Integrated Communications Access Method (ICAM)

Communications Physical Interface (CPI)

OS/3

User Guide

Environment: System 80

SPERRY

UP-9746

PAGE STATUS SUMMARY

ISSUE: UP-9746
RELEASE LEVEL: 8.2 Forward

Part/Section	Page Number	Update Level
Cover/Disclaimer		
PSS	1	
Preface	1 thru 4	
Contents	1 thru 4	
1	1 thru 23	
2	1 thru 27	
3	1 thru 12	
4	1 thru 22	
5	1 thru 16	
Appendix A	1 thru 25	
Index	1 thru 5	
User Comment Sheet		

All the technical changes are denoted by an arrow (⟹) in the margin. A downward pointing arrow (⇓) next to a line indicates that technical changes begin at this line and continue until an upward pointing arrow (⇑) is found. A horizontal arrow (⟹) pointing to a line indicates a technical change in only that line. A horizontal arrow located between two consecutive lines indicates technical changes in both lines or deletions.

Preface

This manual is one of a series that describes the purpose and use of the SPERRY Operating System/3 (OS/3) Integrated Communications Access Method (ICAM). It specifically describes the communications physical interface and how it is used to write user programs at the physical level. This manual is intended for the applications programmer experienced in data communications and assembler programming.

The physical interface requires the least amount of main storage; but, it also provides a minimum amount of support. To use this interface, you must have considerable knowledge of data communications, because your program must initialize the hardware, format all output messages using the appropriate protocol, perform any required translations, acknowledge and process all input messages, and perform all error detection and recovery procedures. In addition, the communications portion of your program must be written in basic assembly language, and your system must include the OS/3 assembler.

The information contained in this manual is presented as follows:

- Section 1. Basic Concepts

  Describes the overall concepts and procedures in writing a user program that uses the physical interface of ICAM. Introduces the control packet (table) format (CPIOCP) and imperative macroinstruction (CCRCALL) necessary to interface your program with ICAM software. Also discusses how to generate an ICAM network for a communications physical interface and how to load it.

- Section 2. Writing a User Program

  Describes how to request the network and lines, initialize the single line communications adapter (SLCA), perform the transmit/receive functions, and release the lines and network.

- Section 3. Additional User Program Features

  Discusses additional useful functions available via the physical interface, such as reading the SLCA words and tables, reading the line link table, chaining, and automatic commands.

- Section 4. Single Line Communications Adapter Subsystem

  Describes the single line communications adapter hardware subsystem and the software procedures necessary to initialize it. The description includes construction of the port control words, character detect tables, and character interpretation table required to support the communications lines and line disciplines necessary for remote devices.

- Section 5. SLCA Table Initialization

  Describes the initialization parameters for an SLCA pertinent to specific SPERRY remote device handlers. This includes the control character detect tables, control character interpretation tables, and port control words for terminals supported by SPERRY software.

- Appendix A. Control Tables

  Describes the control packet and line link table used to present the required information to the system software. The tables are described via the dummy control section (DSECT) labels for the necessary fields and field settings.

As one of a series, this manual is designed to guide you in programming and using the OS/3 integrated communications access method. Depending on your need, you may wish to refer to the current version of one of the other ICAM manuals. Complete manual names, their ordering numbers, and a general description of their contents and use is as follows:

- Integrated Communications Access Method (ICAM) Concepts and Facilities, UP-9744

  Provides an overview of the facilities offered by ICAM including the hardware supported, the types of programs supported (assembler, COBOL, and RPG II), and the services provided (polling, queueing, buffering, etc).

- ICAM Network Definition and Operations User Guide, UP-9745

  Describes how to define an ICAM network, submit it to the system generation procedure, and load and operate the resulting ICAM symbiont. Many sample network definitions are provided to make it easier to define your ICAM network. In addition, most of the required operational functions are described. These functions include loading ICAM, establishing a dynamic session from a terminal, communicating with ICAM, etc.

- ICAM Standard MCP (STDMCP) Interface User Guide, UP-8550

  The standard interface is a logical interface that provides a general communications capability with message queueing and a message processing capability.

  This user guide provides all of the macroinstructions, programming requirements, and terminal information you need to interface with the standard interface.

You will need this user guide only if you are writing your own communications program. Programs that use the standard interface directly must be coded in basic assembly language (BAL), and your system must include the OS/3 assembler.

- ICAM Direct Data Interface (DDI) User Guide, UP-8549

  The DDI interface commonly supports ICAM utility programs and programs written in the RPG II language. If you are using an ICAM utility only, or if your program is written in RPG II, you won't need this user guide because the utility programs and the RPG II compiler automatically convert any requests by your program to the proper instructions needed to work with this interface.

  The DDI interface also enables you to write your own specialized communications program to work with it. If you do this, you must take care of your own message buffering and queueing. If you write a program to interface directly with direct data interface, it must be written in basic assembly language.

- ICAM Utilities User Guide, UP-9748

  Describes the utilities provided by ICAM. These utilities:

  - Enable your processor to emulate a SPERRY 1004 Card Processor System, SPERRY DCT 2000 terminal, or IBM 2780 terminal
  - Provide a facility to enable you to submit batch jobs from a remote terminal
  - Provide the capability to produce printed reports from journal files
  - Supply the software to create a module that converts communications requests in your COBOL program to instructions recognizable by the ICAM standard interface
  - Describe how to run RPG II under ICAM as a utility
  - Describe how to dump and list the contents of an SLCA
  - Help locate the cause of operational problems

- ICAM Message Processing Procedure Specification (MPPS) User Guide, UP-8946

  MPPS enables you to write message processing routines and include them in your ICAM network. This makes it possible for ICAM to analyze and process input messages before they are made available to your program, including the establishment of priority based on message content. Message processing routines can also be used to process output messages, including rerouting, if necessary, due to hardware and software error conditions.

  You do not need to include message processing routines in your network – they are totally optional; hence your need for this user guide depends on your requirements.

- ICAM Programmer Reference, UP-9749

  This reference summarizes the information found in the other ICAM manuals. No introductory information or examples are given; however, it is a useful document when you are familiar with ICAM and you need a quick reference to macroinstructions, formats, and tables.

- NTR Utility User Guide, UP-9502

  Describes how the System 80 can operate as a remote job entry/batch terminal to a SPERRY Series 1100 System via ICAM.

  The utility permits operation of reader, punch, and printer device-dependent files. It also supports user-own-code tasks to process device-independent files (e.g., tape, disk, paper tape).

- Remote Terminal Processor (RTP) User Guide, UP-8990

  The remote terminal processor is a data communications program that permits your SPERRY System 80 processor to function as a remote job entry terminal to one or more IBM host processors. Using the SPERRY OS/3 integrated communications access method (ICAM) software, the remote terminal processor enables you to:

  - send jobs to an IBM host;
  - transmit and receive files on tape, punched cards, or diskette;
  - send messages to the central site; and
  - receive output data and console messages from the IBM host.

  Remote terminal processor operations are directed from the OS/3 system console.

Contents

TABLES

1. Basic Concepts

1.1. GENERAL

Data communications – very simply stated – is the transmission of computer data via communications lines. These lines may be directly connected from one piece of equipment to another (dedicated) or they may operate across telephone transmission switching equipment (switched).

Just as the ordinary telephone converts vibrations (voice) to electrical impulses, an attachment to the telephone, called a data set, converts data bits to electrical frequencies. The household telephone uses diaphragms and magnets to oscillate at audio frequencies and superimposes this signal on a carrier wave. The data set uses a modem to modulate or demodulate a data signal and superimposes it on a carrier wave.

Data input/output equipment (I/O devices) may be connected to a communications line via a data set or they may be directly connected to the line. When these I/O devices terminate a communications line directly, they contain their own internal modem and are commonly known as terminals. There are a large number of I/O devices that can be used as terminals, such as printers, punches, magnetic and paper tapes, disks, and video display terminals. The terminals themselves may have additional I/O equipment attached, such as tape cassette drives and diskettes. This type of equipment is designated as an auxiliary device. Out of this diversity of equipment, analysis will show that there are a number of functions common to all that can be woven together to form a common structure for data communications software. (See Figure 1–1.) The first of these functions is a line request. In software terms, this means that we must specify the kind of line it is that we want. This may include the mode of communication, such as batch or interactive, the line speed and the line type (switched versus dedicated, half duplex versus full duplex, automatic versus manual dial, etc).

In the following paragraphs, we use a simple UNISCOPE display terminal protocol to show how data communications works.

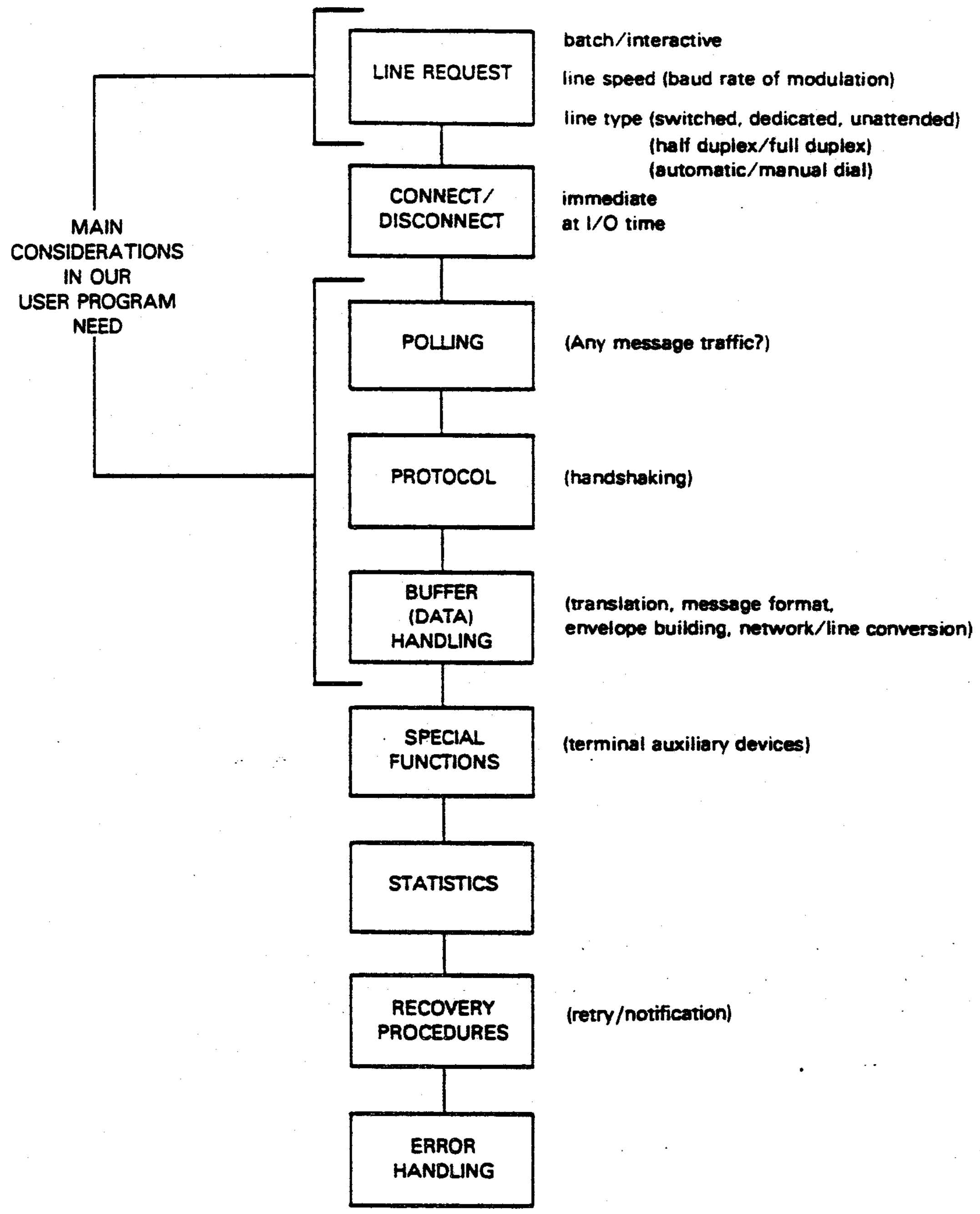

MAIN CONSIDERATIONS IN OUR USER PROGRAM NEED
LINE REQUEST
batch/interactive
line speed (baud rate of modulation)
line type (switched, dedicated, unattended)
(half duplex/full duplex)
(automatic/manual dial)
CONNECT/ DISCONNECT
immediate
at I/O time
POLLING
(Any message traffic?)
PROTOCOL
(handshaking)
BUFFER (DATA) HANDLING
(translation, message format, envelope building, network/line conversion)
SPECIAL FUNCTIONS
(terminal auxiliary devices)
STATISTICS
RECOVERY PROCEDURES
(retry/notification)
ERROR HANDLING

Figure 1–1. Common Structure for Data Communications Software

Once we have requested the specific type of line we require and a connection is made, we must determine how we can converse with the other end of the line. That is, we must have set up some common rules and formats that both ends of the line are aware of and agree on. We must both speak the same language and play the same game. Thus, we have line protocol (the rules) and message envelopes (the format). The protocol (Figure 1–2) determines the sequence or order in which our messages are sent and received and is commonly known in the trade as handshaking. The message envelope (Figure 1–3) contains a prefix and suffix to the message transmitted. This contains basic information about the message, such as what kind of message it is, where it is going, and what is to be done with it.

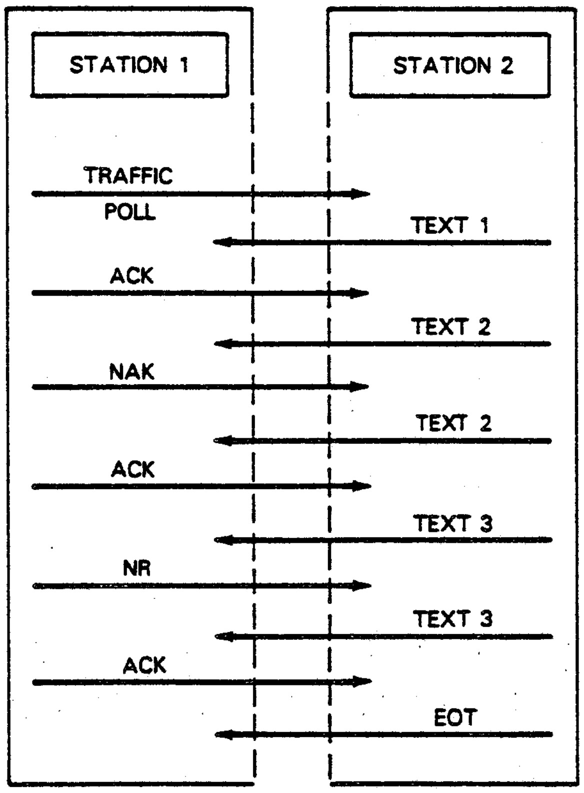


Figure 1–2. Typical Example of Line Protocol

Now that we have formulated a basic set of rules and structure, how do we get our message or conversation going? Well, in a telephone conversation, when we call someone, we generally say "Hello" to find out if anyone is there. The same holds true for our data conversation – we must first send some kind of message to the effect:

Is anyone out there?

Does anyone have a message for me?

or

I'm sending a message, is anyone awake?

This type of message is called a poll. It expects some kind of reply or response, such as an acknowledgment:

Yes, I'm here.

I have a message for you and here it is.

or

I'm awake – go ahead.

Normally, you would respond with a "yes, I am" by sending text. I would then respond with an acknowledgment message (ACK) that says: "Got your first text message – thank you – please send more." You would then send your second text message.

Suppose, however, that I did not receive your second text message correctly – something may have been garbled or did not make sense in the order of things. I would then return a negative acknowledgment (NAK) and you would transmit the second text message again.

Another case might be that you would send me a message and I would not respond. In this case, your program would have to time out and retransmit until it received an acknowledgment. After a certain number of retries and failures, it would disconnect or send an EOT.

If you sent me all the messages you had or I signed off on my end, an EOT message would perform the disconnect notice.

Now, if I'm going to send a poll message and expect an acknowledgment (ACK) or some text, what would these messages really look like? Well, let's look at some specific formats. (See Figure 1–3 and Table 1–1.) First, if I am in synchronous operation, my prefix (header) must have at least two synchronous characters (SYNCs), which are provided by the the single line communications adapter (SLCA).

If I'm in asynchronous operation, I don't have to send these characters, because the transmission equipment itself includes a transmission characteristic with each message transmitted.

My header would now have a start-of-header (SOH) character preceding a destination address that consists of a RID, SID, and DID. A RID specifies a remote installation identifier, a hexadecimal number representing an installation such as the O'Hare Airport or the World Trade Center. A SID specifies a particular terminal station identifier, a hexadecimal number representing a station such as visual display 14. A DID specifies a hexadecimal device address of a specific auxiliary device that may be connected to a specific terminal station. This device address may represent a printer, a diskette, a tape cassette, or other supporting auxiliary devices.

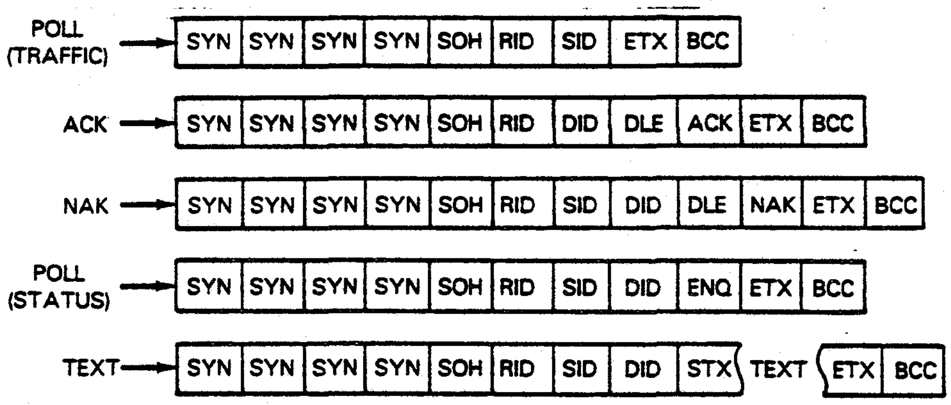

POLL (TRAFFIC)
SYN SYN SYN SYN SOH RID SID ETX BCC
ACK
SYN SYN SYN SYN SOH RID DID DLE ACK ETX BCC
NAK
SYN SYN SYN SYN SOH RID SID DID DLE NAK ETX BCC
POLL (STATUS)
SYN SYN SYN SYN SOH RID SID DID ENQ ETX BCC
TEXT
SYN SYN SYN SYN SOH RID SID DID STX TEXT ETX BCC

Figure 1–3. Typical Communications Message Envelopes

Table 1–1. Control Character Description (Part 1 of 2)

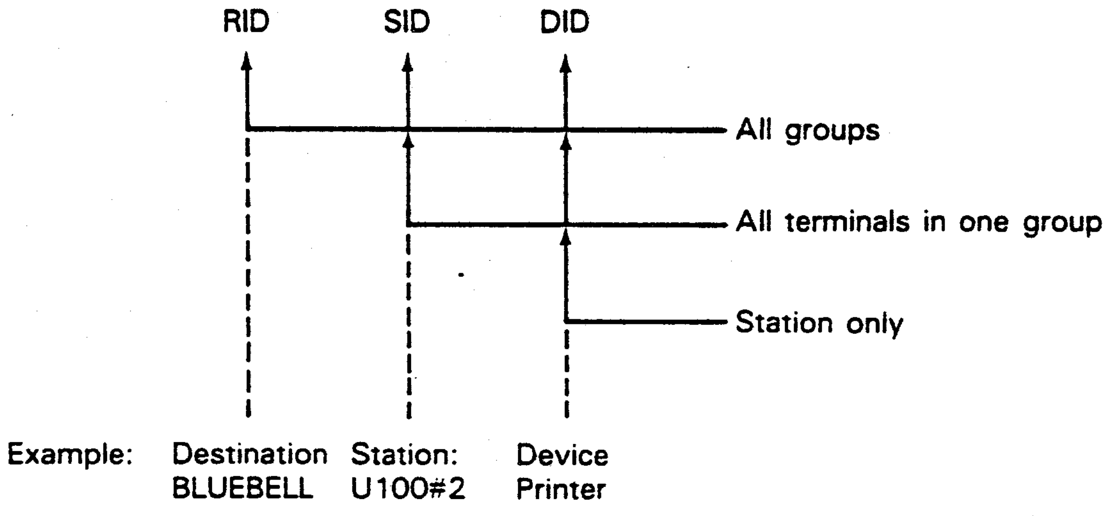
Character	ASCII Hex. Value	Meaning
ACK	31	Acknowledgment – An affirmative reply response used to positively acknowledge receipt of a message. Indicates that line conditions and status of message are normal.
BCC	–	Block check character – A character added at the end of a message or transmission block to facilitate error detection.
DID	20–7F	Device identifier – DID characters specify an auxiliary device. The DID is not a function of the device type but of the station configuration.
DLE	10	Data link escape – A communications control character that will change the meaning of a limited number of contiguously following characters. It is used exclusively to provide supplementary controls in data communications networks.
ENQ	05	Enquiry – A communications controller character used in data communications systems as a request for a response from a remote station. It may be used as a "Who are you?" (WRU) to obtain identification, or may be used to obtain station status, or both. (ENQ is the basic character for the status poll.)
EOT	04	End of transmission – A communications control sequence used to indicate the conclusion of a transmission that may have contained one or more text and any associated headings. The sequences may also be used to indicate a "no traffic" in response to a poll.
ETX	03	End of text – A communications control character used to terminate a sequence of characters started with STX or SOH and transmitted as an entity.

Table 1–1. Control Character Description (Part 2 of 2)

Character	ASCII Hex. Value	Meaning
NAK	15	Negative acknowledgment – A negative reply response used to indicate receipt of a garbled message. The condition may reflect bad status, block check errors, or line problems.
RID	(20–4F)	Remote identifier – RID characters are used to address a remote station, a message switching center, or a group of terminals. The number of RIDs may vary. A minimum address must at least contain a RID character.
SID	(50–6F)	Station identifier – SID characters specify a specific terminal that may be connected to the communications line by appropriate facilities or via a message switching center. The number of SID characters in a station address may vary depending on system requirements.
SOH	01	Start of header – A communications control character sometimes used at the beginning of a sequence of characters that constitutes a machine-sensible address or routing information. This sequence is referred to as the header.
STX	02	Start of text – A communications control character that precedes a sequence of data characters that is to be treated as an entity. Such a sequence is referred to as TEXT. STX may be used to terminate a sequence of characters (headings) started by SOH.
SYN	16	Synchronous idle – A communications control character used by a synchronous transmission system to provide a signal from which synchronism may be achieved or retained.

The RID, SID, DID address trio shown in Figure 1–3 may also make use of a general identifier (GID). This GID may be used in place of the RID, SID, or DID. For the RID, a GID = 20_{16} is recognized by all remote installations as its RID. For the SID, a GID = 50_{16} is recognized by all terminals at an installation. For the DID, a GID = 70_{16} merely addresses the station without selecting an auxiliary device.

Summary of GIDs:

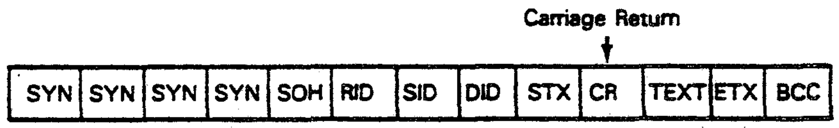


Following the address trio are control characters that determine the type of message being transmitted. For instance, a poll is the name given to a particular message header that solicits or requests input. If an end-of-text (ETX) character was encountered immediately after the address trio, the message is designated as a traffic poll. A traffic poll is one that simply requests the other terminal's identification. If an ENQ character followed by an ETX was encountered, the message would be designated as a status poll. A status poll is one that requests a response from the terminal. If the message is not a poll message, the control character next seen is a data link escape (DLE) character. This signifies that some type of supplementary control information is coming. (See Figure 1–4.) An ACK followed immediately by an ETX signifies that the message is a positive acknowledgment (ACK). If the message is a negative acknowledgment (NAK), a NAK character follows the DLE.

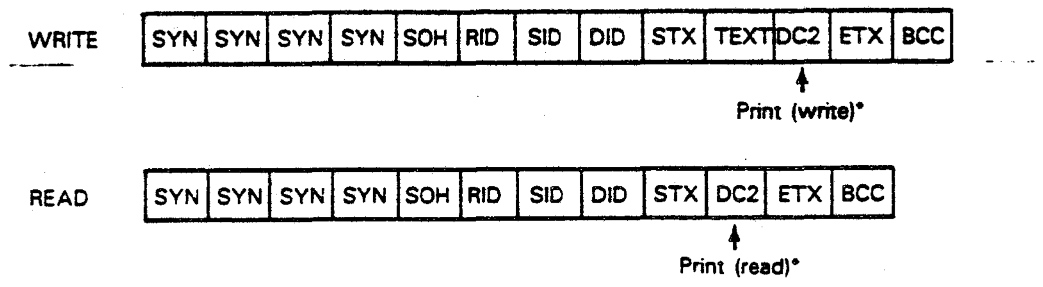
If the message is a text message, a start-of-text (STX) control character follows the DID without a DLE character. This is followed by the text and ends with an end-of-text (ETX) character (tail). All messages then end with a BCC character, which is a parity check character used to detect transmission errors.

The message header and the tail together comprise what is known as a message envelope. It is normally the job of a remote device handler to build this envelope on transmittals and strip it on reception of messages. In the physical interface, it is the job of your program. Once this set of rules and regulations has been solidified, the system is ready to receive data and must now consider how to handle the data coming in and going out of the buffer areas.

Example 1 – Sending an auxiliary device command



Example 2 – Sending to a UNISCOPE tape cassette auxiliary device



*The DID determines interpretation as either a read-head or write-head command.

Figure 1–4. Typical Communications Message Envelopes with Control Characters (Part 1 of 2)

Example 3 – Sending to a UNISCOPE terminal

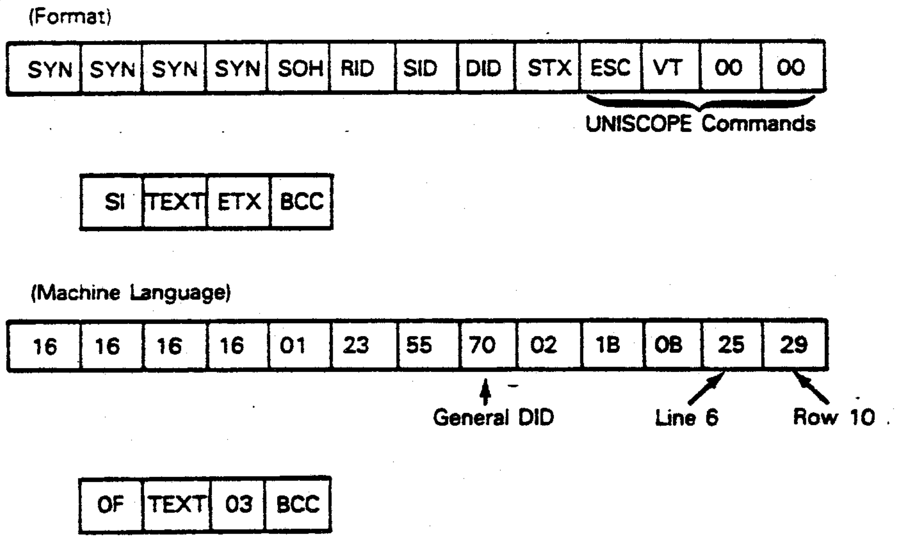
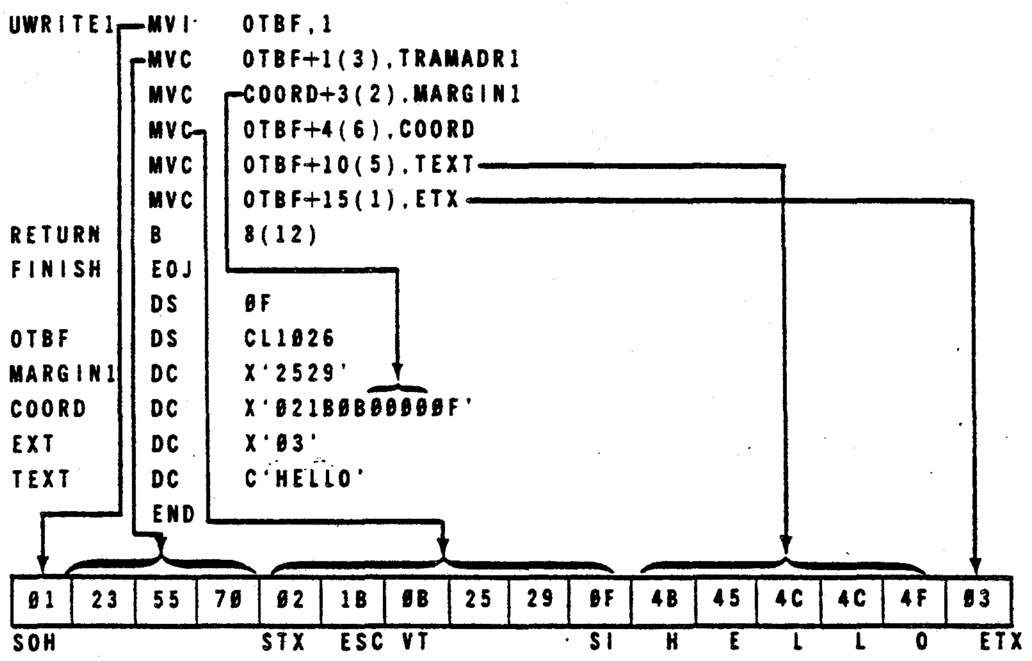


Figure 1–4. Typical Communications Message Envelopes with Control Characters (Part 2 of 2)

In Figure 1–4, example 3 shows a typical message envelope for a UNISCOPE visual display terminal. Following it is a hexadecimal machine language representation of the format. The hexadecimal code equivalents are obtained from Figure 1–3 and Tables 1–1 and 1–20. The following example is typical of what your assembler source code must do to construct this message envelope:

```
BEGIN    START 0
         .
         .
         .
         BAL   12,UWRITE1                      Set up message
TRMADR1  DC    X'23557000'                     Destination terminal address
         BAL   12,SEND                         Go to send routine
         .
         .
         .
         B     FINISH
UWRITE1  MVI   OTBF,1
         MVC   OTBF+1(3),TRAMADR1
         MVC   COORD+3(2),MARGIN1
         MVC   OTBF+4(6),COORD
         MVC   OTBF+10(5),TEXT
         MVC   OTBF+15(1),ETX
RETURN   B     8(12)
FINISH   EOJ
         DS    0F
OTBF     DS    CL1026
MARGIN1  DC    X'2529'
COORD    DC    X'021B0B00000F'
EXT      DC    X'03'
TEXT     DC    C'HELLO'
         END
```



As you can see, in the example we chose a particular installation and station with no auxiliary device addressed (235570). This was placed into our output buffer following the start of headers. Text was introduced by the start of text character (STX=02). Then we set the display terminal screen coordinates in front of our text. This control message is a cursor positioning code sequence but could just as well have been any display terminal function listed in Table 1-2. The cursor control code sequence was terminated with the shift in (SI=0F) character, followed by the text. Following the text, we terminated the message with an end of text. Finally, we would branch to our send routine to transmit the message.

This is the typical kind of communications message for a display terminal that would be transmitted across a physical interface via software. The protocol and format comprising this message is known as the remote device handler discipline or the line discipline. This line discipline will vary for each device. It is the objective of ensuing sections to describe how this line discipline is constructed at machine level by a user program and how it interfaces with OS/3 at this physical level.

Table 1-2. Control Codes and Control Code Sequences for a UNISCOPE Visual Display Terminal (Part 1 of 2)

Function	Code to Terminal	Code From Terminal	Hexadecimal ASCII	Hexadecimal EBCDIC
Cursor positioning	ESC VT Y X SI	-	1B0BYX0F	270B00000F
SOE positioning	-	ESC VT Y X NUL SI	1B0BYX000F	
Cursor return (new line)	CR	CR	0D	0D
Erase to end of display	ESC a	-	1B61	2781
Erase to end of line	ESC b	-	1B62	2782
Delete in line	ESC c	-	1B63	2783
Delete in display	ESC C	-	1B43	27C3
Insert in line	ESC d	-	1B64	2784
Insert in display	ESC D	-	1B44	27C4
Scan left	ESC g	-	1B67	2787
Scan right	ESC h	-	1B68	2788
Scan down	ESC i	-	1B69	2789
Scan up	ESC f	-	1B66	2786
Character erase (space)	SP	SP	20	40
Tab	HT	-	09	05
Tab stop set	ESC HT	-	1B09	2705
Tab stop	-	HT	-	-
Message waiting	BEL	-	07	2F
Cursor to home	ESC e	-	1B65	2785
Insert line	ESC j	-	1B6A	2791
Delete line	ESC k	-	1B6B	2792
Erase field	ESC K	-	1B4B	27D2
Erase display	ESC M	-	1B4D	27D4
Request processor message	-	BEL	07	2F
Start blink marker	FS	FS	1C	1C
End blink marker	GS	GS	1D	1D
Lock keyboard	DC4 or ESC DC4	-	14	3C
Print	DC2	-	12	12
Print transparent	ESC DC2	-	1B12	2712

Table 1–2. Control Codes and Control Code Sequences for a UNISCOPE Visual Display Terminal (Part 2 of 2)

Function	Code to Terminal	Code From Terminal	Hexadecimal ASCII	Hexadecimal EBCDIC
Start of entry (SOE)	RS	RS	1E	1E
Shift in (note 6)	SI	-	0F	0F
Shift out	SO	-	0E	0E
Line feed	LF	LF	0A	25
Form feed	FF	FF	0C	0C
Transmit (unprotected)	DC1	-	11	11
Transmit display	ESC DC1	-	1B11	2711

1.2. GENERAL DESCRIPTION OF THE PHYSICAL INTERFACE

An interface is basically defined as an interconnecting link between two systems. In data processing, this link may be either hardware or software. A *physical* interface is commonly considered to mean the link *between* the hardware (machine) and software (programming). It is often referred to as machine-level programming, since it has direct access to the hardware. In the integrated communications access method (ICAM), the communications physical interface (CPI) is the fundamental link between the single line communications adapter hardware and OS/3 communications software. This physical interface in ICAM is primarily intended to allow *experienced programmers* to write a user program at the physical level. In most instances it is used only when it is necessary to control a device or terminal that is not supported by current ICAM software.

The physical interface is a logical and physical packet-driven interface that gives you access to every command and status capability of the single line communications adapter (SLCA) hardware subsystem. The SLCA provides the hardware interface to a single communications line. The information necessary to control these communications lines is initially loaded into the SLCA. At the physical interface level, you must load this information via your user program.

In the basic System 80 Models 3–6, up to two SLCAs are supported. With a single input/output microprocessor attached, Models 3–6 support one to eight SLCAs and Model 8 supports up to 14 SLCAs. With a dual input/output microprocessor attached, Model 8 supports up to 28 SLCAs.

The physical interface is the lowest level of ICAM support. In essence, it merely provides activity scheduling within OS/3. It takes up the least amount of storage but also provides the least amount of services. Thus, the bulk of the work is left to you. In this interface, *you* must perform all the necessary initialization of the hardware. *You* must request and release the network and lines to be used, provide the device and line discipline, process the data, and perform all error detection and recovery procedures.

ICAM merely provides you with a formatted table and the necessary macroinstructions to schedule your communications task. In addition, *you* must construct the tables (packets) within your user program and supply the proper parameters in the proper sequences. Your user-own-code program is then submitted as a normal user job via a job control stream.

If you need to write a program at this level, you should be thoroughly knowledgeable of assembler language programming and data communications generally, and single line communications adapter hardware specifically. It is strongly recommended that you become familiar with current versions of the following Sperry manuals:

- integrated communications access method (ICAM) concepts and facilities, UP-9744
- interfacing a remote device handler programmer reference, UP-8424
- input/output microprocessor programmer reference, UP-8909

The physical interface actually has the dual role of servicing physical-level user programs and linking higher-level user programs via remote device handlers. This manual is primarily written for the physical-level user program; however, it does contain a great deal of information concerning remote device handlers, since the two are so closely related.

1.3. TERMINOLOGY USED IN THIS MANUAL

To retain a link with former versions of this manual, yet simplify terminology, you can also refer to the CPI as the physical interface. The communications physical input/output control packet (CPIOCP) is also called the control packet. The communications control routine user program is, simply, the user program (your program).

1.4. PHYSICAL INTERFACE MACROINSTRUCTIONS AND PACKETS

If you are familiar with data management or the use of macroinstructions, you probably have some understanding of the relationship between imperative and declarative macroinstructions.

The imperative macroinstruction causes the generation of executable code sequences in the user program. These code sequences are the link between your user program and the input/output portions of the supervisor (in our sphere of interest, the extension of the supervisor known as ICAM). The imperative macroinstructions are used to request services of the supervisor and direct operations of the user program. Operands used with the imperative macroinstruction point to the file described by a declarative macroinstruction and specify the processing action to be taken.

Access to the physical interface is supported by ICAM solely via the imperative macroinstruction CCRCALL, which defines entry to the physical interface through the communications control routines and identifies the associated control packet. This macroinstruction identifies a control packet that must be aligned on a full-word boundary. (The CYIELD and CAWAKE macroinstructions are available to the physical interface user to release or activate the communications task.) Before execution of the instructions generated by a CCRCALL, the referenced control packet fields must have been set according to Appendix A.

Format:

LABEL	△OPERATION△	OPERAND
[symbol]	CCRCALL	{CPIOCP-label / (1)}[,IRL=NO]

Label:

symbol
: An alphanumeric character string up to eight characters long uniquely identifying this instruction.

Parameters:

CPIOCP-label
: Identifies the symbolic name of the control packet.

(1)
: Indicates that the address of the control packet to be accessed is contained in register 1. The high-order byte of the register must be zero (X'00').

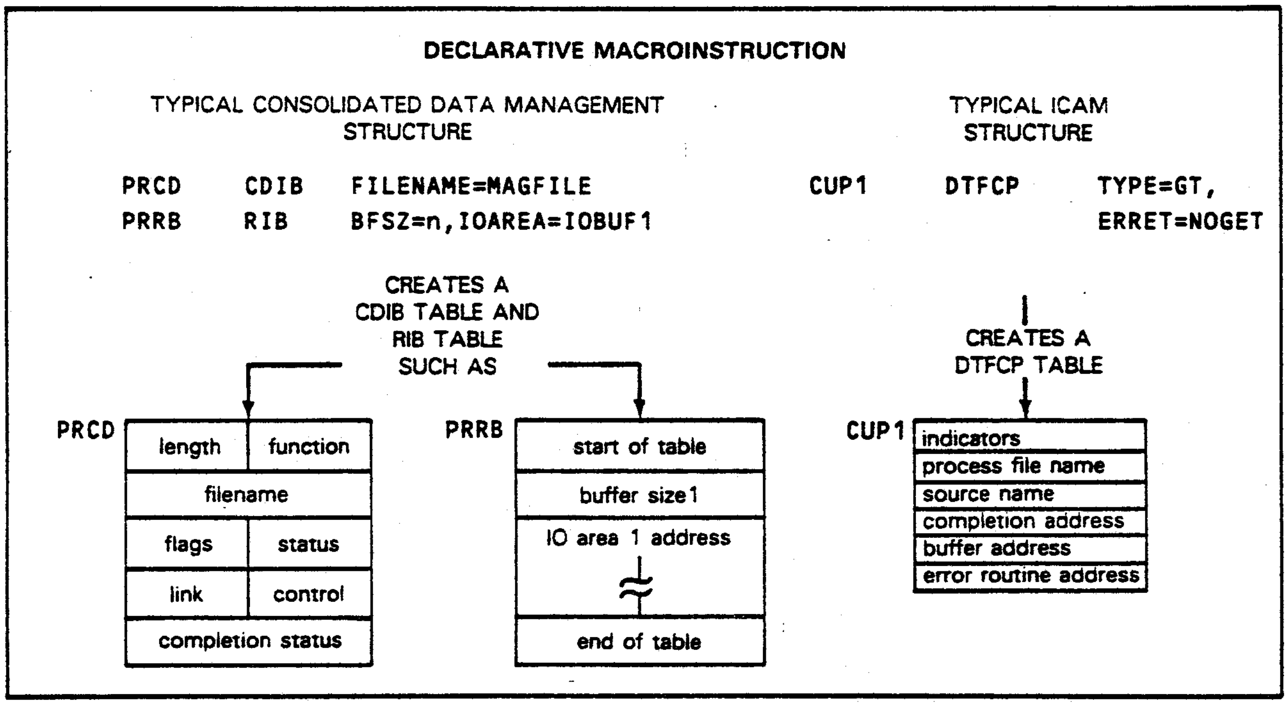
IRL=NO
: May be used by the remote device handler to indicate that the control packet does not request immediate return line (IRL) of control following execution of the instructions generated by the macroinstruction.

NOTE:

The CCRCALL macroinstruction generates a service request interrupt for the physical user program, but it generates a BALR instruction for remote device handlers, since they are an integral part of ICAM.

The declarative macroinstruction provides inline expansion of nonexecutable code, such as define constant (DC) and define storage (DS) statements. Declarative macroinstructions are used to allocate areas in main storage to describe all aspects of a file to be processed.

Figure 1–5 illustrates the parallel structure of imperative and declarative macroinstructions that exists between data management and ICAM, since ICAM, in effect, is a set of data-management-type macroinstructions used to control communications input/output operations.



```
PRCD     CDIB     FILENAME=MAGFILE              CUP1     DTFCP     TYPE=GT,
PRRB     RIB      BFSZ=n,IOAREA=IOBUF1                             ERRET=NOGET
```

IMPERATIVE MACROINSTRUCTION

TYPICAL CONSOLIDATED DATA MANAGEMENT STRUCTURE

```
DMOUT   PRCD,BUFFOUT
```

or

```
DMINP   PRCD,BUFFIN
```

CREATES A SERVICE CALL SUCH AS

```
DC      DY(Ø)
L       1,=A(PRCD)
L       Ø,=A(BUFFOUT)
MVI     49(1),X'2Ø'
L       15,52(1)
SVC     98
```

TYPICAL ICAM STRUCTURE

```
PUTCP     CUP1,BUFFOUT
```

or

```
GETCP     CUP1,BUFFIN
```

CREATES A SERVICE CALL SUCH AS

```
CNOP      Ø,4
DC        X'Ø7ØØØ7ØØ'
SVC       98
DC        AL2((Ø*/8++5)
EOJ
SYSTEM
DS        ØH
SVC       28
SVC       Ø9
```

Figure 1-5. Comparison of Data Management and ICAM Structures

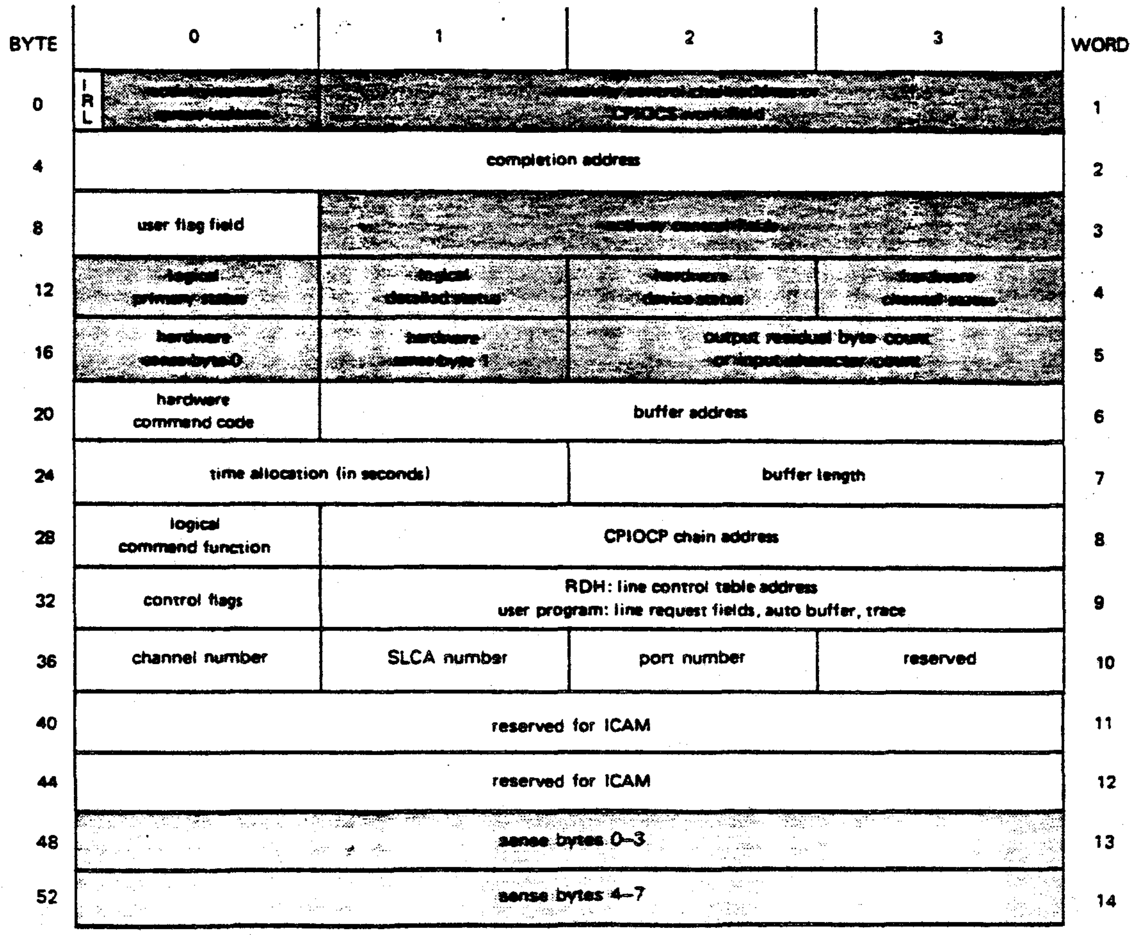
The declarative and imperative macroinstructions operate together similarly for both data management and most levels of ICAM. However, at the physical level, there is no declarative macroinstruction to construct the control table or control packet. You must do this yourself via assembler DC and DS instructions. For example, to construct the DTFCP table (CUP1) in Figure 1–5, we might perform the following code:

Example 1:

```
CUP1     DC     F'Ø'                     Indicators
         DC     A(PRF1)                  Process file name
         DC     A(SRC1)                  Source name
         DC     A(GOODIO)                Completion address
         DC     A(BUFFIN)                Buffer address
         DC     A(ERRI)                  Error routine address
```

Thus, instead of using the declarative macroinstruction to format and construct the needed information packet, we merely build it using standard assembler instructions.

Let's briefly look at the *actual* control packet (Figure 1–6) that is used to pass the required information across the physical interface. This is shown functionally and in detail in Appendix A, but here we just want to look at its structure.



BYTE	0	1	2	3	WORD
0	IRL [illegible]	CPIOCS work field			1
4	completion address				2
8	user flag field	[illegible]			3
12	logical primary status	logical detailed status	hardware device status	hardware channel status	4
16	hardware sense byte 0	hardware sense byte 1	output residual byte count or input character count		5
20	hardware command code	buffer address			6
24	time allocation (in seconds)		buffer length		7
28	logical command function	CPIOCP chain address			8
32	control flags	RDH: line control table address user program: line request fields, auto buffer, trace			9
36	channel number	SLCA number	port number	reserved	10
40	reserved for ICAM				11
44	reserved for ICAM				12
48	sense bytes 0–3				13
52	sense bytes 4–7				14

LEGEND:
▨ System-supplied parameters

Figure 1–6. Communications Control Packet Basic Format

We could build this control packet with the following typical coding, which we might call "hard coding":

Example 2:

```
OTPTPKT  DC    F'0'                          Priority/activity control
         DC    A(COMPADDR)                   Completion address
         DC    F'0'                          Flag field
         DC    F'0'                          Status field
         DC    F'0'                          Sense bytes
         DC    XL1'00'                       Clear command code
         DC    AL3(OUTBUFF)                  Output buffer address
         DC    H'3'                          3-sec time allocation
         DC    H'47'                         Buffer length
SOFTFUNC DC    YL1                           Logical command
         DC    XL3'00'                       Chain address
         DC    F'0'                          Operational flags
SLCANO   DC    XL4'00000000'                 Channel, SLCA number and port number
         DC    F'0'                          Reserved for ICAM
         DC    F'0'                          Reserved for ICAM
         DC    F'0'                          System 80 sense bytes
         DC    F'0'                          System 80 sense bytes
```

When we wish to perform a particular communications input/output function, we now only need an imperative macro that points to this control packet. However, in many programs, multiple control tables are desirable and the hard coding method of constructing them is both repetitive and time consuming. The use of dummy sections (DSECTs) reduces the amount of coding required by providing you with all the symbolic names and values for accessing and setting any portion of the packet that might be required.

1.5. USE OF DUMMY CONTROL SECTIONS (DSECTS)

A special assembler directive is available in OS/3 to generate dummy control sections (DSECTs) that can be shared by a user program. A DSECT permits you to define symbolic addresses in your user program without requiring any additional storage allocation for them. This allows you to map or overlay a storage area in your program with a set of symbolic addresses (labels) that were previously constructed for you in a dummy control section of the system software. Thus, you can map a storage area in your program with a set of labels without allocating space within your program.

Displacement and base addresses are calculated by the assembler for each symbol defined by the DSECT, but they do not become part of your object program. This facility is especially useful in ICAM for control table access and manipulation, thereby considerably reducing the size and complexity of the programmer's coding task. This is most evident in the physical interface user program. This is because, in the physical interface, the use of multiple control tables for various functions is quite advantageous. The format and size of the tables may be identical, but the contents differ for each function. Using DSECTs, you need to perform only a simple allocation statement in your user program for each table required and then map it with the DSECT.

For instance, suppose you wanted to construct an imaginary control packet for a network request and another for a transmit function. Instead of labeling each word and byte in each table, as well as all the commands, flags, and status settings, you could use a single precoded DSECT to map any number of packets. As an example, let's make up a single fictitious DSECT, such as the following:

Example 3:

```
T#PKT    DSECT
T#FUNC   DS    XL1                   Function (command)
T#FLGS   DS    XL1                   Flags
T#RESV   DS    XL2                   Reserved
T#BUFADR DS    XL4                   Buffer address
T#CMPLAD DS    XL4                   Completion address
T#PRIM   DS    XL1                   Primary status
T#DETL   DS    XL1                   Detailed status
T#SENS1  DS    XL1                   Sense byte 1
T#SENS2  DS    XL1                   Sense byte 2
*   EQUATES FOR TABLE SETTINGS
*                                    Commands
T#SEND   EQU   X'Ø1'                 Send
T#PDTR   EQU   X'Ø2'                 Set data terminal ready
T#EDI    EQU   X'Ø3'                 Enable data input
T#LREQ   EQU   X'2C'                 Line request
T#NREQ   EQU   X'2D'                 Network request
*                                    Status settings
T#END    EQU   X'Ø1'                 Message completion
T#PROGE  EQU   X'Ø2'                 Program error
T#PAR    EQU   X'Ø3'                 Parity error
T#BUSY   EQU   X'Ø4'                 Busy
```

If this fictional DSECT existed in the system software, you could call it in for availability to your user program immediately following the START instruction of your program, as follows:

```
START Ø
TN#DSECT T#PKT
```

where T#PKT is an operand that specifies the dummy control section.

This DSECT, in effect, would allocate a table of constants and equates in system storage that would functionally appear as:

T#FUNC	T#FLGS	TN#PRESV	
T#BUFADR			
T#CMPLAD			
T#PRIM	T#DETL	T#SENS1	T#PSENS2

These symbolic names could now be used in your program and you could very simply define multiple control packets by mapping or overlaying this DSECT on the control packet storage allocated in your program. For brevity, we'll assume just two packets: therefore, only two DS statements are required, but it could just as easily be a hundred.

```
SENDPKT  DS    4F'Ø'                  A 4-word control packet
                                           for transmit functions
NTRQPKT  DS    4F'Ø'                  A 4-word control packet
                                           for network request functions
```

To overlay the first packet, first load register 1 with the address of the storage area defined for our transmit control packet (SENDPKT). Then, use a USING statement to specify that register 1 is the cover register for the desired DSECT; e.g.:

```
LA    R1,SENDPKT
USING T#PKT,R1
```

This technique in effect maps (or overlays, if you prefer) the SENDPKT storage area with the labels of the constants and equates of the T#PKT dummy control section. You can now use these labels in your program to fill in the control packet with the information necessary to perform a specific function.

If we wished to use a network request function, we could do this simply by doing a

```
LA    R1,NTRQPKT
USING T#PKT,R1
```

and we would now have our NTRQPKT storage area mapped by our dummy control section.

Putting this all together, if such a DSECT was available, we might use it as in Example 4.

Example 4:

```
                START 0
                TN#DSECT T#PKT                   Call required DSECT
                BALR  R3,0                       Set program relative 0
      BEGIN     USING *,R3                       Begin program here
                USING T#PKT,R1                   DSECT covered by R1
                   .
                   .  } Application code
                   .
      REQNET    LA    R1,NTRQPKT                 Map packet
                MVI   T#FUNC,T#NREQ              Set command
                MVC   T#CMPLAD,=A(NREQCMP)       Set completion address
                B     ISSUE                      Issue I/O
                   .
                   .
                   .
      NREQCMP   EQU   *
                   .
                   .  } Status checking
                   .
                B     SEND
                   .
                   .
                   .
      SEND      LA    R1,SENDPKT                 Map packet
                MVI   T#FUNC,T#SEND              Set command
                MVC   T#BUFADR,=A(MSGBUF)        Set data address
                MVC   T#CMPLAD,=A(SENDCMPL)      Set completion address
                B     ISSUE                      Issue I/O
                   .
                   .
                   .
      SENDCMPL EQU    *
                   .
                   .  } Status checking
                   .
      ISSUE     CCRCALL (1)
                   .
                   .
                   .
                EOJ
      SENDPKT   DS      3F'0'
      NTRQPKT   DS      3F'0'
      MSGBUF    DS      16F'0'
                END
```

As you can see, first the network request control packet is mapped, the DSECT labels are used to fill in the packet with the relevant information, and a branch is made to issue the imperative macro. When the service request is completed, control returns at NREQCMP, which was specified as the completion address. At this point, the available DSECT labels could be used to perform status checking, such as:

```
CLI   T#PRIM,T#END           Good completion?
BE    CONTINUE
CLI   T#PRIM,T#PROGE         Program error?
  .
  .
  .
  etc
```

The program might then branch to SEND where it now maps the SENDPKT control packet and again fills in the relevant data via the DSECT labels provided.

It is important to thoroughly understand the use of dummy control sections before proceeding. As you will see in the following sections, this basic technique is used many times in writing a complete physical interface program.

1.6. GENERATING ICAM

You create your communications system as part of the system generation (SYSGEN) process outlined in the current version of the system installation user guide/programmer reference, UP-8839. You can have as many as 18 different ICAM symbionts that comprise your communications system; you create each one in a separate system generation.

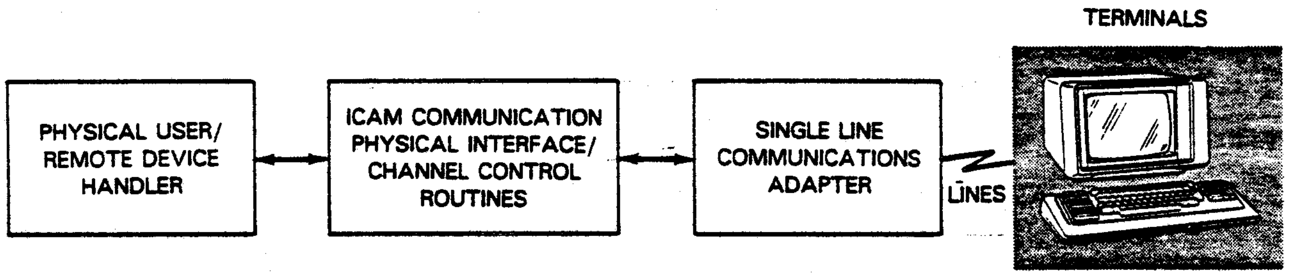
To generate an ICAM symbiont, you code the parameters in the COMMCT phase of system generation. The COMMCT parameters include the network definition macroinstructions and message control program (MCP) parameters described in 1.7. You submit the COMMCT phase to the parameter processor SG$PARAM, which validates the specifications and produces a listing. You then run the prefiled job control stream SG$COMMK to assemble and link the ICAM symbiont into the $Y$LOD library on your system resident disk volume.

1.7. CPI NETWORK DEFINITION

For your program to work with ICAM, you must define the network you are going to use. A network definition consists of a set of macroinstructions you submit to the COMMCT phase of the system generation process of the operating system. The ICAM load module created by this process is a symbiont. When you load this symbiont, it becomes part of the supervisor.

A communications physical interface network is a dedicated network – that is, the resources assigned to it are available only to *your* program.

The following shows the arrangement of physical users in a communications physical interface environment.


TERMINALS
PHYSICAL USER/ REMOTE DEVICE HANDLER
ICAM COMMUNICATION PHYSICAL INTERFACE/ CHANNEL CONTROL ROUTINES
SINGLE LINE COMMUNICATIONS ADAPTER
LINES

Since only the physical interface and channel control routines of ICAM are provided in a communications physical interface environment, your network definition is extremely simple – only two macroinstructions are involved: CCA and ENDCCA.

Format:

LABEL	△OPERATION△	OPERAND
PIOCS	CCA	USERS=n

Label:

PIOCS
: Required for a communications physical interface network only.

Parameter:

USERS=number
: Applies only to the communications physical interface and is the decimal number of physical user programs that will use the physical network.

Example:

```
PIOCS    CCA    USERS=2
         ENDCCA
```

The message control program (MCP) section of the COMMCT phase specifies the volume that the ICAM symbiont is to be placed on and the name of the symbiont. In addition, the CACH parameter identifies each SLCA accessed or supported by the ICAM symbiont. If you specify half-duplex or full-duplex mode operation in word 10 of the CPIOCP, you must also specify the corresponding mode in the CACH parameter. For instance, if you specified asynchronous, 9600 baud, full-duplex mode in the port control word and CPIOCP, the corresponding COMMCT would look like:

```
COMMCT
PIOCS   CCA    USERS=2
        ENDCCA
        MCP
               MCPNAME=M3
        CACH=(08,9600,FULL)                 SLCA#1,9600 baud,full-duplex
END
```

Physical interface network definitions must be the last ones specified when a multiple CCA ICAM is generated.

1.8. ICAM INITIALIZATION

You must load ICAM before you can execute any communications user programs. ICAM resides in main storage until the last communications user program is terminated (unless you specify the KEEP parameter). Then ICAM shuts itself down.

The KEEP parameter keeps the ICAM symbiont loaded until cancelled by the system operator or if ICAM experiences an unrecoverable error. For example, if you load ICAM symbiont M3 by keying in:

```
M3 KEEP
```

ICAM remains loaded after all user programs are terminated.

1.8.1. Loading an ICAM Symbiont

Always load ICAM in an idle system to avoid main storage fragmentation. You load ICAM from the system console by keying in the operator command:

$\begin{Bmatrix}\text{Cn}\\\text{Mn}\end{Bmatrix}\triangle\text{[KEEP]}$

where:

`Cn` or `Mn`
: Is the ICAM symbiont name specified on the MCPNAME parameter in the COMMCT phase of system generation.

`KEEP`
: Keeps the ICAM symbiont loaded until cancelled by the system operator or ICAM suffers an unrecoverable error.

You can also load ICAM from a job control stream with the statement:

```
// CCΔ'{Cn}ΔKEEP'
       {Mn}
```

This allows you to load ICAM by means of a workstation instead of asking the console operator to do it for you.

You can use the CC job control statement in two ways:

1. In a job control stream that only loads ICAM:

```
// JOB jobname
//ΔCCΔ'{Cn}ΔKEEP'
        {Mn}
/&
```

2. In a job control stream that executes your program. The CC job control statement immediately follows the JOB statement.

NOTE:

Even if you load ICAM by job control, ICAM messages are still directed to the operator's console.

1.8.2. Loading Your Program

You load your program the same way as any other user program. That is, you submit a job control stream defining the devices, program name, and any data associated with your program. You must always assign a printer.

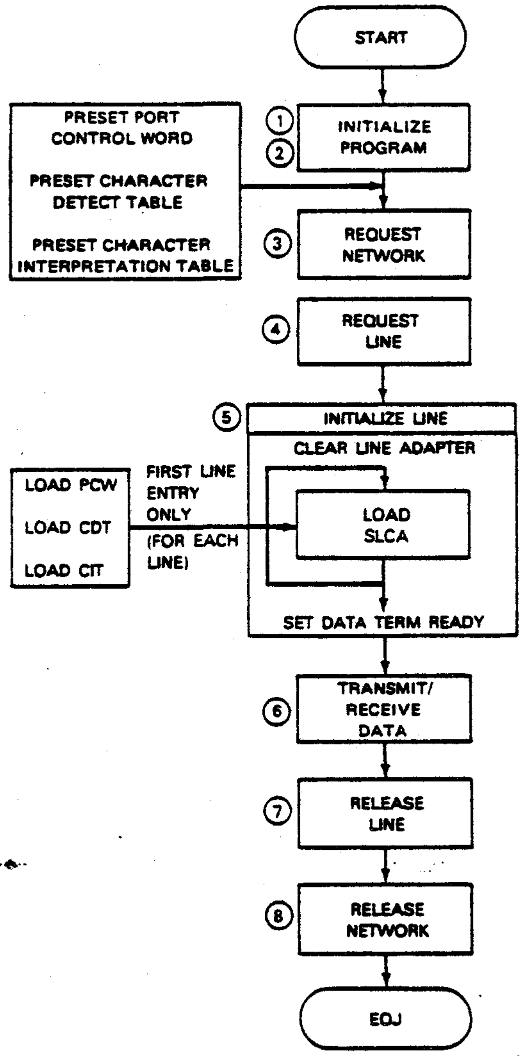
You can execute your program in the same job control stream that loads ICAM or in a separate job control stream. If you execute your program after ICAM is loaded, wait until the ICAM READY message is received.

2. Writing a User Program

2.1. GENERAL

We have seen the similarities in programming structure between data management and ICAM. We have also discussed the use of dummy control sections to aid in constructing our control packets. However, since data communications takes place over communications lines, communications programming has a number of additional functions that data management does not.

As discussed in 1.1, all communications programming is basically the same in structure, so let's put these functions in logical order and then see how they specifically apply to the physical interface. In Section 1, we mentioned such items as loading the single line communications adapter (SLCA), requesting networks and lines, and providing line discipline. Figure 2–1 shows a block diagram of the basic functions required in the physical interface and summarizes the basic steps required to perform these functions.



(1) Initialize the program.

(2) Initialize the SLCA tables, if applicable.

(3) Request the network.
- a. Load the cover register with the request packet.
- b. Fill the packet with the network request command and information.
- c. Issue the CCRCALL.

(4) Request the line.
- a. Load the cover register with the request packet.
- b. Fill the packet with the line request command and information.
- c. Issue the CCRCALL.

(5) Initialize the line.
- a. Clear the line adapter.
  - ■ Load the cover register with the request packet.
  - ■ Fill the packet with the clear line adapter command and information.
  - ■ Issue the CCRCALL.
- b. Load the SLCA port control word.
  - ■ Load the cover register with the request packet.
  - ■ Fill the packet with the load PCW command and information.
  - ■ Issue the CCRCALL.
- c. Load the SLCA character detect table.
  - ■ Load the cover register with the request packet.
  - ■ Fill the packet with the load CDT command and information.
  - ■ Issue the CCRCALL.
- d. Load the SLCA character interpretation table.
  - ■ Load the cover register with the request packet.
  - ■ Fill the packet with the load CIT command and function.
  - ■ Issue the CCRCALL.
- e. Set the data terminal ready.
  - ■ Load the cover register with the request packet.
  - ■ Fill the packet with the set DTR command and function.
  - ■ Issue the CCRCALL.

(6) Transmit the data.
- a. Load the cover register with the input/output packet.
- b. Fill the packet with the send data command and information.
- c. Issue the CCRCALL.

(7) Release the line.
- a. Load the cover register with the request packet.
- b. Fill the packet with the release line command and information.
- c. Issue the CCRCALL.

(8) Release the network.
- a. Load the cover register with the request packet.
- b. Fill the packet with the release network command and function.
- c. Issue the CCRCALL.

Figure 2-1. Structure of a Communications Physical Interface Program

As you can see in Figure 2-1, most of the functions required are simply unique formats of the control packet. The bulk of the task, therefore, simply breaks down to:

- ■ Load the cover register with the desired packet.
- ■ Use the DSECT labels to fill the control packet with the relevant parameters.
- ■ Use the imperative macroinstruction to call the service request for the relevant control packet.

The only exception to this task is on initial line entry. Before each initial line entry, the SLCA must be loaded with the port control word, character detect table, and character interpretation table. We must tell the control packet that we are using our own user discipline and supply the appropriate port control word, character detect table, and character interpretation table to be loaded. These additional tasks will be considered in detail in following sections.

But, for now, let's make up some simple symbolic labels for the functions in Figure 2-1, such as:

Label	Routine
NETREQ1	Request the network.
LNEREQ1	Request the line.
INITLINE	Initialize the line.
SETRMRDY	Set the terminal ready.
SENDMSG	Transmit the message.
LNEREL1	Release the line.
NETREL1	Release the network.

Label	Control Packet Function
REQPK	Request control.
SETDTRPK	Set data terminal ready.
INPTPK	Input
OTPTPK	Output

Using these symbolic labels, we can code a basic outline of our communications program in Figure 2-2. The procedures in Figure 2-1 are numerically keyed to the labels in Figure 2-2 for easy identification (items 1a, 1b, etc). Following this outline, the rest of this section describes how to use the dummy control section labels to set the parameters in the control packet for each function. The dummy control section fields are shown here for ease of use, but are discussed in detail in Appendix A. The fields and the actions performed on them for each function are then numerically keyed to a typical coding example.

Byte	0	1	2	3	Word
0	IRL TN#PRTY	TN#PRESV			1
4	TN#PFCPL				2
8	TN#PTNDX	[illegible]			3
12	[illegible]	[illegible]	[illegible]	[illegible]	4
16	[illegible]	[illegible]	[illegible]		5
20	TN#PCMMD	TN#PBADR			6
24	TN#PTIME		TN#PBLTH		7
28	TN#PFUNC	TN#PLINK			8
32	TN#PFLGS	TN#PLCT			9
36	TN#PCHNL	TN#PPORT	TN#PS80P	TN#PS80F	10
40	[illegible]				11
44	[illegible]				12
48	TN#PS80S				
52	TN#PS80S [illegible]				

LEGEND:

▨ System-supplied parameters

NOTE:

You must specify the channel number (in field TN#PCHNL) and the SLCA number (in field TN#PPORT) for all CPI requests, including a network request. The channel numbers can be hexadecimal 02, 0D, or 0F, and the SLCA number range can be hexadecimal 01 to 0F or 08 to 0F, depending on the System 80 model type (Table A–6). For example, because TN#PMUX is the half-word label for TN#PCHNL AND TN#PPORT, you can specify:

```
MVC TN#PMUX,=X'0208'
```

```
             USING TN#PARP,R1
             .
             .
             .
(3) NETREQ1  LA    R1,REQPK
             .     }
             .     }   Set parameters in control packet for network request
             .     }
             CCRCALL (1)
             .
             .
             .
(4) LNEREQ1  LA    R1,REQPK
             .     }
             .     }   Set parameters in control packet for line request
             .     }
             CCRCALL (1)
             .
             .
             .
(5a) INITLINE LA   R1,REQPK
             .     }
             .     }   Set parameters in control packet for line adapter clear
             .     }
             CCRCALL (1)
             .
             .
             .
(5b) LDPCW   LA    R1,REQPK
             .     }
             .     }   Set parameters in control packet for load port control word
             .     }
             CCRCALL (1)
             .
             .
             .
(5c) LDCDT   LA    R1,REQPK
             .     }
             .     }   Set parameters in control packet for load character
             .     }   detect table

             CCRCALL (1)
             .
             .
             .
```

Figure 2–2. Skeleton Program Outline for the Physical Interface (Part 1 of 2)

```
(5d) LDCIT     LA    R1,REQPK
               .
               .          Set parameters in control packet for load character
               .          interpretation table
               CCRCALL (1)
               .
               .
               .
(5e) SETRMRDY  LA    R1,REQPK
               .
               .          Set parameters in control packet for set data terminal ready
               .
               CCRCALL (1)
               .
               .
               .
(6) SENDMSG    LA    R1,OTPTPK
               .
               .          Set parameters in control packet for send message
               .
               CCRCALL (1)
               .
               .
               .
(7) LNEREL1    LA    R1,REQPK
               .
               .          Set parameters in control packet for release line
               .
               CCRCALL (1)
               .
               .
               .
(8) NETREL1    LA    R1,REQPK
               .
               .          Set parameters in control packet for release network
               .
               CCRCALL (1)
               .
               .
               .
              EOJ
```

Figure 2–2. Skeleton Program Outline for the Physical Interface (Part 2 of 2)

In 2.8, the coding discussed for each function is merged into one basic program to illustrate what an overall physical user program would look like. In this example, completion routines containing status, sense, and condition code checks are not shown to keep the size of the program manageable and the structure visible. Section 4 describes how the port control word, character detect table, and character interpretation tables are constructed for standard and unique user line disciplines.

2.2. REQUESTING A NETWORK

The first function that any communications program must perform is to request the network of lines and terminals desired. Referring to the dummy control section (DSECT) for our control packet, as detailed in Appendix A, the fields that require setting for the network request function are:

	Field	Action
①	TN#PLINK	Set to 0.
②	TN#PFUNC	Set to TN#PNREQ. Bits 0 and 1 are not interpreted.

Using the programming techniques discussed in Section 1, we load the base address of our dummy control section via the USING statement and map the control packet area in our user program by loading the address of the control packet in register 1. We now set the required fields by performing the following typical coding, using the relevant labels of the dummy control section:

```
  NETREQ   EQU    *
           LA     R1,REQPKT                  Map packet
           MVI    TN#PCMMD,0                 Clear command
①          XC     TN#PLINK,TN#PLINK          No chaining
②          MVI    TN#PFUNC,TN#PNREQ          Set function command
           MVC    TN#PFCPL,=A(NTREQCMP)      Set completion address
           LH     R3,TIME3                   Set time-out
           STH    R3,TN#PTIME
           CCRCALL (1)                       Issue I/O request
           .
           .
           .
```

In this coding, the request packet address is first loaded in register 1 for mapping purposes. The hardware command code is then cleared to signal I/O control (CPIOCS) that it must translate the logical command in TN#PFUNC to a hardware command and to overlay it on the TN#PCMMD byte. If this is not done, I/O control uses whatever hardware command currently resides there. Next, data chaining is inhibited by clearing the address field, and the function to be performed is inserted in the function field. The I/O completion address is set and a 3-second time-out is inserted. Finally, an I/O request is issued by the CCRCALL macroinstruction.

The user program is assigned one of activity control's user activity slots when it issues its network request function (or, under the temporary option, when it issues its first CCRCALL for whatever function). If a user activity slot is not available, the user program is cancelled and the control packet identified by register 1 has TN#PRIM set to TN#PROGE and TN#PDETL set to TN#PMNCE. Refer to 3.6 for other conditions that result in the user program being cancelled by the interface code.

If the user program operates as a multitasked job, all access to the physical interface should be via one specific task. Should the user program access the physical interface by means of more than one task for one job, each task would be assigned one of ICAM's task identities.

While the user program is executing communications I/O requests, it must not be executing other I/O requests using the same task. The user program's task identity is deleted and its lines are released when it goes to end-of-job (EOJ) either normally or abnormally.

After the supervisor request call issued via inline expansion of the CCRCALL has been completed successfully, control returns to label NTREQCMP as set in the completion address portion of the control packet.

Normally, at this point in your program, you would want to perform clean-up tasks, such as checking completion status. Again, using the DSECT labels available to you, you could check logical status bytes, such as:

```
         CLC   TN#PRIM(2),=X'0180'         Good completion?
         BE    LNEREQ1                     Yes, perform line request
```

The status bytes are interrogated by assembler statements using labels in the control packet DSECT. The number of checks available to you is rather large; the degree of checking is your option.

2.3. REQUESTING A LINE

Once the network request is completed satisfactorily, you must request the specific line on which you wish to communicate.

All user program attempts to execute functions without prior request and assignment of a line are aborted. Your user program must secure the desired line and a set of communications adapter interpretation tables before it can issue I/O functions. Do not attempt a line request while non-CPI line requests are in progress.

To request a line, the user program must set the following control packet fields as directed. Those fields not specified here are set as specified in the control packet description in Appendix A.

	Field	Action
①	TN#PMUX	Set byte 0 (TN=PCHNL) to 02, 0D, or 0F depending on your System 80 model type (Table A-6). Set byte 1 to the SLCA number and byte 2 to the port number.
②	TN#PFUNC	Set to TN#PLREQ. Bits 0 and 1 are not interpreted.
③	TN#PLINK	Set to 0.
④	TN#PDSPL	Set bit 0 to 1 and set bits 1–3 to any value. A user program key is placed into bits 4–7 to create a unique user remote device handler discipline ID.
⑤	TN#PCAID	Set from 00_{16} to 02_{16}. The SLCA has two control character detect tables and one control interpretation table. When this byte is set to 00_{16}, an available set of tables is located and the ID is placed back in TN#PCAID. The user program can optionally specify the tables it wants (01_{16} – 02_{16}).
⑥	TN#PBLTH	The SLCA buffer size must be specified in the TN#BLTH field of the CPIOCP at the request time. The valid range is 32–256 in decimal. If buffer toggling is required, the line buffer size must be an even multiple of the SLCA buffer size.

Translating these actions into actual coding, we might have something like this:

```
    LNEREQ1  EQU    *
             LA     R1,REQPKT               Map packet
①            MVI    TN#PCHNL,2              Set to channel 2
             MVI    TN#PPORT,8              Set to SLCA#1
             MVI    TN#PCMMD,0              Clear command
②            MVI    TN#PFUNC,TN#PLREQ       Set function command
③            XC     TN#PLINK,TN#PLINK       No data chaining
             LH     R3,TIME3                Set time-out
             STH    R3,TN#PTIME
             MVC    TN#PFCPL,=A(LNREQCMP)   Set completion address
④            MVI    TN#PDSPL,X'80'          Set for user own discipline
⑤            MVI    TN#PCAID,1              Set for CDT1/CIT1
⑥            MVC    TN#PBLTH+1,H'256'       Set SLCA buffer length to 256
             CCRCALL (1)                    Issue I/O request
             .
             .
             .
    LNREQCMP EQU    *                       Line request completion routine
             B      INITLNE                 Continue to next function
             .
             .
             .
```

As you can see in the coding, the channel is set to channel 2; the SLCA is set to SLCA#1. The command is set to line request. For this example, we could have selected a remote device handler discipline (label TN#PDSPL). If we choose a remote device handler that is already supported, we must construct a port control word, character detect table, and character interpretation table according to the parameters pertinent to that handler. These parameters are described in Section 5. (An explanation of how to construct these tables with the relevant parameters is given in 4.2.1 through 4.2.3.) We chose instead to construct our own remote device handler discipline by setting bit 0 equal to 1.

We have also specified that we want the number 1 character detect and character interpret tables. However, we could have requested any available set of tables by performing a

```
    MVI   TN#PCAID,Ø
```

You will notice in the next section that line initialization need only be performed the first time that a particular line is requested.

Some other considerations you should keep in mind when performing line requests are:

- A full-duplex line is assigned only if both ports are unallocated.
- A line with primary and secondary channels can be assigned only one channel at a time. A line request must first be made for the primary channel port; then, a second line request should be made for the secondary channel port. Separate line requests must be issued for autodialer ports and the associated line. The dial commmand request will be illegal if the dial port and the line associated with the autodialer have not both been assigned to you.

NOTE:

Two simultaneous line request operations can cause an attach error condition to be presented (TN#PRIM=TN#PROGE and TN#PDETL=TN#PAATH) in one of the control packets (CPIOCPs). You can recover from this condition by reissuing the line request function for the control packet returned with the error status.

2.4. INITIALIZING A LINE

The following procedure is recommended to initialize a line once the line request function has been completed successfully. This procedure is identical to that performed by remote device handlers for other ICAM interfaces.

- Half-duplex lines:

  1. Issue line adapter (LA) clear (TN#PFUNC=TN#PFSL+TN#PLACL).

  2. Load port control word (TN#PFSL + TN#PLB14).

  3. Load the SLCA control tables if not already loaded.

This consists of loading the character detect and character interpretation tables before the first transmission on the line. These tables are described in Section 5. These tables must be constructed within the nonexecutable code of your program before issuing the load commands. As mentioned in 2.3, if the remote device handler discipline used is one of the supported handlers, the parameters prescribed may be used directly to construct the tables. This construction is described in 4.2.1 through 4.2.3. If your own unique discipline is to be used instead, you must then construct your own unique tables.

4. Issue a set data terminal ready function (TN#PFUNC=TN#PFSL+TN#PDTR). This enables receipt of the BREAK interrupt and permits dial connection.

- Full-duplex line:

  1. Issue clear and load port control word for output port as explained for half-duplex lines.

  2. Load control tables. (Same as item 3 for half-duplex lines)

  3. Clear and load port control word for input port.

     NOTE:

     Port 0 is for special control, port 1 is for output, and port 2 is for input. The 2-way alternate (half-duplex) operation, therefore, assumes port 1 is for both output and input if the set-full-duplex command is not set. For full-duplex operation, you specify port 2 for input by issuing the set-full-duplex command in step 4. See Appendix A.

  4. Issue set-full-duplex command to output port (TN#PFUNC=TN#PFSL + TN#PFLDX).

  5. Issue a set data terminal ready function (TN#PFUNC=TN#PFSL + TN#PDTR).

Breaking each of these functions down to our standard structure and using the half-duplex procedure, we would have:

- clear the line adapter;
- load the port control word;
- load the character detect table;
- load the character interpretation table; and
- set the data terminal ready.

Once the line has been initialized for the first time, these functions do not have to be performed again. Therefore, in the coding of the next section, we will insert a switch that falls through on the first entry and is then set to skip the initialization procedures on ensuing transmissions for this particular line. If other lines are to be used, they also must be initialized before the first entry.

2.4.1. Clear the Line Adapter

To issue a line adapter clear, we must perform the same steps of setting the relevant parameters in the control packet and making a service request via the CCRCALL. To issue this command, you must perform the following actions on the specified fields:

	Field	Action
①	TN#PCHNL	Same as line request (see 2.3)
②	TN#PPORT	Same as line request (see 2.3)
③	TN#PFUNC	Set to TN#PFSL + TN#PLACL (start and end of message with line adapter clear command).

Showing these actions in coding format, we might have:

```
    INITLNE  EQU   *
    SWITCH   BC    0,SENDMSG                    Initialization switch
             LA    R1,REQPKT                    Map packet
             MVI   TN#PCMMD,0                   Clear command
①           MVI   TN#PCHNL,2
②           MVI   TN#PPORT,8
             MVI   TN#PS80P,1
③           MVI   TN#PFUNC,TN#PFSL+TN#PLACL  Set function command
             XC    TN#PLINK,TN#PLINK            No data chaining
             LH    R3,TIME3                     Set time-out
             STH   R3,TN#PTIME
             MVC   TN#PFCPL,=A(LACMPL)          Set completion address
             CCRCALL (1)                        Issue I/O request
             .
             .
             .
    LACMPL   EQU   *                            Line adapter clear completion routine
             MVI   SWITCH+1,X'F0'                 Set to skip after 1st time thru
             B     LDPCW                        Continue to next function
             .
             .
             .
```

2.4.2. Load the Port Control Word

Each port in the SLCA must have a port control word consisting of a set of four control bytes. These control bytes must be coded and loaded by your user program. The settings for these control bytes are determined by the characteristics you require for a particular remote device handler discipline. These characteristics are described in detail in Section 3 of the input/output microprocessor programmer reference, UP-8909 (current version). Briefly, byte 1 concerns details of synchronous/asynchronous operation. Byte 2 references the character detect table to use, the character length, and the asynchronous line speed, if applicable. Byte 3 references the character interpretation table to use, specifies whether to include or exclude the start character in the block check character count, and the type of parity to be used in transmission. Byte 4 specifies whether operation is of the synchronous or asynchronous mode and timing intervals.

Subsection 4.2.1 describes how to construct a port control word. At this point, we are basically concerned with loading it via our user program. So for now, let's assume that we have a table in our nonexecutable code of the form:

```
DCT2PCW  DC  X'80 20 19 02'
```

	Field	Action
①	TN#PFUNC	Set to load port control word (TN#PFSL+TN#PLB14).
②	TN#PBLTH	Set to 4 bytes.
③	TN#PBADR	Set to data buffer address.

In our coding, we perform the following:

```
     LDPCW    EQU    *
              MVC    OUTBUFF,DCT2PCW                Put port control word in buffer
              LA     R1,REQPKT                      Map packet
              MVI    TN#PCMMD,0                     Clear command
 ①            MVI    TN#PFUNC,TN#PFSL+TN#PLB14      Set function to load PCW
 ②            MVI    TN#PBLTH+1,4                   Set buffer length to 4 bytes
              LH     R3,TIME3                       Set 3-sec time-out
              STH    R3,TN#PTIME
 ③            MVC    TN#PBADR,OUTBUFAD              Set output buffer address
              XC     TN#PLINK,TN#PLINK'             No data chaining
              MVC    TN#PFCPL,=A(LPCWCMPL)          Set completion address
              CCRCALL (1)                           Issue I/O request
              .
              .
              .
     LPCWCMPL EQU    *                              Load PCW completion routine
              B      LDCDT1                         Continue to next function
              .
              .
              .
```

Additional tasks performed in this routine are to reset a switch (such that this routine is entered only the first time this line is requested), put the port control word in the output buffer, prescribe the length of the data buffer that is to be transmitted, and set the output buffer address in the control packet. Standard tasks are to insert the load port control word command, and, of course, furnish a completion address.

2.4.3. Load the Character Detect Table

Each port in the SLCA references at least one of two character detect tables. These tables are used to produce start and end character functions, and access a character interpretation table word, if additional functions are required. Again, the character detect table must be coded and loaded by your user program. The settings for these bytes are determined by the characteristics you require. The characteristics are described in detail in Section 3 of the System 80 input/output microprocessor programmer reference, UP-8909 (current version). A discussion on how to construct a typical character detect table is contained in 4.2.2. Since this subsection is concerned only with loading the table, let's assume there is a table in the nonexecutable code of our program that looks like:

```
DCT2CDT  DC    X'0012001317001414'      00 - 0F
         DC    X'0000000000000000'
         DC    X'0014000000141000'      10 - 1F
         DC    X'0000001300001400'
         DC    X'0000000000000000'      20 - 2F
         DC    X'0000000000000000'
         DC    X'0000000000000000'      30 - 3F
         DC    X'0000000000000000'
         DC    X'0000000000000000'      40 - 4F
         DC    X'0000000000000000'
         DC    X'0000000000000000'      50 - 5F
         DC    X'0000000000000000'
         DC    X'0000000000000000'      60 - 6F
         DC    X'0000000000000000'
         DC    X'0000000000000000'      70 - 7F
         DC    X'0000000000000000'
         DC    X'0000001300001400'      80 - 8F
         DC    X'0000000000000000'
         DC    X'0014000000000000'      90 - 9F
         DC    X'0000000000000000'
         DC    X'0000000000000000'      A0 - AF
         DC    X'0000000000000000'
         DC    X'0000000000000000'      B0 - BF
         DC    X'0000000000000000'
         DC    X'0000000000000000'      C0 - CF
         DC    X'0000000000000000'
         DC    X'0000000000000000'      D0 - DF
         DC    X'0000000000000000'
         DC    X'0000000000000000'      E0 - EF
         DC    X'0000000000000000'
         DC    X'0000000000000000'      F0 - FF
         DC    X'0000000000000000'
```

To load this table into the SLCA, you perform the following actions on the specified fields:

	Field	Action
①	TN#PFUNC	Set to load character detect table number 1. (The CD table is always 1 or 2 (00_{16} or 01_{16}) for the SLCA.)
②	TN#PBLTH	Set to 256 bytes maximum.
③	TN#PBADR	Set to the address of the data output buffer.

To perform these actions on the specified fields, you do the following coding:

```
    LDCDT1   EQU    *
             LA     R1,REQPKT                 Map packet
             MVC    OUTBUFF,DCT2CDT           Put character detect table in output
                                              buffer
             MVI    TN#PCMMD,0                Clear command
 ①           MVI    TN#PFUNC,TN#PLCD1         Set function to load CD table 1
                                              (See NOTE.)
 ②           MVC    TN#PBLTH,W256             Set buffer length to 256 bytes
             LH     R3,TIME3                  Set 3-sec time-out
             STH    R3,TN#PTIME
 ③           MVC    TN#PBADR,OUTBUFAD         Set output buffer address
             XC     TN#PLINK,TN#PLINK         No data chaining
             MVC    TN#PFCPL,=A(LCDTCMPL)     Set completion address
             CCRCALL (1)                      Issue I/O request
             .
             .
             .
    LCDTCMPL EQU    *                         Load CD table completion routine
             B      LDCIT1                    Continue to next function
             .
             .
             .
```

NOTE:

This command loads only character detect table 1. The SLCA allows two character detect tables.

As in the previous section, the data to be transmitted to (loaded into) the SLCA is placed in the data output buffer, and the length of the output buffer and the address of the output buffer are placed in the control packet. Finally, the function and completion address are inserted and the I/O service request is performed via the CCRCALL macroinstruction.

2.4.4. Load the Character Interpretation Table

There is only one character interpretation table in the SLCA. It is used to provide multiple sequences of action when addressed by the character detect table. That is, the character detect table may initiate a single hard-wired function or it may be used to initiate a function and address one word of the character interpretation table. Each individual bit in the character interpretation table word can then indicate a particular action or sequence of actions in the SLCA. Thus, multiple actions can be initiated by a single character. As with the port control word and the character detect table, you must code and load the character interpretation table via your user program. The settings for these bytes are determined by the characteristics you require. A discussion on how to construct a typical character interpretation table is contained in 4.2.3. Again, since we're concerned here with only the loading of the table, let's assume we have constructed a table in our nonexecutable code that looks like the following:

```
DCT2CIT  DC    X'08010000'
         DC    X'11014800'
         DC    X'01000000'
         DC    X'00000100'
         DC    X'00000000'
         DC    X'00000000'
         DC    X'00000000'
         DC    X'00000000'
```

To load this table into the SLCA, you have to perform the following actions on the specified fields:

	Field	Action
①	TN#PFUNC	Set to load character interpretation table number 1.
②	TN#PBLTH	Set to 32 bytes maximum (16 words).
③	TN#PBADR	Set to address of data output buffer.

To perform these actions on the specified fields, we use the same format as before to construct our table; the coding to do this looks like:

```
    LDCIT1  EQU   *
            LA    R1,REQPKT              Map packet
            MVC   OUTBUFF,DCT2CIT  .     out character interpret table in
                                         output buffer
            MVI   TN#PCMMD,0             Clear command
①           MVI   TN#PFUNC,TN#PLCI1      Set function to load CIT table 1
            XC    TN#PLINK,TN#PLINK      No data chaining
②           MVI   TN#PBLTH+1,32          Set for 32-byte buffer
③           MVC   TN#PBADR,OUTBUFAD      Set output buffer address
            LH    R3,TIME3               Set time-out
```

(continued)

```
          STH    R3,TN#PTIME
          MVC    TN#PFCPL,=A(LCITCMPL)      Set completion address
          CCRCALL (1)                       Issue I/O request
          .
          .
          .
LCITCMPL  EQU    *                          Load CI table completion routine
          B      SETRMRDY                   Continue to next function
          .
          .
          .
```

2.4.5. Set the Data Terminal Ready

After the SLCA is loaded, line initialization is complete and we can now issue the function to set the data terminal ready. This allows a BREAK interrupt to be received and permits dial connection. To issue this command, you must perform the following actions on the specified fields:

	Field	Action
①	TN#PFUNC	Set to TN#PFSL+TN#PDTR (start and end of message with set data terminal ready command)

With the exception of entering this particular command, the coding again remains basically the same:

```
SETRMRDY  EQU    *
          LA     R1,REQPKT                  Map packet
          MVI    TN#PCMMD,0                 Clear command
①         MVI    TN#PFUNC,TN#PFSL+TN#PDTR   Set function to set data terminal
                                            ready
          XC     TN#PLINK,TN#PLINK
          MVC    TN#PBADR,AFAKEBUF          Set fake buffer
          LH     R3,TIME3                   Set time-out
          STH    R3,TN#PTIME
          MVC    TN#PFCPL,=A(SDTRCMPL)      Set completion address
          CCRCALL (1)                       Issue I/O request
          .
          .
          .
SDTRCMPL  EQU    *                          Set data terminal ready completion
                                            routine
          B      SENDMSG                    Continue to next function
```

The only unusual action in this coding might be use of a fake buffer (contents of zero) to clear the output buffer. The set data terminal ready command does not require use of the output buffer, but clearing the buffer prevents any possible transmission of spurious messages or commands to a terminal.

2.5. TRANSMITTING THE DATA

Transmission or reception of data are basically the same structurally. They differ only in the command and data buffer address supplied to the control packet. Reception of a message might differ only in performing a poll before performing the actual receive. In this case, a poll would merely be a send command control packet with a poll message in the data buffer. For the sake of simplicity, however, we'll only show a send text procedure. To do a send, we would perform the following actions on the specified fields of our control packet:

	Field	Action
①	TN#PFUNC	Set to send data command (TN#PSEND+TN#PFSL).
②	TN#PBADR	Set to the data output buffer address.
③	TN#PBLTH	Set to the length of the output buffer.

To perform these actions on the specified fields, we might code:

```
    SENDMSG  EQU    *
             LA     R1,OTPTPKT                Map packet
             MVI    TN#PCMMD,Ø                Clear command
①            MVI    TN#PFUNC,TN#PFSL+TN#PSEND Set function to send data
             MVC    OUTBUFF,MSG1              Put text in output buffer
             XC     TN#PLINK,TN#PLINK         No data chaining
             LH     R3,TIME3                  Set time-out
             STH    R3,TN#PTIME
②            MVC    TN#PBADR,OUTBUFAD         Set buffer address
③            MVI    TN#PBLTH+1,8Ø             Set buffer length
             MVC    TN#PFCPL,=A(SENDCMPL)     Set completion address
             CCRCALL (1)                      Issue I/O request
             .
             .
             .
    SENDCMPL EQU    *                         Send completion routine
             B      LNEREL1                   Continue to next function
             .
             .
             .
```

If we were performing a receive function, at this point, instead of continuing to the next function, we might want to construct a loop that would send a poll message for text. The text could then be scanned for an end message and, if obtained, a branch would continue to the next function to release the line.

2.6. RELEASING THE LINE

The line release function deallocates a line for assignment to other users. As with line requests, primary and secondary channels must be released separately (secondary before primary). An autodialer port and associated lines must also be released separately.

Before a line release is issued, the user program must ensure that there are no outstanding control packets. If there are any outstanding control packets for the line, they may be retrieved by executing a CCRCALL with TN#PFUNC=TN#PFSL + TN#PIOFF. (The turn-off command does not provide an immediate status. The I/O completion interrupt must be awaited.) If the user program interface code detects an outstanding control packet, it will abort the line release function, setting TN#PRIM=TN#PROGE and TN#PDETL=TN#PFRMT.

For this function, then, we'll actually perform two functions – one to issue the turnoff and one to issue the line release command.

Field	Action
① TN#PFUNC	Set to complete message with turnoff (TN#PFSL+TN#PIOFF).

To perform the turnoff, we need only the following:

```
    LNEREL1  EQU    *
             LA     R1,REQPKT                 Map packet
             MVI    TN#PCMMD,0                Clear command
①            MVI    TN#PFUNC,TN#PFSL+TN#PIOFF Set function to turnoff
             XC     TN#PLINK,TN#PLINK         No data chaining
             LH     R3,TIME3                  Set time-out
             STH    R3,TN#PTIME
             MVC    TN#PFCPL,=A(NOMOPKTS)     Set completion address
             CCRCALL (1)                      Issue I/O request
             .
             .
             .
    NOMOPKTS EQU    *                         Turnoff completion routine
             B      LNEREL2                   Continue to next function
             .
             .
             .
```

At the completion address, as just mentioned, one of the things you would want to check is if the outstanding packet check bit is set in the control packet. This could be done, for instance with a:

```
CLI    TN#PRIM,TN#PROGE            Program error?
BNE                                No
CLI    TN#PDETL,TN#PFRMT           Outstanding packet?
BNE                                No
```

Once the turnoff has been completed successfully, the line release function can be issued.

	Field	Action
①	TN#PFUNC	Set to line release command (TN#PLREL)

This function would be performed by:

```
    LNEREL2  EQU    *
             LA     R1,REQPKT                  Map packet
             MVI    TN#PCMMD,Ø                 Clear command
①            MVI    TN#PFUNC,TN#PLREL          Set function to line release
             XC     TN#PLINK,TN#PLINK          No data chaining
             LH     R3,TIME3                   Set time-out
             STH    R3,TN#PTIME
             MVC    TN#PFCPL,=A(LRELCMPL)      Set completion address
             CCRCALL (1)                       Issue I/O request
             .
             .
             .
    LRELCMPL EQU    *                          Line release completion routine
             B      NETREL1                    Continue to next function
```

2.7. RELEASING THE NETWORK

The last CCRCALL that the user program executes should be a user program network release function to avoid abnormal termination by the supervisor. The user program network release function releases all lines that the user program has not yet released and disengages the user program from ICAM. The only field that needs setting for a user program network release other than the standard ones is:

	Field	Action
①	TN#PFUNC	Set to TN#PNREL. Bits 0 and 1 are not interpreted.

You can perform the function of network release with the following typical coding:

```
    NETREL1  EQU    *
             LA     R1,REQPKT                  Map packet
             MVI    TN#PCMMD,0                 Clear command
    (1)      MVI    TN#PFUNC,TN#PNREL          Set function to network release
             XC     TN#PLINK,TN#PLINK          No data chaining
             LH     R3,TIME3                   Set time-out
             STH    R3,TN#PTIME
             MVC    TN#PFCPL,=A(NETRLCMP)      Set completion address
             CCRCALL (1)                       Issue I/O request
             .
             .
             .
    NETRLCMP EQU    *                          Network release completion routine
             .
             .
             .
             EOJ
```

At this point in the program, there are no further functions to be performed and the next instruction would be an EOJ to end the job and start the constant area to define the required constants for the program. The communications task is now complete for this simple program.

2.8. EXAMPLE OF A BASIC USER PROGRAM

To illustrate all of the functions described thus far, Figure 2–3 shows all of the statements coded from 2.2 through 2.7 integrated as a simple program. The dummy completion routines are all consolidated to save space and clarify the flow, especially since they are included only for completeness of explanation. Tables necessary for this program are included in the example. They are discussed in detail in Section 3.

```
*
*  PROGRAM  INITIALIZATION
*
         START Ø
         CSECT
         TN#DSECT XCPIOCP
         TN#DSECT LLT
         RGEQU
         USING TN#PARP,R1
         USING TN#PLINE,R7
BEGIN    EQU   *
         BALR  R2,Ø
         USING *,R2
         MVI   SWITCH+1,Ø
*
* REQUEST A NETWORK
*
NETREQ   EQU   *
         LA    R1,REQPKT
         MVI   TN#PCMMD,Ø
         XC    TN#PLINK,TN#PLINK
         MVI   TN#PFUNC,TN#PNREQ
         MVC   TN#PFCPL,=A(NTREQCMP)
         LH    R3,TIME3
         STH   R3,TN#PTIME
         CCRCALL (1)
*
* REQUEST A LINE
*
LNEREQ1  EQU   *
         LA    R1,REQPKT
         MVI   TN#PCHNL,2
         MVI   TN#PPORT,8
         MVI   TN#PS8ØP,1
         MVI   TN#PCMMD,Ø
         MVI   TN#PFUNC,TN#PLREQ
         XC    TN#PLINK,TN#PLINK
         LH    R3,TIME3
         STH   R3,TN#PTIME
         MVI   TN#PBLTH+1,256
         MVC   TN#PFCPL,=A(LNREQCMP)
         MVI   TN#PDSPL,3
         MVI   TN#PCAID,1
         CCRCALL (1)
```

Figure 2-3. Basic User Program Example (Part 1 of 6)

```
*
* INITIALIZE THE LINE - (1) CLEAR THE LINE ADAPTER
*
INITLNE  EQU    *
SWITCH   BC     0,SENDMSG
         LA     R1,REQPKT
         MVI    TN#PCMMD,0
         MVI    TN#PFUNC,TN#PFSL+TN#PLACL
         XC     TN#PLINK,TN#PLINK
         LH     R3,TIME3
         STH    R3,TN#PTIME
         MVC    TN#PFCPL,=A(LACMPL)
         CCRCALL (1)
*
* INITIALIZE THE LINE - (2) LOAD PORT CONTROL WORD
*
LDPCW    EQU    *
         MVC    OUTBUFF,DCT2PCW
         LA     R1,REQPKT
         MVI    TN#PCMMD,0
         MVI    TN#PFUNC,TN#PFSL+TN#PLB14
         MVI    TN#PBLTH+1,4
         MVC    TN#PBADR,OUTBUFAD
         XC     TN#PLINK,TN#PLINK
         LH     R3,TIME3
         STH    R3,TN#PTIME
         MVC    TN#PFCPL,=A(LPCWCMPL)
         CCRCALL (1)
*
* INITIALIZE THE LINE - (3) LOAD THE CHARACTER DETECT TABLE
*
LDCDT1   EQU    *
         LA     R1,REQPKT
         MVC    OUTBUFF,DCT2CDT
         MVI    TN#PCMMD,0
         MVI    TN#PFUNC,TN#PLCD1
         MVC    TN#PBLTH,W256
         MVC    TN#PBADR,OUTBUFAD
         XC     TN#PLINK,TN#PLINK
         LH     R3,TIME3
         STH    R3,TN#PTIME
         MVC    TNPFCPL,=A(LCDTCMPL)
         CCRCALL (1)
*
* INITIALIZE THE LINE - (4) LOAD CHARACTER INTERPRETATION TABLE
```

Figure 2–3. Basic User Program Example (Part 2 of 6)

```
LDCIT1   EQU    *
         LA     R1,REQPKT
         MVC    OUTBUFF,DCT2CIT
         MVI    TN#PCMMD,Ø
         MVI    TN#PFUNC,TN#PLCI1
         XC     TN#PLINK,TN#PLINK
         MVC    TN#PBADR,OUTBUFAD
         MVI    TN#PBLTH+1,32
         LH     R3,TIME3
         STH    R3,TN#PTIME
         MVC    TN#PFCPL,=A(LCITCMPL)
         CCRCALL (1)
*
*  INITIALIZE THE LINE - (5) SET DATA TERMINAL
SETRMRDY EQU    *
         LA     R1,REQPKT
         MVI    TN#PCMMD,Ø
         MVI    TN#PFUNC,TN#PFSL+TN#PDTR
         XC     TN#PLINK,TN#PLINK
         MVC    TN#PBADR,AFAKEBUF
         LH     R3,TIME3
         STH    R3,TN#PTIME
         MVC    TN#PFCPL,=A(SDTRCMPL)
         CCRCALL (1)
*
* TRANSMIT A MESSAGE
*
SENDMSG  EQU    *
         LA     R1,OTPTPKT
         MVI    TN#PCMMD,Ø
         MVI    TN#PFUNC,TN#PSEND+TN#PFSL
         MVC    OUTBUFF,MSG1
         XC     TN#PLINK,TN#PLINK
         LH     R3,TIME3
         STH    R3,TN#PTIME
         MVC    TN#PBADR,OUTBUFAD
         MVI    TN#PBLTH+1,8Ø
         MVC    TN#PFCPL,=A(SENDCMPL)
         CCRCALL (1)
*
* RELEASE THE LINE - (1) ISSUE LINE TURNOFF
*
```

Figure 2–3. Basic User Program Example (Part 3 of 6)

```
LNEREL1  EQU   *
         LA    R1,REQPKT
         MVI   TN#PCMMD,Ø
         MVI   TN#PFUNC,TN#PFSL+TN#PIOFF
         XC    TN#PLINK,TN#PLINK
         LH    R3,TIME3
         STH   R3,TN#PTIME
         MVC   TN#PFCPL,=A(NOMOPKTS)
         CCRCALL (1)
*
* RELEASE THE LINE - (2) ISSUE LINE RELEASE
*
LNEREL2  EQU   *
         LA    R1,REQPKT
         MVI   TN#PCMMD,Ø
         MVI   TN#PFUNC,TN#PLREL
         XC    TN#PLINK,TN#PLINK
         LH    R3,TIME3
         STH   R3,TN#PTIME
         MVC   TN#PFCPL,=A(LRELCMPL)
         CCRCALL (1)
*
* RELEASE THE NETWORK
*
NETREL1  EQU   *
         LA    R1,REQPKT
         MVI   TN#PCMMD,Ø
         MVI   TN#PFUNC,TN#PNREL
         XC    TN#PLINK,TN#PLINK
         LH    R3,TIME3
         STH   R3,TN#PTIME
         MVC   TN#PFCPL,=A(NETRLCMP)
         CCRCALL (1)
*
* DUMMY COMPLETION ROUTINES
*
NTREQCMP EQU   *
         B     LNEREQ1
LNREQCMP EQU   *
         B     INITLNE
LACMPL   EQU   *
         MVI   SWITCH+1,X'FØ'
         B     LDPCW
```

Figure 2–3. Basic User Program Example (Part 4 of 6)

```
LPCWCMPL EQU   *
         B     LDCDT1
LCDTCMPL EQU   *
         B     LDCIT1
LCITCMPL EQU   *
         B     SETRMRDY
SDTRCMPL EQU   *
         B     SENDMSG
SENDCMPL EQU   *
         B     LNEREL1
NOMOPKTS EQU   *
         B     LNEREL2
LRELCMPL EQU   *
         B     NETREL1
NETRLCMP EQU   *
ENDCPI   EOJ
*
         CNOP  0,4
* START CONSTANT AREA
*
         CNOP  0,8
REQPKT   DS    14F'0'
INPTPKT  DS    14F'0'
OTPTPKT  DS    14F'0'
OUTBUFF  DS    CL80
FAKEBUFF DS    F'0'
OUTBUFAD DC    AL3(OUTBUFF)
AFAKEBUF DC    AL3(FAKEBUFF)
         CNOP  0,4
TIME3    DC    H'3'
W256     DC    X'0100'
ZEROES   DC    X'000000'
MSG1     DC    C'FOUR SCORE AND S'
         DC    C'EVEN YEARS AGO O'
         DC    C'UR FATHERS BROUG'
         DC    C'HT FORTH ON THIS'
         DC    C' CONTINENT...END'
         CNOP  0,8
*
* PORT CONTROL WORD
*
DCT2PCW  DC    X'80201902'
```

Figure 2–3. Basic User Program Example (Part 5 of 6)

```
* CHARACTER DETECT TABLE
*
DCT2CDT  DC    X'0012001317001414'
         DC    X'0000000000000000'
         DC    X'0014000000141000'
         DC    X'0000001300001400'
         DC    X'0000000000000000'
         DC    X'0000000000000000'
         DC    X'0000000000000000'
         DC    X'0000000000000000'
         DC    X'0000000000000000'
         DC    X'0000000000000000'
         DC    X'0000000000000000'
         DC    X'0000000000000000'
         DC    X'0000000000000000'
         DC    X'0000000000000000'
         DC    X'0000000000000000'
         DC    X'0000000000000000'
         DC    X'0000001300001400'
         DC    X'0000000000000000'
         DC    X'0014000000000000'
         DC    X'0000000000000000'
         DC    X'0000000000000000'
         DC    X'0000000000000000'
         DC    X'0000000000000000'
         DC    X'0000000000000000'
         DC    X'0000000000000000'
         DC    X'0000000000000000'
         DC    X'0000000000000000'
         DC    X'0000000000000000'
         DC    X'0000000000000000'
         DC    X'0000000000000000'
         DC    X'0000000000000000'
         DC    X'0000000000000000'
*
* CHARACTER INTERPRETATION TABLE
*
DCT2CIT  DC    X'08010000'
         DC    X'11014800'
         DC    X'41000000'
         DC    X'00004100'
         DC    X'00000000'
         DC    X'00000000'
         DC    X'00000000'
         DC    X'00000000'
         END
```

Figure 2–3. Basic User Program Example (Part 6 of 6)

3. Additional User Program Features

3.1. CONTROL PACKET CHAINING

The physical interface supports both data buffer chaining and message chaining. Data buffer chaining is the linking of control packet buffers for input or output of a single message. The buffers may be chained through the TN#PLINK field of the control packet at the time of CCRCALL execution, or they may be linked one or more at a time. The communications physical I/O control system (CPIOCS) will then chain a following control packet behind one that is already queued on a particular line. Invalid packet chaining links cause unpredictable errors. Message chaining is the same as the linking of data buffers with the exceptions that more than one complete message (TN#PFUNC/TN#PFSL) is set.

For data buffer chaining, the first control packet of a message or function must have TN#PFUNC set to TN#PFS (start of message). Output messages must have TN#PFUNC set to TN#PFL for the last control packet of a message (end of message). You must set up the first two CPIOCPs before issuing the first CCR call. Failure to do so causes a chaining check error. It is also required that all control packets within one message have TN#PMUX set to the same channel and SLCA numbers.

Whenever the function or buffer of a specified control packet reaches completion, the control packet is scheduled back to your program unless TN#PFLGS is set to TN#PCIX (buffer completion suppressed), or TN#PESO (error schedule only). If either of these flags is set and the condition indicated by the flag is met, the packet is released from the active line queue and placed on a delayed queue for the same line. If the flags are not set, CPIOCS checks the completion status. If TN#PRIM=TN#PEND (message/function completion) and TN#PDETL=TN#PBCI (buffer completion), CPIOCS breaks the link to the chained control packet and proceeds to check for any control packets on the delayed queue. If TN#PRIM=TN#PEND and TN#PDETL=TN#EOM (successful) message/function (completion), CPIOCS then checks to see whether any other control packet is chained to this control packet. If there is, a link is kept intact to the next control packet that has TN#PFUNC set to TN#PFS. The last control packet in the chain is indicated with TN#PLINK set to 0. If the completed control packet has any error status in TN#PRIM and TN#PDETL, the chain is kept intact regardless of the number of control packets having TN#PFS set.

NOTE:

If buffer toggling is required, TN=PESO (error schedule only) must not be set in the TN#PFLGS field of the CPIOCP because it would prevent the buffer completion interrupt from being reported. The line buffer size must be an even multiple of the SLCA buffer size specified in TN#PBLTH if using buffer toggling.

Before scheduling the completed control packet back to your program, together with any chained control packets, CPIOCS checks if there are any packets on the delayed queue. If there are, the currently completed packet is chained behind the last packet on the delayed queue. Status in the currently completed packet is then moved to the head control packet on the delayed queue. The head packet on the delayed queue is now scheduled back to your program at the completion address specified by its TN#PFCPL field together with all control packets chained to it.

As an example, good use of message chaining is accomplished when chaining an output poll message to the first buffer of an input response. The poll control packet has TN#PFLGS set to TN#PESO. The first control packet of the input message does not have TN#PFLGS set to TN#PESO. If the output poll is successfully issued, your program will be notified at the completion address of the output control packet when the first input message control packet is complete. This technique saves scheduling your program until significant information is available requiring your attention, and it provides for issuance of the input function immediately upon successful completion of the poll.

To clarify the preceding discussion on data chaining, we'll give a coding example of each of the two types, including the special case of chaining output polls to input response, and point out the major differences of each.

For data buffer chaining, let's take a case where we want to send three buffers chained together to make one complete message:

Packet	Field	Action
1	TN#PFUNC	Set to send data with data chaining first packet indicator (TN#PSEND + TN#PFS).
	TN#PLINK	Set to address of second control packet.
2	TN#PFUNC	Set to send data (TN#PSEND) (not first packet or last packet).
	TN#PLINK	Set to address of third control packet.
3	TN#PFUNC	Set to send data with data chaining last packet indicator (TN#PSEND + TN#PFL).
	TN#PLINK	Set to zeros.

Naturally, all three packets must have the same channel and SLCA number. Control is returned at the completion address of the first packet, and completion or error status will be contained in it. Showing these actions in code we have:

```
SENDMSG1 EQU   *
         LA    R1,OTPTPKT1
         MVI   TN#PCHNL,2
         MVI   TN#PPORT,8
         MVI   TN#PS8OP,1
         MVI   TN#PFUNC,TN#PSEND + TN#PFS
         MVC   TN#PLNKF,=A(SENDMSG2)
         MVC   TN#PBADR,ATEXT1
         MVI   TN#PFLGS,TN#PESO
         MVC   TN#PBLTH,=H'15'
         MVC   TN#PFCPL,=A(OUTCMPL)
         LH    R3,TIME3
         STH   R3,TN#PTIME
          .
          .
SENDMSG2 EQU   *
         LA    R1,OTPTPKT2
         MVI   TN#PCHNL,2
         MVI   TN#PPORT,8
         MVI   TN#PS8OP,1
         MVI   TN#PFUNC,TN#PSEND
         MVC   TN#PLNKF,=A(SENDMSG3)
         MVC   TN#PBADR,ATEXT2
         MVC   TN#PBLTH,=H'17'
         MVC   TN#PFCPL,=A(OUTCMPL)
         LH    R3,TIME3
         STH   R3,TN#PTIME
          .
          .
SENDMSG3 EQU   *
         LA    R1,OTPTPKT3
         MVI   TN#PCHNL,2
         MVI   TN#PPORT,8
         MVI   TN#PS8OP,1
         MVI   TN#PFUNC,TN#PSEND + TN#PFL
         MVC   TN#PLNKF,=X'0000'
         MVC   TN#PBADR,ATEXT3
         MVC   TN#PBLTH,=H'34'
         MVC   TN#PFCPL,=A(OUTCMPL)
         LH    R3,TIME3
         STH   R3,TN#PTIME
         LA    R1,OTPTPKT1
         CCRCALL (1)
          .
          .
```

(continued)

```
OUTCMPL  EQU   *                         completion routine
         .
         .
         EOJ
OTPTPKT1 DS    14F'0'
OTPTPKT2 DS    14F'0'
OTPTPKT3 DS    14F'0'
ATEXT1   DC    AL3(TEXT1)
TEXT1    DC    C'NOW IS THE TIME'
ATEXT2   DC    AL3(TEXT2)
TEXT2    DC    C'FOR ALL GOOD MEN'
ATEXT3   DC    AL3(TEXT3)
TEXT3    DC    C'TO COME TO THE AID'
         DC    C' OF THEIR PARTY'
         END
```

As you can see, this has the effect of chaining the data buffers together with one service request call. The messages or text shown in the buffer are small for the sake of brevity – this type of chaining is normally performed where the message is too large for one buffer – therefore, it is usually only part of the overall message.

Message chaining, as opposed to data buffer chaining, consists of chaining complete messages together – still with the use of only one service request call. Let's use three messages again (as opposed to three buffers) to show the differences.

Packet	Field	Action
1	TN#PFUNC	Set to send complete message (TN#PSEND + TN#PFSL).
	TN#PLINK	Set to address of second control packet.
	TN#PFLGS	Set to suppress notification until all messages complete (TN#PESO).
2	TN#PFUNC	Set to send complete message (TN#PSEND + TN#PFSL).
	TN#PLINK	Set to address of third control packet.
	TN#PFLGS	Set to suppress notification until all messages complete (TN#PESO).
3	TN#PFUNC	Set to send complete message (TN#PSEND + TN#PFSL).
	TN#PLINK	Set to zeros.

Again, all three packets must have the same channel and SLCA number. Control and status are returned to the first packet. As you can see, the only difference in the coding would be to set the function byte to send complete messages.

Finally, message chaining can be used to perform the special case of chaining an output poll to an input response. In this case, the messages are still complete messages; however, the output packet uses the TN#PLINK field to chain the input packet address. For this special case, the TN#PESO bit is set to suppress notification of successful completion until the input packet without this setting has been received. Therefore we would have:

Packet	Field	Action
1	TN#PFUNC	Set to complete output message (TN#PSEND + TN#PFSL).
	TN#PFLG	Suppress notification until input complete (TN#PESO).
	TN#PLINK	Set to address of input packet.
2	TN#PFUNC	Set to complete input message (TN#PEDI + TN#PFSL).
	TN#PFLG	Set to zero.
	TN#PLINK	Set to zero.

Since a poll is actually an output message with no text, an example of buffer chaining a poll to the input response might look like the following:

```
SENDPOLL EQU   *
         LA    R1 OTPTPKT              Map
         MVI   TN#PCHNL,2
         MVI   TN#PPORT,8
         MVI   TN#PS8OP,1
         MVI   TN#PFUNC,TN#PSEND + TN#PFSL
         MVC   OUTBUFF,POLL
         MVC   TN#PLINK,RCVPKT         Chain packet address
         MVI   TN#PCMMD,0              Clear command byte
         MVI   TN#PFLGS,TN#PESO        Suppress until chaining complete
         MVC   TN#PBADR,OUTBUFAD
         MVC   TN#PBLTH,=H'5'
         MVC   TN#PFCPL,=A(INPUTMSG)
         LH    R3,TIME3
         STH   R3,TN#PTIME
         LA    R1,INPTPKT
         MVI   TN#PCMMD,0
         MVI   TN#PCHNL,2
         MVI   TN#PPORT,8
         MVI   TN#PS8OP,1
```

(continued)

```
         MVI   TN#PFUNC,TN#PEDI + TN#PFSL
         XC    TN#PLINK,TN#PLINK          Clear chaining
         MVC   TN#PBADR,INPTBFAD
         MVC   TN#PBLTH,=H'256'
         MVC   TN#PFCPL,=A(INPUTMSG)
         LH    R3,TIME3
         STH   R3,TN#PTIME
         LA    R1,OTPTPKT                 Reload first packet
         CCRCALL (1)
           .
           .
           .
INPUTMSG EQU   *                          Completion routine
           .
           .
           .
         EOJ
TIME3    DC    H'3'
OTPTPKT  DS    14 F'0'
INPTPKT  DS    14 F'0'
OUTBUFAD DC    AL3(OUTBUFF)
INBUFAD  DC    AL3(INPTBUFF)
OUTBUFF  DS    CL64
INPTBUFF DS    CL64                      /S  R S D E\
POLL     DC    X'0120507003'             |O  I I I T|
                                         \H  D D D X/
ZEROES   DC    X'000000'
         END
```

Note that the completion address of the first packet is the place in your program where control is passed. When control is returned, the logical status of the last packet (or the packet with the error) will overlay that of the first packet.

3.2. READING THE SLCA WORDS AND TABLES

Sometimes it may be desirable to know the particular line discipline that is already resident on a particular port. Or, for diagnostic purposes, it may be desirable to know the conditions that existed both before and after an input/output operation on a port. The physical interface of ICAM gives you the capability of obtaining this information with the diagnostic commands to:

- read a port control word;
- read a character detect table; and
- read a character interpretation table.

3.2.1. Reading the Port Control Words

The READ PORT CONTROL WORD command is a diagnostic command that causes the SLCA to transfer the control bytes associated with the addressed SLCA to the processor. When it is issued, the 32 bytes associated with the port control word are read.

Therefore, if we allocate space in our program for the port control word information and construct a control packet with this command, we can look at our control information at any time for any port. This may be done both before and after any input or output function completion for comparison purposes. To perform this procedure, we would first do a network request and line request as described in 2.2 and 2.3. Then, when we initialize our line, we perform a read port control word function in our control packet. This action is, in effect, the opposite operation of loading the port control word described in 2.4.2.

Field	Action
TN#PFUNC	Set to read port control word (TN#PFSL + TN#PRPCW).
TN#PBLTH	Not required, CPIOCS automatically transfers 24 bytes in order shown.
TN#PBADR	Set to address of buffer that is to contain the desired information.

These actions can be performed with the following typical coding:

```
RDPCW    EQU   *
         LA    R1,REQPKT                   Map
         MVI   TN#PCMMD,0                  Clear
         MVI   TN#PFUNC,TN#PFSL+TN#PRPCW   Set to read PCW
         MVC   TN#PBADR,APCW               Set buffer address
         XC    TN#PLINK,TN#PLINK           No chaining
         MVC   TN#PFCPL,=A(RDPCWCMP)       Set completion address
         CCRCALL (1)
          .
          .
          .
RDPCWCMP EQU   *                           Completion routine
```

In your nonexecutable code, you would allocate 24 bytes with a label of PCW.

```
APCW     DC    AL3(PCW)
PCW      DC    6 F'0'
```

3.2.2. Reading the Character Detect Table

The READ CHARACTER DETECT TABLE command is a diagnostic command that transfers a command-specified 256-byte character detect table to the processor. When it is issued, the table words are read sequentially beginning with address 00_{16} through FF_{16}. The command requires that an SLCA number referencing the table be given. (You will recall that each SLCA references at least one of the two character detect tables.)

To perform this function, we would do a network request and line request as described in 2.2 and 2.3. When we initialize our line, we would then perform a read character detect control table function in our control packet. This action is, in effect, the opposite operation of loading the character detect table described in 2.4.3.

Field	Action
TN=PFUNC	Set to read character detect table number 1 (TN#PFSL+TN#RCDn).
TN=PBLTH	Not required. CPIOCS automatically transfers 256 bytes sequentially.
TN=PBADR	Set to address of buffer that is to contain the desired information.

These actions can be performed with the following typical coding:

```
RDCDT1   EQU   *
         LA    R1,REQPKT                  Map
         MVI   TN#PCMMD,Ø                 Clear
         MVI   TN#PFUNC,TN#PFSL+TN#PRCD1  Set to read character detect
                                          table no. 1
         MVC   TN#PBADR,ACDT1
         XC    TN#PLINK,TN#PLINK
         MVC   TN#PFCPL,=A(RDCD1CMP)
         CCRCALL (1)
          .
          .
          .
RDCD1CMP EQU   *
```

Again, sufficient storage would have to be allocated in your nonexecutable code to hold the number of tables you want to read (at 256 bytes per table) either totally or one at a time. For one table, we would have merely:

```
ACDT1    EQU   AL3(CDT1)
CDT1     DC    64 F'Ø'
```

3.2.3. Reading the Character Interpretation Table

The READ CHARACTER INTERPRETATION TABLE command is a diagnostic command that transfers a command-specified 16-word character interpretation table to the processor. When it is issued, the table words are read sequentially beginning with word 00_{16} and ending with word $0F_{16}$. The command requires that an SLCA number referencing the table be given, since each SLCA references one of the character interpretation tables. (See 4.2.1.)

To perform this function, we would do a network request and line request as described in 2.2 and 2.3. When we initialize our line, we would then perform a read character interpretation table function in our control packet. This action is, in effect, the opposite operation of loading the character interpretation table as described in 2.4.4.

Field	Action
TN#PFUNC	Set to read character interpretation table number 1 (TN#PFSL +TN#RCI1).
TN#PBLTH	Not required. CPIOCS automatically transfers 16 words sequentially.
TN#PBADR	Set to address of buffer that is to contain the desired information.

These actions can be performed with the following typical coding:

```
RDCIT1   EQU   *
         LA    R1,REQPKT                 Map
         MVI   TN#PCMMD,Ø                Clear
         MVI   TN#PFUNC,TN#PFSL+TN#PRCI1 Set to read character interpretation
                                         table no. 1
         MVI   TN#PBADR,ACIT1
         XC    TN#PLINK,TN#PLINK
         MVC   TN#PFCPL,=A(RDCICMPL)
         CCRCALL (1)
          .
          .
          .
RDCICMPL EQU   *
```

Sufficient storage must be allocated in your nonexecutable code to hold the number of tables you desire to read (at 16 words per table). For one table, this would merely be:

```
ACIT1    DC    AL3(CIT1)
CIT1     DC    64 F'Ø'
```

3.3. READ LINE LINK TABLE

The read line link table function transfers the last 16 bytes of the line link table pertaining to the port specified to the user program's buffer area. To issue a read line link table command, the following actions must be performed on the specified fields:

Field	Action
TN#PMUX	Set byte 0 (TN#PCHNL) to 02, 0D, or 0F, depending on your System 80 model type (see Table A–6). Set byte 1 to the SLCA number and byte 2 to the port number.
TN#PBADR	Set to the buffer address into which the information is to be read.
TN#PBLTH	Not interpreted. The 16 bytes are transferred regardless of value.
TN#PFUNC	Set to TN#PRLLT. Bits 0 and 1 not interpreted.
TN#PLINK	Set to 0.

Again, we would load our control packet and map it. To perform the actions, our coding might appear as:

```
         LA    R1,REQPKT                Map
         MVI   TN#PCHNL,2               Set to channel 2
         MVI   TN#PPORT,8               Set to port 8
         MVI   TN#PS80P,1
         MVI   TN#PCMMD,0               Clear
         MVI   TN#PFUNC,TN#PRLLT        Set command
         XC    TN#PLINK,TN#PLINK        No chaining
         MVC   TN#PBADR,LNELNKTB        Set buffer address
         MVC   TN#PFCPL,=A(RDLLTCMP)    Set completion address
         CCRCALL (1)
          .
          .
          .
RDLLTCMP EQU   *                        Completion routine
```

After checking our status for good completion, we can look at our data in our input buffer. However, using our dummy control section technique, we must take into account the fact that the line link table is a work area and only the last 16 bytes are read into your user program. If you load a register with the address of the 16-byte area into which the line link information is to be read, the value must be modified by 20_{16} to be used properly as a cover for the DSECT. That is, TN#PLINE is the label of the DSECT, but to map the labels on your storage area (arbitrarily named LNELNKTB in the example), we would have to load the register with the storage area label and a using statement with an address of TN#PLINE+20. This would align the relevant DSECT labels with our buffer for the line link table.

Looking at our coding, we could have something like the following:

```
LA     R7,LNELNKTB
USING  TN#PLINE+20,R7
CLC    TN#PFLGT,TN#PDOWN          Line down
B
CLC    TN#PFLGT,TN#PALLO          Line allocated
B
CLC    TN#PFLGT,TN#PAPND          Allocation pending
B
CLC    TN#PFLGT,TN#PEON           EON required
B
CLC    TN#PFLGT,TN#PNEP           No physical line
B
```

or we could interrogate any other bytes of our line link table buffer area using the relevant DSECT labels.

3.4. CPIOCS TRACE

ICAM has a trace facility that saves information at critical points within CPIOCS and user program interface code.

For details on the ICAM trace facility, see the ICAM utilities user guide, UP-9748 (current version).

3.5. AUTOMATIC COMMANDS

Unless a diagnostic user specifies that automatic commands be suppressed (TN#PFLGS=TN#PS2F), CPIOCS executes commands automatically in response to the following conditions:

1. a control packet time-out occurs; and

2. a unit check interrupt occurs.

CPIOCS does the following:

- In response to a control packet time-out (TN#PTIME decremented to 0), CPIOCS issues a halt-device command.

- In response to an error interrupt, a sense command is issued. All sense bytes are placed in the CPIOCP.

Whenever there is an outstanding attention condition for a line, CPIOCS takes the next control packet you issue and forces the sending of the sense command. This is done to fulfill an SLCA requirement that a sense command be issued following detection of an unsolicited status.

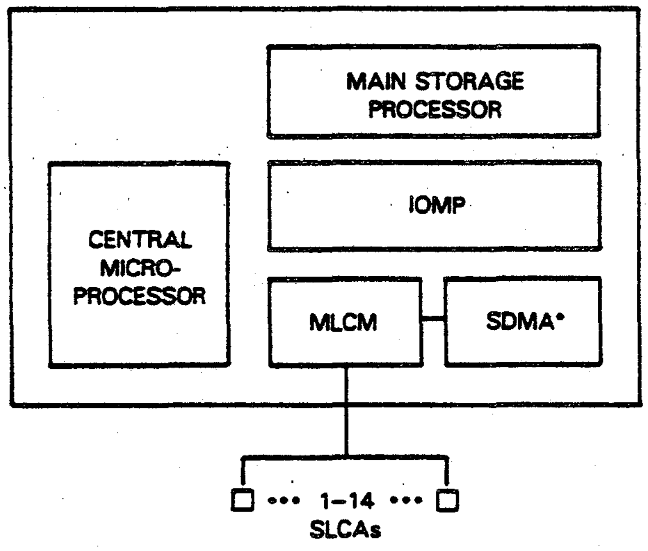
3.6. CANCEL CONDITIONS

There are three error conditions for which user program interface code will cancel the user program following execution of a CCRCALL. The first condition is when register 1 or the address in TN#PLINK of any chained control packet does not specify a full-word boundary for the control packet address. The second condition is when register 1 or the address in TN#PLINK of any chained control packet indicates that the entire control packet is not within the user program boundary. The third condition is when there is no available task identity. Under this condition, TN#PRIM is set to TN#PROGE, and TN#PDETL is set to TN#PMNCE. Refer to the OS/3 system messages programmer/operator reference, UP-8076 (current version) for a listing of ICAM cancel identification codes.

4. Single Line Communications Adapter Subsystem

4.1. GENERAL

The single line communications adapter (SLCA) is a microprocessor communications controller that coordinates data transfer, status, and commands between the central processor and remote devices via communications lines. In the basic System 80 Models 3–6, up to two SLCAs are supported. With a single input/output microprocessor attached, Models 3–6 support one to eight SLCAs and Model 8 supports up to 14 SLCAs (Figure 4–1). With a dual input/output microprocessor attached, Model 8 supports up to 28 SLCAs. An IOMP is connected to the SLCAs via the multiple line communications multiplexer (MLCM), also known as the communications multiplexer channel. For a more detailed description of the SLCA, see Section 3 of the System 80 IOMP programmer reference, UP-8909.



*Shared direct memory access channel (device channel)

Figure 4–1. System 80 Model 8 with Single Input/Output Microprocessor

The SLCAs can be configured to support half-duplex or full-duplex communications lines. Logic functions can be dynamically configured by down-line loading the SLCAs random access memory (RAM) by ICAM. The RAM can be loaded by ICAM and configured to operate in the basic read only memory function, or according to one of the following preset features (Table 4–1):

- Communications multiplexer module synchronous mode
- Communications multiplexer module asynchronous mode
- Universal data link control (UDLC) mode

In the communications physical interface, you must construct your own tables of logic functions in an ICAM user program and load them into the read only memory via your user program.

Table 4–1. Single Line Communications Adapters

Feature Number	Interface Specification	Transmission Rate (bits/s)	Half Duplex (HD) Full Duplex (FD)	Async/ Sync	Bit/ Byte	Protocol (See notes.)	Microcode Name 4
F2788-02	RS-232	9600 4800	HD FD	Sync	Byte	1	CMM1
F2788-03	MIL-STD-188	9600 4800	HD FD	Sync	Byte	1	CMM1
F2788-03	RS-232	9600 9600	HD FD	Sync	Byte	NTR	NTR1
F2788-04	MIL-STD-188	9600 9600	HD FD	Sync	Byte	NTR	NTR1
F2798-00	RS-232	19200 9600	HD FD	Sync	Bit	2	UDLC
F2799-00	RS-232	9600 4800	HD FD	Sync	Bit	2	CMM2
F2799-01	MIL-STD-188	9600 4800	HD FD	Async	Byte	3	CMM2
F2986-05	CCITT V.35	56000 56000	HD FD	Sync	Byte	NTR	NTR5
F3794	RS366	Autodial					
F2986 00	CCITT V.35	56000	FD	Sync	Byte	BSC	BSC5

NOTES

1. Any standard synchronous byte protocol, e.g., UNISCOPE, BSC, 1004, NTR, or remote workstation
2. Handles any standard bit-oriented protocol, e.g., X.25 on UDLC ABM
3. Any standard asynchronous protocol, e.g., UNISCOPE display terminal, teletypewriter, DCT 500, UTS 10
4. The microcode name is actually eight characters. The first four specify the microcode name, the next two specify the release version, and the last two are zeros; for example, CMM1F000. The release version is supplied in the software release description (SRD) for each release.

4.2. CONSTRUCTING THE SLCA WORDS AND TABLES

As discussed in 2.4, each port in the SLCA requires a port control word, a control character detect table, and a control character interpretation table. A table of each type is allocated in the SLCA. One of the functions of the port control word is to reference one of the character detect and character interpretation tables. These tables are constructed to detect and interpret the control characters necessary for the particular line discipline you wish to support. At the physical level interface, *you* must construct these tables within your own user program.

In the following sections, we'll discuss how to construct these words and tables from listings of line discipline common characteristics. These listings are shown within the text as we proceed to build our words and tables. Section 5 contains table summaries of the initialization parameters created from these lists for currently supported terminals. Therefore, we'll first discuss how to generate the words and tables from the characteristics for the DCT 2000, since this was the terminal used in our example in 2.4. Then we'll show how the tables of initialization parameters may also be used to directly code the control tables when you are using ICAM-supported terminals. In this case, you will see that the tables arrived at by either method are identical.

4.2.1. How to Construct a Port Control Word

A port control word consists of four port control bytes that give specific information to the SLCA. Its basic format is:

byte 1	byte 2	byte 3	byte 4

where:

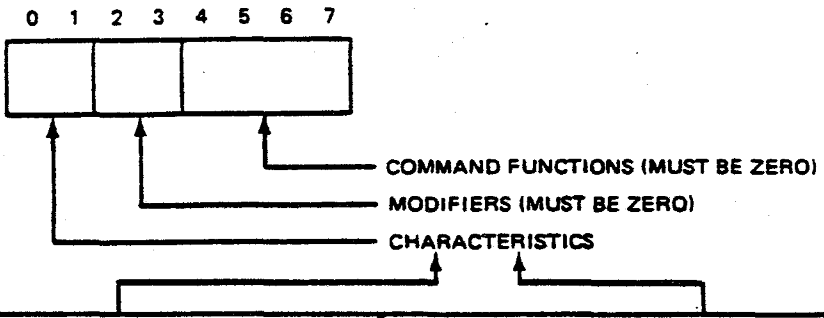
byte 1
: Contains synchronous/asynchronous characteristics.

byte 2
: Indicates the character detect table to be used and contains character length and asynchronous line speed, if applicable.

byte 3
: Indicates the character interpretation table to be used, the start character for BCC accumulation, and parity type.

byte 4
: Concerns synchronous/asynchronous selection and timing values and ranges.

Figure 4–2 shows byte 1 in more detail.



Synchronous Operation		Asynchronous Operation	
Bit 0 1	Characteristic	Bit 0 1	Characteristic
X 0	Input (specific start characters)	0 0	Invalid*
X 1	Input (any nonsync start)	0 1	Output (1-unit interval stop element)
0 X	Output (2 pads + 2 syncs)	1 0	Output (1-1/2-unit interval stop elements)
1 X	Output (4 syncs)	1 1	Output (2-unit interval stop elements)

* Results in program-alert sense and unit-check sense.

NOTE:

x indicates a "don't care" bit.

Figure 4–2. Byte 1 of Port Control Word

Therefore, if, for example, our terminal were a synchronous operation with four sync characters on output, and assuming zeros whenever a bit setting is not specified, we would have:

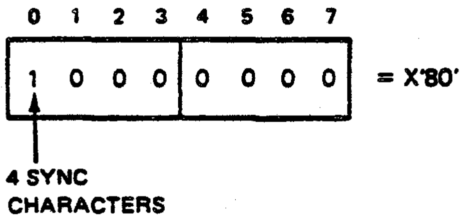
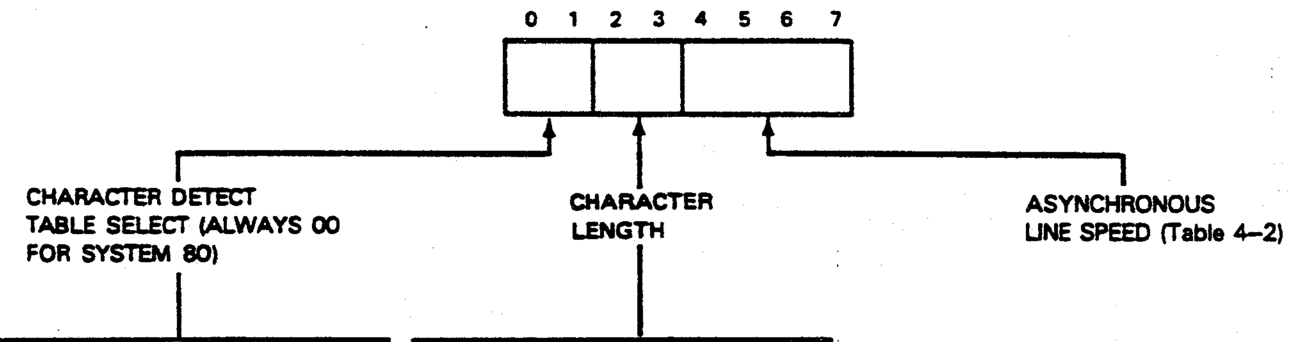


Figure 4-3 shows the format and characteristics for byte 2. Table 4-2 gives asynchronous lines speeds.


0 1 2 3 4 5 6 7
CHARACTER DETECT TABLE SELECT (ALWAYS 00 FOR SYSTEM 80)
CHARACTER LENGTH
ASYNCHRONOUS LINE SPEED (Table 4-2)

Bit 0 1	Character Detect Table
0 0	Table 1
0 1	Table 2

Bit 2 3	Character Length (excluding parity bit)
0 0	5 bits per character
0 1	6 bits per character
1 0	7 bits per character
1 1	8 bits per character

Figure 4-3. Byte 2 of Port Control Word

Table 4-2. Asynchronous Line Speeds

Bit 4 5 6 7	Asynchronous Speed (bps)
0 0 0 0	Not used (invalid *)
0 0 0 1	50
0 0 1 0	75
0 0 1 1	110
0 1 0 0	134.5
0 1 0 1	150
0 1 1 0	300
0 1 1 1	600
1 0 0 0	900
1 0 0 1	1200
1 0 1 0	1800
1 0 1 1	2400
1 1 0 0	3600
1 1 0 1	4800
1 1 1 0	7200
1 1 1 1	9600

*Results in program-alert-sense and unit-check status

First, we specify character detect table 1 and assume our terminal needs 7-bit character length. The second byte of our port control word would now look like:

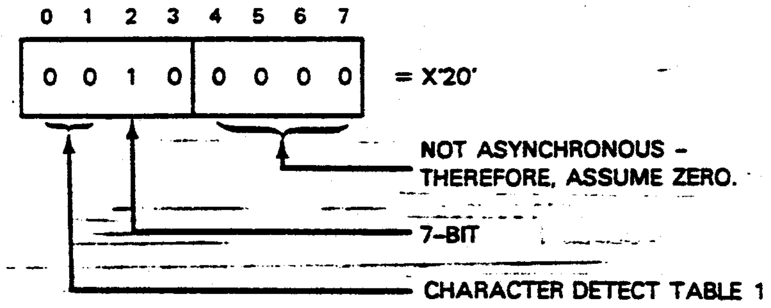


Figure 4–4 shows the format and characteristics for byte 3 and Table 4–3 gives its parity functions.

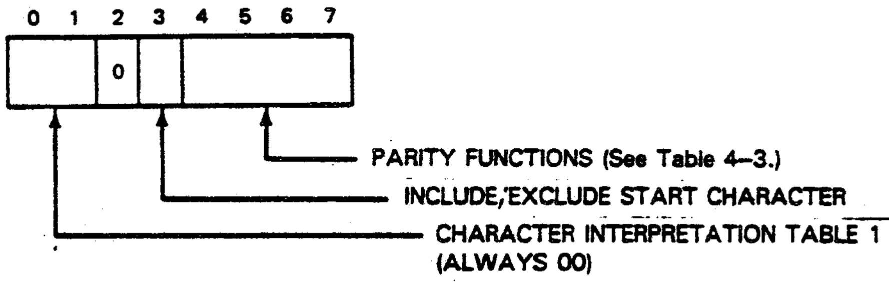

0 1 2 3 4 5 6 7
0
PARITY FUNCTIONS (See Table 4–3.)
INCLUDE/EXCLUDE START CHARACTER
CHARACTER INTERPRETATION TABLE 1 (ALWAYS 00)

Figure 4–4. Byte 3 of Port Control Word

Table 4-3. Parity Functions for Port Control Word Byte 3

Bit 4 5 6 7	VRC	Control Character	LRC Character Parity	LRC Selection	CRC Selection (Note 1)	Application
0 0 0 0	None	None	None	None	None	
0 0 0 1	Odd	Even	None	None	None	CTMC/DCS compatible
0 0 1 0	Odd	Odd	None	None	None	CTMC/DCS compatible
0 0 1 1	Even	Even	None	None	None	CTMC/DCS compatible
0 1 0 0	Even	Odd	None	None	None	CTMC/DCS compatible
0 1 0 1	Even	Even	Even	Even	None	ANSI async standard
0 1 1 0	Invalid	Invalid	Invalid	Invalid	None	
0 1 1 1	Invalid	Invalid	Invalid	Invalid	None	NTR/1004 (synchronous)
1 0 0 0	Odd*	Even*	Even*	Even*	None	1004 (DLT-1 or DLT-3)
1 0 0 1	Odd	Odd†	Even††	Odd	None	ANSI sync standard
1 0 1 0	Odd	Odd	Even	Even	None	U300 (synchronous)
1 0 1 1	Odd*	Odd*	Even*	Odd*	None	BSC ASCII nontransparent
1 1 0 0 (Note 2)	Odd†	Odd*	Invalid	Invalid	1	BSC ASCII transparent
1 1 0 1 (Note 2)	Invalid	Invalid	Invalid	Invalid	1, 4	BSC EBCDIC or 6-bit code
1 1 1 0	Invalid	Invalid	Invalid	Invalid	2	CCITT
1 1 1 1	Invalid	Invalid	Invalid	Invalid	3	Selectable CRC (not used)

* Invalid in asynchronous mode

† Even in asynchronous mode

†† Odd in asynchronous mode

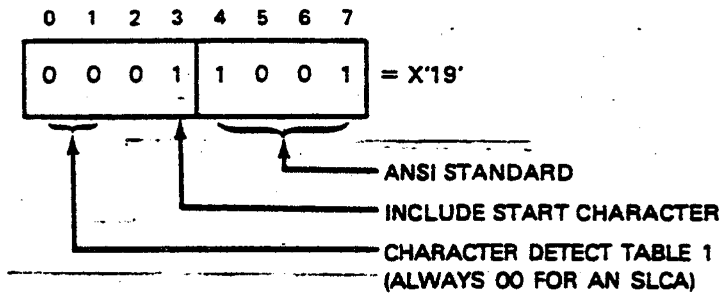
NOTES (apply to synchronous mode only):

1. CRC 1: $x^{16} + x^{15} + x^2 + 1$ (CRC-16)
   CRC 2: $x^{16} + x^{12} + x^5 + 1$ (CCITT)
   CRC 3: 16-bit strap selected (special order)
   CRC 4: $x^{12} + x^{11} + x^3 + x^2 + x + 1$ (CRC-12)

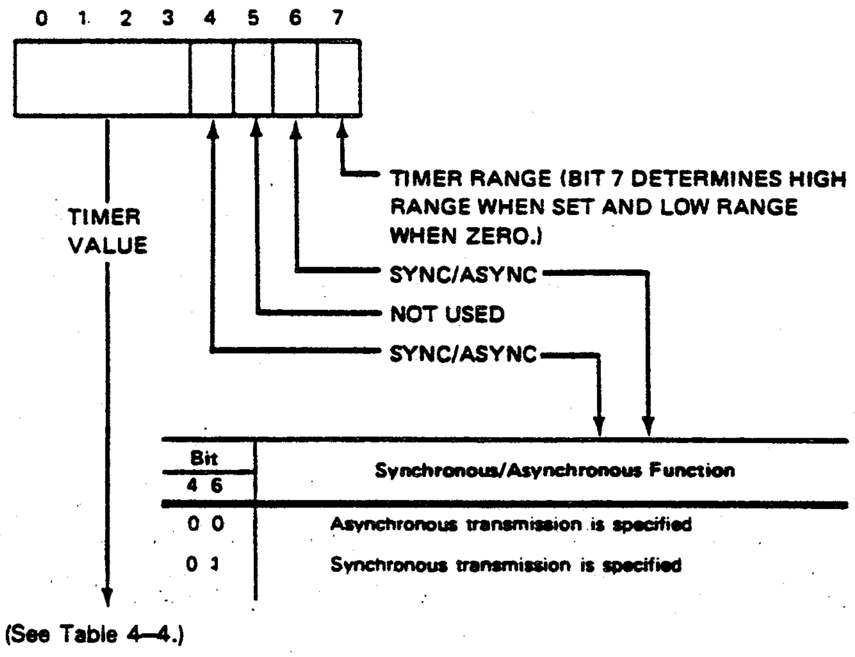
2. When parity function 1100_2 is used, 7-bit character size must be specified. When parity function 1101_2 is used, 6-bit or 8-bit character size must be specified. When parity function 1110_2 is used, 8-bit character size must be specified.

Next, we must use character interpretation table 1. For our discipline, we want to include the start character in the block check character count (BCC) and we want ANSI standard parity, which means even longitudinal redundancy character (LRC) and odd vertical redundancy check (VRC).

Going to our tables and selecting the relevant settings, we have for byte 3:



Finally, Figure 4–5 shows port control byte 4. Table 4–4 gives high and low elapsed time ranges.


0 1 2 3 4 5 6 7
TIMER RANGE (BIT 7 DETERMINES HIGH RANGE WHEN SET AND LOW RANGE WHEN ZERO.)
SYNC/ASYNC
NOT USED
SYNC/ASYNC
TIMER VALUE
(See Table 4–4.)

Bit 4 6	Synchronous/Asynchronous Function
0 0	Asynchronous transmission is specified
0 1	Synchronous transmission is specified

Figure 4–5. Byte 4 of Port Control Word

To get the data placed in the SLCA buffer when a CPIOCP input time-out occurs, you must specify a line procedure timer value.

Table 4–4. High and Low Elapsed Time Ranges

High Range*		
Control Byte 4 Bit	**Elapsed Time (seconds)**	
0 1 2 3	**Maximum**	**Minimum**
0 0 0 0	(Timer stopped)	(Timer stopped)
0 0 0 1	(Timer stopped)	(Timer stopped)
0 0 1 0	0.142	0.071
0 0 1 1	0.213	0.142
0 1 0 0	0.284	0.213
0 1 0 1	0.355	0.284
0 1 1 0	0.426	0.355
0 1 1 1	0.710	0.426
1 0 0 0	0.994	0.710
1 0 0 1	1.278	0.994
1 0 1 0	1.562	1.278
1 0 1 1	1.846	1.562
1 1 0 0	2.130	1.846
1 1 0 1	2.414	2.130
1 1 1 0	2.698	2.414
1 1 1 1	3.834	2.698

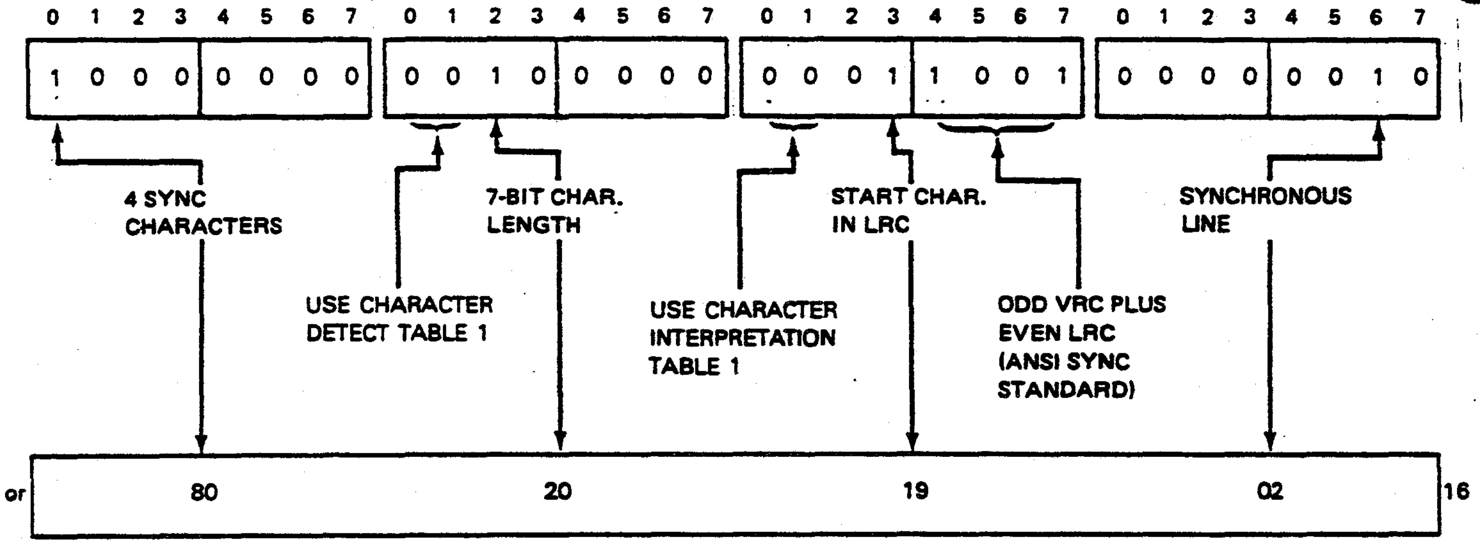
Low Range		
Control Byte 4 Bit	**Elapsed Time (seconds)**	
0 1 2 3	**Maximum**	**Minimum**
0 0 0 0	(Timer stopped)	(Timer stopped)
0 0 0 1	(Timer stopped)	(Timer stopped)
0 0 1 0	4.544	2.272
0 0 1 1	6.816	4.544
0 1 0 0	9.088	6.816
0 1 0 1	11.360	9.088
0 1 1 0	13.632	11.360
0 1 1 1	15.904	13.632
1 0 0 0	18.176	15.904
1 0 0 1	20.448	18.176
1 0 1 0	22.720	20.448
1 0 1 1	24.992	22.720
1 1 0 0	27.264	24.992
1 1 0 1	29.536	27.264
1 1 1 0	31.808	29.536
1 1 1 1	34.080	31.808

* For this timer range, the timer will reset and continue timing when a SYN character is sent or received in addition to those reset conditions previously discussed.

So, for this byte we'll select synchronous transmission, and, for the sake of simplicity, we'll not use the timer. Therefore, we have merely:

0 1 2 3	4 5 6 7	
0 0 0 0	0 0 1 0	= X'02'

Summing it all up, our port control word looks like:



Thus, we now add a label and place this word in the nonexecutable portion of our code as:

```
DCT2PCW  DC    X '80201902'
```

This word is placed in our output buffer at line initialization time with:

```
            MVC    OUTBUFF,DCT2PCW
```

The port control word can then be loaded as described in 2.4.2.

If you are using a standard remote device handler (that is, a handler written for a device currently supported by ICAM), the tables in Section 5 can be used directly to construct your port control word. Again, using the DCT 2000 as our example, Table 5–7 shows the following:

Byte	Bit	Value (binary)	Function
1	0, 1	10	Four SYN characters on output
2	2, 3	10	Seven-bit character length
3	3 4–7	1 1001	Include start character in LRC Odd VRC plus even LRC
4	4–6	001	Synchronous line

Coding this directly gives us:

BYTE 1							
0	1	2	3	4	5	6	7
1	0						

BYTE 2							
0	1	2	3	4	5	6	7
		1	0				

BYTE 3							
0	1	2	3	4	5	6	7
			1	1	0	0	1

BYTE 4							
0	1	2	3	4	5	6	7
				0	0	1	

and, assuming all zeros for any bits not specified, we would again have X'80201902' – the same value as obtained from our previous characteristic tables for our hypothetical case. Thus, it is easier to use the tables already constructed for us when we have one of the supported devices; we must use the characteristics table when we have a unique device.

4.2.2. How to Construct a Character Detect Table

The character detect table is used to detect single or multiple function characters. It uses the low-order 5 bits of a byte to signify the particular function. When bit 3 is *not* set, it signifies a single function; when it *is* set, it signifies a multiple function. For instance:

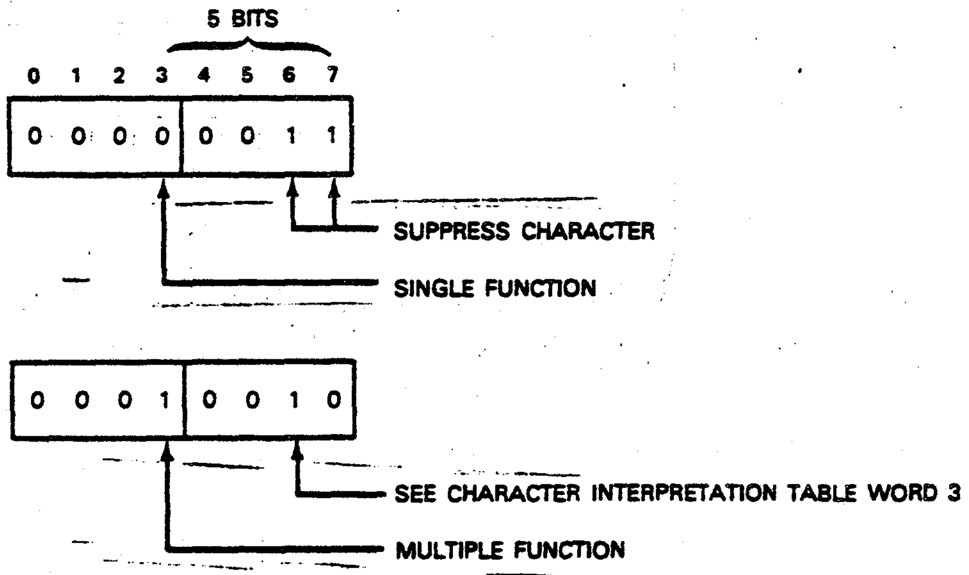


Table 4–5 gives the functions in effect when bit 3 is zero and the detected character *is* preceded by a DLE character.

Table 4–5. CD Table Functions When Preceded by DLE Character

Bit Position 3 4 5 6 7	Designation	Function
0 1 0 0 0	NO-OP	Produces no control function
0 1 0 0 1	NO-OP	Produces no control function
0 1 0 1 0	END-CHARACTER	Stops input data and causes DEVICE END status to be set. Ends two character sequences, DLE 0 and DLE NAK. This function is active only if a BCC is not being accumulated when this character is detected.
0 1 0 1 1	NO-OP	Produces no control function
0 1 1 0 0	NO-OP	Produces no control function
0 1 1 0 1	MONITOR	Causes UNIT CHECK status and the MONITOR sense bit to be set. Active only after detection of a START CHARACTER and before detection of an END CHARACTER. Input or output data flow stops until a new command is received.
0 1 1 1 0	NO-OP	Produces no control function
0 1 1 1 1	NO-OP	Produces no control function

Table 4–6 gives the functions in effect when bit 3 is zero and the detected character is *not* preceded by a DLE character.

Table 4–6. CD Table Functions When Not Preceded by DLE Character (Part 1 of 2)

Bit Position 3 4 5 6 7	Designation	Function
0 0 0 0 0	NO-OP (Data)	Produces no control function
0 0 0 0 1	NO-OP	Produces no control function
0 0 0 1 0	NO-OP	Produces no control function
0 0 0 1 1	SUPPRESS CHARACTER	The character is suppressed and not transferred to the processor. Active on input only.
0 0 1 0 0	START-END	Produces both START CHARACTER and END CHARACTER functions and is used for single character replies, for example, ACK and NAK. This function is active only if a BCC is not being accumulated when this character is detected. Generates DEVICE END status.

Table 4–6. CD Table Functions When Not Preceded by DLE Character (Part 2 of 2)

Bit Position	Designation	Function
3 4 5 6 7		
0 0 1 0 1	MONITOR	Causes UNIT CHECK status and the MONITOR sense bit to be set. Active only after detection of a START CHARACTER and before detection of an END CHARACTER. Input or output data flow stops until a new command is received. Normally used to monitor output text for illegal text characters
0 0 1 1 0	NO-OP	Produces no control function
0 0 1 1 1	NO-OP	Produces no control function

When bit 3 is set for multiple functions, the second half-byte now refers to a specific word of a character interpretation table to supply the additional information. Table 4–7 describes the format in this case.

Table 4–7. CD Table Codes Accessing the CI Table

Bit Position	Function
3 4 5 6 7	
1 0 0 0 0	SYN* – See word 0 (CI table access for asynchronous mode).
1 0 0 0 1	DLE* – See word 1.
1 0 0 1 0	CI table access only – See word 2.
1 0 0 1 1	CI table access only – See word 3.
1 0 1 0 0	CI table access only – See word 4.
1 0 1 0 1	CI table access only – See word 5.
1 0 1 1 0	CI table access only – See word 6.
1 0 1 1 1	EOT* – See word 7.
1 1 0 0 0	CI table access only – See word 8.
1 1 0 0 1	CI table access only – See word 9.
1 1 0 1 0	CI table access only – See word 10.
1 1 0 1 1	CI table access only – See word 11.
1 1 1 0 0	CI table access only – See word 12.
1 1 1 0 1	CI table access only – See word 13.
1 1 1 1 0	CI table access only – See word 14.
1 1 1 1 1	CI table access only – See word 15.

* These codes access the CI table and also produce hardwired functions; therefore, they must be used only for the indicated control character. An input DLE EOT sequence always causes a disconnection when the 11_{16} and 17_{16} codes appear in the CI table.

Armed with this format information, as each input or output character is received, it is used as an address to access the character detect table. The control code is read from the table and the relevant functions are produced. Since either parity may be used, this fact must be taken into consideration when constructing the table. That is, output characters address a table before a parity bit is inserted (when used). Input characters are used to address the table before the parity bit is stripped.

To make it easier for you to see an example of this, the control code ASCII hexadecimal values are shown here with their functions.

00	NUL	08	BS	10	DLE	18	CAN
01	SOH ✓	09	HT	11	DC1 ✓	19	EM ✓
02	STX	0A	LF	12	DC2	1A	SUB
03	ETX ✓	0B	VT	13	DC3	1B	ESC
04	EOT ✓	0C	FF	14	DC4	1C	FS
05	ENQ	0D	CR	15	NAK ✓	1D	GS
06	ACK ✓	0E	SO	16	SYN ✓	1E	RS
07	BEL ✓	0F	SI	17	ETB	1F	US

In constructing a table then, let's assume we need the codes checked, which in reality are the codes required for the DCT 2000. We also require that, on input, we can have only an ETX, ACK, and DC1. Since input means before parity is stripped, on input these characters would look like 83, 86, and 91, respectively, since the most significant bit would be set.

As we mentioned, the character itself is used as an address to access the table; therefore, the table must be constructed similar to a translate table (Figure 4–6).

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	00	12	00	13	17	00	14	14	00	00	00	00	00	00	00	00
10	00	14	00	00	00	14	10	00	00	03	00	00	00	00	00	00
20	00															→ 00
30	00															→ 00
40	00															→ 00
50	00															→ 00
60	00															→ 00
70	00															→ 00
80	00	00	00	13	00	00	14	00	00	00	00	00	00	00	00	00
90	00	14	00	00	00	00	00	00	00	00	00	00	00	00	00	00
A0	00															→ 00
B0	00															→ 00
C0	00															→ 00
D0	00															→ 00
E0	00															→ 00
F0	00															→ 00

Figure 4–6. Character Detect Table

This tells us that, when an SOH (X'01') is detected, the SLCA will go to location X'01' in our character detect table to find the functions to be executed. In this case, the function is a X'12'. Since bit 3 is set in a X'12', we know from our previous discussion that it signifies a multiple function, which is contained in word 2 of the relevant character interpretation table. An ETX (X'03') would access location X'03' of our table, which in turn signifies a multiple function contained in word 3 of a character interpretation table.

Coding our table as shown then, assuming all locations not used to be zeros, and giving the table a label, we have:

```
DCT2CDT  DC    X'00 12 00 13 17 00 14 14 00 00 00 00 00 00 00 00' 00-0F
               X'00 14 00 00 00 14 10 00 00 03 00 00 00 00 00 00' 10-1F
               X'00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00' 20-2F
               X'00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00' 30-3F
               X'00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00' 40-4F
               X'00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00' 50-5F
               X'00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00' 60-6F
               X'00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00' 70-7F
               X'00 00 00 13 00 00 14 00 00 00 00 00 00 00 00 00' 80-8F
               X'00 14 00 00 00 00 00 00 00 00 00 00 00 00 00 00' 90-9F
               X'00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00' A0-AF
               X'00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00' B0-BF
               X'00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00' C0-CF
               X'00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00' D0-DF
               X'00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00' E0-EF
               X'00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00' F0-FF
```

Again, as in the port control word, if you are using a standard supported device, such as the DCT 2000, the relevant table in Section 5 can be used directly to construct your character detect table. For the DCT 2000, the following initialization parameters are summarized:

ASCII Character (hexadecimal)	Name	CD Table Setting② (hexadecimal)	Meaning of Setting
01	SOH	12	See CI word 2
03	ETX	13	See CI word 3
04	EOT	17	See CI word 7
06	ACK	14	See CI word 4
07	BEL	14	See CI word 4
11	DC1	14	See CI word 4
15	NAK	14	See CI word 4
16	SYN	10	See CI word 0
19	EM	03	Suppress character (input)
83①	ETX	13	See CI word 3
83①	ACK	14	See CI word 4
91①	DC1	14	See CI word 4

NOTES:

① These characters occur during input only (before parity is stripped from the character).

② All other CD table settings will be zero, implying data characters.

With this method, the value contained in the CD table setting column is placed in our coded table at the location corresponding to the value in the ASCII character column:

location 01 = 12
location 03 = 13
location 04 = 17

.
.
.

location 91 = 14

Assuming all locations other than those specified to be zero, you would derive the same table that was constructed by our previous method. Thus, it is easier to use the tables already constructed for us when we have one of the supported devices, but we must construct our own from scratch when we have a unique device.

As before, we place our table in the nonexecutable portion of our program and move it into our output buffer at line initialization time with:

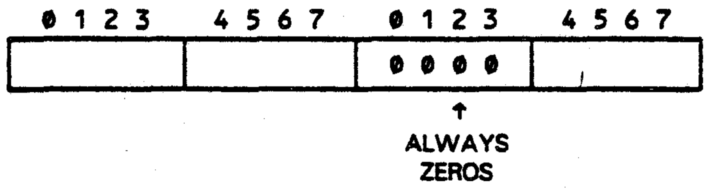
```
MVC   OUTBUFF,DCT2CDT
```

The character detect table can then be loaded as described in 2.4.3.

4.2.3. How to Construct a Character Interpretation Table

The character interpretation table is composed of sixteen 12-bit words. These words are accessed by the character detect table to support multiple functions. Each bit of a character interpretation table word produces a single function when set. Thus, each word can cause multiple functions to be performed.

The format of a CI table word is:



where the bit settings cause the functions to be performed as given in Table 4–8.

Table 4–8. Character Interpretation Table Functions (Part 1 of 2)

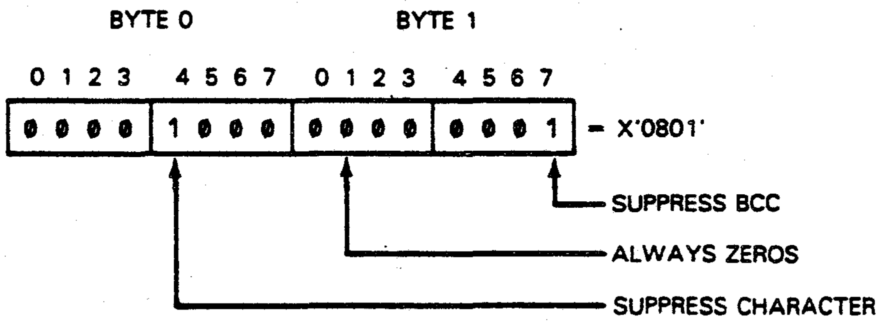
Bit	Designation	Function
BYTE 0		
0	INTERMEDIATE END CHARACTER	This bit is normally used to detect an ITB character. During an input, the following BCC is checked and results are stored until the end-character code (ETB or ETX) is detected, at which time the appropriate status and sense bits are set. No automatic look-for-sync function occurs, and input data flow is not stopped regardless of the BCC checking results. A new BCC accumulation is started and includes the next non-SYN character received. During an output, detection of the ITB character causes transmission of BCC. Output data flow is not stopped and a new BCC is started on the next character received from the MLCM to be an output.
1	END CHARACTER	Channel-end, device-end status is set. If applicable, the following BCC is checked. Automatic look-for-sync function commences after the BCC is received. This bit stops input or output data flow until a new command is received.
2	ABORT	Must be preceded by a start character. Stops input data flow and sets channel-end, device-end, as well as unit-check status bits. Also sets abort sense bit. This bit is active only on input. Automatic look-for-sync function occurs immediately.
3	START BCC	Starts BCC accumulation, but ignored if accumulation has already been started. This bit is active on input as well as output.
4	SUPPRESS CHARACTER	Character is not transferred to the MLCM by the SLCA. This bit is active on input only. The character is included in the BCC unless bit 11 is also set to 1.
5	Not used	Set to 0.
6*	MONITOR	Causes unit-check status and monitor sense bits to be set. This bit is active only on output.
7	START CHARACTER	First specific non-SYN character that starts input data. If a start character has already been detected, this bit has no effect.

Table 4–8. Character Interpretation Table Functions (Part 2 of 2)

Bit	Designation	Function
BYTE 1		
0–3	Not used	
4	EXCLUDE DLE CHARACTER	This character and the preceding DLE character are excluded from the BCC accumulation when they occur after a start character. This function is active only if the detected character was preceded by a DLE character. (These characters are used in ANSI control procedures for embedding acknowledgment in the text). Active only when LRC type parity function is specified.
5	RESET TRANSPARENT	Places SLCA port into nontransparent mode. This bit is used only for input. This function is active only if the detected character was preceded by a DLE character. Active only when set in conjunction with end (bit 1) or ITB (bit 0)
6	SET TRANSPARENT	Places SLCA port into transparent mode. This bit is active for input as well as output. This function is active only if the detected character was preceded by a DLE character.
7	SUPPRESS BCC	The accumulated BCC is suppressed for this character. This bit is not active in transparent mode. The character is transferred to the MLCM unless bit 4 also is set to 1.

* Bits 1 and 6 both set to 1 in the same CI word, upon detection of the control character in the data stream, would result in performance of the functions specified for bit 0, but functions specified individually for bits 1 and 6 would be ignored by the SLCA.

To illustrate the use of this table, we might want to have the functions of suppressing the block character check count and also suppressing the character on input. Using our table and setting the relevant bits we would have:



Considering the DCT 2000 discipline in our example, let's make a list of some of the multiple functions required:

1. A SYN function, when hardwired, requires an additional suppress character and suppress BCC.

2. An SOH requires start character, start BCC, and suppress BCC functions.

3. An ETX requires a suppress character and end character functions.

4. An EOT requires a start character function.

5. The BEL, DC1, NAK, and ACK require an additional start character function.

Putting all these requirements into our character interpretation table, we have the table shown in Figure 4–7.

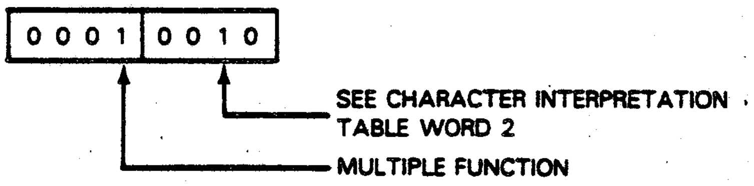
			ALWAYS ZERO			
	0 1 2 3	4 5 6 7	0 1 2 3	4 5 6 7	HEX.	FUNCTION
WORD 0	0 0 0 0	1 0 0 0	0 0 0 0	0 0 0 1	= X'0801'	(SYN)
		↑ SUPPRESS CHARACTER		↑ SUPPRESS BCC		
1	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	= X'0000'	
2	0 0 0 1	0 0 0 1	0 0 0 0	0 0 0 1	= X'1101'	(SOH)
	↑ START BCC	↑ START CHARACTER		↑ SUPPRESS BCC		
3	0 1 0 0	1 0 0 0	0 0 0 0	0 0 0 0	= X'4800'	(ETX)
4	0 1 0 0	0 0 0 1	0 0 0 0	0 0 0 0	= X'4100'	(BEL,DC1,) (NAK,ACK)
	↑ END CHARACTER	↑ START CHARACTER				
5	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	= X'0000'	
6	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	= X'0000'	
7	0 1 0 0	0 0 0 1	0 0 0 0	0 0 0 0	= X'4100'	(EOT)
	↑ END CHARACTER	↑ START CHARACTER				
8–15	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	= X'0000'	

Figure 4–7. Character Interpretation Table

Putting this all together, and adding a label, we have:

```
DCT2CIT  DC    X'0801'
         DC    X'0000'
         DC    X'1101'
         DC    X'4800'
         DC    X'4100'
         DC    X'0000'
         DC    X'0000'
         DC    X'4100'
         DC    X'0000'
         DC    X'0000'
         DC    X'0000'
         DC    X'0000'
         DC    X'0000'
         DC    X'0000'
         DC    X'0000'
         DC    X'0000'
```

Once the character interpretation table is complete, we may then go back to the character detect table and fill in the word numbers that we need to fulfill our list of requirements. For example, our first requirement was that an SOH required the additional functions of start of data, start BCC, and suppress BCC. Therefore, we can now see why we used a X'12' in location 01 of the character detect table. When an SOH is detected, the SLCA accesses location 01 of the character detect table and finds the value X'12'. This value is determined to mean



Summing the functions then, we have:

SOH, start character, start BCC, suppress BCC

which is what we require.

Again, going to Section 5, we find that the initialization parameters for all the *currently supported* devices are contained in tabular form. If we are using a supported device, we may use these tables to *directly* code our character interpretation table. Using our DCT 2000 example, we have the following:

CI Word	CI Table Setting* (hex)	Meaning of Setting
0	0801	SYN character (hardwired function), suppress character, suppress BCC
1	-	Not used
2	1101	Start characters plus start BCC plus suppress BCC
3	4800	Suppress character plus end character
4	4100	Start character plus end character
5	-	Not used
6	-	Not used
7	4100	Start character plus end character
8 through 15	-	Not used

*All other CI table settings will be zero.

As you can see, this is identical to the table arrived at by our previous method. It is just a table preconstructed for us to make it easier to create our coding tables when using supported devices. If we are using unique devices, we must construct our tables from the first method, whereby we constructed our words from the character interpretation table bit settings. As before, once our table is coded, we place it in the nonexecutable portion of our program. At line initialization time, we then place it in our output buffer via

```
MVC   OUTBUFF,DCT2CIT
```

before performing the load character interpretation function. The character interpretation table can then be loaded as described in 2.4.4.

5. SLCA Table Initialization

5.1. GENERAL

The initialization parameters for the SLCA that are pertinent to the terminal handlers are listed in the following tables. The definitions in each paragraph are the numeric values to be preset into the control character detect table, control character interpretation table, and port control word for each terminal currently supported by SPERRY software.

5.2. DCT 500 SERIES/TELETYPE* TELETYPEWRITER REMOTE DEVICE HANDLER

The DCT 500 series/TELETYPE Teletypewriter remote device handler provides the initialization parameters listed in Tables 5–1 through 5–5.

Table 5–1. DCT 500 Series/TELETYPE Teletypewriter ASCII Control Character Detect Table

ASCII Character① (hexadecimal)	Name	CD Table Setting② (hexadecimal)	Meaning of Setting
00	NUL	03	Suppress input character
03	ETX	15	See CI word 5
05	ENQ	18	See CI word 8
06	ACK	19	See CI word 9
10	DLE	11	See CI word 1
15	NAK	1A	See CI word 10
7F	DEL	03	Suppress input character

NOTES:

① Depending upon the parity of the characters transmitted by the terminal, it may be necessary to have duplicate settings for both parities of the characters; for example, an odd-parity DCT 500 must have the setting 15_{16} for both 03_{16} and 83_{16}.

② All other CD table settings will be zero, implying data characters.

Trademark of Teletype Corporation

Table 5-2. DCT 500 Series/TELETYPE Teletypewriter ASCII Control Character Interpretation Table

CI Word	Setting (hexadecimal)	Meaning of Setting
0	0000	DLE EOT disconnects line
1	–	Not used
2	–	Not used
3	–	Not used
4	–	Not used
5	4000	End character
6	–	Not used
7	–	Not used
8	4000	End character
9	4000	End character
10	4000	End character
11	4000	End character
12	4000	End character
13	–	Not used
14	–	Not used
15	–	Not used

Table 5-3. DCT 500 Series/TELETYPE Teletypewriter Baudot Control Character Detect Table

Baudot Character (hexadecimal)	Name	CD Table Setting(1) (hexadecimal)	Meaning of Setting
00	NUL	03	Suppress input character
14	H/	11	See CI word(1)
1F	LTRS	0A	H-LTRS ends input(2)

NOTES:

(1) All other CD table settings will be zero, implying data characters.

(2) The table assumes that the sequence H-LTRS will be chosen by the user to signal end of input.

Table 5-4. DCT 500 Series/TELETYPE Teletypewriter Baudot Control Character Interpretation Table

CI Word	Setting (hexadecimal)	Meaning of Setting
0	-	Not used
1	0000	Set DLE
2 through 15	-	Not used

Table 5-5. DCT 500 Series/TELETYPE Teletypewriter Port Control Word (Part 1 of 2)

Byte	Bit	Value (binary)	Meaning
1	0, 1	11	Two-unit interval stop element (for 110 bps operation)
		10	One-unit interval stop element (for all other transmission rates)
2	2, 3	00	For Baudot terminals
		10	For ASCII terminals
	4-7	0000	Not used (invalid*)
		0001	50 bps
		0010	75 bps
		0011	110 bps
		0100	134 bps
		0101	150 bps
		0110	300 bps
		0111	600 bps
		1000	900 bps
		1001	1200 bps
		1010	1800 bps
		1011	2400 bps
		1100	3600 bps
		1101	4800 bps
		1110	7200 bps
		1111	9600 bps

*Permitted values

Table 5–5. DCT 500 Series/TELETYPE Teletypewriter Port Control Word (Part 2 of 2)

Byte	Bit	Value (binary)	Meaning
3	3–7	0000	No parity checking*
		0010	Odd parity*
		0011	Even parity*
4	4–6	000	Asynchronous transmission

* Permitted values

5.3. DCT 2000 REMOTE DEVICE HANDLER

The initialization parameters provided in the DCT 2000 remote device handler for the control character tables and port control word are listed in Tables 5–6 through 5–8.

Table 5–6. DCT 2000 Control Character Detect Table

ASCII Character (hexadecimal)	Name	CD Table Setting (2) (hexadecimal)	Meaning of Setting
01	SOH	12	See CI word 2
03	ETX	13	See CI word 3
04	EOT	17	See CI word 7
06	ACK	14	See CI word 4
07	BEL	14	See CI word 4
11	DC1	14	See CI word 4
15	NAK	14	See CI word 4
16	SYN	10	See CI word 0
19	EM	03	Suppress character (input)
83 (1)	ETX	13	See CI word 3
86 (1)	ACK	14	See CI word 4
91 (1)	DC1	14	See CI word 4

NOTES:

(1) These characters occur during input only (before parity is stripped from the character).

(2) All other CD table settings will be zero, implying data characters.

Table 5-7. DCT 2000 Control Character Interpretation Table

CI Word	CI Table Setting* (hexadecimal)	Meaning of Setting
0	0801	SYN character (hardwired function), suppress character, suppress BCC
1	–	Not used
2	1101	Start character plus start BCC plus suppress BCC
3	4800	Suppress character plus end character
4	4100	Start character plus end character
5	–	Not used
6	–	Not used
7	4100	Start character plus end character
8 through 15	–	Not used

*All other CI table settings are zero.

Table 5-8. DCT 2000 Port Control Word

Byte	Bit	Value (binary)	Meaning
1	0, 1	10	Four SYN characters on output
2	2, 3	10	Seven-bit character length
3	3	1	Include start character in LRC
	4–7	1001	Odd VRC plus even LRC
4	4–6	001	Synchronous line

5.4. 1004/9200/9300 REMOTE DEVICE HANDLER

Tables 5–9 through 5–11 list initialization parameters provided in the 1004/9200/9300 remote device handler for the control character tables and port control word. The same initialization parameters are used for the 1004 slave mode handler.

Table 5–9. 1004/9200/9300 Control Character Detect Table – DLT-1 and DLT-3 Boards

ASCII Character (1) (hexadecimal)	Name	CD Table Setting (2) (hexadecimal)	Meaning of Setting
35	SYN	10	SYN character, see CI word 0
55	EOM	13	EOM on input, see CI word 3
95	EOM	13	EOM on output, see CI word 3

NOTES:

(1) The SOM on input is 40_{16}, which is considered to be a data character because of its odd parity. The port control word is set to start input and start BCC on the first nonsynchronous input character.

(2) All other CD table settings will be zero, implying data character.

Table 5–10. 1004/9200/9300 Control Character Interpretation Table

CI Word	Setting (hexadecimal)	Meaning of Setting
0	0800	SYN character, suppress data character on input
1	–	Not used
2	–	Not used
3	4801	Suppress BCC, end character, suppress character on input
4 through 15	–	Not used

Table 5-11. 1004/9200/9300 Port Control Word

Byte	Bit	Value (binary)	Meaning
1	0, 1	11	Four SYN characters on output, start input on first nonsync
2	2, 3	01	Six-bit character length
3	3 4-7	1 1000	Start character included in LRC Parity option 8: Data = odd parity Control characters = even parity LRC = even LRC parity = even (not checked on input)
4	0-3 6 7	- 1 -	Line procedure timer value as required (0's if not used) Synchronous Line procedure timer range as required

5.5. UNISCOPE 100/UNISCOPE 200/DCT 1000/UTS 400/UTS 4000 REMOTE DEVICE HANDLER

Tables 5-12 through 5-14 list the initialization parameters provided in the UNISCOPE 100/UNISCOPE 200/DCT 1000/UTS 400/UTS 4000 remote device handler for the control character tables and port control word.

Table 5-12. UNISCOPE 100/UNISCOPE 200/DCT 1000/UTS 400/UTS 4000 Control Character Detect Table (Part 1 of 2)

ASCII Character (hexadecimal)	Name	CD Table Setting* (hexadecimal)	Meaning of Setting
01	SOH	12	See CI word 2
02	STX	13	See CI word 3
03	ETX	15	See CI word 5
04	EOT	16	See CI word 6
05	ENQ	OA	End input data (DLE sequence)
10	DLE	11	See CI word 1
15	NAK	OA	End input data (DLE sequence)
16	SYN	10	See CI word 0
17	ETB	14	See CI word 4

* All other CD table settings are zero, implying data characters.

Table 5-12. UNISCOPE 100/UNISCOPE 200/DCT 1000/UTS 400/UTS 4000 Control Character Detect Table (Part 2 of 2)

ASCII Character (hexadecimal)	Name	CD Table Setting* (hexadecimal)	Meaning of Setting
30	0	OA	End input data (DLE sequence)
31	1	OA	End input data (DLE sequence)
3F	?	OA	End input data (DLE sequence)
81	SOH	12	See CI word 2
82	STX	13	See CI word 3
83	ETX	15	See CI word 5
84	EOT	16	See CI word 6
85	ENQ	OA	End input data (DLE sequence)
90	DLE	11	See CI word 1
95	NAK	OA	End input data (DLE sequence)
96	SYN	10	See CI word 0
97	ETB	14	See CI word 4
B0	0	OA	End input data (DLE sequence)
B1	1	OA	End input data (DLE sequence)
BF	?	OA	End input data (DLE sequence)

* All other CD table settings are zero, implying data characters.

Table 5-13. UNISCOPE 100/UNISCOPE 200/DCT 1000/UTS 400/UTS 4000 Control Character Interpretation Table

CI Word	Setting (hexadecimal)	Meaning of Setting
0	0801	Suppress character, suppress BCC accumulation for this character
1	0100	Start input data
2	1101	Start input data, start BCC, and suppress BCC accumulation for this character
3	1100	Start input data and start BCC
4	4800	Suppress character and end character
5	4800	Suppress character and end character
6	1100	Start input data and start BCC
7	–	Not used
8 through 15	–	Not used

Table 5-14. UNISCOPE 100/UNISCOPE 200/DCT 1000/UTS 400/UTS 4000 Port Control Word

Byte	Bits	Value (binary)	Meaning
1	0, 1	10	Specific start character on input, four SYN characters on output
		01	One-unit interval stop element (asynchronous)
2	2, 3	10	Seven-bit character length (both synchronous and asynchronous)
	4-7	0110 1001 1010 1011	300 bps* 1200 bps* 1800 bps* 2400 bps*
3	3	1	Start character included in LRC
	4-7	1001	ANSI standard (synchronous) parity function
		0101	ANSI standard (asynchronous) parity function
4	4-6	0X1 0X0	Synchronous Asynchronous

*Asynchronous line speed

5.6. BSC REMOTE DEVICE HANDLER

Tables 5-15 through 5-23 list the initialization parameters provided in the BSC remote device handler for the control character tables and port control word.

Table 5-15. BSC ASCII Nontransparent Control Character Detect Table (Part 1 of 2)

ASCII Character (hexadecimal)	Name	CD Table Setting* (hexadecimal)	Meaning of Setting
01	SOH	13	See CI word 3
02	STX	13	See CI word 3
03	ETX	15	See CI word 5
83	ETX	15	See CI word 5
04	EOT	17	See CI word 7
05	ENQ	12	See CI word 2
85	ENQ	12	See CI word 2
10	DLE	11	See CI word 1

*All other CD table entries will be zero, implying data characters.

Table 5–15. BSC ASCII Nontransparent Control Character Detect Table (Part 2 of 2)

ASCII Character (hexadecimal)	Name	CD Table Setting* (hexadecimal)	Meaning of Setting
15	NAK	04	Start and end
16	SYN	10	See CI word 0
17	ETB	14	See CI word 4
19	EM	03	Suppress character
97	ETB	14	See CI word 4
1F	ITB	16	See CI word 6
30	0	0A	End input (DLE sequence) (ACK 0)
B0	0	0A	End input (DLE sequence) (ACK 0)
31	1	0A	End input (DLE sequence) (ACK 1)
3B	;	0A	End input (DLE sequence) (WACK)
3C	<	0A	End input (DLE sequence) (RVI)

*All other CD table entries will be zero, implying data characters.

Table 5–16. BSC ASCII Nontransparent Control Character Interpretation Table

CI Word	Setting (hexadecimal)	Meaning of Setting
0	0801	SYN character, suppress on input, no BCC
1	0100	DLE, start input
2	4100	Start character plus end character
3	1100	SOH, STX, start input, start BCC
4	4000	ETB, end character
5	4000	ETX, end character
6	8000	ITB, intermediate end character
7	4100	EOT, start input, end character
8 through 15	–	Not used

Table 5-17. BSC ASCII Nontransparent Port Control Word

Byte	Bit	Value (binary)	Meaning
1	0, 1	10	Specific start character on input, four SYNs on output
2	2, 3	10	Seven-bit character length
3	3	0	Start character omitted from LRC
	4-7	1011	BSC ASCII parity option
4	4-6	0X1	Synchronous

Table 5-18. BSC EBCDIC Control Character Detect Table

EBCDIC Character (hexadecimal)	Name	CD Table Setting* (hexadecimal)	Meaning of Setting
01	SOH	13	See CI word 3
02	STX	13	See CI word 3
03	ETX	15	See CI word 5
10	DLE	11	See CI word 1
19	EM	03	Suppress character
1F	ITB	16	See CI word 6
26	ETB	14	See CI word 4
2D	ENQ	12	See CI word 2
32	SYN	10	See CI word 0
37	EOT	17	See CI word 7
3D	NAK	04	Start at end character
6B	,	0A	End input (DLE sequence) (WACK)
70	None	0A	End input (DLE sequence) (ACK 0)
71	None	0A	End input (DLE sequence) (ACK 1)
7C	@	0A	End input (DLE sequence) (RVI)

*All other CD table settings will be zero, implying data characters.

Table 5–19. BSC EBCDIC Control Character Interpretation Table

CI Word	Setting (hexadecimal)	Meaning of Setting
0	0801	SYN character, suppress on input, no BCC
1	0100	DLE, start input
2	4100	ENQ, start input, end character
3	1102	SOH, STX, start input, start BCC, set transparent mode
4	4004	ETB, end character, clear transparent mode
5	4004	ETX, end character, clear transparent mode
6	8000	ITB, intermediate end character
7	4100	EOT, start input, end character
8 through 15	–	Not used

Table 5–20. BSC EBCDIC Port Control Word

Byte	Bits	Value (binary)	Meaning
1	0, 1	10	Specific start character on input, four SYNs on output
2	2, 3	11	Eight-bit character length
3	3 4–7	0 1101	Start character omitted from CRC BSC CRC-16
4	4–6	001	Synchronous

Table 5-21. BSC Transcode Control Character Detect Table

Transcode Character (hexadecimal)	Name	CD Table Setting* (hexadecimal)	Meaning of Setting
00	SOH	13	See CI word 3
0A	STX	13	See CI word 3
0F	ETB	14	See CI word 4
1D	ITB	16	See CI word 6
1E	EOT	17	See CI word 7
1F	DLE	11	See CI word 1
20	–	0A	End input (DLE sequence) (ACK 0)
23	T	0A	End input (DLE sequence) (ACK 1)
2D	ENQ	12	See CI word 2
2E	ETX	15	See CI word 5
32	2	0A	End input (DLE sequence) (RVI)
3A	SYN	10	See CI word 0
3D	NAK	04	Start and end character
3E	EM	03	Suppress character

*All other CD table settings will be zero, implying data characters.

Table 5-22. BSC Transcode Control Character Interpretation Table

CI Word	Setting (hexadecimal)	Meaning of Setting
0	0801	SYN character, suppress on input, no BCC
1	0100	DLE, start input
2	4100	ENQ, start input, end character
3	1100	SOH, STX, start input, start BCC
4	4000	ETB, end character
5	4000	ETX, end character
6	8000	ETB, intermediate end character
7	4100	EOT, start input, end character
8 through 15	–	Not used

Table 5-23. BSC Transcode Port Control Word

Byte	Bit	Value (binary)	Meaning
1	0, 1	10	Specific start character on input, four SYNs on output
2	2, 3	01	Six-bit character length
3	3 4–7	0 1101	Start character omitted from LRC BSC CRC-12
4	4–6	001	Synchronous

5.7. AUTOMATIC DIALING

Control tables are not used for the autodial adapter. The port control word for the autodial adapter operation must have bit 6 of control byte 4 equal to 1. All other bits in the four control bytes must be set to 0.

When a DIAL command (05) is sent to the autodial adapter, the autodial adapter will check that the DATA LINE OCCUPIED signal is off and the POWER INDICATION signal is on prior to turning on the CALL REQUEST signal. If these conditions are not met, the CALL REQUEST signal is not turned on. In addition, if the DATA LINE OCCUPIED signal is on and DATA SET STATUS is off, the autodial adapter will return UNIT CHECK status and set the ABANDON CALL AND RETRY sense bit. If the DATA LINE OCCUPIED signal and DATA SET STATUS signal are both on when the DIAL command is received, the SLCA does not return UNIT CHECK status; however, this situation is avoided with proper software design (software should not issue a DIAL command when the associated modem is in the data mode). UNIT CHECK and ABANDON CALL AND RETRY will also be generated if the POWER INDICATION signal from the ACU is off at the time the DIAL command is received by the autodial adapter.

To avoid a potential race condition when aborting a call attempt (the CALL REQUEST signal is turned off before the DATA SET STATUS signal is turned on), the DATA TERMINAL READY signal to the associated modem must also be turned off when the CALL REQUEST signal is turned off.

Dialing information from the System 80 buffer is presented to the autodial adapter by the SLCA one byte at a time. No CD tables are referenced for these bytes.

The four most significant bits of the byte are stripped by the CMM, and the four least significant bits are interpreted by the autodial adapter as shown in Table 5-24.

It should be noted that CCITT V.25 automatic calling equipment may require an end-of-number (EON) code following the dial digits. The EON option normally is not used with the Bell System 801 ACU.

The format of the dial buffer should be as follows:

First digit

Second digit

.

.

.

Last digit

EON (if required by the ACU)

End of dial (EOD)

Dummy character (requested by the SLCA but does not go to the autodial adapter)

Dummy character (to prevent the byte count from going to zero)

The last dummy character is required for the autodial adapter port to remain active so the ABANDON CALL AND RETRY signal or DATA SET STATUS signal from the automatic calling unit will be reported by the SLCA. When either of these status conditions is reported, software should issue an LA CLEAR command to reset the CALL REQUEST signal.

Table 5-24. Dialing Information

Bit 4 5 6 7	Autodial Adapter Interpretation
0 0 0 0	Digit 0
0 0 0 1	Digit 1
0 0 1 0	Digit 2
0 0 1 1	Digit 3
0 1 0 0	Digit 4
0 1 0 1	Digit 5
0 1 1 0	Digit 6
0 1 1 1	Digit 7
1 0 0 0	Digit 8
1 0 0 1	Digit 9
1 0 1 0	Do not use
1 0 1 1	End of Dial (EOD)
1 1 0 0	End of Number (EON)
1 1 0 1	Do not use
1 1 1 0	Do not use
1 1 1 1	Delay*

*If sent to the autodial adapter, the delay digit will be discarded but will result in a pause of approximately 1.0 ÷ 0.6 second in the dialing sequence. A number of delay digits may be loaded in a dial buffer to effect time delays (for example, to wait for the second dial tone if dialing over a tieline).

Appendix A. Control Tables

A.1. COMMUNICATIONS CONTROL PACKET

This control packet is an activity request packet for the physical inteface. You must construct a packet of the following structure in your user program area and enter the specific parameters you require before execution of each CCRCALL macroinstruction.

Figure A-1 shows a functional field description and summary of the related DSECT symbolic labels. A detailed description of the control packet and usage is provided in Table A-1 and in the following text. The control packet is described in terms of the TN#PARP dummy control section, which may be obtained from the OS/3 system library by:

```
    TN#DSECT CPIOCP
```

Each label of this DSECT begins with the prefix TN#P; therefore, in the table, only the label suffixes are shown.

Byte	0	1	2	3	Word
0	I R L · activity control queue indexes (TN#PRTY)	▨ activity control chain address or CPIOCS work field (TN#PRESV)			1
4	completion address (TN#PFCPL)				2
8	user flag field (TN#PTNDX)	▨ activity control fields (TN#PLOCA)			3
12	▨ logical primary status (TN#PRIM)	▨ logical detailed status (TN#PDETL)	▨ hardware device status (TN#PSTAT)	▨ hardware channel status (TN#PSTAT+1)	4
16	▨ hardware sense byte 0 (TN#PSEN1)	▨ hardware sense byte 1 (TN#PSEN2)	▨ output residual byte count or input character count (TN#PRBC)		5
20	hardware command code (TN#PCMMD)	buffer address (TN#PBADR)			6
24	time allocation (in seconds) (TN#PTIME)		buffer length (TN#PBLTH)		7
28	logical command function (TN#PFUNC)	CPIOCP chain address (TN#PLINK)			8
32	control flags (TN#PFLGS)	RDH: line control table address user program: line request fields (TN#PLCT)			9
36	channel number (TN#PCHNL)	SLCA number (TN#PPORT)	port number (TN#PS80P)	reserved (TN#PS80F)	10
40	reserved for ICAM (TN#PBFST)				11
44	reserved for ICAM (TN#PBCUR)				12
48	▨ sense bytes 0–3 (TN#PS80S)				13
52	▨ sense bytes 4–7 (TN#PS80S) (cont)				14

LEGEND:

▨ System-supplied parameters

Figure A–1. Communications Physical Input/Output Control Packet (CPIOCP) Functional Field Description

Table A-1. Control Packet Detailed Field Description (Part 1 of 10)

Word	Field				Content	Comment	Byte
	Word/Bit Label Suffix	Type and Length	Set by				
			User	CCR			
1	RTY	XL1	X	X	IRL and activity control fields Bit 0 = IRL Bit 1-7 = activity control fields	The IRL bit is set only by you. When set, you get an immediate return in-line after CPIOCS has acted upon the CCRCALL. If the CCRCALL references chained CPIOCPs, the IRL only has significance in the first CPIOCP. While CPIOCS is executing a CPIOCP or has CPIOCPs on its line queues, it uses activity control's chain field for work fields: Byte 0 = bits 0-1 of TN#PFUNC bits 2-3 used to control buffers greater than 1024 Byte 1 = bits 2-7 of TN#PFUNC. Byte 2 = bits 0-6 of TN#PMUX	0 1
	RESV	XL3	X	X	Activity control chain field and CPIOCS work fields	CPIOCS sets TN#PRESV to 0 before returning CPIOCPs to you. You must set TN#PRESV to 0 in all chained CPIOCPs before executing CCRCALL.	
2	FCPL	XL4	X		Completion address	When chained CPIOCPs are scheduled back to you, only the completion address of the first CPIOCP in the chain is used. If you set byte 1 to EE$_{16}$, you will not be scheduled control when this CPIOCP function is complete. The CPIOCPs will be unchained normally and then dropped from all CCR queues. This facility is NOT recommended for normal user operations.	4
3	TNDX TRTS LIDL LOCA	XL1 AL3	X		User flag LOCAP address	 Clear REQUEST TO SEND signal No line turnaround	8 9
4	RIM ITST	XL1		X	Primary logical status Idle time status - line information not especially related to message completion.	This field will be set to 0 by CPIOCS when CPIOCP is first received.	12

Table A-1. Control Packet Detailed Field Description (Part 2 of 10)

Word	Field				Content	Comment	Byte
	Word/Bit Label Suffix	Type and Length	Set by				
			User	CCR			
4 (cont)	HDWR				Hardware error - do not retry.		
	ROGE				Program error		
	LLE				Link level exception - line error but retry is indicated.		
	ERND				Erroneous function/message completion.		
	END				Function/message completion tion		
	DETL	XL1		X	Detailed logical status	This field will be set to 0 by CPIOCS when CPIOCP is first received.	13
	For ITST						
	RING				Ring interrupt		
	BRK				Break		
	For HDWR						
	CREJ				Command reject		
	BUSO				Bus-out check		
	CHS1				Auto sense error		
	CHS2				Incorrect length		
	CHS3				Program check		
	CHS4				Protection check		
	CHS5				Data check		
	CHS6				Control check		
	CHS7				Interface control check		
	CHS8				Chaining check		
	DSRF				Data set ready off		
	OPNL				Open line		
	DCON				Disconnect		
	NOOP				Device (SLCA not operational, unit exception)		
	For ROGE						
	FRMT				CPIOCP format or procedure error	Refer to Table A-3. When this error is reported by user program interface code, even more detailed status is reported in TN#PSEN2.	
	AATH				Attach error		
	DTCH				Detach error		
	LPHE				Load program phase error		
	LRLE				User program line request line error	Requested line is already allocated.	
	MNCE				Maximum number of user program task identities exceeded	Refer to 3.6. ICAM cannot process another user program task.	

Table A-1. Control Packet Detailed Field Description (Part 3 of 10)

Word	Field				Content	Comment	Byte
	Word/Bit Label Suffix	Type and Length	Set by				
			User	CCR			
4 (cont)	For LLE						
	ORUN				Input overrun		
	URNO				Space to mark		
	ASS				Output sequence abort		
	DCHK				Data Check: LRC/CRC or character parity error		
	IORN				Idle overrun		
	ABRT				Abort		
	TIMD				CPIOCP time-out		
	TIMP				SLCA time-out		
	CARF				Carrier off		
	GSSE				General subsystem error		
	For ERND						
	IMPR				Status condition improper	An example is no hardware sense bits returned for a SENSE command.	
	For END						
	EOM				Successful function/message completion		
	BCI				Buffer completion notification	The interpretation of the value (bits) in this field is dependent on the value of TN#PRIM.	
	GIVB				Return of CPIOCPs due to subsequent CPIOCP requesting immediate turn-off function		
	STAT	XL2			Hardware status Byte 0 = device status Byte 1 = channel status	See Table A-2.	
5	SENS	XL2			Sense bytes		
	SEN1			X	Hardware sense byte 2	These fields will be set to 0 by CPIOCS when CPIOCP is received.	16
	SEN2			X	Hardware sense byte 3	When TN#PRIM = TN#PROGE, and TN#PDETL = TN#PFRMT, then TN#PSEN2 will be set to the values in Table A-3 when an error has been detected by the user program interface code. If device errors occur, sense bytes 2 and 3 are reported here. See Table A-4.	17
	or						
	CFEC			X	CCRU CPIOCP format error code		17

Table A-1. Control Packet Detailed Field Description (Part 4 of 10)

Word	Field				Content	Comment	Byte
	Word/Bit Label F Suffix	Type and Length	Set by				
			User	CCR			
5 (cont)	RBC	XL2		X	Residual byte count for output, or input character count for input	This field will be set to 0 by CPIOCS when CPIOCP is first received. When the input character count is reported in more than one CPIOCP (buffer chaining) for a particular message, each CPIOCP contains the total accumulated count thus far. This field is not for immediate commands (commands that do not result in device or channel status).	18
6	CMMD	XL1	X	X	Hardware command code	NOTE: When this field is *not* set to 0 by you, CPIOCS will *not* translate the logical command function in TN#PFUNC (bits 2-7) to obtain proper hardware command code. Whatever value was last here or whatever value you set will be issued to the SLCA.	20
	BADR	XL3	X	X	Line buffer address		21
7	TIME	XL2	X	X	Time allocation in seconds	The CPIOCP at the head of any CPIOCS line queue, when this field is not 0, is decremented by 1 each time CCRs timer services is notified that 1 second has elapsed. When this field decrements to 0, a HALT DEV command is issued, and the CPIOCPs are scheduled back to you with TN#PRIM = TN#PLLE and TN#PDETL = TN#PTIMD. If, however, the logical command function (TN#PFUNC, bits 2-7) was set to TN#PIDLT, a HALT DEV command is issued, but the CPIOCP is scheduled back to you with TN#PRIM = TN#PEND and TN#PDETL = TN#PEOM. The accuracy of this field can be off by minus 1 second, depending on when CCRs timer service is next notified that 1 second has elapsed.	24

Table A-1. Control Packet Detailed Field Description (Part 5 of 10)

Word	Field				Content	Comment	Byte
	Word/Bit Label | Suffix|	Type and Length	Set by				
			User	CCR			
7 (cont)						NOTE: Because this field is decremented by CCRs timer services, you must be careful to reset this field before reusing this CPIOCP for another CCRCALL.	
	BLTH	XL2	X	X	Line buffer length	Both TN#PTIME and TN#PBLTH are used by the user program for Line Request.	26
8	FUNC	XL1	X		Logical command function plus start and end of message/ function flags		28
	FS				Start of message	When TN#PFS is set to 1, it indicates that this CPIOCS is the first CPIOCP for a particular message or function. When it is set to 0, it means that chained CPIOCPs are being used to output or input a message and this particular CPIOCP is not the first one. Refer to 3.1 for information regarding the setting of this bit for message chaining.	
	FL				End of message	When TN#PFL is set to 1, it indicates that this CPIOCP is the last CPIOCP for a particular message or function. The setting of this bit for the last CPIOCP of an output message is particularly critical. When TN#PFL is set to 0, it means CPIOCP is being used in buffer chaining and that it is not the last CPIOCP of the chain. When you know that a particular CPIOCP is the only one used to perform a function, TN#PFS and TN#PFL should both be set (TN#PFSL).	
	FSL				Start and end of message/function Bits 2-7 specify the logical command function	Refer to Table A-5 for a cross-referencing of the logical command function, the hardware command code, and the hardware command code as it is represented in TN#PCMMD.	

Table A-1. Control Packet Detailed Field Description (Part 6 of 10)

Word	Field				Content	Comment	Byte
	Word/Bit Label Suffix	Type and Length	Set by				
			User	CCR			
8 (cont)	WAIT				Wait for interrupt	No SIO instruction is issued. A CPIOCP for this function is issued only to receive notification of an interrupt when no message processing is in progress, such as the RING interrupt. If an interrupt other than unit check is received for a line with no CPIOCP active on the CPIOCS line queue, the interrupt will be ignored. Refer to 3.5 for automatic issuing of a SENSE command in response to unit check.	
	IOFF				Immediate port turn off	Return all CPIOCPs currently on the CPIOCS line queue, marking the head CPIOCP with a status of TN#PRIM = TN#PEND and TN#PDETL = TN#PGIVB. Then, perform a HALT DEV command.	
	SEND				Enable data output for output/input mode of operation	This output function should be used instead of TN#PRTS for all output messages unless you know that subsequent output messages will follow with no intervening input. This function enables the SLCA fast turnaround logic which drops the request-to-send load and prepares for input.	
	DIAL SPAC				Dial Send space (asynchronous only)	If either a CPIOCP or an SLCA time-out occurs for this function, the status returned is TN#PRIM = TN#PEND and TN#PDETL = TN#PEOM.	
	SMRK				Send mark (asynchronous only)	This command disables interrupts and must be used with caution.	
	SIDL				Send idle		
	EDI				Enable data input		
	ON				Same as TN#PEDI		
	NSYN				New sync		
	LSYN				Immediate look for sync	No data transfer is processed.	
	OFF				Port turn off	No immediate status	

Table A-1. Control Packet Detailed Field Description (Part 7 of 10)

Word	Field				Content	Comment	Byte
	Word/Bit Label - Suffix	Type and Length	Set by				
			User	CCR			
8 (cont)	DISC				Disconnect		
	SBSY				Set busy		
	LACL				LA clear		
	DSRM				Enable data set ready monitor		
	FLDX				Set full duplex		
	TEST				LA test		
	MODT				Modem test		
	LATO				Line adapter turn off		
	RTS				Enable data output for output/ output sequence of message	This output function does not clear request-to-send as does TN#PSEND.	
	IDLT				Send idle and wait for CPIOCP time-out - not an immediate command.	This send idle function does not result in an immediate command. Status is not returned to you until the hardware interrupts or until a CPIOCP time-out occurs. If a CPIOCP or an SLCA time-out occurs for this function, the status returned is TN#PRIM = TN#PEND and TN#PDETL = TN#PEOM.	
	LB14				Load port control word (bytes 1-4)		
	LB24				Load control bytes 2, 3, 4		
	LB34				Load control bytes 3, 4		
	LB4				Load control byte 4		
	LCD1				Load control character detect table 1		
	LCD2				Load control character detect table 2		
	LCI1				Load control interpretation table 1		
	RPCW				Read port control word		
	RCD1				Read control character detect table 1		
	RCD2				Read control character detect table 2		
	RCI1				Read control interpretation table 1		
	BNOP				NO-OP		
	BRRM				read random access memory		
	BLMA				Load memory address		
	BLRM				Load random access memory		

Table A–1. Control Packet Detailed Field Description (Part 8 of 10)

Word	Field				Content	Comment	Byte
	Word/Bit Label Suffix	Type and Length	Set by				
			User	CCR			
8 (cont).	BDIA				Diagnostic modem test		
	LREL				User program line release	Refer to 2.6	
	LREQ				User program line request	Refer to 2.3.	
	RLLT				User program read line link table	Refer to 3.3.	
	NREQ				User program network request	Refer to 2.2.	
	NREL				User program network release	Refer to 2.7.	
	LINK	XL3	X	X	CPIOCP chain address field	Refer to 3.1 for a complete description of chaining and unchaining CPIOCPs to support data buffer chaining and message chaining. NOTE: Do not chain a CPIOCP to itself either directly or later in the chain.	29
9	FLGS	XL1	X	X	Control flags		32
	CIX		X		- Suppress buffer completion interrupt scheduling	Whenever a buffer completion interrupt occurs, the CPIOCP for that buffer will be scheduled back to you unless this flag is set. When this flag is set, the scheduling will be delayed. Refer to 3.1 for a statement about scheduling suppressed CPIOCPs.	
	ESO		X		Error schedule only	When this flag is set, the scheduling of the CPIOCP back to you will be delayed if it has a successful completion status (TN#PRIM = TN#PEND and TN#PDETL = TN#PEOM or TN#PBCI). NOTE: The last CPIOCP in a chain must not have this flag set. Refer to 3.1 for an explanation of how suppressed CPIOCPs are scheduled back to you.	

Table A-1. Control Packet Detailed Field Description (Part 9 of 10)

Word	Field				Content	Comment	Byte
	Word/Bit Label Suffix	Type and Length	Set by				
			User	CCR			
9 (cont)	TRNP				BSC is now operating in transparent mode.		
	CC			X	Condition code	The SIO condition code for the last command issued for this CPIOCP is saved here. 00_2 = successful 01_2 = unsuccessful, status pending 10_2 = busy 11_2 = illegal port (e.g., 10_{16} or 13_{16}) or SLCA not operational.	
	LCT	XL3	X X X	X X	Multipurpose field: RDH line control table address; or User program line request fields	The RDH line control table is where RDH saves line information. Refer to 2.3 for information about the user program line request fields.	33
	RQFG RFDQ RFUL	XL1	X		Line request flag field TN#PRQFG = TN#PLCT - Request full-duplex queueing - Request full-duplex line		33
	DSPL UDID	XL1	X	X	CA tables discipline ID TN#PDSPL = TN#PLCT + 1 - User discipline ID		34
	CAID	XL1	X	X	CA tables ID TN#PCAID = TN#PLCT + 2		35
	LCTF				Full word address	TN#PLCTF=TN#PFLGS+TN#PLCT	
10	MUX						36
	CHNL	XL1		X	Channel number		
	PORT	XL1		X	SLCA number		
	S80P	XL1		X	Port number		
	S80F	XL1		X	X-21 control flag		
	MOTO				X.21 map port 0 to 0		
	ICIO				X.21 implicit chain IORBs		

Table A-1. Control Packet Detailed Field Description (Part 10 of 10)

Word	Field				Content	Comment	Byte
	Word/Bit Label Suffix	Type and Length	Set by				
			User	CCR			
11	BFST	XL4		X		Reserved for ICAM	40
12	BCUR	XL4		X		Reserved for ICAM	44
13	S80S	XL4		X	Sense bytes (0-3)	Reserved for ICAM	48
14		XL4		X	Sense bytes (4-7)	Reserved for ICAM	52

The following paragraphs describe the control packet on a word and byte basis.

- Word 1 – Priority and Activity Control (TN#PRTY and TN#PRESV)

  Byte 0 is used to designate an immediate return line. An IRL transfers control inline immediately following the CCRCALL macro; however, control will also be passed to your completion address as specified in word 2. You set this byte to regain control while ICAM is performing input/output functions. Your program is disconnected from ICAM at this point until you issue a non-IRL macro or a CYIELD instruction. (See example 1.) Bytes 1 through 3 are restricted to activity control usage and are unusable by your program.

  Example 1:

```
LBLPACKT DC    XL4'80000000'            IRL
LBLPACKT DC    F'0'                     No IRL
```

- Word 2 – User Completion Address (TN#PFCPL)

  The user completion address, the label of an entry point in your program where status notification is to be scheduled, must be written as a full-word address constant as shown in example 2:

  Example 2:

```
*
         DC    A(COMPADDR)              Entry point for status notification
*
```

- Word 3 – User Flag Field and Activity Control Address

  Byte 8 – User flag field (TN#PTNDX)

  TN#PTRTS

  Clear REQUEST TO SEND signal in output/output messages sequences.

  TN#PLIDL

  No line turnaround after input completion.

  Byte 9 – LOCAP address (TN#PLOCA)

  Used by activity control.

- Word 4 – Status Word (TN#PRIM, TN#PDETL, and TN#PSTAT)

  Word 4 is initialized to zero by CPIOCS, since it is dynamically filled whenever user status notification is scheduled. Bytes 0 and 1 are not hardware-generated codes but software-generated logical codes to facilitate some degree of hardware independence. Bytes 2 and 3 are filled with the actual hardware-generated codes. Byte 2 is reserved for device status and byte 3 for channel status. See Table A–2.

Table A–2. Standard Processor Hardware Status Byte Settings

Status	Bit	Hexadecimal Value	Status	Bit	Hexadecimal Value
Device status			Channel status		
Attention	0	80	Unassigned	0	80
Status modifier	1	40	Incorrect length	1	40
Control unit end	2	20	Program check	2	20
Busy	3	10	Invalid address	3	10
Channel end	4	08	Channel data check	4	08
Device end	5	04	Interface control check	5	04
Unit check	6	02	Channel control check	6	02
Unit exception	7	01	Buffer terminate	7	01

Remote device handlers primarily use the software-generated status half word, while diagnostic routines supplied by Sperry use the hardware status half word.

Table A-3. Control Packet Format Error Code Specifications

Hexadecimal Value in TN#PSEN2*	Interpretation of Error Code
00	Error condition not detected by user program interface code**
EC	No entry in SLCA load table to release
ED	Received user RDH discipline ID instead of requested system RDH discipline ID
EE	Invalid channel and/or SLCA number
EF	Illegal channel value
F0	Buffer length field greater than 1024†
F1	TN#PLINK not zero as required for this function
F2	Illegal buffer address for read LLT or read SLCA tables function
F3	CPIOCS line queue not empty before line release function
F4	Illegal set of SLCA tables specified for line request
F5	CPIOCP chained to itself or TN#PRESV + 1 not zero when CPIOCP issued
F6	TN#PLCT has illegal address.
F7	Illegal logical command function
F8	Buffer address plus length exceeds program boundary.
F9	Load SLCA control tables function for tables not assigned to you
FA	SLCA number above highest SLCA specified by system generation
FB	SLCA number below first SLCA specified by system generation
FC	Not used
FD	SLCA number not assigned to you
FE	Line request for secondary channel port before that of primary channel
FF	Line release for primary channel before secondary channel

* When TN#PRIM = TN#PROGE and TN#PDETL = TN#PFRMT, then TN#PSEN2 should be interpreted according to this table.

** This error condition will occasionally occur if two or more CPIOCPs are toggled (buffer chained) in order to input a message of unknown length. For example, when CPIOCP No. 1 has been returned to you with TN#PRIM = TN#PEND and TN#PDETL = TN#PBCI, in order for you to process the data (if your program is interrupted by a message end condition for CPIOCP No. 2) you will reissue CPIOCP No. 1, not yet knowing the message is over. CPIOCP No. 1 will be scheduled back with this status because TN#PFUNC will not have had bit 0 (TN#PFS) set.

- Word 5 – Sense Bytes and Output Residual Byte Count/Input Character Count (TN#PSENS and TN#PRBC)

  Word 5 is divided into two half-word fields. If user interface completion is not successful, the first field contains a control packet format error code. If completion is successful, see Table A–4.

  Table A–4 defines the mapping of the functional sense bytes to the primary and detailed status fields in word 4.

Table A–4. Sense Bytes Settings for CPIOCP Word 5

Sense Byte	Bit	TN#PRIM ARP Status Primary	TN#PDETL ARP Status Detail
2	0	TN#PHDWR	TN#PCREJ
	1	TN#PLLE	TN#PIORN
	2	TN#PHDWR	TN#PBUSO
	3	TN#PLLE	TN#PABRT
	4	TN#PLLE	TN#PDCHK
	5	TN#PLLE	TN#PORUN
	6	TN#PITST	TN#PRING
	7	TN#PLLE	TN#PCARF
3	0	TN#PITST	TN#PBRK
	1	TN#PHDWR	TN#PDCON
	2	TN#PHDWR	TN#POPNL
	3	TN#PLLE	TN#PTIMP
	4	TN#PERND	TN#PIVCB
	5	TN#PLLE	TN#PURNO
	6	TN#PHDWR	TN#PDSRF
	7	TN#PLLE	TN#PASS

TITLE PAGE TEXT

The second half word is used as a residual byte count (RBC) for output and an accumulated character count for input. When control packets are chained, the count is transferred to the various chained packets and reflects the total characters received thus far. If a 3-packet chain of control packets, each specifying buffer length, was used to handle a 54-byte input message, the packets would be marked as follows:

CPIOCP1 0014_{16} (decimal 20)

CPIOCP2 0028_{16} (decimal 40)

CPIOCP3 0036_{16} (decimal 54)

The output RBC is the difference between the length of the buffer and the number of characters transferred over the line. (If a user specifies a 40-byte buffer and the end-of-message character is located at byte 32, an RBC count of 8 results.) This word is initialized to zero by CPIOCS because it is dynamically filled by CPIOCS.

- Word 6 – Hardware Command Code and Buffer Address (TN#PCMMD and TN#PBADR)

Word 6 is divided into two fields: a 1-byte hardware command code field and a 3-byte buffer address field. (See example 3.) The hardware command can be a user-specified or ICAM-generated field. If you clear (zero) this field, CPIOCS looks at the logical function field of word 8 and translates the appropriate hardware command into this field. When the field is not zero, no translation is performed. Hardware independence is furthered by zeroing the field. (This can also effectively cause the second field to become a 4-byte address field.) Refer to Table A–5.

It is strongly recommended that both the remote data handler and the user program always zero the hardware command code before issuing the CCRCALL. In this way, recoding and errors can be eliminated from user programs when ICAM is executed on new or different communications subsystems.

The buffer address field contains the address where the data will reside for sending or receiving. The data must be in the output code expected by the remote terminal. The proper code (ASCII, Baudot, or transparent EBCDIC, must be used, as well as all message control characters. Input messages are delivered in the code sent by the remote terminal, and translation by your program is required.

Example 3:

```
         DC    XL1'01'              Physical command send data
         DC    AL3(OUTBUFF)         Buffer address
* (or)
         DC    A(OUTBUFF)           4-byte address in N leading zero
* (or)
         DC    X'00'                Zero physical command
         DC    AL3(INBUFF)          3-byte input buffer address
```

Table A–5. Cross-Reference of Logical Command Functions and Hardware Command Codes (Part 1 of 3)

TN#PFUNC (Bits 2–7) Hardware DSECT Label	TN#PFUNC (Bits 2–7) Hexadecimal Value	Logical Command Function	Hardware Command Code in TN#PCMMD	Name
TN#PWAIT	3F	Wait for interrupt.	–	–
TN#PIOFF	00	Immediate port turnoff	2E	Clear active.
TN#PDTR	01	Set data terminal ready.	02 2E	Set data terminal ready. Clear active.
TN#PSEND	02	Enable data output (half-duplex line). Enable data output (full-duplex line).	81 81	EDO with fast turnaround clear of request to send EDO without fast turnaround clear of request to send
TN#PDIAL	03	Dial.	05	Dial.

Table A–5. Cross-Reference of Logical Command Functions and Hardware Command Codes (Part 2 of 3)

TN#PFUNC (Bits 2–7) Hardware DSECT Label	TN#PFUNC (Bits 2–7) Hexadecimal Value	Logical Command Function	Hardware Command Code in TN#PCMMD	Name
TN#PSPAC	04	Send space.	11	Send space.
TN#SMRK	05	Send mark.	2D	Send mark.
TN#PSIDL	06	Send idle (immediate command).	29	Send idle.
TN#PEDI	07	Enable data input (half-duplex line).	02	Enable data input.
		Enable data input (full-duplex line).	13	Disconnect.
TN#PNSYN	08	New sync	2A	New sync
TN#PLSYN	09	Look for sync.	26	Look for sync.
TN#POFF	0A	Port turn off.	2E	Clear active.
TN#PDISC	0B	Disconnect.	33	Disconnect.
TN#PSBSY	0C	Set busy.	3F	Set busy.
TN#PLACL	0D	Line adapter clear	2F	Line adapter clear
TN#PDSRM	0E	Enable data set ready monitor.	37	Enable data set ready monitor.
TN#PFLDX	0F	Set full duplex.	3B	Set full duplex.
TN#PTEST	10	Line adapter test	2B	Line adapter test
TN#PMODT	11	Modem test	27	Modem test
TN#PLATO	12	Line adapter turnoff	23	Turnoff
TN#PRTS	13	Enable data output/output sequence of messages.	01	Enable data output with no fast turnaround clear of REQUEST TO SEND.
TN#PIDLT	14	Send idle (not immediate).	09	Send idle.
TN#PLB14	15	Load port control word (bytes 1–4)	15	Load control bytes 1, 2, 3, and 4.
TN#PLB24	16	Load control bytes 2, 3, and 4.		Load control bytes 2, 3, and 4.
TN#PLB34	17	Load control bytes 3 and 4.	95	Load control bytes 3 and 4.
TN#PLB4	18	Load control byte 4.	D5	Load control byte 4.
TN#PLCD1	19	Load control character detect table 1.	19	Load control character detect table 1.
TN#PLCD2	1A	Load control character detect table 2.	59	Load control character detect table 2.
TN#PLCI1	1D	Load control interpretation table 1.	1D	Load control interpretation table 1.

Table A–5. Cross-Reference of Logical Command Functions and Hardware Command Codes (Part 3 of 3)

TN#PFUNC (Bits 2–7) Hardware DSECT Label	TN#PFUNC (Bits 2–7) Hexadecimal Value	Logical Command Function	Hardware Command Code in TN#PCMMD	Name
TN#PRPCW	21	Read port control word.	16	Read port control word.
TN#PRCD1	22	Read control character detect table 1.	1A	Read control character detect table 1.
TN#PRCD2	23	Read control character detect table 2.	5A	Read control character detect table 2.
TN#PRCI1	26	Read control interpretation table 1.	1E	Read control interpretation table 1.
	2A	Undefined	2E	Clear active.
TN#PLREL	2B	User program line release	2F	Line adapter clear
TN#PLREQ	2C	User program line request	–	–
TN#PRLLT	2D	User program read line link table	–	–
TN#PNREQ	2F	User program network request	–	–
TN#PNREL	30	User program network release	2F	Line adapter clear
TN#PBSLC	33	SLCA clear		
TN#PBCLA	34	Clear active.		
TN#PBSEN	35	Sense		
TN#PBNOP	36	No-op		
TN#PBRRM	37	Read memory (RAM).		
TN#PBLMA	38	Load memory address.		
TN#PBLRM	39	Load memory (RAM).		

- Word 7 – Time Allocation and Buffer Length (TN#PTIME and TN#PBLTH)

  Word 7 is composed of a half word of time allocation in seconds and a half word of buffer length, respectively. (See example 4.) The systems timer, in conjunction with CCR timer services, controls the passage of 1-second intervals. Time-outs present a primary status of line link exception. No timing out of the control packet function will be performed if the initial timer value is zero. A recommended minimum time allotment is 2 seconds, based on the accuracy of the timer and rapid interrupt notification.

  The maximum buffer length that CPIOCS supports without restrictions is 1024_{10} (400_{16}). For configurations of CPIOCS which support physical user programs and transient ICAM, CPIOCS will support message buffer lengths of any size up to $65{,}535_{10}$ ($FFFF_{16}$) bytes with one restriction. If a buffer exceeds 1024 bytes, it must be for an entire message. (Buffers greater than 1024 cannot be chained together by the user to transfer a single message.)

  Refer to 3.2 for special use of the time and length fields for user program line request.

  Example 4:

```
WORD7    DC    H'3'                       3-second time allotment
         DC    H'47'                      47-byte buffer
* (or)
         DC    F'47'                      No time, 47-byte buffer
* (or)
         DC    H'0'                       Same as above
         DC    H'47'
```

- Word 8 – Logical Command Function and Chain Address Field (TN#PFUNC and TN#PLINK)

  Word 8 comprises a 1-byte logical command function followed by a 3-byte control packet chain address field. The least significant six bits of the logical command function field are used to obtain the hardware command code only if the hardware command field is zero.

  As shown in example 5, the labels rather than the actual values are used in the logical command function field. The use of the logical operator + (plus) allows the AND function to be performed on labels to effect multiple specifications. The labels are not subject to change, while the value could possibly change; therefore, the DSECT labels should be used as a general convention.

  When chained control packets are required, the 3-byte address constant format should be used as shown in word 6.

Example 5:

```
SOFTFUNC DC    YL1(TN#PSEND + TN#PFSL)   The line labeled SOFTFUNC causes
*                                        a send-data command to be executed;
*                                        it also states by logical operator + TN#PFSL
*                                        that this is the first and last CPIOCP.
*                                        (Thus a complete message-sized
                                         buffer is implied.)
CONTFUNC DC    YL1(TN#PEDI)              The line labeled CONTFUNC implies the
*                                        continuation of an input function.
*
*
*
ENDFUNC  DC    YL1(TN#PSEND + TN#PFL)    The line labeled ENDFUNC says to end
*                                        an output sequence with the depletion
*                                        of the current buffer.
```

- Word 9 – Flags and Special Tasks (TN#PFLGS and TN#PLCT)

Word 9 has a 1-byte operational flag field set by your program and a 3-byte field labeled TN#PLCT, which has four uses.

The flag field has six major processing directives under control of user settings. The labels used (DSECT) are shown as follows, with a description of their usage:

TN#PESO

Suppress notification of successful completion until a packet in a chained sequence is encountered without this bit setting or until an error occurs. The original intent of this setting was to take a complete polling sequence of output and input chained packets and delay the notification of your program until a response to a traffic poll has been received by the computer. This method eliminates notification of the remote device handlers whenever good status is presented. The completion address of the first packet is the place in your program where control is passed. When control is returned to you, the logical status (TN#PRIM and TN#PDETL) of the last control packet in the chain (or the control packet that had the error) overlays that of the head control packet of the chain.

TN#PCIX

Suppress the notification of buffer completion interrupts until a terminating device status is presented (complete message or error).

TN#PTRNP

Informs CPIOCS that binary synchronous communication is now using transparent mode.

The field labeled TN#PLCT is used differently, depending on the user.

- If the user is a remote device handler, it contains the address of the line control table.

- If the user program is performing a line request, this field is used to assign access to the SLCA discipline ID and control character tables (2.3).

■ Word 10 – Channel and Port Numbers (TN#PMUX)

Byte 0 is for the channel number and byte 1 is for the SLCA number. A System 80 Model 4 or 6 supports one to eight SLCAs. A System 80 Model 8 can support up to 14 SLCAs with an attached input/output microprocessor (IOMP) or 28 SLCAs with a dual IOMP (14 full-duplex mode/28 half-duplex mode).

Table A–6 details how the channel and SLCA numbers can be assigned. The SLCA ID range for Model 8 is from 01_{16} to $0F_{16}$, although the maximum assignable is 14 SLCAs.

Table A–6. Channel and SLCA ID Assignments

Model	CHANNEL (Byte 1)	SLCA ID (Byte 2)
3-6	02	08–0F
8 (single IOMP)	0D	01–0F
(dual IOMP)	0D/0F	01–0F

Byte 2 is for the port number. Port 0 is for special control; port 1 is for output; and port 2 is for input. The 2-way alternate (half-duplex) operation, therefore, assumes port 1 is for both output and input if the set-full-duplex command is not set. For full-duplex operation, you specify port 2 for input by issuing the set-full-duplex command.

Example 7 shows a typical coding for an SLCA without an IOMP. Example 8 shows a typical coding for SLCAs with a dual IOMP.

Example 7:

```
SLCA8     MVC TN#PMUX,=X'02080000'     For half-duplex operation
SLCA8     MVC TN#PMUX,=X'02080200'     For full-duplex operation (input)
```

Example 8:

```
                                                IOMP#1
    SLC01      MVC TN#PMUX,=X'0D0102'           Full duplex
    SLC02      MVC TN#PMUX,=X'0D0202'           Full duplex
    SLC03      MVC TN#PMUX,=X'0D0301'           Half duplex
    SLC04      MVC TN#PMUX,=X'0D0401'           Half duplex
    SLC05      MVC TN#PMUX,=X'0D0501'           Half duplex
    ...
    ...
    ...
    SLC14      MVC TN#PMUX,=X'0D0E01'           Half duplex

                                                IOMP#2
    SLC15      MVC TN#PMUX,=X'0F0102'           Full duplex
    SLC16      MVC TN#PMUX,=X'0F0201'           Half duplex
    ...
    ...
    ...
    SLC28      MVC TN#PMUX,=X'0F0E01'           Half duplex
```

- Words 11 and 12 – Reserved
- Words 13 and 14 – Sense bytes (TN#PS80S). This data is system-supplied.

A.2. LINE LINK TABLE

The line link table is a configurable table based on your physical network. It is composed of a 48-byte element for each SLCA defined at SYSGEN time. The first element is assigned to the lowest numbered SLCA defined, and the last element is assigned the highest numbered SLCA. Intervening SLCAs are assigned whether or not they are defined.

Figure A–2 depicts the packets. It usage is illustrated by an English-language field description within the allocated fields. Table A–6 gives a detailed byte description of these fields, relating them to the applicable DSECT byte and bit labels.

DSECT TN#PLINE governs each 48-byte element of the line link table. This DSECT may be obtained from the OS/3 user library by the proc call as shown in the following example:

```
          TN#DSECT LLT
```

Each label has two parts: a prefix and a suffix. Table A–7 contains only the suffix; all labels are prefixed by TN#P. Detailed instructions are provided in 3.3 for the user program to read some portions of the line link table that are useful to you.

Word	0	1	2	3	Byte
1	Thirty-two bytes of information for icam use only. These bytes are not transferred by the read LLT function.				0
2					4
3					8
4					12
5					16
6					20
7					24
8					28
9	flags	task control block address			32
10	device type	companion port ID (primary or secondary channel)	not used	not used	36
11	line type	(SLC ID + 1)	flag and line speed	flags	40
12	port control word				44

LEGEND:

▨ System-supplied parameters

Figure A–2. Line Link Table Functional Field Description

Table A-7. Line Link Table Detailed Field Descriptions (Part 1 of 2)*

Word	Field				Content	Comment	Byte
	Word/Bit Label Suffix**	Type and Length	Set by				
			User	CCR			
9	FXFG	XL1		X	CPIOCS line flags		32
	DSRO				Data set ready off		
	DIAG				Diagnostic trace enabled	Set by interface code for line request	
	FDO				Full-duplex queueing	Set by DUST for RDH	
	FULL				Full-duplex line	Set by interface code (line request) for user program	
	LTCB	XL3		X	Task control block address		33
10	DLID	XL1		X	SLCA ID for dial adapter or its companion data SLCA †	Set by SYSGEN and ICAM initialization	36
	PSID	XL1		X	Companion SLCA ID for this primary or secondary channel SLCA ††	Set by SYSGEN	37
	OFST	XL1		X			39
	IOFF						
	COFF						
11	TYPE	XL1		X	Line type		
	SDL				Single line dial adapter SLCA	Set by SYSGEN	40
	MDL				Multiline dial adapter SLCA		
	PRIM				Primary channel SLCA		
	SEC				Secondary channel SLCA		
	IOST				PIOST SLCA set by SYSGEN		
	DEV	XL1		X	Device type	Set by DUST	41
	FPOL				Polled DCT 500 flag		
	FASC				ASCII DCT 500/TTY Flag		
	A500				Automatic 500 or DCT 524		
	U100				UNISCOPE 100/DCT 1000		
	DCT2				DCT 2000		
	DCT1				9000 computer - 1004 mode		
	DCT9				9000 computer - 929 mode		
	1004				1004		
	9200				9200		
	9300				9300		
	BTMT				BSC terminal, TRANSCODE		
	BCBT				BSC CPU batch, TRANSCODE		
	BCIT				BSC CPU interactive, TRANSCODE		
	BTME				BSC terminal, EBCDIC		
	BCBE				BSC CPU batch, EBCDIC		
	BCIE				BSC CPU interactive, EBCDIC		
	BTMA				BSC terminal, ASCII		
	BCBA				BSC CPU batch, ASCII		
	BCIA				BSC CPU interactive, ASCII		
	TN4E				1004 EMULATION line		
	NTR				Series 1100		

Table A-7. Line Link Table Detailed Field Descriptions (Part 2 of 2)*

Word	Field				Content	Comment	Byte
	Word/Bit Label Suffix**	Type and Length	Set by User	Set by CCR			
11 (cont)	SPED	XL1		X	Flags and line speed	Set by SYSGEN	42
	FDX				Full-duplex line		
	SWT				Switched line		
	ACU				Autodialer associated with this SLCA		
	SYN				Synchronous line		
					Time speed (4 bits)		
	RATE				- Asynchronous value is identical to value specified in port control word		
					- Synchronous value:		
					B'0000' = 2000 bps		
					B'0001' = 2400 bps		
					B'0010' = 4800 bps		
	9600				B'0011' = 9600 bps		
	LHSP				B'0100' = over 9600 bps		
	FLGT	XL1		X	Flags		43
	DOWN				Line down		
	ALLO				Line allocated	Set by DUST	
	APND				Line allocation pending		
	EON				End-of-number digit required for use with this autodialer	Set by SYSGEN	
	NEP				No physical line		
12	PCW	XL4		X	Port control word	Set by DUST during line request processing	44

* The DSECT label of this table is TN#PLINE.

** All labels have the prefix TN#P.

† TN#PTYPE, subfield TN#PSDL or TN#PMDL indicates SLCA is for autodial adapter.

†† TN#PSPED, subfield TN#PACU indicates SLCA is associated with an autodialer. TN#PTYPE, subfields TN#PPRIM and TN#PSEC indicate whether this port is a primary or secondary channel.

Index