklist entry.

TIP WORKLIST ENTRIES

The format of TIP worklist entries from the service module is similar to those from the multiplex subsystem except that only the first two words of the service module worklist entry are meaningful. It is assumed that no TIP has a worklist entry with fewer than two words.

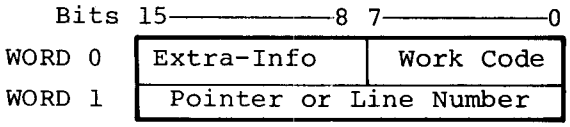
The format of a worklist entry from the service module is as follows:



The line number is available as an integer or as two bytes (port and subport). The work code is of type integer and is interpreted as follows:

Code	Meaning
AOSMEN	Service module requests TIP to enable the line indicated. TIP should respond COLINOP or COLNINOP (described below).
AOSMTCB	Service module has attached first (or only) TCB to the indicated line. No response is expected.
AOSMDA	Service module requests TIP to disable the line indicated. After disabling the line (and after Data Set Ready (DSR) drops on a switched line), the TIP should respond with COLNDA. The service module purges the TCBs.

The format of a worklist entry to the service module is as follows:



where the workcode meanings are as follows:

Code	Meaning
COSMSGR	Downline service message received. Word 1 is a pointer to the service message.
COTCBREL	Release TCB. When a TIP receives a TERMINATE, the associated TCB is removed from the chain of TCBs and passed to the service module. The service module releases the TCB and any attached queue and sends a TERMINATE to the host. Word 1 is a pointer to the TCB released.

Code	Meaning
COLINOP	Sends line operational, indicating the TIP has enabled a line and successfully connected and identified a terminal. The left byte of Word 0 specifies the current terminal type and Word 1 is the line number.
COLNINOP	Sends line inoperative, indicating the TIP has detected a line failure. The left byte of Word 0 contains the line error code and Word 1 is the line number.
COLNDA	Indicates line disabled by TIP. TIPs must send this as a response to AOSMDA when the line is disabled. The left byte of Word 0 is not used and Word 1 is the line number.

STATISTICS AND ERROR MESSAGES

Statistics and error messages can be sent to the host via service module procedures that are available at all software priority levels.

STATISTICS DUMP

When incrementation of a statistics counter causes an overflow (to zero), the counter must be set to all ones ($FFFF) and the statistics dump (PNDSTAT) procedure must be called. PNDSTAT dumps the particular statistics block to the host and clears that block. The calling sequence is as follows:

```
PNDSTAT (parm1, parm2)
```

where parm1 is of type C3STAT and indicates the type of statistics block to be dumped (C3NPU, C3LINE, or C3M4TER) and parm2 is either a TCB pointer or an LCB. Where parm1 equals C3NPU, parm2 is not used and should be NIL. Where parm1 equals C3LINE, parm2 is the LCB whose statistics are to be dumped. Where parm1 equals C3M4TER, parm2 is a pointer to the TCB whose statistics are to be dumped.

CE ERROR FILE

Upon detection of any hardware abnormality, users should report to the host via a CE error file message (PNCEFILE). To do so, an error message must be prepared and sent as an upline message to the host. Such message must begin with a 1-byte error code and continue for a maximum message length of 28 bytes (characters). The calling sequence for PNCEFILE is as follows:

```
PNCEFILE (parm1, parm2)
```

where parm1 is of type integer and indicates the message length in bytes and parm2 is of type PACKED ARRAY OF CHAR that begins with the error type code and follows with the error message text of up to 28 characters.

UPLINE BLOCK HANDLER (HEADER BUILD)

The upline block handler is a header build procedure. It is supplied as a standard POI sequence used by TIPs and the service module at the post-input POI to complete message headers. Header build assumes that the TIP has provided room for the header in the message and provides the following functions:

1. Inserts DN and CN as indicated by the input TCB (BITCB).
2. Inserts SN from the global variable CKLOCNODE.
3. Inserts block type as supplied by B1BT.

4. Inserts block serial number (BSN) from the TCB (upline or downline).

5. Increments BSN in the TCB (module H3BSNL+1, upline or downline).

6. Posts an entry for the block in the internal processing worklist.

7. For block-types 2, 3, and 4 (BLK, MSG, CMD, or SVM), enters the block in the source retention queue (SRQ).

The upline block handler (header build) calling sequence is as follows:

```
PNHDRBLD
```

The parameters set up for this sequence are as follows:

```
B1TCB    (address of source TCB)
B1BUFF   (address of block)
B1BT     (type of block)
```

On entry, the data block FCD is set for the first byte of user data. On exit, the FCD is decremented by four. Should the new FCD be less than six, a PBHALT occurs.

DOWNLINE BLOCK HANDLER

The downline block handler (PNDLBH) is a standard system-supplied POI sequence used by the HIP at the internal-in POI. It queues the BLK, MSG, and CMD block types and processes RESET, BREAK, TERMINATE, and BACK type blocks. It also maintains the source retention queue (SRQ) and overflow source retention queue (overflow SRQ). PNDLBH communicates with the TIPs via worklist entries of an event, depending upon the Boolean setting of the associated terminal characteristics table (TCT) fields. The downline block handler calling sequence is:

```
PNDLBH
```

INPUTS

Global inputs to PNDLBH are as follows:

B1TCB	-	address of destination TCB
B1BUFF	-	address of block

OUTPUTS

For block types BLK, MSG, and CMD, PNDBLH queues the block in the TCB output queue and, depending upon the TCT-defined Boolean values, generates an output-queued worklist entry if the output acceptable flag in the TCB is set. If this flag is not set, the block is discarded.

For BREAK blocks, PNDLBH generates an upline RESET and resends all blocks for which a BACK has not been received. A BREAK-received worklist entry is made if the associated Boolean value in the TCT is set.

For BACK blocks, if BSN matches, the top entry in the SRQ is released. If the overflow SRQ is not empty, the top entry from the OSRQ is placed at the end of the SRQ and is sent to internal processing.

For TERMINATE blocks, the buffers associated with the block are released and a terminate worklist entry is built if the associated Boolean value in the TCT is set.

For RESET blocks, the buffers associated with the block are released and the output acceptable flag is set in the TCB.

APPENDIX A
GLOSSARY

INTRODUCTION

This glossary defines terms (both English language and mnemonic) unique to the descriptions contained in this manual or common terms whose definitions are different from or more constrained than definitions commonly held. A glossary of English language terms is presented first, followed by a glossary of mnemonic terms.

ENGLISH LANGUAGE TERMS

ACCEPTANCE TEST PERIOD (ATP)

The period of time, following a failed-to-operational transition, during which a facility must remain continuously operational, prior to accepting the facility for handling traffic.

ADDRESS INFORMATION

That information within a block or message that identifies the source or intended destination of the associated data.

APPLICATION PROCESS (AP)

A process resident in a host computer which provides an information storage, retrieval and/or processing service to a remote user via data communications.

BLOCK

A portion or all of a message. A message is divided into blocks to facilitate buffering, transmission, error detection and correction of variable length data streams. A transmission block includes the protocol envelope, consisting of the transmission header and transmission trailer information. The envelope is used to delimit and control transmission of the block over the communications channel.

BLOCKING

The process of dividing a contiguous data stream into units of generally fixed length.

BREAK

1. A method which a terminal operator employs to indicate that he desires to interrupt output in progress. This is accomplished by pressing the "break" key on a teletype. This causes the line to remain in the spacing state while the key is depressed, causing a framing error to be detected by the line adapter which terminates the asynchronous line. For terminals operating in half-duplex mode, break causes transition from output mode to input mode.

2. An element of the block protocol which indicates an interruption of the data stream.

BUFFER

Consecutive bytes of 2550 Communications Controller Storage used to hold a portion of an information stream.

BYTE

A group of bits. Unless prefixed (e.g., 6-bit byte), the term implies 8-bit groups. When used for encoding character data, a byte represents a single character.

CLUSTER

A group consisting of a controller and all terminals which it supports.

COMMAND

Information passed to a process which performs control of the process rather than being data destined for transmission by the process.

COMMUNICATIONS LINE

A communications circuit between a terminal and its network processing unit.

COMMUNICATIONS LINE ADAPTER (CLA)

A unit of hardware which interfaces a communications line to the storage of an NPU.

CONNECTION NUMBER

An 8-bit number which represents a pair of processes within the context of the two nodes associated with the communicating processes.

CONTENTION

The situation which exists when a device has information ready to place on a communications line which is busy.

CONTROL INFORMATION

Information which is not part of a message, but which must be transmitted in support of the communications protocol.

CONTROLLED TERMINAL

A terminal whose input device will place data on the communications line only in response to a poll. The maximum input rate of the device can thus be regulated by control of the polling rate.

CONTROLLER

A hardware device which interfaces multiple terminals to a single communications line, and performs some common functions for those terminals (such as protocol handling).

DATA

Any portion of a message as created by the source, exclusive of any information used to accomplish the transmission of such message.

DEAD START

The process of loading and starting a processor.

DEMAND FLOW CONTROL (DFC)

A method of regulating the rate at which information is forwarded by a host to the terminal node for an output device, such that storage requirements at the terminal node are minimized while keeping the device operating at capacity, by using transmission completion events to request further output data.

DESTINATION

The terminal or host which is designated to receive the message.

DESTINATION PROCESS

The process at the destination node which delivers a message to the destination.

DESTINATION NODE (DN)

The node which directly interfaces to the destination.

DEVICE

An input-only or output-only portion of a terminal.

DISCONNECT

The state transition which occurs when a terminal connected to the terminal node via the switched network ceases to be so connected, because its modem goes "on hook", or because of a failure of the switched network.

DOWNLINE

The flow direction of output from host to terminal.

FAILURE VERIFICATION PERIOD (FVP)

The period of time, following an operational-to-failed transition, during which a facility must remain continuously failed, prior to taking the facility out of service and reporting the failure. The FVP is used to isolate the host from high frequency transients in the state of a communications line.

FREEWHEELING TERMINAL

A terminal which can input at the discretion of the terminal operator and whose input rate cannot be controlled by the terminal node. Contrast with controlled terminal.

FRONT END

A network processing unit which directly interfaces one or more hosts.

FULL DUPLEX (FDX)

Two-way simultaneous transmission, when applied to a communications line. Simultaneous, independent operation of the input and output devices, when applied to a terminal.

HALF DUPLEX (HDX)

Two-way alternate transmission, when applied to a communications line. When applied to a terminal, it means that the terminal cannot simultaneously send one message while receiving another, usually because the output device locally copies the input while the terminal is in input mode.

HOST

A digital computer which executes the programs of an application process.

HOST COUPLER

An element of hardware which interfaces a host computer to a Communications Controller.

HOST INTERFACE PACKAGE (HIP)

The collection of programs resident in a Communications Controller which controls the transfer of blocks between one or more hosts and NPUs.

HOST NODE (HN)

The node ID associated with an interface between an NPU and a host.

INFORMATION

A one-dimensional stream of bits which is communicated from one point to another, exclusive of synchronizing patterns which establish the sample point for the receiver.

INOPERATIVE

Not operational.

INPUT

Information flowing upline from terminal to host.

LINE CONTROL BLOCK (LCB)

A control block in the terminal node which records the status and operational parameters of the associated line.

LINE CONTROL PROTOCOL

The conventions for encoding, blocking and formatting messages, redundancy generation and error

detection, and error correction procedures, and the formats and interpretation of the address and control information used to effect such procedures, as applied to the communication of data between a terminal and its terminal node.

LINE NUMBER

The identifier of a specific terminal line, consisting of a CLA hardware address (port) and, where necessary, a multiplexer subport.

LINE PROCESS

That process in a terminal node which controls line (as opposed to terminal) functions, such as modem interface, connect and disconnect.

LINK

A full-duplex point-to-point communications connection between two nodes, consisting of one or more trunks.

LINK CONTROL BLOCK (LKCB)

A control block which maintains the operational parameters and status of a particular link.

LINK CONTROL PROTOCOL

The conventions for grouping packets, redundancy generation and error detection, and error correction procedures, and the formats and interpretation of the control information used to effect such procedures, as applied to the communication of packets between neighboring nodes.

LINK FAILURE

The failure of all trunks of a link.

LINK INTERFACE PACKAGE (LIP)

The collection of programs resident in the communications controller which controls the transfer of blocks over one or more links.

LINK PROTOCOL ENVELOPE

That information which is placed both before and after a collection of packets to form a link transmission block. The envelope performs the functions of: delimiting the start and end of the packet group, identifying the group, carrying error detection redundant information, and carrying error control information.

LINK QUEUE

A queue of packets to be transmitted to the CNP.

LINK RETENTION QUEUE (LRQ)

A queue of packets which have been transmitted to the CNP but not acknowledged as received.

LOAD REGULATION

The mechanism of controlling the rate of information input to a node by selectively inhibiting particular inputs when the number of available buffers decreases past a preset threshold.

LOGICAL CHANNEL

The total mechanism provided by the NPU for the bidirectional transfer of data and commands between two particular processes.

LOGICAL CONNECTION

The association of two particular processes via the assignment of a network address.

NETWORK ADDRESS

A set of three 8-bit numbers, consisting of two node IDs followed by a connection number. The first node ID is the destination node. The second node ID is the source node.

NODE

A network element which creates, absorbs, switches and/or buffers blocks.

NODE ID

An 8-bit binary serial number which represents a node.

OFF-LINE

The state in which a terminal cannot be physically accessed by the network because power is off, or terminal is in local mode, or terminal is not connected via communications line to the network.

OPERATIONAL

In a condition to perform intended function of communicating data, with no further status change being necessary.

OUTPUT

Information flowing downline from host to terminal.

PORT

The interface between a communications line and a network processing unit.

PRIORITY, INPUT

As free buffers in an NPU are placed into service, and the free buffer level crosses succeedingly lower thresholds, certain terminals are not permitted to input. Those terminals permitted to input at the lowest free buffer level have highest input priority.

QUIESCENT

When applied to a line, neither is data in transit nor is polling scheduled.

RESEQUENCING

Storing received elements by serial number so that the elements can be forwarded in the order that the serial number was assigned.

SERVICE MESSAGE (SM)

A command transferred between service modules.

SERVICE MODULE

A set of processes which are not directly involved in the transmission or processing of data, but which assist in establishing and maintaining an environment for the communication of data between processes.

SOURCE

The terminal or host which created the message.

SOURCE PROCESS

The process at the source node which obtains messages from the source station.

SOURCE NODE (SN)

The node which directly interfaces to the source station.

STATION

A provider and/or recipient of messages.

TERMINAL

An entity, external to the network, but connected to it via a communications line, which supplies input messages to, and/or accepts output messages from, an application process.

TERMINAL CONTROL BLOCK (TCB)

A control block in the terminal node which records the status and operational parameters of the associated terminal.

TERMINAL CONFIGURATION

That collection of information which identifies the addresses

(if any), device types and characteristics, and operational mode of all terminals connected to a given communications line.

TERMINAL INTERFACE PROGRAM (TIP)

The collection of programs resident in a communications controller which controls the transfer of messages between a communications line (to which are connected one or more terminals and/or clusters) and the Communications Controller.

TERMINAL MODE

The specification of which unit is the input device and which unit is the output device, and how those units are coupled, if the option exists.

TERMINAL NODE (TN)

Network processing unit which supports one or more terminal interface programs and to which terminals are directly connected via communications lines.

TERMINAL OPERATOR

That person who is operating the controls of a terminal. Contrast with user.

TERMINAL PHYSICAL ADDRESS

A sequence of numbers which represent the terminal node, line number and identification of the terminal on the line.

TIP CONTROL COMMAND

A command which is communicated via a logical connection. Contrast with service message.

TRUNK

A circuit which is used to carry packets between computers.

TRUNK BLOCK

A transmission block on a trunk circuit.

TRUNK CONTROL BLOCK (TKCB)

A control block in a Network Processing Unit which records the status and operational parameters of the associated trunk.

TRUNK FAILURE

Failure of the link control protocol to obtain an error free input from a trunk during a predetermined interval. Such a failure may be caused by failure of the communications facility, or by failure of the neighboring node.

TURN AROUND

A state transition wherein a half-duplex circuit changes from communicating information in one direction, to communicating information in the opposite direction.

UNIT

One of multiple components of a device. Only one unit of a device may be active at a time.

UPLINE

The flow direction of input from terminal to host.

USER

That person or group of people who are the preparers and/or recipients of messages communicated with an application process via the network. A user may interface with one or more terminals, or with no terminals. Contrast with terminal operator.

MNEMONIC TERMS

AP	Application Process
BACK	Block Acknowledge
BCC	Block Check Character
BCD	Binary-coded Decimal

BFC	Block Format Code
BLK	All but last block of message (see MSG)
BP	Breakpoint
BSN	Block Serial Number
BSNL	Block Serial Number Limit
BT	Block Type
CC	Command Code
CCP	Communications Control Program
CLA	Communications Line Adapter
CN	Connection Number
CRC	Cyclic Redundancy Check
CRT	Cathode Ray Tube
CTS	Clear to Send
DCD	Data Carrier Detect
DMA	Direct Memory Access
DN	Destination Node
DP	Destination Process
DSR	Data Set Ready
DTR	Data Terminal Ready
FCD	First Character Displacement
FCR	Function Control Register
FDX	Full Duplex (see HDX)
HDX	Half Duplex (see FDX)
HIP	Host Interface Program
ID	Identification Number
IDC	Internal Data Channel
LCB	Line Control Block
LCD	Last Character Displacement
LEC	Line Error Code
LPC	Longitudinal Parity Character
LRC	Longitudinal Redundancy Check
MC	Master Clear
MLIA	Multiplex Loop Interface Adapter
MSG	Last Block of Message (see BLK)
MTI	Message Type Indicator
NPU	Network Processing Unit
ODD	Output Data Demand
PCB	Program Control Block
POI	Point of Interface
PPU	Peripheral Processing Unit
RAM	Random Access Memory
RB	Reason For Break
ROM	Read Only Memory
RTS	Request to Send
SC	Service Code
SN	Source Node
SOH	Start of Header
SP	Source of Process
TCB	Terminal Control Block
TDP	Time Dependent Program
TIP	Terminal Interface Program
TPN	Terminal Position Number
TTY	Teletype
TUP	Test Utility Program
U/L	Upper/Lower Indicator

APPENDIX B
CE ERROR MESSAGE AND SYSTEM ERROR CODES

CE ERROR MESSAGES

Format:

Bytes	0	1	2	3	4	5	6	7	8	34
	DN	SN	00	4	BSN	00	04	00	EC	TEXT

where:

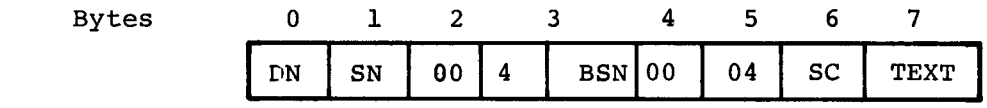
DN = Destination Node (2 hexadecimal characters)
SN = Source Node (2 hexadecimal characters)
BSN = Block Sequence Number (1 hexadecimal character)
EC = CE Error Code (2 hexadecimal characters)
TEXT = Error Code Dependent Text (see below)

ERROR CODE	REPORTED BY	DESCRIPTION	TEXT
01	CLA STATUS HANDLER	DISCONNECT SWITCHED LINE	PT SP S1 S2 PT - PORT NUMBER (CLA ADDRESS) SP - SUBPORT NUMBER (NOT USED = 00) S1 - CLA STATUS BYTE 1 (LOGICAL FORMAT0 FORMAT) S2 - CLA STATUS BYTE 2 (LOGICAL FORMAT) CLA STATUS BYTE 1 CLA DSR DCD SCDC QM SQD RI SPARE CLA STATUS BYTE 2 SPARE SPARE ILE OLE DTO NCNA PES FES CTS - CLEAR TO SEND DSR - DATA SET READY DCD - DATA CARRIER DETECT SDCD - SECONDARY DATA CARRIER DETECT QM - QUALITY MONITOR SQD - SIGNAL QUALITY DETECTOR RI - RING ILE - INPUT LOOP ERROR OLE - OUTPUT LOOP ERROR DTO - DATA TRANSFER OVERRUN NCNA - NEXT CHARACTER NOT AVAILABLE PES - PARITY ERROR STATUS FES - FRAMING ERROR STATUS
02	CLA STATUS HANDLER	ABNORMAL DSR OR CTS OPERATION	
03	CLA STATUS HANDLER	ABNORMAL DATA CARRIER DETECT OPERATION	
04	WORKLIST PROCESSOR	UNSOLICITED ODD	
05	WORKLIST PROCESSOR	CLA ADDRESS OUR OF RANGE	
06	WORKLIST PROCESSOR	ILLEGAL LOOP CELL FORMAT	
07	WORKLIST PROCESSOR	UNSOLICITED INPUT	
08	CLA STATUS HANDLER	INPUT LOOP ERROR	
09	CLA STATUS HANDLER	OUTPUT LOOP ERROR	
0A	PTTER	ODD TIMEOUT	
0B	PTTER	MODEM TIMEOUT	
0D	CLA STATUS HANDLER	CLA STATUS OVERFLOW	
0E	CLA STATUS HANDLER	FRAMING ERROR	
0F	CLA STATUS HANDLER	NEXT CHARACTER NOT AVAILABLE	
10	CLA STATUS HANDLER	DATA TRANSFER OVERRUN	

Error Code	Reported By	Description	Text
11	PBMLIA	MLIA Error Status	ET | LE | LD | AL ET - Error Type 00 - Error Condition Restored 01 - Error Counts Given 02 - MLIA Failure LE - Input Loop Error Count } only listed if ET = 01 LD - Lost Data Count AL - Alarm Count
12	Mode 4 TIP	Upline Break From Overflowed Error Counter	00 | RB | PP | SS | CA | TA RB - Reason for Break 01 - No Response Counter Overflowed 02 - Bad Response Counter Overflowed 03 - Error Response Counter Overflowed PP - Port Number SS - Subport Number CA - Cluster Address TA - Terminal Address
18	Real-Time Clock (RTC)	Real-Time Clock Error Status	SS | SS SS - Clock Status
20	PTSTART	Deadman Timeout	LS | NS LS - Last State NS - Next State
21	PTINTPROC	Spurious Interrupt	
22	AOPT2	Chain Address Zero	CP | ST
23	AOPT2	Hardware Timeout on Input	
24	AOPT2	Input Data Transfer Terminated by PPU	CP ST - Coupler Status Word
25	AOPT3	Illegal Orderword	CP | ST | OR | WD CP ST - Coupler Status Word OR WD - Orderword Received
27	AOPT5	Output Data Transfer Terminated by PPU	
28	AOPT5	Hardware Timeout on Output	CP | ST
29	AOPT5	EOP Missing	CP ST - Coupler Status Word
2A	AOPT6	Unexpected Status	

STATISTICS SERVICE MESSAGE

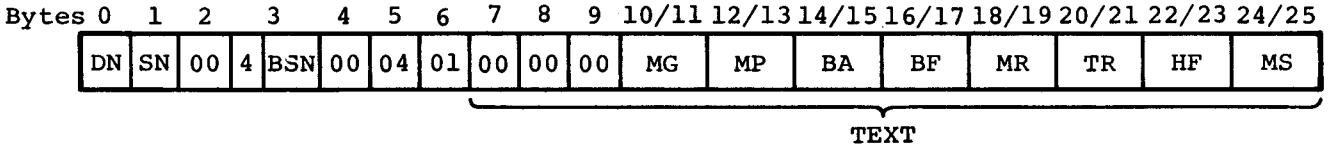
Format:



where:

DN = Destination Node
SN = Source Node
BSN = Block Sequence Number
SC = Service Code
01 = NPU Statistics
02 = Line Statistics
03 = Mode 4 Terminal Statistics
TEXT = Service Code Dependent Text (see below)

NPU STATISTICS

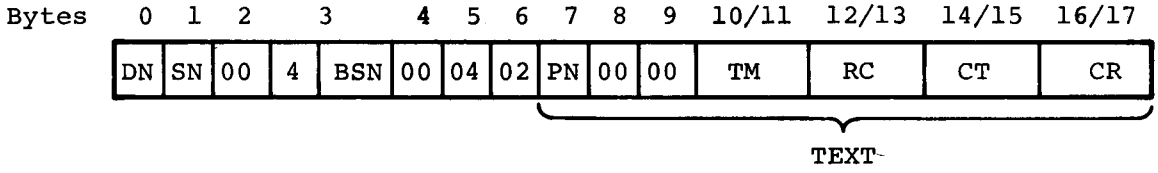


where:

DN = Destination Node
SN = Source Node
BSN = Block Sequence Number
MG = Messages Generated Count
MP = Messages Processed Count
BA = Bad Address Count
BF = Bad Format Count
MR = Mode 4 Input Regulation Started Count
TR = Teletype Input Regulation Started Count
HF = Host Failure Count
MS = Messages Out-of-Sequence Count

} Two bytes each

LINE STATISTICS



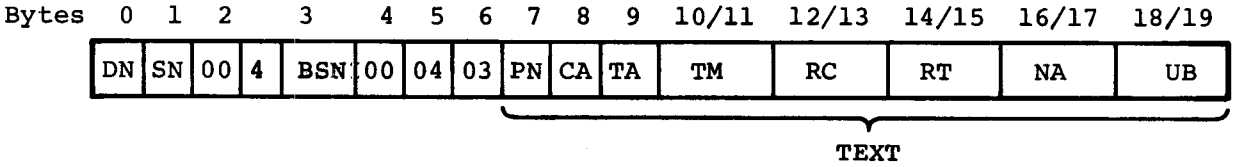
where:

DN = Destination Node
SN = Source Node
BSN = Block Sequence Number
PN = Port Number
TM = Blocks Transmitted Count
RC = Blocks Received Count
CT = Characters Transmitted Count
CR = Characters Received Count

} Good blocks only

} Two bytes each

MODE 4 TERMINAL STATISTICS



where:

DN = Destination Node
SN = Source Node
BSN = Block Sequence Number
PN = Port Number
CA = Controller Address
TA = Terminal Address
TM = Blocks Transmitted Count (does not include retransmission)
RC = Blocks Received Count (does not include blocks with errors)
RT = Blocks Retransmitted Count
NA = Blocks Not Accepted Count
UB = Upline Break Count

} Because of error only

} Two bytes each

SYSTEM HALT CODES

000	Not a valid halt code
001	Power Fail
002	Memory Parity
003	Program Protect
004	Interrupt Count < 0
005	Timal worklist error
006	Active LCB list error
007	No buffers left
008	Size error in stamp
009	Duplicate GET
00A	Duplicate RELEASE
00B	Chain error
00C	Buffer out of range
00D	Bad command, not Type 1 or Type 2
00E	PMWOLP not called from P3
00F	Attempted to clear an enabled line
010	Wrong terminal type specified
011	Bad MLIA status (initialization)

012 Duplicate CLA address (initialization)

013 Attempt to redefine an existing DN Directory entry

014 Attempt to redefine an existing CN Directory entry

015 Attempt to remove a non-existent DN Directory entry

016 Attempt to remove a non-existent SN Directory entry

017 Attempt to remove a non-existent CN Directory entry

019 Illegal POT Key

01A Attempted to add zero CN to the directories

01B Program selected to run is not in core

01C Monitor did not run for B2TIME/2 sec.

01D Service Module called with worklist empty

01E Service Module workcode out of range

01F MLIA failure

020 Pointer to read next loop cell from CIB exceeded the present line frame pointer

021 Reserved for Firmware use

022 Reserved fro Firmware use

023 Reserved for Firmware use

024 Reserved for Firmware use

025 Reserved for Firmware use

026 Reserved for Firmware use

027 Reserved for Firmware use

028 Coupler alarm condition

029 No Queue Control Block available for TCB Build

02A Bad line number from TIP

02B Unknown TASKNR selected

02C Unknown Block/CMD received

02D Improper MUX-Sub operation

02E Improper M4 - TIP operation

02F Control for disabled line

030 Reserved for M4 - TIP

031 Error in PNHDRBLD

032 Error in PNDLBH

033 Illegal line status detected by PTCLAS

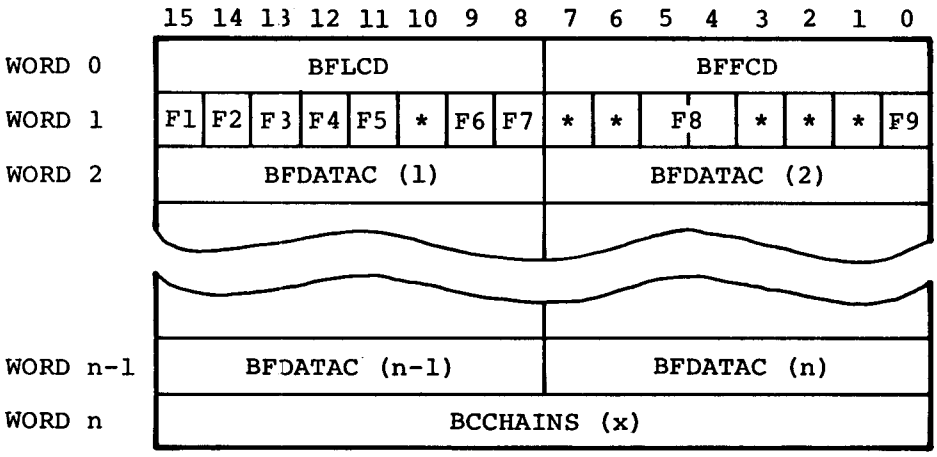
034 Illegal call to Queue Services

035 Attempt to queue message to NPU console in system without console

036 Directory function attempted with DN out of range

APPENDIX C
DATA BUFFER-GENERAL FORMAT

DATA BUFFER - GENERAL FORMAT



*	Spare
BFLCD	Last Character Displacement
BFFCD	First Character Displacement
F1(BFEOTFLG)	End of Transmission Flag
F2(BFSOTT)	Start of Transparent Text Flag
F3(BFSONT)	Start of Non-Transparent Text Flag
F4(BFSUPCHAIN)	Suppress Buffer Chaining Flag
F5(BFEOBFLG)	End of Block Flag
F6(BFPRTK)	Buffer Protect Flag
F7(BFPERM)	Permanent Buffer Flag
F8(BFQCNT)	Source Retention Queue and HIP Queue Count
F9(BFDBSIZE)	Data Buffer Size (0 = small, 1 = large)
BFDATAC (n)	Data Character
BCCHAINS (x)	Buffer Chain Word (Chain to Next Buffer)

INDEX

COMMENT SHEET

MANUAL TITLE Communications Control Program Version 1 – Software Reference Manual

PUBLICATION NO. 60470000 REVISION B

FROM: NAME: _____

BUSINESS ADDRESS: _____

_____

COMMENTS:

CUT ALONG DOTTED LINE

FOLD ON DOTTED LINES AND STAPLE

STAPLE

STAPLE

FOLD

FOLD

FIRST CLASS
PERMIT NO. 8241

MINNEAPOLIS, MINN.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

CUT ALONG LINE

CONTROL DATA CORPORATION
Publications and Graphics Division
3519 West Warner Avenue
Santa Ana, California 92704

FOLD

FOLD