Honeywell

LEVEL 6

GCOS

COMMUNICATIONS PROCESSING

SERIES 60 (LEVEL 6)

COMMUNICATIONS PROCESSING

SUBJECT

Descriptions and User Procedures for Communications Processing Software

SPECIAL INSTRUCTIONS

This revision supersedes Revision 0 of the manual dated January 1978. Change bars are omitted because of extensive reorganization in sections and content that are too numerous to identify separately.

SOFTWARE SUPPORTED

This manual supports Release 0110 of the Series 60 (Level 6) GCOS 6 MOD 400 Operating System. See the Manual Directory of the latest *GCOS 6 MOD 400 System Concepts* manual (Order No. CB20) for information as to later releases supported by this document.

ORDER NUMBER

CB03, Rev. 1

July 1978

Honeywell

PREFACE

This manual describes the operation and use of GCOS communications software for Honeywell-supported Series 60 (Level 6) communications devices and protocols. The term GCOS as used in the manual refers to GCOS 6 software. The term Level 6 refers to a specific series of Series 60 (Level 6) hardware models on which GCOS software is executed.

Section 1 is a brief overview of GCOS software in general and its communications subsystem.

Section 2 summarizes the Monitor and file system macro calls and services.

Sections 3 and 4 discuss the use of communications with COBOL and FORTRAN application programs, respectively.

Section 5 describes the use of communications in assembly language applications, using the GCOS file system interface.

Section 6 describes the use of communications in assembly language applications, using GCOS physical I/O for more direct access to data structures and physical devices.

Sections 7, 8, 9, and 10 describe the operation and use of Honeywell line protocol handlers for teleprinter-type (TTY), visual-information projection (VIP), polled VIP emulator (PVE), and binary synchronous communication (BSC) device/protocols, respectively.

Appendix A provides more details about communications subsystem functions. Appendix B contains tables of possible values for the STTY command and $STTY macro call. Appendix C describes the system's resource control table (RCT), used as an interface between the software and the devices that use it. Appendix D contains various examples, intended for illustration only, of communications application programs for COBOL, FORTRAN, and assembly language.

Appendix E lists communictions control characters and character code sets. Appendix F lists the various device control characters and corresponding device keys. Appendix G describes how to obtain a dump of the multiline communications processor's (MLCP) and/or the dual communications processor's (DLCP) memory.

How to Use the Manual

The following are general guidelines to using the manual according to the reader's interests and responsibilities:

Sections	Applicable To:
1	All users
2, 3, 4, 5	Applications programmers/analysts using higher-level languages
6	Those responsible for system building; applications programmers/analysts using assembly language
7, 8, 9, 10	All users, but according to the device or protocol being used
Appendix G	All users.
Remaining appendixes	Users of corresponding numbered sections

MANUAL DIRECTORY

The following publications comprise the GCOS 6 manual set. The Manual Directory in the latest GCOS 6 MOD 400 Systems Concepts manual (Order No. CB20) lists the current revision number and addenda (if any) for each manual in the set.

Order Number	Manual Title
CB01	GCOS 6 Program Preparation
CB02	GCOS 6 Commands
CB03	GCOS 6 Communications Processing
CB04	GCOS 6 Sort/Merge
CB05	GCOS 6 Data File Organizations and Formats
CB06	GCOS 6 Systems Messages
CB07	GCOS 6 Assembly Language Reference
CB08	GCOS 6 System Service Macro Calls
CB09	GCOS 6 RPG Reference
CB10	GCOS 6 Intermediate COBOL Reference
CB20	GCOS 6 MOD 400 System Concepts
CB21	GCOS 6 MOD 400 Program Execution and Checkout
CB22	GCOS 6 MOD 400 Programmer's Guide
CB23	GCOS 6 MOD 400 System Guilding
CB24	GCOS 6 MOD 400 Operator's Guide
CB25	GCOS 6 MOD 400 FORTRAN Reference
CB26	GCOS 6 MOD 400 Entry-Level COBOL Reference
CB27	GCOS 6 MOD 400 Programmer's Pocket Guide
CB28	GCOS 6 MOD 400 Master Index
CB30	Remote Batch Facility User's Guide
CB31	Data Entry Facility User's Guide
CB32	Data Entry Facility Operator's Quick Reference Guide
CB33	Level 6/Level 6 File Transmission Facility User's Guide
CB34	Level 6/Level 62 File Transmission Facility User's Guide
CB35	Level 6/Level 64 (Native) File Transmission Facility User's Guide
CB36	Level 6/Level 66 File Transmission Facility User's Guide
CB37	Level 6/Series 200/2000 File Transmission Facility User's Guide
CB38	Level 6/BSC 2780/3780 File Transmission Facility User's Guide

Order Number	Manual Title
CB39	Level 6/Level 64 (Emulator) File Transmission Facility User's Guide
CB40	IBM 2780/3780 Workstation Facility User's Guide
CB41	HASP Workstation Facility User's Guide
CB42	Level 66 Host Resident Facility User's Guide
CB43	Terminal Concentration Facility User's Guide

The following documents provide general hardware information:

Order Number	Manual Title
AS22	Honeywell Level 6 Minicomputer Handbook
AT04	Level 6 System and Peripherals Operation Manual
AT97	MLCP Programmer's Reference Manual
FQ41	Writable Control Store User's Guide

CONTENTS

CONTENTS (cont)

CONTENTS (cont)

CONTENTS (cont)

CONTENTS (cont)

CONTENTS (cont)

CONTENTS (cont)

CONTENTS (cont)

CONTENTS (cont)

ILLUSTRATIONS

ILLUSTRATIONS (cont)

TABLES

TABLES (cont)

TABLES (cont)

SECTION 1

COMMUNICATIONS OVERVIEW

GCOS SOFTWARE OVERVIEW

The GCOS 6 Operating System includes the Monitor, file system, physical input/output (P I/O), and communications software.

The Monitor controls loading of user programs, supports execution of user applications tasks, and provides system services for users to control execution of separate tasks. Monitor functions are obtained through commands, through system macro calls, and through statements in higher level languages.

The operating system has two levels of interface with remote and local terminals; they may be accessed indirectly through the sequential file interface of the file system's file management facility, or directly through the system's physical I/O facility.

The file system, which is based on a tree-like hierarchical directory/pathname structure, provides software to create and maintain that structure, to create and manage files, and to provide the logical transfer of data between an application and an external device. These functions are available through commands, and for an assembly language programmer, through the system service macro calls of the file system.

The physical input/output (or physical I/O) driver software (for peripheral devices), and similar line protocol handler software (for communications devices) work at the physical hardware level. Physical I/O is used with assembly language programs to call device drivers and line protocol handlers directly.

Communications software, through the file system, uses system service macro calls for communications data operations with all languages. For assembly language applications, communications software, through physical I/O, provides the data operations that are provided by the file system, plus additional controls over terminal functions at the hardware physical level.

The System Concepts manual describes the file system and file system structure in detail, and is necessary in understanding system terms, directory/pathname structures, and system functions that may be referred to in this manual.

GCOS 6 File System

The file system includes an extensive set of logical input/output access methods that handle logical input/output for all supported peripheral devices and terminals. The file system provides sequential file processing for communications, treating communications devices as sequential files. A file is the basic, or lowest level structural unit that can be referred to in the file system software. Within the file system, a file can be generally defined as a peripheral device, as a terminal device, or as an aggregate of data.

Section 2 summarizes the file system macro calls and data structures that are used in communications processing. Sections 3, 4, and 5 discuss the file system interface in communications processing in COBOL, FORTRAN, and assembly language, respectively.

Physical Input/Output (Physical I/O)

Physical I/O provides all services that are availble through the file system, plus other services that permit user control over data structures that affect terminals' hardware and operating characteristics. With the physical I/O interface, assembly language applications can call line protocol handlers directly, rather than through the indirect interface provided by the file system.

GCOS COMMUNICATIONS SUBSYSTEM OVERVIEW

GCOS communications software can be considered as a functional group of components known as the communications subsystem, which when specified at system building, defines the communications environment of the operating system.

The communications subsystem interacts with the Monitor to service applications programs, and provides all the communications software needed with Honeywell-supported communications devices, so that the user need not write his own. Communications software is user-driven, responding to connects, reads, or writes issued by user programs. Through the request I/O ($RQIO) macro calls, the communications subsystem provides a common physical I/O interface with user programs.

The communications subsystem comprises the communications supervisor, the line protocol handlers (one for each class of supported communication device), the multiline communications processor (MLCP) driver, and the MLCP itself.

Appendix A describes the overall functions of the communications subsystem in more detail. The line protocol handlers for specific devices and protocols are described in Sections 7 through 10.

Communications Supervisor

The communications supervisor, which resides in the central processor's main memory, provides the interface at the physical I/O level to communications applications programs. It queues user programs' requests for services, activates the appropriate line protocol handler, interacts with a user application through system software when a transaction is complete, and services connect/disconnect requests and timeouts for line protocol handlers.

Line Protocol Handler (LPH)

A communications protocol is a set of conventions for transmitting data over a communications line. A line protocol handler (usually referred to as an LPH) is the memory-resident reentrant and interrupt-driven program that transfers data between a communications device and the application program or system that uses that device. Each LPH supports a specific class of device, e.g., teleprinter-compatible terminal (TTY), or supports a communications protocol, e.g., binary synchronous communications (BSC). Other functions of an LPH are:

- o Handling error recovery (by parity or block control check)
- o Initializing the LPH and the channel control program of the multiline communications processor
- o Processing interrupts, timeouts, and I/O requests
- o Handling affirmative or negative acknowledgments

Defined at system building, an LPH can be any of the following:

TTY

Supports asynchronous terminal devices generically classified as teleprinter-compatible (TTY), including certain ASR, KSR, and visual information projection (VIP) terminals.

VIP

Supports synchronous VIPs and receive-only printers (ROPs)

PVE

Services the polled VIP emulator (PVE), or keyboard/ screen features of the VIP 7700 operating according to the polled VIP protocol

BSC

Supports a station (device) operating under binary synchronous communication (BSC) 2780 or 3780 compatible protocol.

Appendix A has a more detailed description of line protocol handler functions.

The user may write his own line protocol handler provided it conforms to the same internal interface requirements used by the Honeywell-supplied line protocol handlers.

Multiline Communications Processor (MLCP) and MLCP Driver

The multiline communications processor includes a channel control program (CCP) for each class of supported device. The MLCP driver, which resides in main memory when defined at system building, sets up and processes input/output orders from the line protocol handlers, and services MLCP interrupts. The Series 60 (Level 6) MLCP Programmer's Reference Manual describes the multiline communications processor in detail.

Communications Subsystem Interface With Applications Programs

FILE SYSTEM INTERFACE

The file system interface, operating between the application program and the terminal, provides, through communications software, system service file management macro calls that:

- Open the file
- Read data from the file (or device)
- Write to the file (or device)
- Test for completion of processing
- Wait for completion of processing
- Close the file

COBOL and FORTRAN run-time routines issue these macro calls according to the corresponding input/output statements in the compiled programs (see Sections 3 and 4). File system services are available also to assembly language programs (see Section 5).

Section 2 describes these system services macro calls and data structures briefly, the System Service Macro Calls manual describes all GCOS 6 macro calls and related data structures in detail.

PHYSICAL INPUT/OUTPUT INTERFACE

The physical I/O interface permits direct user control over communications processing. The physical I/O interface can be used only with assembly language programs, which can call a line protocol handler directly rather than indirectly through the file system interface.

Physical I/O macro calls used in communication between an application and line protocol handler are:

- o Request I/O transfer ($RQIO)
- o Input/output request block, generate ($IORB)
- o Set terminal characteristics ($STTY)

Section 6 discusses physical I/O, the macro calls, and data structures in more detail.

TTY and VIP Line Protocol Handler Device Support

Asynchronous devices supported by the TTY line protocol handler are referred to throughout the manual as teleprinter-compatible or TTY devices.

Synchronous devices supported by the VIP line protocol handler are referred to throughout the manual as VIP devices. The VIP designation applies also to receive-only printers (ROPs) associated with a VIP terminal.

BSC and PVE Host-Communications Support

Binary synchronous communications (BSC) permits communication between a Level 6 and another computer system that suppoorts the 2780/3780 protocols.

The polled VIP emulator (PVE) permits a Level 6 computer to communicate with another Level 6, Level 66, or any other Honeywell host system.

Sections 9 and 10 have detailed descriptions of the BSC and PVE line protocol handlers.

SECTION 2

FILE SYSTEM FUNCTIONS AND MACRO ROUTINES

This section discusses those macro routines and related data structures that pertain to communications processing and are often referred to throughout this manual. The System Service Macro Calls manual describes in detail the format, functional description, and arguments for each macro routine, and corresponding data structures.

The macro routines summarized and listed in this section have the following file system functions, which are organized according to the following major functional groups:

- o File/management
- o Data management
- o Storage management

The file management macro routines provide service functions at the file level (i.e., reserving files, opening and closing files, testing the status of I/O activity, etc.). Data management macro routines supply service functions at the record level, such as read, write, delete, and rewrite. Storage management macro routines furnish service functions such as read and write at the block (unit of transfer) level. Since terminal files are are considered to be simple, unblocked sequential files, storage and data management functions are equivalent.

FILE MANAGEMENT MACRO CALLS

The file management macro calls let the user manipulate his files within the file system hierarchy (described in the System Concepts manual). File management macro functions that apply to communications processing are:

- o Get a file (reserve a file for processing) ($GTFIL)
- o Open a file ($OPFIL)
- o Close a file ($CLFIL)

- o Remove a file from processing ($RMFIL)
- o Associate a logical file number with a pathname ($ASFIL)
- o Dissociate a logical file number from a pathname ($DSFIL)
- o Get information about a file ($GIFIL)
- o Test the status of an outstanding I/O activity (terminal) ($TIFIL/$TOFIL)
- o Wait for the completion of an asynchronous I/O activity (terminal) ($WIFIL/$WOFIL)

The file reservation function (get-file) can be done outside program execution by the GET command.

DATA MANAGEMENT MACRO CALLS

The data management macro calls allow manipulation of logical records within a file. The macro calls that apply to communications processing are:

- o Write a record ($WRREC)
- o Read a record ($RDREC)

Arguments required by these functions are passed in a file information block (FIB), described later in this section. The macro calls to generate and change FIBs and to define FIB offsets are discussed in the System Service Macro Calls manual.

Before any data management macro calls can be executed, the terminal file must have been reserved and opened with the LFN supplied in the FIB (get file ($GTFIL) and open file ($OPFIL) macro calls).

STORAGE MANAGEMENT MACRO CALLS

The storage management macro calls provide a primitive interface for transferring blocks directly between the user buffer and a file. Storage management itself is used by data management to perform input/output.

The complexities of blocking and deblocking logical records, and conforming at the same time to the various file organizations and formats, recommend against using storage management when dealing with I/O at the logical record level. To ensure maximum efficiency in terms of space and access, let the system (i.e., data management) handle the records.

However, for unblocked records or large blocks with simple fixed-length records to be blocked by the user, the storage management macro calls can be used to perform I/O transfers between the user buffer and the file.

Storage management macro functions are:

- o Read a block ($RDBLK)
- o Write a block ($WRBLK)
- o Wait for the completion of an I/O activity ($WTBLK)

FILE INFORMATION BLOCK (FIB)

Some macro routines, particularly for data and storage management, use a data structure called the file information block (FIB), which provides the interface between a user program and the system for data and storage management. In order for the file to be accessed, there must be one FIB for each file.

The $FIB macro call is used to build a file information block, alter its contents, or to provide labels for its entries.

The FIB must be provided to each of the following macro calls:

$OPFIL:	open file
$CLFIL:	close file
$TIFIL:	test file for input
$TOFIL:	test file for output
$RDREC:	read record
$WRREC:	write record
$RDBLK:	read block
$WRBLK:	write block

FIB Format and Contents

Figure 2-1 shows the format of the FIB; Table 2-1 shows its contents.

Figure 2-2 shows the format of the FIB for data management applications; Table 2-2 shows its contents.

Figure 2-3 shows the format of the FIB for storage management applications; Table 2-3 shows its contents.

Word	Label	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
0	F_LFN	LOGICAL FILE NUMBER
1	F_PROV	PROGRAM VIEW
2	F_URP/F_UBP	USER RECORD/BUFFER POINTER
3		
4	F_IRL/F_BFSZ	INPUT RECORD LENGTH/BUFFER SIZE
5	F_ORL/F_BKSZ	OUTPUT RECORD LENGTH/BLOCK SIZE
6	F_LIRT/F_BKN01	RECORD TYPE RANGE/BLOCK NUMBER
7	F_HIRT/F_BKN02	RECORD TYPE RANGE/BLOCK NUMBER
8	F_ORT	RESERVED
9	F_IKP	INPUT KEY POINTER
10		
11	F_IKF/F_IKL	INPUT KEY FORMAT/INPUT KEY LENGTH
12	F_ORA1	(LEFT) OUTPUT RECORD ADDRESS
13	F_ORA2	(RIGHT) OUTPUT RECORD ADDRESS
14	F_RFU	RESERVED
15		

Figure 2-1. File Information Block (FIB)

Table 2-1. Contents of File Information Block (FIB)

Item	Label	Bit(s)	Contents
0	F_LFN	0-15	Logical file number (LFN)
1	F_PROV	0	Access level - set on for storage management, off for data management.
		1-4	Process rules - bit 1 for $RDREC/$RDBLK, bit 2 for $WRREC/$WRBLK, bit 3 for $RWREC, bit 4 for $DLREC.
		5-9	Key type - bit 5 for primary keys, bit 8 for relative keys, bit 9 for simple keys (bits 6 and 7 must be 00).
		10	Record class - set on for fixed-length records only, off for fixed- and variable-length records.
		11	Record visibility - set on if deleted records are to be visible, off if invisible.
		12	Key storage alignment - set on if storage area begins at odd-byte boundary, off if even-byte boundary.

Table 2-1 (cont). Contents of File Information Block (FIB)

Item	Label	Bit(s)	Contents
1 (cont)	F_PROV (cont)	13	Record storage area/buffer alignment - set on if record storage area (or buffer) begins on odd-byte boundary, off if even-byte boundary.
		14	Transcription mode - set on if data transferred in binary transcription mode, off if ASCII mode.
		15	Synchronous/asynchronous indicator - set on if $RDBLK/$WRBLK calls executed asynchronously, off if executed synchronously.
2 3	F_URP/ F_UBP	0-31	Start address of user record area data management) or start address of buffer area (storage management).
4	F_IRL/ F_BPSZ	0-15	Input record length (data management) or transfer size (storage management).
5	F_ORL/ F_BKSZ	0-15	Output record length (data management) or block size (storage management).
6	F_LIRT/ F_BKN01	0-15	must be 0000 for data management; is the left half of the block number (F_BKN01) for storage management.
7	F_HIRT/ F_BKN02	0-15	Must be FFFF for data management; is right half of the block number for storage management.
8	F_ORT	0-15	Must be 0000.
9	F_IKP	0-31	Start address of user key area.
11	F_IKF/ F_IKL	0-7	Input key format (0 for none specified, 1 for primary key, 2 for simple key)
		8-15	Input key length.
12	F_ORA1	0-15	Output record address (left half).
13	F_ORA2	0-15	Output record address (right half).
14	F_RFU	0-31	Reserved for future use.

Program View Entry in the FIB

The FIB's program view entry (item 1 in the FIB) describes to the file system how the file is to be accessed, and what the file looks like from the programmer's point of view. The file system uses the FIB's contents to ensure that the file is accessed only as intended.

The bits in the program view entry are read when the file is opened. After the file is opened, the user can change only bits 11, 12, and 13. Other bits cannot be changed until the file is closed and then reopened.

Table 2-1 above shows the contents of the program view entry indicated as item 1 and labeled F_PROV. The System Service Macro Calls manual describes the program view entry in detail, with reference to its usage for specific file system services and macro calls.

FIB Displacement Definitions

Displacement definition macro calls are used to refer to specific locations in the FIB and in the various macro call argument structures. These calls define standard displacement tags.

The $TFIB macro call defines tags for the FIB for the following macro calls:

Open file ($OPFIL)
Close file ($CLFIL)
Test file ($TIFIL, $TOFIL)
Read record ($RDREC)
Write record ($WRREC)
Rewrite record ($RWREC)
Delete record ($DLREC)
Write block ($WRBLK)
Wait block ($WTBLK)

Word	Label	
0	F_FLN	LOGICAL FILE NUMBER
1	F_PROV	PROGRAM VIEW
2	F_URP	USER RECORD POINTER
3		
4	F_IRL	INPUT RECORD LENGTH
5	F_ORL	OUTPUT RECORD LENGTH
6	F_RFU1	RESERVED
7	F_IRT	INPUT RECORD TYPE
8	F_ORT	OUTPUT RECORD TYPE
9	F_IKP	INPUT KEY POINTER
10		
11	F_IKF/F_IKL	INPUT KEY FORMAT INPUT KEY LENGTH
12	F_ORA	OUTPUT RECORD ADDRESS
13		
14	F_RFU2	
15		RESERVED

Figure 2-2. Format of File Information Block (FIB) for Data Management

Table 2-2. Contents of FIB for Data Management

Word	Label	(Bits)	Contents	Applicable Macros
0	F_LFN	0-15	Logical file number (LFN)	
1	F_PROV	0	Access level - OFF to indcate to access via data management	$OPFIL
		1-4	Access rules - Bit 1: ON if $RDREC will be issued Bit 2: ON if $WRREC will be issued Bits 3, 4: does not apply - set to OFF	$OPFIL
		5-9	Do not apply - set OFF	
		10	Record length verification - ON when expecting fixed length record and OFF for variable length record	$RDREC
		11-12	Do not apply - set OFF	
		13	User record area alignment - ON if user record record area begins on odd-byte boundary, off if even-byte boundary.	$RDREC $WRREC
		14-15	Do not apply - set OFF	
2,3	F_URP	0-31	Start address of user record area	$RDREC $WRREC
4	F_IRL	0-15	Input user record area size in bytes	$RDREC
5	F_ORL	0-15	Output user record area size bytes	$RDREC
			Actual record size in bytes filled by data management on each macro call	$RDREC $WRREC
6	F_RFU1	0-15	Reserved - set to 0	
7	F_IRT	0-15	Do not apply - set to FFFF	
9	F_ORT	0-15	Do not apply - set to 0	

Table 2-2 (cont). Contents of FIB for Data Management

Word	Label	Bit(s)	Contents	Applicable Macros
9,10	F_IKP	0-31	Do not apply - set to 0	
11	F_IKF F_IKT	0-7 8-15	Do not apply - set to 0 Do not apply - set to 0	
12,13	F_ORL	0-31	Output record address - line sequence number filled by data management on each macro call	$RDREC $WRREC
14,15	F_RFU2	0-31	Reserved - set to 0	

Word	Label	
0	F_LFN	LOGICAL FILE
1	F_PROV	PROGRAM VIEW
2	F_UBF	USER BUFFER POINTER
3		
4	F_BFSZ	USER BUFFER SIZE
5	F_BKSZ	USER BLOCK SIZE
6	F_BKNO	BLOCK NUMBER
7		
8	F_RFU3	
9		
10		RESERVED
11		
12		
13		
14		
15		

Figure 2-3. Format of File File Information Block (FIB) For Storage Management

Table 2-3. Contents of FIB for Storage Management

Word	Label	Bit(s)	Contents	Applicable Macros
0	F_LFN	0-15	Logical File Number (LFN)	
1	F_PROV	0	Access level - ON (to indicate access via storage management)	$OPFIL
		1-4	Access Rules: Bit 1: ON IF $RDBLK will be issued Bit 2: ON if $WRBLK will be issued Bits 3-4: Does not apply - set to OFF	$OPFIL
		5-12	Do not apply - set to OFF	
		13	User buffer area alignment - ON if user buffer area begins on odd-byte boundary, OFF if even-byte boundary	$RDBLK $WRBLK
		14-15	Do not apply - set to OFF	
2,3	F_UBP	0-31	Start address of user buffer area	$RDBLK $WRBLK
4	F_BFSZ	0-15	User buffer size in bytes	$RDBLK $WRBLK
			Actual transfer size in bytes filled by storage management on each macro call	$RDBLK $WRBLK
5	F_BKSZ	0-15	Do not apply - set to 256	
6,7	F_BKNO	0-31	Block Number - does not apply	
			Line sequence number filled by storage management on macro call	$RDBLK $WRBLK
8-15			Reserved - set to 0	

FILE SYSTEM CONSIDERATIONS IN COMMUNICATIONS

The file system provides device independent facilities so that terminals can be reserved, removed, opened, closed, read and written just like standard sequential files. In addition, asynchronous I/O facilities are provided for efficient processing in a multiterminal environment. Asynchronous I/O refers to the capability of the file system to perform I/O between a terminal and a system buffer while the application program executes in parallel. Facilities are available for the application program to test whether or not the I/O is complete and, alternatively, to give up control of the central processor until the I/O is comlete. This buffering capability is a device attribute and can be set at system build time or dynamically via the STTY command. The system buffer is actually acquired when the terminal is opened and returned when it is closed.

From the application program point of view:

- An application program can be written to be device independent. The terminals, whether or not buffered, whenever a logical read or write is issued, control returns only to the application program when data has been moved to or from the application area. Buffering improves performance by providing the same level of asynchronous I/O as for unit record devices like the card reader or line printer -that is, while the application is processing one message the file system may be reading the next. This kind of application is efficient in a single terminal environment.

- A more complex level of asynchronous I/O is necessary when the application program must interact with multiple terminals, establish its own polling priorities and run efficiently with high response time. One example is the traditional online/batch environment where, when terminal input is available, the online task has highest priority with respect to CP time, memory, etc., with batch processing operating efficiently while online processing is dormant. Facilities are available to schedule I/O without waiting for its completion, to continue task execution in parallel with the I/O transfer, to test to see if the I/O is complete, and to wait until I/O is complete.

- For interactive terminals an open causes an asynchronous physical connect to be performed while the application continues executuion. The application can then test to determine if the connect is complete and input is available, or if the device is ready for output.

- Before reading, the application task can test the file status to see if a read can be done without stalling task execution. File status remains busy until the system buffer is full (i.e., the anticipatory read is complete). When the file status is not busy the application can issue a read with the assurance of receiving data immediately. The anticipatory read allows an application to control input from more than one terminal, each of which represents a data entry terminal. By testing the status of the system buffer before a read (FORTRAN,assembly) or by checking for the 9I status after a COBOL READ, the application will not be stalled and it can continue to poll other terminals. The user can establish the order of the tests and thus the polling priority.

- The application can also wait for input from a list of terminals. CP time is then made available to lower priority tasks until input is available from one or more terminals in the list.

- A buffered write operation to a terminal works on behalf of the application program in the same logical manner as the read, that is, the program is allowed to execute in parallel with the physical transfer to the device. Each write call is completed by moving data from the application area to the file system buffer (with detabbing if required), initiating the output transfer and returning control to the application program. If the program performs a second write while the system buffer is still in use for the previous transfer, the application is stalled until the buffer is available and new data moved into it again. The application can avoid the stalling the execution by testing the status of the system buffer before issuing a write (FORTRAN,assembly) or by testing for the 9I status return after a WRITE in COBOL.

- The application program can also issue a wait for output to a list of terminals. CP time is then made available to lower priority tasks until output is complete to one or more terminals in the list.

DEFINING FILE/TERMINAL CHARACTERISTICS

There are these considerations in defining terminal file characteristics for the file system. The first deals with a file's operational characteristics (with respect to the device) when the system is first build. The DEVICE directive permits the user to specify among others the default record size of the file and the use of an intermediate buffer (this option is specified by the buffered/unbuffered argument). Buffered device operation is advantageous in synchronous operations against a file, and is mandatory in asynchronous operations against a file.

The second consideration involves the secondary specialization of a file's device's operational characteristics. This specialization can be done at system build by using the STTY directive, from the user's terminal via an STTY command, and during program execution with the $STTY macro call. In each case the $STTY macro call or STTY command permits the following:

- Modification of default record size.

- Specification of the device-specific word which determines the operational characteristics of the device (e.g., whether a control byte is used or a disconnect will force a queue abort).

- Specification of the file indicator word which determines the operational characteristics of the file system (e.g., if the file system is to support input and/or output operations, and whether these operations are synchronous/asynchronous).

The final consideration deals with specifying selected file characteristics at open time. Of particular interest is the program view word of the file information block (FIB), which defines whether the file system is to support input and/or output operations against a file.

SECTION 3

COMMUNICATIONS VIA COBOL

The file system interface (see Sections 1 and 2) provides the logical transfer between the COBOL program and an external device (terminal or another computer). The COBOL run-time routines issue file system macro calls according the the corresponding input/output statements in the compiled programs.

INTERACTIVE DEVICES AND FILES

The operating system defines communications devices and local TTY terminals in COBOL communications processing as "interactive."

Interactive devices can be considered as logical repositories of sequential files in COBOL. Data is read or written with the same COBOL read/write interface as for a file on a noninteractive device.

FILE SYSTEM CONSIDERATIONS

Aside from the use of various COBOL I/O statements the user should be aware of other considerations in using the file system within a communications environment. These considerations are detailed in Section 2.

SOURCE PROGRAM ENTRIES IN COMMUNICATIONS

This subsection refers to certain COBOL source program entries in the context of COBOL communications. The appropriate COBOL Reference manual describes COBOL source program language in detail.

Specifying Files in the Source Program

The user must describe every file with a separate SELECT statement in the FILE-CONTROL paragraph of the Environment Division. File organization and access mode must be stated as sequential.

Each file must have a unique name and, and in the ASSIGN clause, be identified by a 2-character COBOL internal file name (IFN) consisting of a combination of the letters A through I and the digits 0 through 9; one letter must be included. The logical file number (LFN) is specified in the ASSOC or GET commands (before execution) to connect the COBOL internal file name to the external file. This LFN is the same as the COBOL internal file name with letters A through I replaced by the digits 0 through 9. For example, a COBOL IFN of 0C would correspond to an LFN of 03 and an IFN of 0D to an LFN of 04, as in the commands.

```
ASSOC 03   >SPD>VIP1
GET   04   >SPD>TTY1
```

Use of ASSOC or GET Commands

In addition to connecting the internal file name to the external file, the GET command reserves the interactive file for processing until it is removed via the REMOVE command. GET allows the user to guarantee exclusive use of the file prior to program execution and maintain use of the file until the corresponding REMOVe command.

ASSOC, on the other hand, merely connects the internal file name to the external file, without reserving it for use. Each COBOL OPEN statement will cause the file to be reserved exclusively while each COBOL CLOSE statement will remove this reservation.

In a multi-user environment the use of ASSOC command may cause an OPEN to fail because some other user has reserved the file exclusively while the GET command guarantees that OPEN will not fail as a result of some other user's reservation request.

ASSIGNING A FILE TO A DEVICE/TERMINAL

A device-type name of MSD used in the ASSIGN clause of the SELECT statement is the way that the user informs COBOL that the internal file is assigned to a terminal/device file.

For data entry applications (TTY or VIP) the file should be opened in INPUT mode.

For output-only terminals such as the Receive Only Printer (ROP) the file should be opened in OUTPUT mode. Bidirectional devices, such as the BSC 2780 can be opened in INPUT mode or OUTPUT mode but not for both INPUT and OUTPUT at the same time.

For interactive applications (TTY, VIP or BSC3780), the file can be opened in I-O mode allowing both input and output operations.

SELECT and ASSIGN Examples

Figure 3-1 shows an example of a FILE-CONTROL paragraph with SELECT and ASSIGN statements for the input file COMIN and the output file COMOUT. The internal file name for COMIN is 0C and for COMOUT is 0D. Before the program is executed, the user must associate these files with the appropriate device(s) with either an ASSOC or GET command. In this example, the commands could be:

```
GET 03   >SPD>TTY1
GET 04   >SPD>TTY1
```

Although these are different files, they can be associated with the same interactive device, i.e., TTY1, by matching the logical file numbers (03 and 04 for the device pathname >SPD>TTY1) with the internal file name 0C and 0C, respectively.

```
FILE-CONTROL

     SELECT COMIN

          ASSIGN TO OD-MSD
          ORGANIZATION IS SEQUENTIAL WITH VLR
          ACCESS MODE IS SEQUENTIAL
          FILE STATUS IS IN-STAT.


     SELECT COMOUT

          ASSIGN TO OD-PRINTER
          ORGANIZATION IS SEQUENTIAL WITH VLR
          ACCESS MODE IS SEQUENTIAL
          FILE STATUS IS OUT-STAT.
```

Figure 3-1. COBOL SELECT and ASSIGN Examples

Carriage Control

Some devices can be configured such that print carriage control is visible on output to the application program. If the device-type name is MSD, then the application program controls the carriage directly by inserting a program-accessible control byte as the first character in each output record. This byte is the first character in each level-01 record description entry for the output file. It is counted as part of the record area and is directly accessible through statements in the COBOL application program.

Printer Emulation

The user can pretend the device is a printer and more automatically control the carriage. If the device-type name is PRINTER in the ASSIGN clause then COBOL will automatically generate the carriage control byte as a result of an ADVANCING phase in the WRITE statement. This one byte print control character is inserted before each data record being written to the file. It is not counted as part of the record area and is not directly accessible tot he application program.

Specifying Asynchronous or Synchronous Read and Write Execution

If the device is configured with the asynchronous I/O attribute then READ and WRITE statements may be executed synchronously or asynchronously, as indicated by the programmer through calls to the COBOL run-time routines ZCASYN (asynchronous execution) or ZCSYNC (synchronous execution). If neither call is specified, reads and writes are executed asynchronously.

A separate call to ZCSYNC or to ZCASYN is not necessary for each read or write, but when first issued, remains effective until changed by another call. However, if the same run unit is to execute several COBOL programs, each program must separately define its own synchronous or asynchronous condition.

SYNCHRONOUS READ AND WRITE OPERATION (CALL "ZCSYNC")

In synchronous operation, the COBOL routine issues a read or write order without any file status checks. This causes the application program to be put in the wait state until the read or write operation is complete, thus allowing other tasks to be executed.

The source language for synchronous read and write execution is:

```
CALL "ZCSYNC"
```

Synchronous operation is not very useful in a multiterminal environment since each read or write to a terminal must be satisfied before the next terminal can be processed.

ASYNCHRONOUS READ AND WRITE OPERATION (CALL "ZCASN")

In asynchronous operation COBOL READ/WRITE run-time routines issue a test-file call prior to issuing a read or write order. For READ orders, a 9I return status is returned to the application if no data is available to be read. Likewise, for a WRITE order, a 9I status is returned to the application if the device is busy with the previous output. This permits the COBOL program to support terminal I/O without giving up control of the central processor until the I/O is complete.

WAIT for Completion -- Asynchronous Input and Output

In a multi-terminal system the user can control asynchronous read and write operations by calling the COBOL run-time routines ZCWIN and ZCWOUT.

A call to ZCWIN results in a wait file ($WIFIL) macro call which waits until input is available from one or more of the specified terminals.

A call to ZCWOUT results in a wait-file ($WOFIL) macro call which waits until output is complete to one or more of the specified terminals.

The System Service Macro Calls manual describes the wait file macro calls, their format and arguments, in detail. Note that the macro call arguments are similar to the values for the data-name description for the CALL statements (see below).

The source language to call ZCWIN or ZCOUT is:

$$\text{CALL} \left\{ \begin{array}{l} \text{"ZCWIN"} \\ \text{"ZCWOUT"} \end{array} \right\} \text{USING data-name}$$

Data-name is defined as follows:

```
01  data-name
    02  out-LFN  USAGE COMP-1.
    02  list-length  USAGE COMP-1.
    02  LFN-entry-1  USAGE COMP-1.
                   .
                   .
                   .
    02  LFN-entry-n  USAGE COMP-1.
```

The values for out-LFN, list-length, LFN-entry-1 and LFN-entry-n are identical to those for the wait file ($WIFIL and ($WOFIL) macro calls, and are passed by the ZCWIN or ZCWOUT routine to the file system.

When CALL "ZCWIN" is specified, the list of LFNs may refer only to hose devices for which READ statements have been issued. When call "ZCWOUT" is specified, the list of LFNs can refer only to those devices for which WRITE statements have been issued.

When an input/output operaton is completed on any device in the list of LFNs, the application program resumes execution following the CALL statement. The LFN for the device for which input/output is complete is stored in the out-LEN data item.

Figure 3-2 provides simplified program logic for processing multiple terminals. The call to "ZCWIN" stalls program execution until input is available from at least one of the terminals.

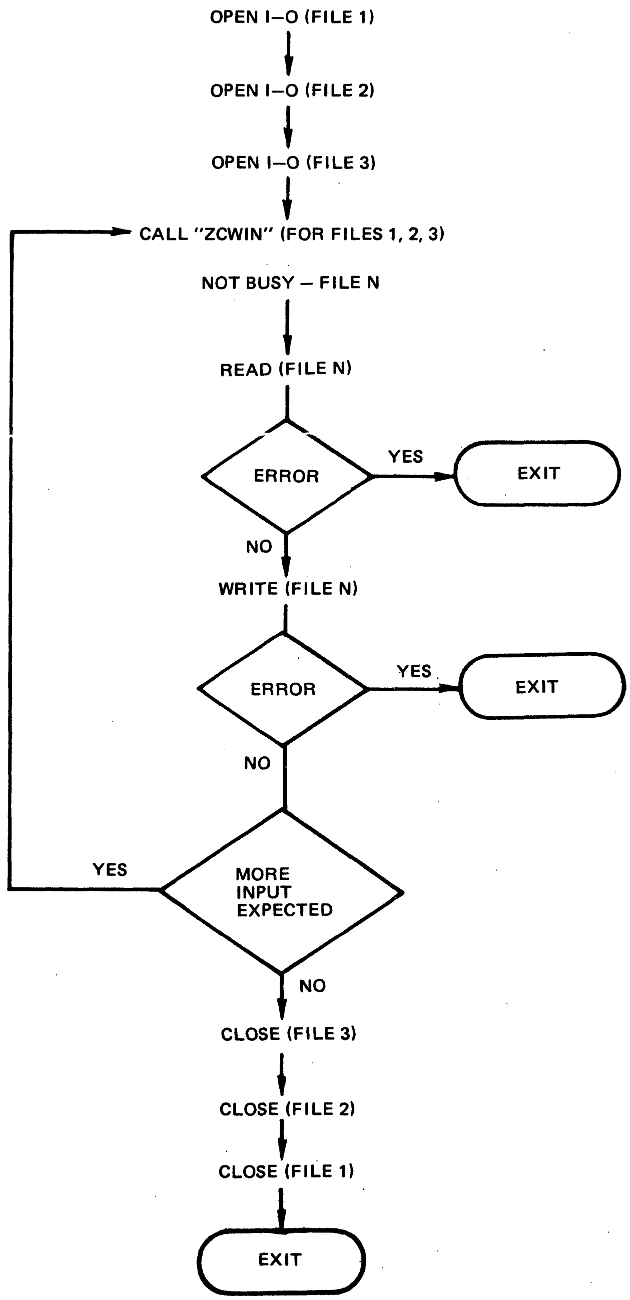


Figure 3-2. Simplified COBOL Program Logic for Multiple Interactive Terminals

The following is an example of a COBOL program which processes two terminals which have been configured to allow asynchronous input and synchronous output operations. The call to ZCWIN gives up control of the central processor until input is available from one of the terminals.

```
     FILE-CONTROL.

       SELECT COM1

         ASSIGN TO 0C-MSD
         ORGANIZATION IS SEQUENTIAL WITH VLR
         ACCESS MODE IS SEQUENTIAL
         FILE STATUS IS C1-STAT.

       SELECT COM2

         ASSIGN TO 0D-MSD
         ORGANIZATION IS SEQUENTIAL WITH VLR
         ACCESS MODE IS SEQUENTIAL
         FILE STATUS IS C2-STAT.

     PROCEDURE DIVISION.

         OPEN I-O COM1.
         OPEN I-O COM2.

       RD1.

         CALL "ZCWIN" USING FLN-LIST.
         READ COM1.
         IF C1-STATE "9I" GO TO RD2.
         IF C1-STATE "00" GO TO WR1.
         GO TO ERROR.

       RD2.

         READ COM2.
         IF C2-STAT "00" GO TO WR2.
         GO TO ERROR.

       WR1.

         WRITE COML1.
         IF C1-STAT "00" GO TO RD1.
         GO TO ERROR.

       WR2.

         WRITE COM2.
         IF C2-STAT "00" GO TO RD1.
         GO TO ERROR.
```

Before program execution, specify these commands to connect the LFNs to the specific terminal files.

```
GET 3  >SPD>TTY1 (for IFN 0C-MSD)
GET 4  >SPD>TTU2 (for IFN 0D-HSD)
```

Binary Synchronous Communication (BSC) With COBOL

Binary Synchronous Communication (BSC), operating in 2780 or 3780 mode, permits a COBOL program to transmit data over communications lines from one Level 6 system to another Level 6, to a Level 66 system, or to a non-Honeywell host system.

BSC DATA TRANSMISSION CONVENTIONS

BSC Data Codes

Data can be in alphanumeric ADCII, alphanumeric EBCDIC, or binary format. In communication between Level 6 and remote host, each system must use the same code set (either ASCII or EBCDIC). When EBCDIC is used, the application programs must know whether transmission is nontransparent or transparent (i.e., BSC control characters are interpreted as data).

BSC Data Transmission Modes

There are two BSC transmission modes: basic and advanced.

In basic transmission mode there is no control byte. The absence of a control byte limits the functionality of the protocol (e.g., an application cannot send or receive two message blocks or cannot initiate a reverse interrupt (RVI) sequence).

In advanced transmission mode there is a control byte which is the first byte in the program's input or output buffer. The control byte is used to control the transmission of data and is used to convey information concerning the receipt of data. With the control byte, the application has complete control over the transmission and reception of data to a remote host.

BSC 2780 and BSC 3780

BSC 2780 is a subset of BSC 3780. Technical differences between the two protocols can be summarized as a set of extensions to the 2780 protocol which are as follows:

- o The ability to receive a conversational reply without a preliminary bid sequence.
- o The ability to receive and transmit selected BSC control characters.

From a user's point of view the differences between the two protocols can be summarized below:

- o BSC 2780

  - Specified at system building time by the BSC device directive.

  - Operates in basic or advanced mode.

  - The file system supports bidirectional usage of BSC 2780 communication line. A CLOSE/OPEN sequence must be initiated prior to the reversal of the communication line.

- o BSC 3780

  - Specified a system building time by the XBSC directive.

  - Operates only in advanced mode.

  - The file system supports interactive usage of the BSC 3780 communication line. To terminate a transmission the application must initiate an EOT sequence by setting the appropriate bit within the control byte. An ETX message transmission sequence can also be terminated if the other application sends a conversational reply. The receipt of conversational reply is indicated by a bit setting within the transmit control byte. The receipt of a conversational reply forces the application to issue a read order to receive the conversational response. The termination of a read sequence is indicated by the AT END condition.

<u>Macro Call Procedures for BSC 2780 in Basic Transmission Mode</u>

The following conditions apply in the use of binary synchronous communications in basic data transmission mode:

- o An application cannot send an RVI (reverse interrupt) control character through the file system.

- o BSC devices in basic transmission mode cannot initiate double (ITB) message transmission (see Section 10).

- o An application can send only the ETB (end of transmission block) BSC control character, not the ETX (end of text) BSC control character.

- o An application can send data in either transparent or nontransparent mode.

o An application can send EOT (end of transmission) control characters by a CLOSE call.

o BSC operation assumes that the detab option is set off.

Figure 3-3 illustrates the necessary logic to support a BSC 2780 application in basic transmission mode.

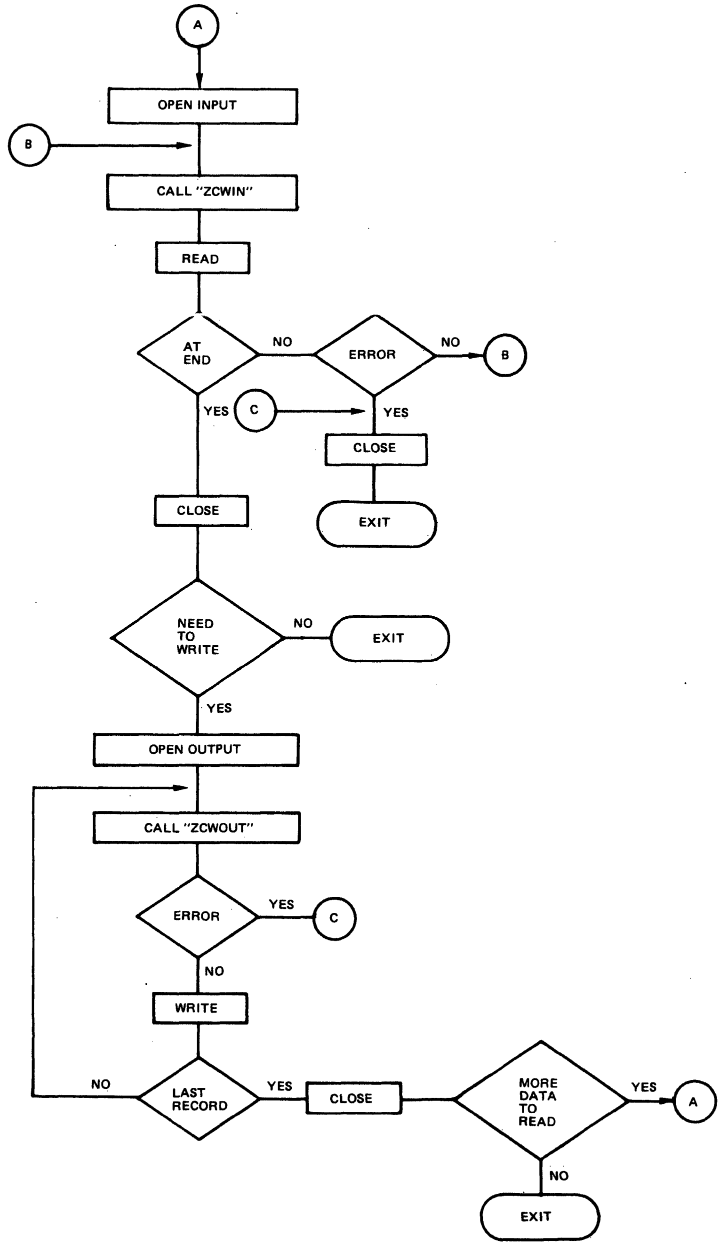


Figure 3-3. Simplified Program Logic for 2780 BSC

Macro Call Procedures for BSC 2780 in Advanced Data Transmission Mode

In the BSC advanced data transmission mode, the first byte of the application program's input or output buffer is a control byte that controls or supplies information about read/write operations. This byte can indicate, for example, whether data is to be transferred in transparent or nontransparent mode, or whether an ETB (end of transmission block) or ETX (end of text) control character is to be sent or received. Section 8 describes the control byte formats.

The following conditions apply in using the file system in 2780 binary synchronous communications in advanced data transmission mode:

> It is not necessary to send EOT control characters through the control byte since the user must close the file in output mode before attempting to read. Closing the file forces BSC if not in idle mode, to send an EOT control character.

Macro Call Procedures for BSC 3780 in Advanced Data Transmission Mode

The first byte of the application program's input or output buffer is a control byte. The control byte controls or supplies information about read/write operations.

The following conventions apply in using 3780 binary synchronous communication in advanced data transmission mode:

- The receipt of an optional conversational reply is indicated by a bit setting in the transmit control byte. (This can occur if the application has transmitted the last (ETX) block of a message). The application must issue a read in order to receive the conversational response.

- The termination of a transmit sequence is signaled (via control byte) by the transmission of an EOT control character following the last block of a message. Once this has been done a read macro call will be needed to receive transmissions from the remote system. (It is not necessary to close and reopen the file to turn the line around).

- The termination of a receive sequence is indicated by the AT END condition. A transmission sequence can be reinitiated by issuing another write macro call. (It is not necessary to close and reopen the file to turn the line around).

- o A line turnaround (receipt of an EOT) is indicated at the AT END condition. At this point the application can use the line for data transmission by issuing another write request. It is also possible to receive an EOT control character which indicates the abortion of the current transmission sequence by the remote host. Such an occurrence is indicated by an AT END condition. If this occurs the application must close the line.

Figure 3-4 illustrates the necessary logic to support a BSC 3780 application.

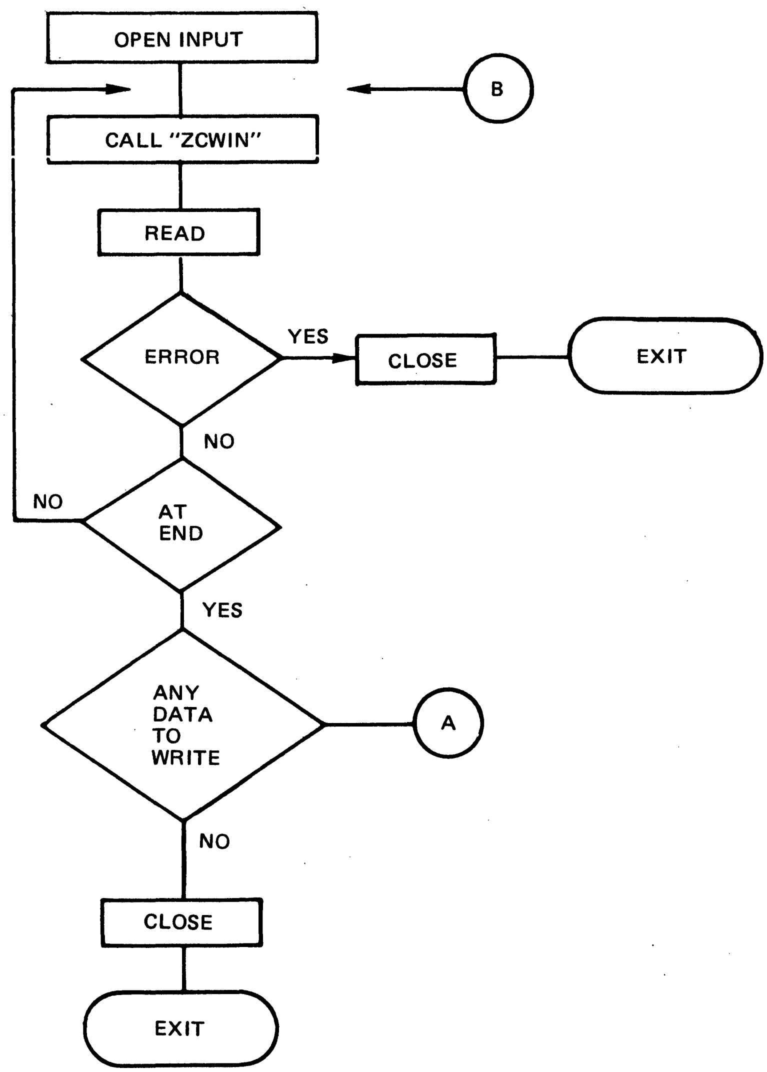


Figure 3-4. Simplified Program Logic for BSC 3780

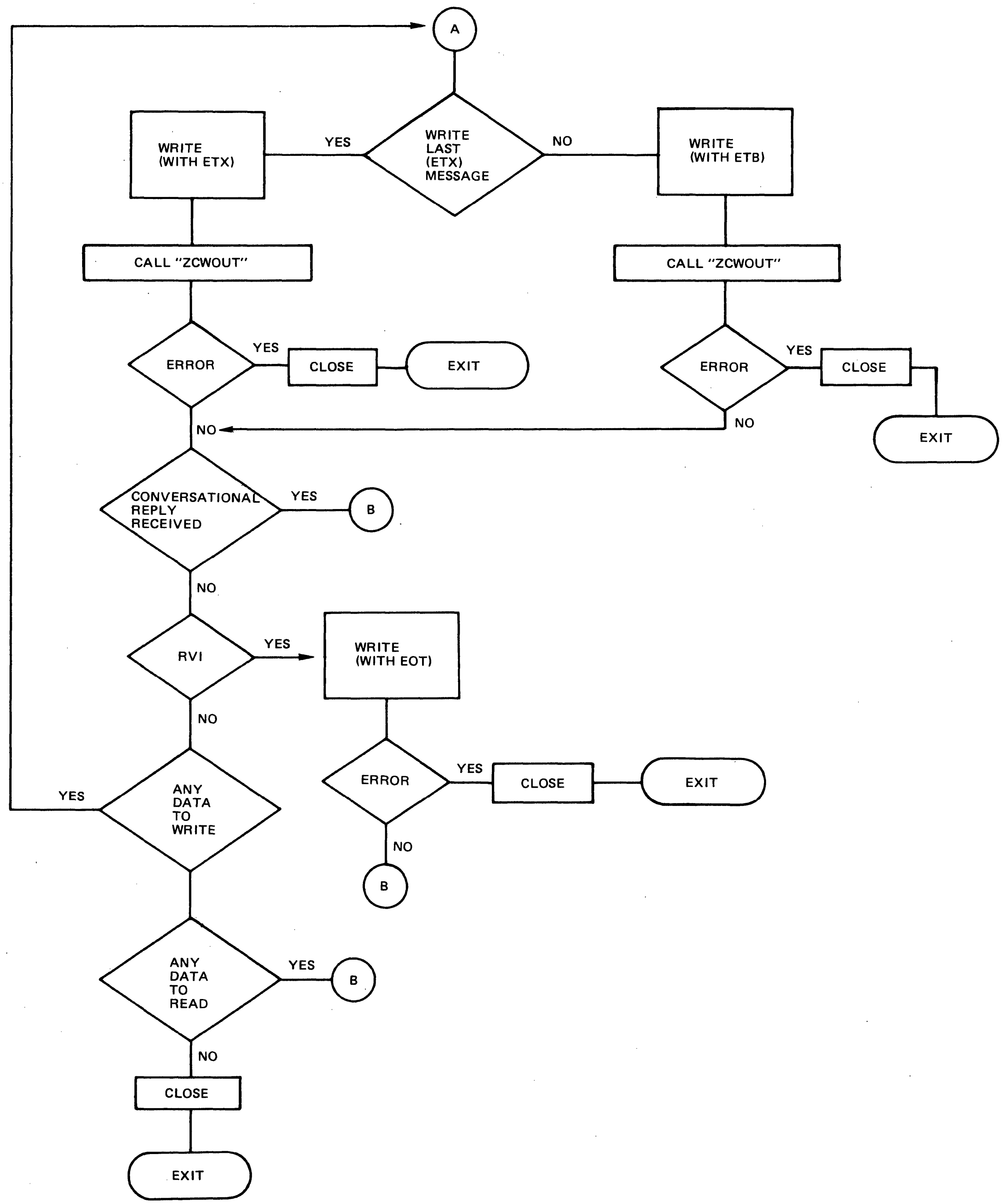


Figure 3-4 (cont). Simplified Program Logic for BSC 3780

SECTION 4

COMMUNICATION VIA FORTRAN

The file system interface (see Sections 1 and 2) provides the logical transfer between the FORTRAN program and an external device (terminal or another computer) in FORTRAN communications. The FORTRAN run-time routines issue file system macro calls according to the corresponding input/output statements in the compiled programs.

INTERACTIVE DEVICES AND FILES

The operationg system defines communications devices and local TTY terminals in FORTRAN communications processing as "interactive." Interactive devices can be considered as logical repositories of sequential files in FORTRAN. Data is read or written with the same FORTRAN read/write interface as for a file on a noninteractive device.

FORTRAN PROGRAM EXECUTION WITH COMMUNICATIONS

Assigning Interactive Devices at Execution

Before the compiled FORTRAN progran can be executed, the user must specify the actual interactive device for the specified file, using the system command ASSOC (associate path). The logical file number (LFN) specified in the command must be the same as the unit specifier (u) that was included in the control information list (clist) in the FORTRAN input/output statement READ, WRITE, or PRINT for that file. See the FORTRAN Reference manual for descriptions of FORTRAN statements and the unit specifier. See the Commands manual for descriptions of the ASSOC and other system commands.

Changing Terminal's File Characteristics

The user can change the file characteristics of a terminal e.g., line length (or record size), detabbing, device type (input, output, etc.,) with the system command STTY (set terminal characteristics), or with the $STTY macro call. This permits the user to modify the characteristics established at system building, and is issued before program execution.

Appendix B shows possible values for the device-specific word and file-indicator word arguments of the STTY command and $STTY macro call.

FORTAN FILE STATUS CHECK (ZFSTIN AND ZFSTOT)

Before a FORTRAN file can be used in communications, the FORTRAN statement OPEN must be specified before any other input/output statement.

The FORTRAN subroutines ZFSTIN (for input files) and ZFSTOT (for output files) enable the application program to check the status of the input or output communications device (file) before issuing a READ or WRITE statement.

When the program issues an I/O request statement (a READ or WRITE), it stalls until that request is completed.

The FORTRAN subroutines ZFSTIN and ZFSTOT, when called before an I/O request is issued, check the availability of the communications device (file), and can prevent the problem of program inactivation or program execution due to file or device unavailability.

The subroutine ZFSTIN checks the status of the input file, ZFSTOT checks the output file. Their use monitors the status of the files without loss of program control and prevents the imposition of file system waits.

A CALL statement to either subroutine should be issued before the application issues any I/O requests to ascertain (1) whether the file (device) is available, and (2) any device error status.

The subroutine ZFSTIN or ZFSTOT, when called, issues a request to the file system, which in turn (without waiting for any pending I/O request to be completed) returns status information about the file's availability. When the file is not busy, the file system will return status information about the previous I/O request.

CALL Statement for ZFSTIN or ZFSTOT

The CALL statement for subroutine ZFSTIN or ZFSTOT is specified as:

$$\text{CALL} \left\{\begin{matrix}\text{ZFSTIN}\\ \text{ZFSTOT}\end{matrix}\right\} \text{(lfn,arg)}$$

lfn

The logical file number, in an ASSOC systems command, that identifies the unit specifier (u) for the file to be checked.

arg

The symbolic integer variable into which the file system will return one of the following statis values:

000_{10}

File is available (READ or WRITE can be issued). The last request, if a READ or WRITE, was successful.

512_{10}

Request rejected; undefined LFN was used, or the file system is not available.

516_{10}

File is busy (READ or WRITE in progress). If ZFSTIN, then a READ is in progress and not yet complete. If ZFSTOT, the previous WRITE is not yet complete.

519_{10}

File is not open; last request was not successful. Issuance of another READ or WRITE will result in an error return.

A call to ZFSTIN or ZFSTOT made to a noncommunications file always results in a 000 (not busy) status return. Such a call allows a user to debug the application program by first using noncommunicsatons files, then write the program so that it can use either communications or noncommunications files.

The FORTRAN subroutine ZFSTIN, when called before issuing a READ request, checks for the availability of input. It prevents the loss of program control until data is available in a file system buffer. When ZFSTIN indicates that the file is not busy then a READ can be issued to move the data just read from the file system to the application program area.

The FORTRAN subroutine ZFSTOT, when called before issuing a WRITE request, checks to see if previous output is complete and the terminal is free to accept more data. When ZFSTOT indicates that the file is not busy then a WRITE can be issued to move data from the application program area to a file system buffer and schedule it to be written to the terminal.

ZFSTIN and ZFSTOT Programming Examples

The following are examples of (1) coding that causes the program to stall when input from a terminal is not completed before a second READ is issued, and (2) a call to subroutine ZFSTIN to check the file status before the second READ is issued. Note that in each case the first FORTRAN statement is OPEN.

Example 1:

```
      OPEN(UNIT=8)
      READ(8,100)IN
      READ(8,199)IN
100   FORMAT(12)
```

Example 2:

```
      OPEN(UNIT=8)
      READ(8,200)IN
 50   CALL ZFSTIN(8,ISTAT)
      IF(ISTAT .EQ. 0) GO TO 100
      IF(ISTAT .EQ. 512) GO TO 900
      IF(ISTAT .EQ. 519) GO TO 900
      GO TO 50
100   READ(8,200)IN
200   FORMAT(15)
900   WRITE(4,910)
910   FORMAT(ERROR FOUND)
```

Appendix D contains an example of a FORTRAN communications program.

SECTION 5

ASSEMBLY LANGUAGE COMMUNICATIONS USING THE FILE SYSTEM

This section discusses the use of file system macro calls in writing communications programs.

FILE SYSTEM CONSIDERATIONS

Aside from the use of macro calls, the user should be aware of other considerations in using the file system within a communications environment. These considerations are detailed in Section 2.

FILE-PROCESSING MACRO CALLS IN ASSEMBLY LANGUAGE APPLICATIONS

The following describe the use of the get file ($GTFIL), open file ($OPFIL), test file ($TIFIL and $TOFIL), and wait file ($WIFIL and $WOFIL) macro calls in assembly language communications processing with the file system.

Get File ($GTFIL) Macro Call Guidelines

The get file function reserves a file for processing and connects a file to a logical file number (LFN). The LFN is used in other file system calls ($OPFIL, $RDREC, $WRREC, etc.) to reference the file in question. Normally the get file function is involved via a GET command outside of program execution.

The arguments for the get file ($GTFIL) macro call in an assembly language communications program must have the values shown in Table 5-1.

Table 5-1. Arguments for Get File ($GTFIL) Macro Call

Argument	Argument Value
Pathname pointer	Must point to a pathname of a communications device (e.g., >SPD>TTY01)
Concurrency control	According to how the application uses the device (normally zero for exclusive use)
Remaining arguments	Zero

Open File ($OPFIL) Macro Call Guidelines

The open file function allocates buffer space (if required) and physically connects the device or terminal.

The open file macro call $OPFIL, when used in communications, must include the location of the file information block (FIB) which in turn must contain a valid program view item.

Table 5-2 indicates bit settings in the program view item for the $OPFIL macro call, such settings are dependent on the actions taken by the communications application program.

Test File ($TIFIL, $TOFIL) Macro Call Guidelines

Before the application issues a $RDREC or $RDBLK macro call, it can issue the test input file ($TIFIL) macro call to check whether input is available. Note that when the operator terminal is checked, the $TIFIL macro call always returns a not busy status.

Before the application issues a $WRREC or $WRBLK macro call, it can issue the test output file ($TOFIL) macro call to check whether the preceding output operation was completed.

Wait File ($WIFIL, $WOFIL) Macro Call Guidelines

The use of the wait file macro call will permit an application to wait for the completion of an outstanding read or write order. The wait file macro call can be used against a set of multiple terminals or devices. Test and wait file macro calls differ in terms of when control is returned to the calling routine. A test file call will return immediately with a busy or not busy status. An application can block the execution of lower level tasks with repeated test file calls to a busy file. Such problems can be avoided by issuing a wait macro call in lieu of successive test macro calls.

$WIFIL is used to wait for input from any device/terminal; $WOFIL to wait for completion of output to any device/terminal.

Table 5-2. Program View Bit Settings for $OPFIL Macro Call

Bit Number	Actions by Assembly Language Application Program	Set Bit(s) To
0	Will use read record ($RDREC) and write record ($WRREC) macro calls	0
	Will use read block ($RDBLK) and write block ($WRBLK) macro calls	1
1 (read bit)	Will read data from the device (see note 1)	1
	Will not read data from the device	0
2 (write bit)	Will write data to the device (see note 1)	1
	Will not write data to the device	0
3 through 12		0
13	As appropriate (see Table 2-1)	0 or 1
14		0
15	Synchronous/asynchronous indicator (see note 2)	0

Notes:
1. Bit value must be consistent with device type being used.
2. When application uses $RDBLK or $WRBLK macro calls, execution of the calls indicates asynchronous.

Device Dependent Macro Call Procedures

The following subsections describe the procedures for issuing device dependent file system macro calls.

Device Modes and Device Types

There are four basic processing modes for communications devices:

Input only (TTY or VIP data entry applications);
Output only (receive only printer application (ROP);
Bidirectional - either the device is opened for input or output, but not both applications (BSC 3780);
Interactive (TTY, VIP or BSC 3780 applications).

Macro Call Procedures for Data Entry Terminals

Table 5-3 shows the procedure for using file system macro calls in communications application involving data-entry terminals.

Table 5-3. Macro Call Procedures for Data Entry Terminals

Procedure Step	Action by Application Program	System Actions
1	Issue $GTFIL macro call (see Table 5-1)	Bit 2 program view
2	Issue &OPFIL macro call (see Table 5-2) with 1 set to 1, bit 2 set to 0.	Issues asynchronous connect, returns a normal status to the program.
3	Issue $WIFIL macro call to wait unil connect is complete and input is available. (With multiple devices, the $WIFIL macro call can be issued with a list of LFNs, effectively giving up control until input is available from one or more devices in the list.) Otherwise, if application is to do other processing (not giving up control), issue $TIFIL macro call.	Will return when a read has been satisfied. If connect is not complete return a busy status. If connect is complete, issue an asynchronous read and return a busy status until read is complete.
4	If not busy status is returned, issue $RDREC or $RDBLK macro call.	With read operation complete, move data from system buffer to application's buffer, issues another asynchronous read, and returns a normal status to the program.
5	If an error status is returned, exit from the procedure.	

Table 5-3 (cont). Macro Call Procedures for Data Entry Terminals

Procedure Step	Action by Application Program	System Actions
6	When read is successful, return to step 3 to request more data from the device.	
7	When application processing completed, issue $CLFIL macro call.	Issues a disconnect.

Macro Call Procedures for Output Only Terminals

Table 5-4 shows the procedure for using macro calls in communications applications involving output only terminals.

Table 5-4. Macro Call Procedures for Output Only Terminals

Procedure Step	Action by Application Program	System Actions
1	Issue $GTFIL macro call (see Table 5-1).	
2	Issue $OPFIL macro call (see Table 5-2) with bit 1 set to 0, bit 2 set to 1.	Issues a asynchronous connect, returns a normal status to the program.
3	Issue $WOFIL macro call to wait until connect is complete and output can be transmitted. (With multiple devices, the $WOFIL macro call can be issued with a list of LFNs, effectively giving up control until output can be sent to one or more of the devices in the list.) Otherwise, if the application is to do other processing (not give up control), issue a $TOFIL macro call.	Will return when output can be transmitted If connect is not complete return a busy status. If connect is complete return a not busy status if output can be transmitted.

Table 5-4 (cont). Macro Call Procedures For Output Only Terminals

Procedure Step	Action by Application Program	System Actions
4	If not busy status is returned, issue $WRREC or $WRBLK macro call.	Moves data from application buffer to system buffer. Issues asynchronous write and returns a normal status to the application.
5	If error status is returned, exit from the procedure.	
6	When write is successful, return to step 3 to transmit more data to the device.	Issues disconnect according to device type.

Macro Calls For a Single Interactive Terminal

Table 5-5 describes the procedures for using macro calls in communications applications involving only one interactive terminal which has been configured for synchronous input/output operation.

Figure 5-1 illustrates the procedure's flow.

Table 5-5. Macro Call Procedures for Single Interactive Terminal

Procedure Step	Action by Application Program	System Actions
1	Issue $GTFIL macro call (see Table 5-1)	
2	Issue $OPFIL macro call (see Table 5-2) with program view bit 1 set to 1, program view bit 2 set to 1.	
To read from the terminal followed by a write to the terminal:		
3	Issue $RDREC or $RDBLK macro call. (This effectively gives up control until the read is satisfied.) If error status returned, exit from the procedure.	Data is read directly into the application buffer.
4	Process the data just read.	
5	Issue $WRREC or $WRBLK. (This effectively gives up control until the write is complete.) If an error status is returned, exit from the procedure.	Data is written directly from the application buffer
6	If additional input is expected refer to step 3.	
7	When application processing is complete, issue $CLFIL macro call.	Issues a disconnect.

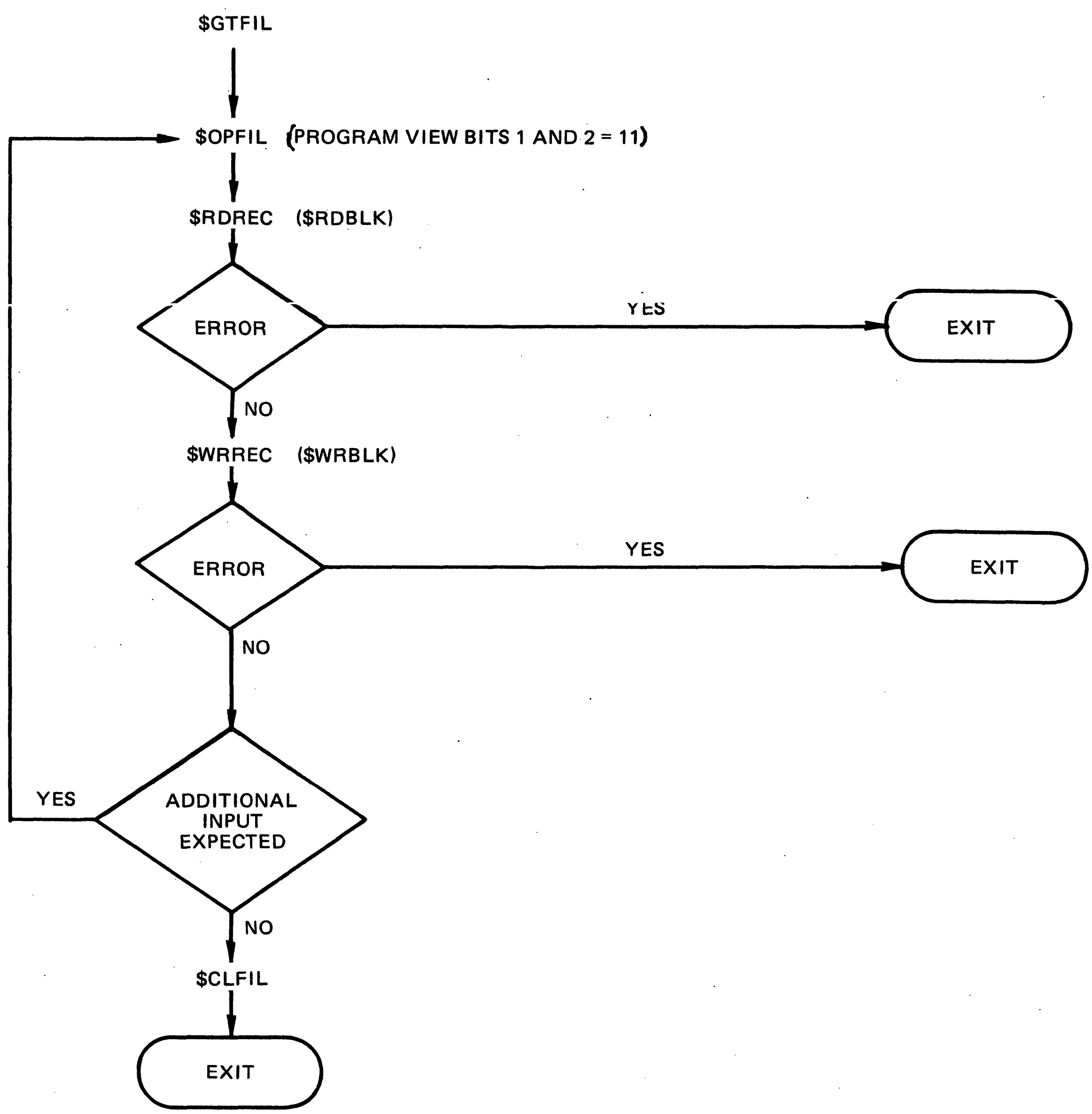


Figure 5-1. Simplified Program Logic for Single Interactive Terminal

Macro Call Procedures for Multiple Interactive Terminals

Table 5-6 describes the procedures for using macro calls in communications applications involving multiple terminals.

Figure 5-2 illustrates the procedure's flow.

Table 5-6. Macro Call Procedures for Multiple Terminals

Procedure Step	Action by Application Program	System Actions
1	Issue $GTFIL macro call to each terminal (see Table 5-1).	
2	Issue $OPFIL macro call to each terminal (see Table 5-2 with program view bit 1 set to 1, bit 2 set to 1.	Issues asynchronous connect, returns normal status to the program.
To read from a terminal followed by a write to a terminal:		
3	Issue $WIFIL macro call with a list of LFNs. (This will effectively give up control until input is available from one or more terminals in the list.)	Will return when a read is complete and data is available. Returns the LFN of the first terminal in the list for which data is available.
4	Issue $RDREC or $RDBLK macro call.	Moves data from system buffer to application's buffer, issues another asynchronous read, and returns a normal status to the program.
5	If an error status is returned, process the error.	
6	Process the data just read.	
7	Issue $WRREC or $WRBLK macro can. (This will give up control unitl output can be sent to terminal.)	Waits until output can be sent; moves data from the application's buffer to system bu fer and issues an asynchronous write.
8	If additional input is expected from any terminal see step 3.	

Table 5-6 (cont.) Macro Call Procedures for Multiple Terminals

Procedure Step	Action by Application Program	System Actions
9	When application processing is complete, issue $CLFIL call.	Issues disconnect.

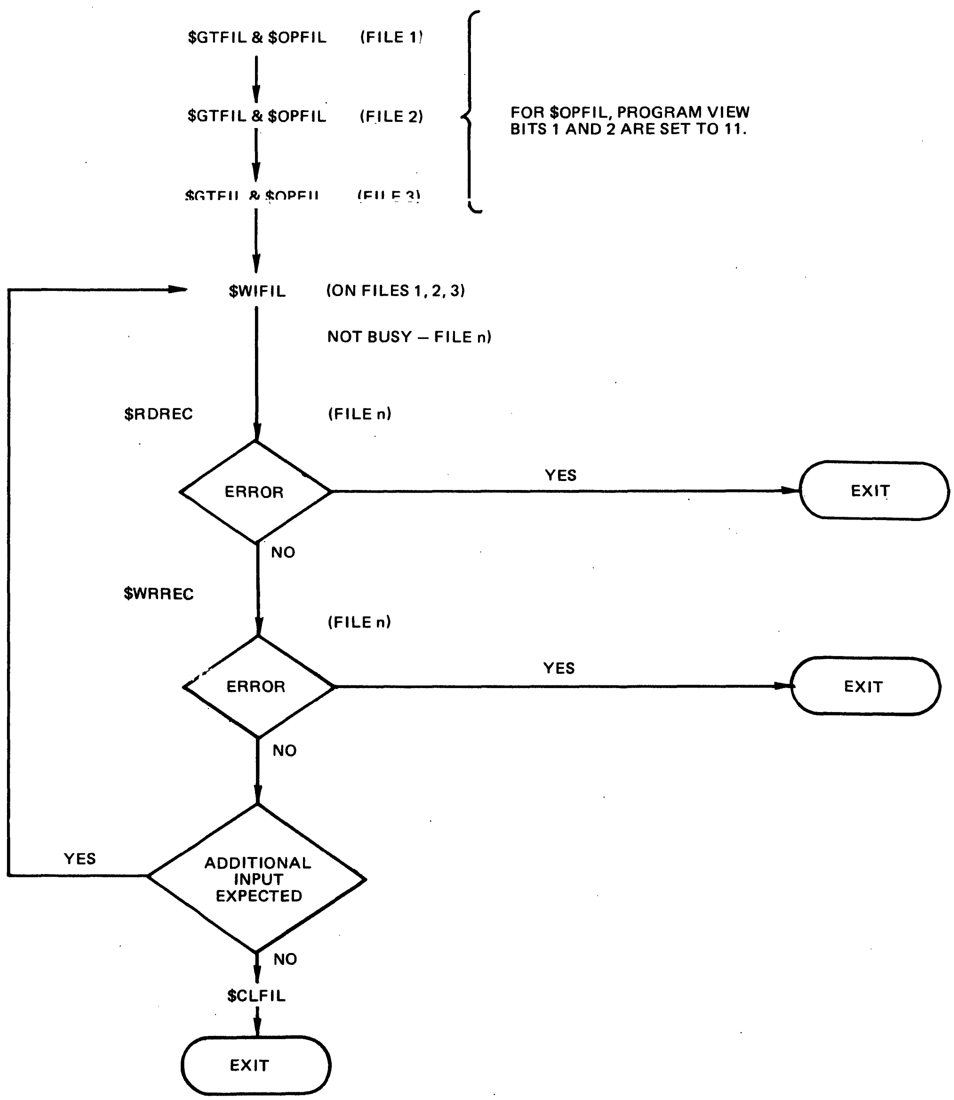


Figure 5-2. Simplified Program Logic for Multiple Interactive Terminals

Binary Synchronous Communication (BSC)

Binary synchronous communication (BSC), operating in 2780 or 3780 mode, permits a program to transmit data over communications lines from one Level 6 to another Level 6, or a Level 66 system, or to a non-Honeywell host system.

BSC DATA TRANSMISSION CONVENTIONS

BSC Data Codes

Data can be in alphanumeric ASCII, alphanumeric EBCDIC, or binary format. In communication between Level 6 and a remote host, each system must use the same code set (either ASCII or EBCDIC). When EBCDIC is used, the application programs must know whether transmission is nontransparent or transparent (i.e., BSC control characters are interpreted as data).

BSC Data Transmission Modes

There are two BSC transmission modes: basic and advanced.

In basic transmission mode there is no control byte. The absence of a control byte limits the functionality of the protocol (e.g., an application cannot send or receive two message blocks or cannot initiate a reverse interrupt (RVI) sequence).

In advanced transmission mode there is a control byte which is the first byte in the program's input or output buffer. The control byte is used to control the transmission of data and to convey information concerning the receipt of data. With the control byte the application has almost complete control (subject to limitations imposed by the protocol) over the transmission and reception of data to and from a remote host. (The control byte formats are detailed in Section 10).

BSC 2780 and BSC 3780

BSC 2780 is a subset of BSC 3780. Technical differences between the two protocols can be summarized as a set of extensions to the 2780 protocol which are as follows:

- o The ability to receive a conversational reply without a preliminary bid sequence.

- o The ability to receive and transmit selected BSC control characters.

From a user's point of view the differences between the two protocols can be summarized as follows:

BSC 2780

- o Specified at system building time by the BSC device directive.
- o Operates only in advanced mode.
- o The file system supports bidirectional usage of BSC 2780 communications line. A CLOSE/OPEN sequence must be initiated prior to the reversal of the communication line.

BSC 3780

- o Specified at system building time by the XBSC directive.
- o Operates only in advanced mode.
- o The file system supports interactive usage of the BSC 3780 communications line. To terminate a transmission the application must initiate an EOT sequence by setting the appropriate bit within the control byte. An ETX message transmission sequence can also be terminated if the other application sends a conversational reply. The receipt of conversational reply is indicated by a bit setting within the transmit control byte. The receipt of a conversational reply forces the application to issue a file system read order to receive the conversational response. The termination of a read sequence is indicated by a EOF return code (021F) and by the EOT bit being set in the receive control byte. (Note that the terms EOF (end of file) and EOT (end of transmission) are synonymous).

Macro Call Procedures for BSC 2780 in Basic Transmission Mode

The following conditions apply in the use of the file system in binary synchronous communications in basic data transmission mode:

- o An application cannot send an RVI (reverse interupt) control character through the file system.
- o BSC devices in basic transmission mode can operate only in single-buffer mode (see Section 10).

- o An application can send only the ETB (end of transmission block) control character, not the ETX (end of text) character.

- o An application can send data in either transparent or nontransparent mode.

- o An application can send EOT (end of transmission) control characters only after it has issued a $CLFIL macro call.

- o BSC operation assumes that the detab option is set off.

Table 5-7 shows the procedure for using macro calls in applications that use BSC in basic data transmission mode.

Figure 5-3 illustrates a simplified program logic for these procedures.

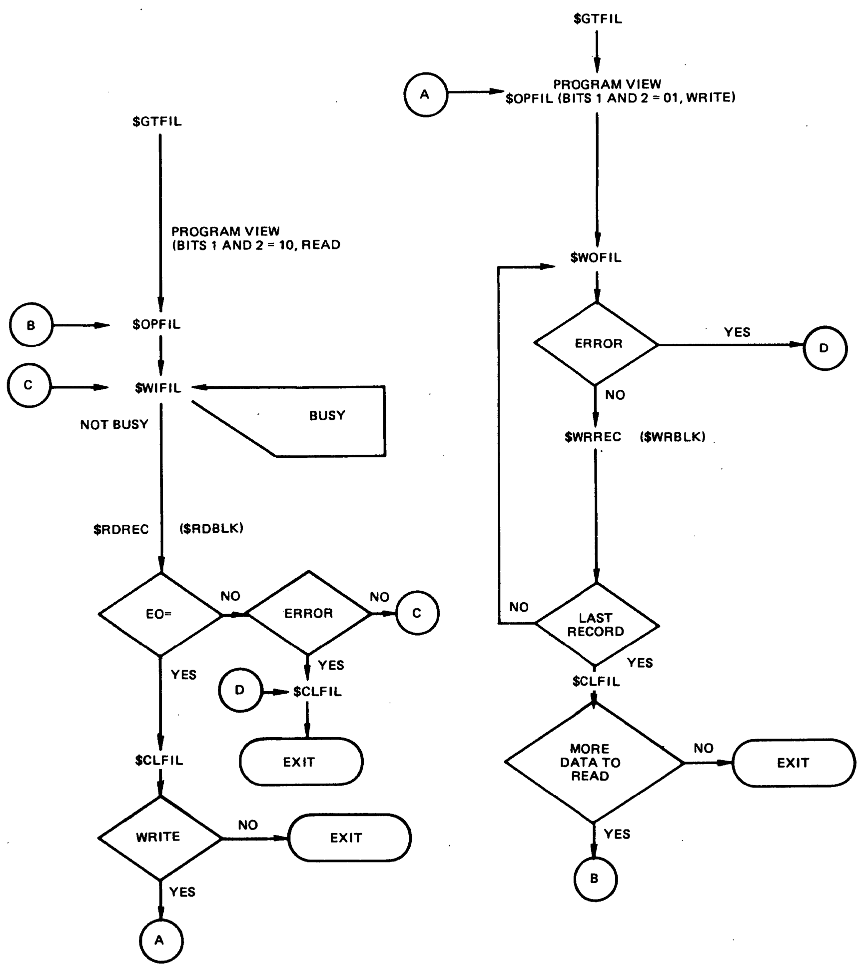


Figure 5-3. Simplified Program Logic for BSC 2780 in Basic Transmission Mode

Table 5-7. Macro Call Procedures for BSC 2780 in Basic Transmission Mode

Procedure Step	Action by Application Program	System Actions
1	Before a file is first opened issue $GTFIL macro call (see Table 5-1).	
	To read data from a BSC line:	
2	Issue $OPFIL macro call (see Table 5-2, with program view bit 1 set to 1, program view bit 2 set to 0.	Issue asynchronous connect; returns a status to the program.
3	Issue $WIFIL macro call to wait until connection is complete and input available. If application is to do other procesisng (not give up control) issue $TIFIL macro call.	If connect is not complete, $TIFIL returns a busy status or, issues an asynchronous read and returns a busy status until read is complete.
4	Issue $RDREC or $RDBLK macro call. If error status other than EOF (end of file) is returned, exit from the procedure. (An EOF status indicates that EOT (end of transmission) control character was received, indicating sender completed its transmission.	Moves data from system buffer to the application's buffer, and again issues an asynchronous read. If there are no errors, returns a normal status.
5	Test for EOF return status. If status is normal, do other processing and return to step 3 if more data expected.	
6	If application is to send data, issue $CLFIL macro call and continue with step 7. If application is not to send or receive data, issue $CLFIL macro call and continue with other processing.	

Table 5-7 (cont). Macro Call Procedures for BSC 2780 in Basic Transmission Mode

Procedure Step	Action by Application Program	System Actions
To write data to a BSC line:		
7	Issue $OPFIL macro call (see Table 5-2) with program view bit 1 set to 0, program view bit 2 set to 1.	
8	Issue $TOFIL macro call to test that connection is complete. If file was already opened, and closed without a phone hangup, the line is still connected; $TOFIL is not required.	
9	Issue $WRREC or $WRBLK macro call. If an error status is returned, exit from the procedure.	If no writes are pending, moves data from application's buffer to system buffer, issues asynchronous write to the BSC line, and returns a normal status.
10	Issue $WOFIL macro call to wait for completion of previously scheduled output. Issue $TOFIL to continue other processing while write is in progress.	If the write is not complete $TOFIL returns a busy status.
11	Can now issue another $WRREC or $WRBLK macro call, or issue a $CLFIL macro call if the preceding write macro call was the last one, or if $CLFIL macro call was issued, and more data is to be read from the line, return to step 2.	When $CLFIL macro call is issued, the system: sends an EOT (end of transmission) character if the BSC is in send or receive mode for that line. Sends nothing if the BSC line is idle.

Macro Call Procedures for BSC 2780 in Advanced Data Transmission Mode

In the BSC advanced data transmission mode, the first byte of the application program's input or output buffer is a control byte that controls or supplies information about read/write operations. This byte can indicate, for example, whether data is to be transferred in transparent or nontransparent mode, or whether an ETB (end of transmission block) or ETX (end of text) control character is to be sent or received. (Section 10 details the usage of BSC control characters).

The following condition applies in using the file system in 2780 binary synchronous communications in advanced data transmission mode:

- It is not necessary to send EOT control characters through the control byte since the user must close the file in output mode before attempting to read. Closing the file forces the BSC, if not in idle mode, to send an EOT control character.

Table 5-8 shows the procedure for using macro calls in applications that use BSC lines in 2780 advanced data transmission mode.

Figure 5-4 illustrates the program logic for these procedures.

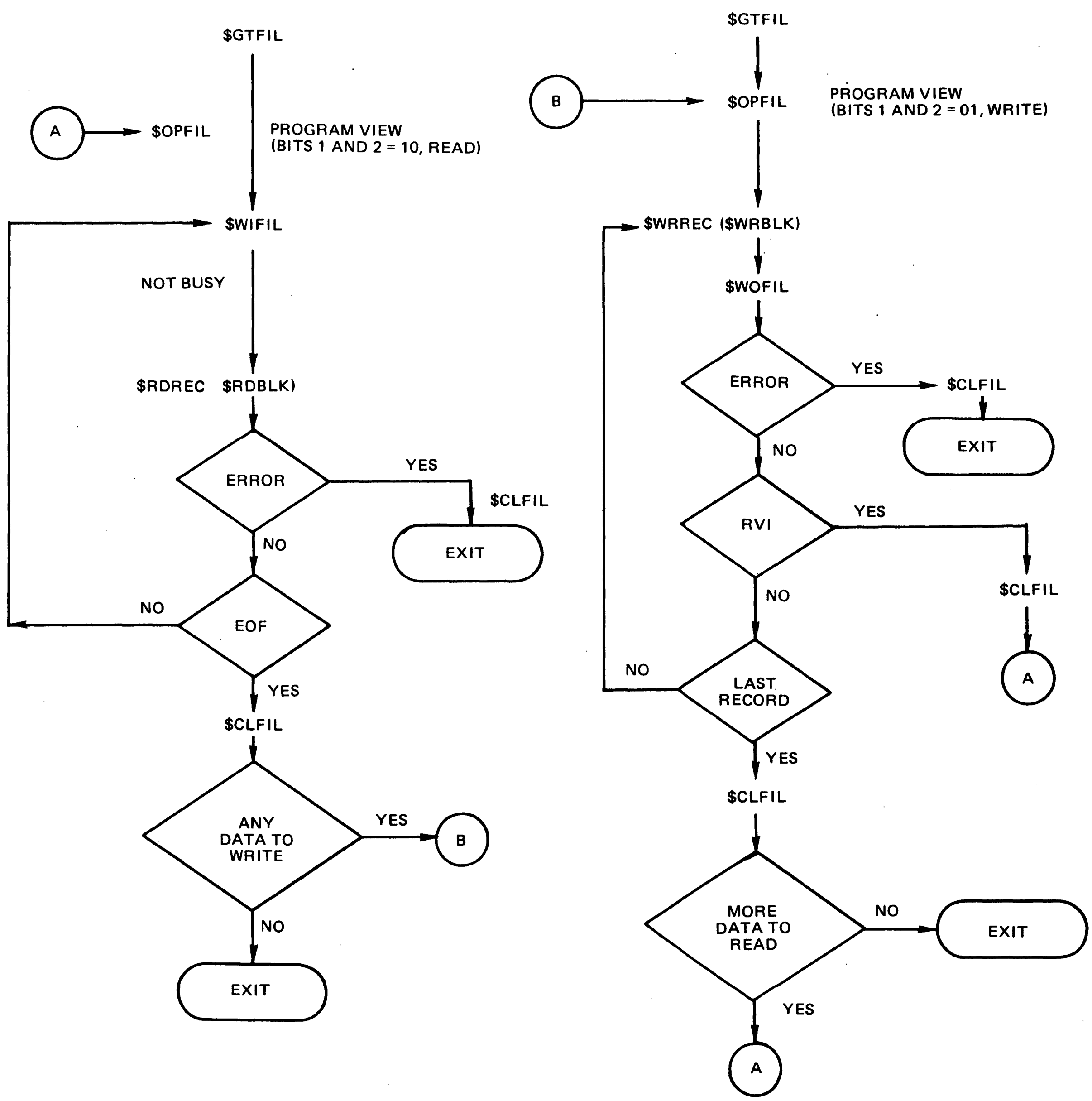


Figure 5-4. Simplified Program Logic for 2780 BSC in Advanced Transmission Mode

Table 5-8. Macro Call Procedures for BSC 2780 in Advanced Transmission Mode

Procedure Step	Action by Application Program	System Actions
1	Before the file is first opened issue $GTFIL macro call.	
	To read data from a BSC 2780 line:	
2	Issue $OPFIL macro call (see Table 5-2) with program view bit 1 set to 1, program view bit 2 set to 0.	Issues an asynchronous connect; returns a normal status to the program.
3	Issue $WIFIL macro call to wait unitl connect is complete and input is available. If application is to do other processing (not give up control), issue $TIFIL macro call.	If connect is not complete, returns a busy status, $TIFIL issues an asynchronous read, and returns a busy status unitl the the read is complete.
4	Issue $RDREC or $RDBLK macro call. If error status other than EOF (end-of-file) is returned, exit from the procedure. (An EOF status indicates that an EOT (end of transmission) control character was received, denoting that the sender has completed its transmission.)	Moves the data from the system buffer to the application's buffer, and again issues an asynchronous read. If there are no error, returns a normal status.
5	Test for EOF return status. If return status is normal, an application can check for ETB or ETX control characters, or for transparent or non-transparent processing, and return to step 3.	
6	When EOF or EOT status is returned, and more data is expected, return to step 3.	

Table 5-8 (cont). Macro Call Procedures for BSC 2780 in Advanced Transmission Mode

Procedure Step	Action by application Program	System Actions
7	If application is to send data, issue a $CLFIL macro call and continue with step 8. If application is not to send or receive data, issue $CLFIL macro call and continue with other processing.	
	To write data to a BSC line:	
8	Issue $OPFIL macro call (see Table 5-2) with program view bit 1 set to 0, program view bit 2 to set to 1.	
9	Issue $WRREC or $WRBLK macro call. Application can set control byte to control transmission (send ETB or ETX concontrol characters, or send in normal or transparent EBCDIC mode).	If no writes are pending, moves the data from the application's buffer, issues an asynchronous write to the BSC line, and returns a normal status.
10	Issue $WOFIL macro call to wait for completion of previously scheduled output. Issue $TOFIL to continue other processing while write is in progress.	If the write is not complete $TOFIL returns a busy status.
11	If an error status is returned, close the file and exit from the procedure.	
12	Can now test for RVI-received bit in the control byte of the record that was just sent. If the bit is set on, can either: a. Close the file and return to step 2, or b. Ignore the RVI condition and continue to write.	

Table 5-8 (cont). Macro Call Procedures for 2780 BSC in Advanced Transmission Mode

Procedure Step	Action by Application Program	System Actions
13	After the write is complete, the application can: If there is more data to be written, issue another $WRREC or WRBLK by returning to step 9, or If more data is expected, issue a $CLFIL macro call and return to step 2, or Simply issue a $CLFIL macro call and exit the procedure.	

Macro Call Procedures for BSC 3780 in Advanced Data Transmission Mode

The first byte of the application program's input or output buffer is a control byte. The control byte controls or supplies information about read/write operations.

The following conventions apply when using the file system with 3780 binary synchronous communication in advanced data transmission mode:

- o The receipt of an optional conversational reply is indicated by a bit setting in the transmit control byte. (This can occur if the application has transmitted the last (ETX) block of a message). The application must issue a read macro call in order to receive the conversational response.

- o The termination of a transmit sequence is signaled (via control byte) by the transmission of an EOT control character following the last block if a message. Once this has been done, a read macro call will be needed to receive transmissions from the remote system. (It is not necessary to close and reopen the file to turn the line around.)

- o The termination of a receive sequence is indicated by the receipt of an EOF return status or an EOT status in the receive control byte. A transmission sequence can be reinitiated by issuing another write macro call. (It is not necessary to close and reopen the file to turn the line around).

- A line turnaround (receipt of an EOT) is indicated by an 021F EOF return code (and the setting of the EOT bit in the receive control byte). At this point the application can use the line for data transmission by issuing another write request. It is also possible to receive an EOF/EOT status, which indicates the abnormal termination of transmit/receive sequence. (This can occur for a variety of reasons, most notably hardware problems.) Such an occurrence is also indicated by an 021F return code. The application can differentiate between these end-of-file conditions by considering when the EOF status was re-received. For example, two applications agree that the last record of a message transmission is demarked by an ETX control character. If the transmission is terminated by the receipt of an EOT and the last record of the transmission was not marked with an ETX control character, the application can assume that the transmitter aborted the transmission sequence. If such a condition is detected, the application must close the line by issuing a close file macro call (all other file system requests will be rejected.

Table 5-9 shows the procedure for using macro calls that use BSC lines in 3780 advanced data transmission mode.

Figure 5-5 illustrates the program logic for these procedures.

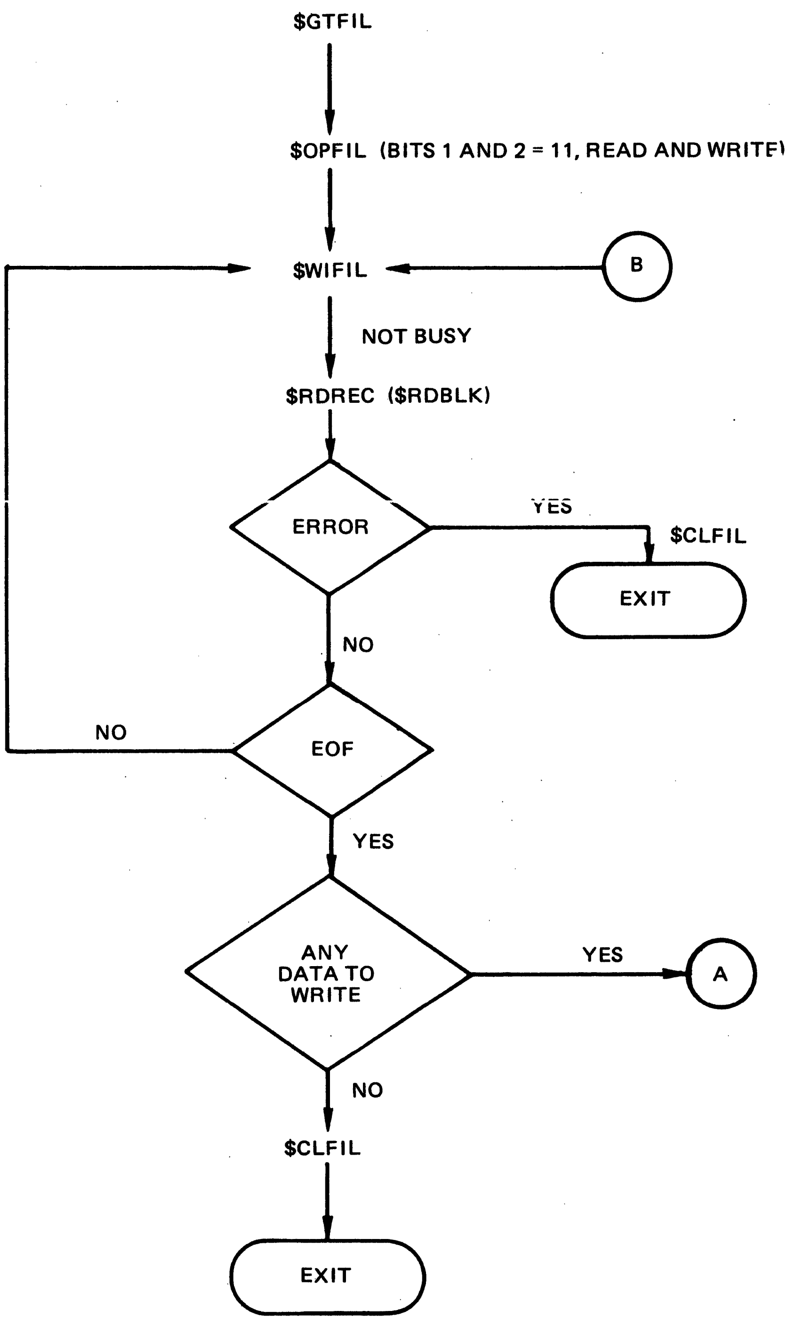


Figure 5-5. Simplified Program Logic for BSC 3780 in Advanced Transmission Mode

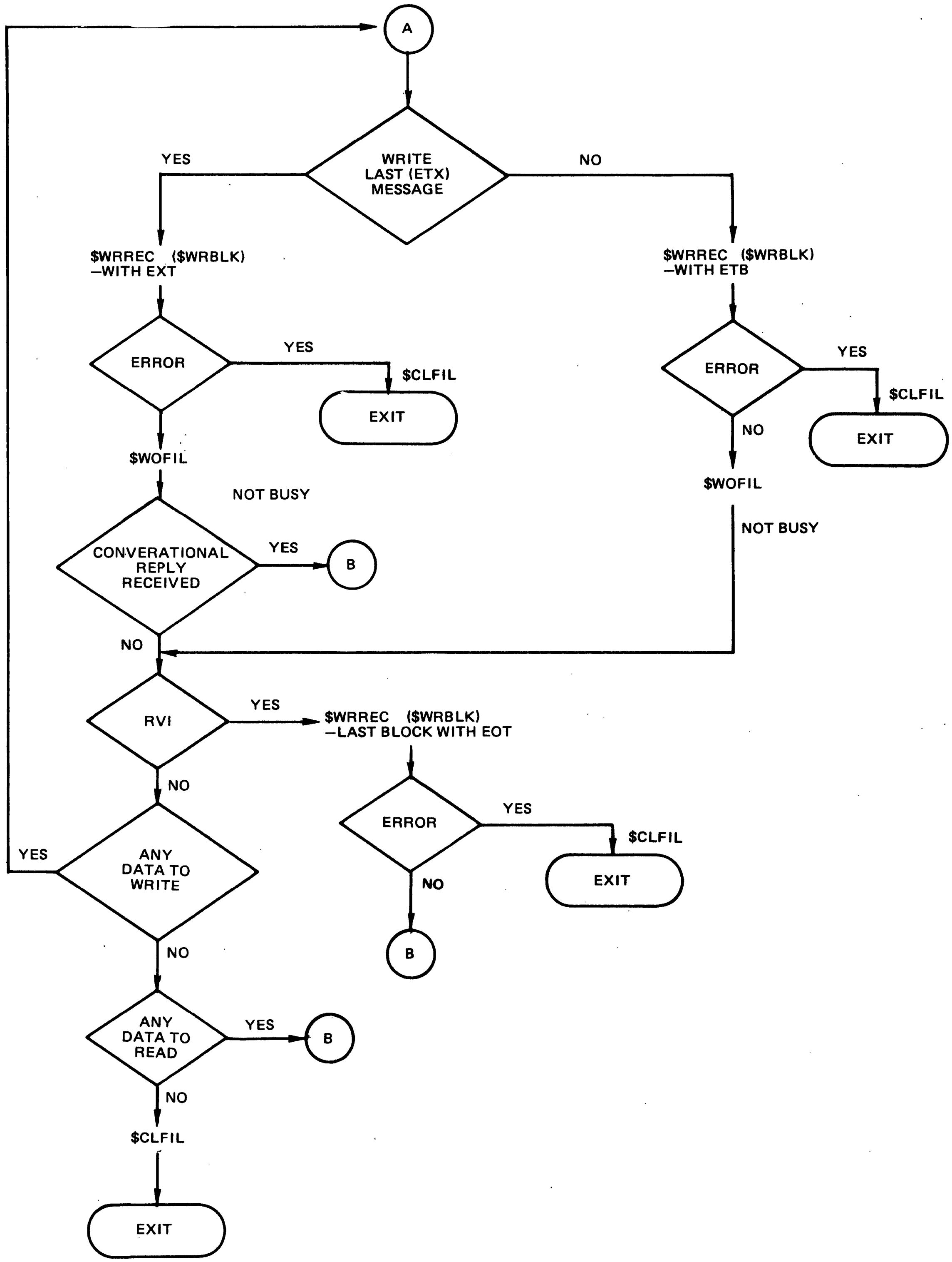


Figure 5-5 (cont). Simplified Program Logic for 3780 BSC in Advanced Transmission Mode

Table 5-9. Macro Call Procedures for BSC 3780 in Advanced Transmission Mode

Procedure Step	Action by Application Program	System Action
1	Before the file is first opened, issue $GTFIL macro call (see Table 5-1).	
	To read data from a BSC line:	
2	Issue $OPFIL macro call (see Table 5-2) with program view bit 1 set to 1, program view bit 2 set to 1.	Issues an asynchronous connect; returns a normal status to the program.
3	Issue $WIFIL macro call to wait until connect is complete and input is available. If application is to do other processing (not give up control), issue $TIFIL macro call.	If connect is not complete, $TIFIL returns a busy status. If connect is complete, issues an asynchronous read, and returns a busy status until the read is complete.
4	Issue $RDREC or $RDBLK macro call. If error status other than EOF (end-of-file) is returned, exit from the procedure. (An EOF status indicates that an EOT (end of transmission) control character was received, denoting that the sender has completed its transmission.	Moves the data from the system buffer to application's buffer, and again issues an asynchronous read. If there are no errors, returns a normal status.
5	Test for EOF return status. If return status is normal, the application can check for ETB or ETX control characters, or for transparent or non-transparent processing, and return to step 3.	
6	If the application has data to send continue with step 8.	
7	If the applicastion has no data to send, issue a $CLFIL macro call and continue with other processing.	

Table 5-8 (cont). Macro Call Procedures for BSC 3780 in Advanced Transmission Mode

Procedure Step	Action by Application Program	System Action
To write data to a BSC line:		
8	If the application wishes to send the last (ETX) block of message, continue with step 16.	
9	Issue $WRREC or $WRBLK macro call. Application can set control byte to control transmission of an ETB control character. If an error status is returned close the file and exit from the procedure.	If no writes, moves the data from the application's buffer to the system buffer, issues an asynchronous write to the BSC line, and returns a normal status.
10	If application is to do other processing (not give up control) issue $TOFIL. Else, issue $WOFIL macro call to give up control of the central processor until the write is completed.	If the write is not complete, returns a busy status. Returns a not busy status when the write is complete.
11	Can now test the transmit control byte for the receipt of a conversational reply. If the bit is set on, initiate another read sequence by returning to step 3.	
12	Can now test for RVI-received bit in the control byte of the record that was just sent. If the bit is set on, can either: a. Close the file and initiate another read sequence by returning to step 3, or b. Ignore the RVI condition and continue to write.	

Table 5-9 (cont). Macro Call Procedures for BSC 3780 in Advanced Transmission Mode

Procedure Step	Action by Application Program	System Action
13	If there is any more data to transmit, continue with step 8.	
14	If data is expected from the remote host, initiate another read sequence by returning to step 3.	
15	Transmission and reception sequences are complete. Issue a $CLFIL macro call and exit from the procedure.	
16	Issue $WRREC or $WRBLK macro call. Application can set control byte to control transmission of an ETX control character. If an error status is returned close the file and exit from the procedure.	Moves the data from the application's buffer to the system buffer, issues an asynchronous write to the BSC line, and returns a normal status.
17	If application is to do other processing (not give up control) issue $TOFIL. Else issue $WOFIL macro call to give up control of the central processor until the write is completed.	If the write is not complete, returns a busy status. Returns a not busy status when the write is completed.
18	Continue with common processing of transmit sequence by continuing with step 12.	

SECTION 6

ASSEMBLY LANGUAGE COMMUNICATIONS USING PHYSICAL INPUT/OUTPUT

The physical input/output (I/O) interface permits more direct user control over communications processing than does the file system.

Used only with assembly language programs, the physical I/O interface enables communications applications to:

- o Call appropriate line protocol handlers (LPH) more directly through the communications subsystem rather than through the file system.

- o Control the data structure, specifically the input/output request block (IORB), that directly controls device operations and/or characteristics. (See "Data Structures" below for description of the IORB.)

COMMUNICATIONS SUBSYSTEM CONVENTIONS

The following conventions apply to use of the communications subsystem:

- o The I/O request block (IORB) is the standard control structure used by an LPH of the communications subsystem.

- o Use the request I/O ($RQIO) macro call in the application program to request an I/O transfer.

- o The B4 register contains the address of the IORB supplied by the application program; the IORB contains the logical resource number (LRN) of the device to be used.

- o The I/O-specific words of the IORB (I_CT2 through I_DVS) are not modified by the line protocol handler.

o The communications subsystem maps the hardware return status into the software status word I_ST of the application's IORB before the line protocol handler gives up control.

Table 6-1 lists the return error status codes that indicate logical result of an I/O request.

USING PHYSICAL I/O

Two fields within the IORB specify the operation to be performed.

1. The function code (Table 6-4), indicated by bits C through F of I_CT2 in the IORB (see Table 6-2), specifies the particular operation.

2. The I_DVS item in the IORB, used with the function code, specializes the input/output order.

For example, in TTY processing, the user can specify a write request (function code 1), with or without a carriage return at end-of-message, as indicated by the C-bit of the I_DVS (see Table 7-3).

To request execution of an I/O operation, the application, with the $RQIO macro call, must transfer control to the physical I/O interface. At the time of the request the $B4 register must contain the address of the IORB being requested. The $RQIO macro routine executes the I/O operations, then returns to the requesting application.

The IORB may define either synchronous or asynchronous control. When the IORB specifies synchronous I/O (W (wait) bit in I_CT1 reset), return to the calling application is delayed by the Monitor until the I/O operation is complete. On return, the return status field of the IORB, and the $R1 register, will contain one of the status codes shown in Table 6-1.

When the IORB specifies asynchronous I/O (W (wait) bit set in I_CT1), control returns immediately without waiting for I/O completion, and the instruction at the return point is executed as soon as the system queues the IORB. To obtain the return status (in $R1 register), when using asynchronous I/O, the application should issue a $WAIT macro call.

At completion of the I/O operation, the application should first check the $R1 register to see that the I/O request was successful. Any error will be defined there. Hardware errors will be indicated in the IORB software status word I_ST (see Table 6-3).

Residual range, indicated in the IORB, shows how much of the requested data was transferred. With a write request, the residual range value is the number of bytes remaining to be transmitted. With a read request, the residual range value is the number of bytes remaining to be received. The residual range value in I_RSR of the IORB is meaningful only when the A-bit in the I_ST item (Table 6-3) of the IORB has been set on.

Table 6-1. Return Status Error Codes for Logical Result of I/O Request

Code Number (Hexadecimal)	Meaning
0	No error, operation complete
1	Request block already busy (T=1)
2	Invalid LRN
3	Illegal wait
4	Invalid arguments
5	Device not ready
6	Device time-out
7	Hardware error, check IORB status word (see Table 6-3)
8	Device disabled[a]
9	File mark encountered
A	Controller unavailable
B	Device unavailable[b]
C	Inconsistent request[c]
F	EOT received (for BSC 3780 only)

[a] This status is returned on an I/O request when the application has disabled the logical resource, and for a communications resource, when the result of either a connect or disconnect for this logical resource is pending.

[b] When these codes are found in I_CT1 (IORB), or in $R1 on a resume after wait, look at I_ST (IORB) to identify the specific error. The status B is returned with every read or write IORB that has been aborted by a disconnect request with queue abort.

[c] This status indicates illogical device requests: read or write before connect, duplicate connect or disconnect requests; write after disconnect.

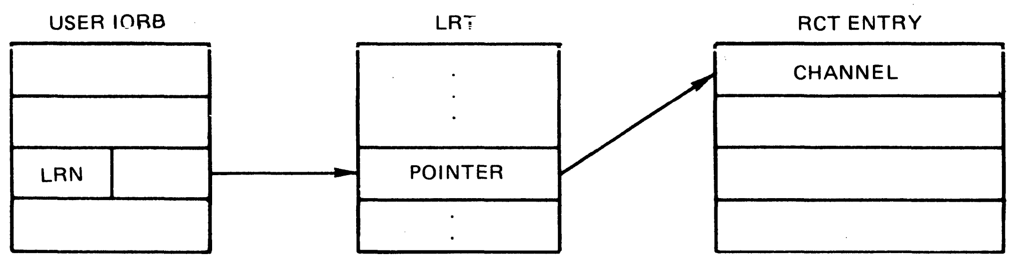
DATA STRUCTURES

Two data structures control the interactions among an application program, its line protocol handlers, and the devices it uses: (1) the input/output request block (IORB), and (2) the resource control table (RCT). The IORB is the interface between the application and line protocol handler; the RCT is the interface between the line protocol handler and its devices.

This section describes the input/output request block (IORB) in general. Later sections describe device-specific fields in the IORB for the TTY, VIP, PVE, and BSC line protocol handlers.

Resource Control Table (RCT)

The device's resource control table (RCT) contains a channel number and level entry, whose values were initially defined at system building. The logical resource number (LRN) supplied by the application in the IORB serves as an index into a system logical resource table (LRT), which in turn contains a pointer to the RCT entry defining the device, as illustrated below.



Thus, with the logical resource number, a line protocol handler can indirectly access the RCT entry that defines the specific device that the application is to use.

Appendix C describes the resource control table (RCT).

Input/Output Request Block (IORB)

The IORB is the standard means for requesting a physical I/O service. Generated by the input/output request block macro call ($IORB), the IORB contains all the information that an application requesting an I/O service must specify to define the operation to be performed. In addition, the IORB includes the following:

- Logical resource number (LRN) that identifies the I/O device being addressed.
- Location and size of the buffer to be used for physical I/O transfers.
- Information returned by the line protocol handler to the application, concerning results of the I/O request.

Figure 6-1 shows the format of the IORB. Table 6-2 defines the separate entries in the IORB. Later sections in the manual describe the device-specific word (I_DVS) and software status word I_ST for each line protocol handler.

NOTES: 1. The IORB as described here is as it appears for short address format (SAF) central processors, namely with one-word items. For long address format (LAF) processors, the same structure would have two-word entries for all pointers.

2. The labels (I_CT1, I_ADR, etc.) used in the IORB are only for easier presentation. The labels cannot be used for programming purposes.

3. The asterisk (*) in the formulas in the "Item" column of Table 6-2 is a multiplication sign.

4. The shaded fields in Figure 6-1 are for system use only. The field I_FCS is used only by the VIP and PVE line protocol handlers. Fields not shaded must be initialized by the application requesting the I/O operation.

When the IORB is used with a $RQIO macro call, the device named in the IORB should have been initially reserved. The logical resource number (LRN) required by the IORB can be obtained by issuing a get file information ($GIFIL) macro call. See the description of the request I/O ($RQIO) macro call in the System Service Macro Calls manual for details.

Offset	Label	0 1 2 3 4 5 6 7	8	9	A	B	C	D	E	F
–$AF –1	I_RRB I_SEM	REQUEST BLOCK POINTER/SEMAPHORE NAME								
0		RESERVED FOR SYSTEM USE AS A POINTER								
$AF	I_CT1	RETURN STATUS		W	U	S	0	R	D	1
1+$AF	I_CT2	LRN	0	B	P	0	FUNCTION			
2+$AF	I_ADR	BUFFER ADDRESS								
2+2*$AF	I_RNG	RANGE								
3+2*$AF	I_DVS	DEVICE SPECIFIC WORD								
4+2*$AF	I_RSR	RESIDUAL RANGE								
5+2*$AF	I_ST	STATUS WORD								
6+2*$AF	I_FCS	FUNCTION CODE 1 (VIP AND PVE)	FUNCTION CODE 2 (VIP AND PVE)							

Figure 6-1. Communications Input/Output Request Block (IORB)

Table 6-2. Contents of Communications Input/Output Request Block (IORB)

Item	Label	(Bits)	Contents
-$AF -1	I_RRB/ I_SEM	0 through 15 (SAF) 0 through 31 (LAF)	Depending on the condition of the S- or R-bits of I_CT1, this word contains a request block pointer (R-bit on), or a semaphore name (S-bit on). Set by user; used by system at termination of request.
0		0 through 15 0 through 31	Reserved for system use; one-word pointer (SAF); two-word pointer (LAF).
$AF	I_CT1	0 through 7	Return status. (See Table 6-1).
		8 (T)	This bit is set (on) while the request using this block is executing; it is reset when the request terminates. The system controls this bit; user should not change it.
		9 (W)	Wait bit - set if the requesting task is not to be suspended pending the completion of the request that uses this block.
		A (U)	User bit - user may or may not use this bit; system does not change it.
		B (S)	Release semaphore indicator. Values: 0=No release, 1=Release, on time-out, of item named in named in I_RRB.
		C	Must be zero.
		D (R)	Return request block indicator. Values: 0=No dispatch, 1=Dispatch of request block named in I_RRB, after timeout of this request. System executes $RQTSK, using I_RRB upon task termination.
		E (D)	Delete I/O request block. Values: 0=No delete, 1=Return memory to the pool where IORB is the first entry of its memory block.

Table 6-2 (cont). Contents of Communications Input/Output Request Block (IORB)

Item	Label	(Bits)	Description
$AF (cont)	I_CT1 (cont)	F	I/O bit - must be set.
1+$AF	I_CT2	0 through 7	Logical resource number (LRN); identifies device to be used.
		8	Reserved for later use.
		9 (B)	Byte index; 0=buffer begins in leftmost byte of word, 1=buffer begins in rightmost byte.
		A (P)	Reserved for system use.
		B	Reserved for later use.
		C through F	Function code. See Table 6-4.
2+$AF	I_ADR	0 through 15	Buffer address; SAF mode, 1-word pointer.
		0 through 31	Buffer address, LAF mode; 2-word pointer.
2+2*$AF	I_RNG	0 through 15	Range - number of bytes to be transferred.
3+2*$AF	I_DVS	0 through 15	Device-specific information.
4+2*$AF	I_RSR	0 through 15	Residual range. Indicates the number of bytes <u>not</u> transferred. Filled in by the system on completion of the order.
5+2*$AF	I_ST	0 through 15	Status word. Reflects the mapping of the hardware status into software status format. See Table 6-3.
6+2*$AF	I_FCS	0 through 7 8 through 15	Function code 1 (VIP and PVE only). Function code 2 (VIP and PVE only).

IORB Software Status Word (I_ST)

The line protocol handler maps into the IORB software status word I_ST (see Table 6-3) the return status of the hardware, obtained from the device status field R_STTS of the resource control table (RCT). (Appendix C describes the resource control table.)

The bit settings in the software status word I_ST indicate to the application the status of the hardware, as shown in Table 6-3.

The meanings of bit settings in the software status word I_ST for specific devices are shown in tables in later sections that describe the line protocol handlers for those devices.

Table 6-3. Software (I_ST) Status Codes

Bit in IORB's I_ST	Meaning When Bit Set On
0	-
1	VIP, PVE read error
2	Data service rate error
3	Lost line bid; RVI received (BSC only)
4	Communication control block service error
5	No stop bit on character input (TTY only); conversational reply received (BSC 3780 only)
6	Long record
7	For BSC: 0=ITB/ETB received; 1=ETX received For VIP and PVE: poll failure
8	For VIP and PVE: NAK limit reached
9	For VIP and PVE: Checksum or parity error limit reached
A	Nonzero residual range
B	Phone disconnect
C	BSC only: End-of-transmission (EOT) received

Table 6-3 (cont). Software (I_ST) Status Codes

Bit in IORB's I_ST	Meaning When Bit Set On
D	For VIP: page overflow For BSC: transparent message received
E	For VIP: busy or not available For BSC: NAK limit reached
F	Nonexistent resource; bus parity error; fatal uncorrectable memory error

COMMUNICATIONS FUNCTION CODES

All line protocol handlers perform similar functions for the devices and applications they service. These functions are performed by the line protocol handler's request and interrupt processing codes.

An application can request specific functions by providing a function code in the IORB supplied when it requests I/O service. The application uses the last four bits of its IORB's I_CT2 entry (see Figure 6-1) to enter the function code for the functions summarized in Table 6-4.

The connect and disconnect functions may be used with noncommunications devices (processed as no-ops) for program compatibility; i.e., no matter how connected to the Level 6 system, all TTY devices and noninteractive (e.g., card reader and printer) devices can be controlled by the same application program. This is useful for program development and test purposes.

Table 6-4. Communications LPH Function Codes

Function Code in IORB	Communications Function
0	Wait online
1	Write
2	Read
A	Connect
B	Disconnect

Wait Online Function (Code 0)

The wait online function, is used to synchronize task operation with device availability, and allows a caller to wait until a device becomes ready for use, or until a specific time interval has passed before using it.

When an LPH receives a service request from a task using the wait online function code in the IORB that is supplied (0000 in the last four bits of I_CT2), and the device is not ready, the driver sets a timer for 5 minutes and suspends. When the LPH is reactivated, either by a ready interrupt from the device or by a time-out, it deactivates the timer, checks the device-ready bit in the hardware status word and places a 0 or 6 value in the return status field of the IORB depending on the condition of that bit. See the return status codes for the $RQIO (request I/O) macro call; the rightmost hexadecimal character is placed in the return status field. See Table 6-1.

The wait online function should not be issued to a device that is currently ready for use unless it is expected to become temporarily unavailable.

NOTE: For compatibility with higher level languages, using the wait for operation complete macro call ($WAIT) results in an immediate return of 0.

Write Function (Code 1)

This function allows data to be written to a specific device. When a line protocol handler (LPH) receives a write request, it transfers the indicated data from the application's buffer to the device, according to the specifications supplied in the device-specific word of the application's IORB.

Read Function (Code 2)

This function allows data to be read from a specific device. When the LPH receives a read request, it transfers data from the device to the application's buffer, according to the information supplied in the device-specific word of the application's IORB.

Connect Function (Code A)

The connect function provides a logical and physical connection between an application program and a communications device.

As a logical function, the connect function is a request to use the specified communications device. If that resource is being used, an error return results. In that case the application must determine whether that resource is sharable (as established by the installation's procedures), and proceed accordingly.

As a physical function, the connect function establishes a physical path to the communications device associated with the specified logical resource number (LRN). This implies, when the device is to be connected over a switched line, that the system software should answer the telephone on the line associated with that device. The request times out after five minutes.

If the connect function is not completed, the system will not process any requests for communication devices, and will return an error status.

The connect function must be requested before any other function, since communications devices are configured into the system in a disconnected state.

Disconnect Function (Code B)

The disconnect function provides both the logical (normal and abnormal) and physical disconnection between the application and a communications device.

As a logical function, the disconnect function indicates that the use of the designated device is to be terminated.

For a logical disconnect, issue a disconnect request (function code B) with a queue abort (E-bit in I_DVS set on), and no phone hangup (F-bit in I_DVS set on). (See Table 7-3.) At this point, any pending read or write requests are returned to the application program with a B status (device unavailable). Continued use of the device requires that the application program issue a connect.

As a physical function, the disconnect function must specify the physical disconnection of a line.

Requesting Communications Functions

The following is the sequence for an application to request a transaction with a communications resource:

1. Set up an IORB with the connect function (code A).

2. Call the physical I/O interface (request I/O macro call $RQIO).

3. When the connection is complete, supply the appropriate IORBs for those operations that the application will perform.

4. Perform the functions, e.g., read, write, and/or wait online required by the application's logic.

5. When application processing is complete, supply an IORB with the disconnect function (code B) and issue the request I/O macro call ($RQIO) to execute the function.

PHYSICAL I/O MACRO CALLS FOR COMMUNICATIONS

The input/output request block ($IORB) and request I/O ($RQIO) macro calls provide direct communication from a communications application to the appropriate line protocol handler (LPH). The System Service Macro Calls manual describes these and related macro calls in detail.

SECTION 7

TTY LINE PROTOCOL HANDLER

The TTY line protocol handler supports asynchronous terminal devices, generically classified as teleprinter-compatible (TTY), that include certain ASR, KSR, and visual information projection (VIP) terminals.

A basic TTY terminal consists of either a printer and keyboard or a VIP 7100/7200/7800 display and keyboard. (Paper tape is not supported.) Each type of TTY terminal has an asynchronous communications interface that permits operation at up to 9600 baud.

GENERAL TTY LINE PROTOCOL HANDLER OPERATION

TTY Message Formats

Figure 7-1 illustrates TTY message formats. On input, the application receives only the text portion of the message. On output messages, the application can control print format with a control byte that is specified as the first character of the output buffer (in the IORB device-specific word I_DVS, described later). At connect, read, or write, the application can, with the I_DVS word, dynamically specify which message format is to be used.

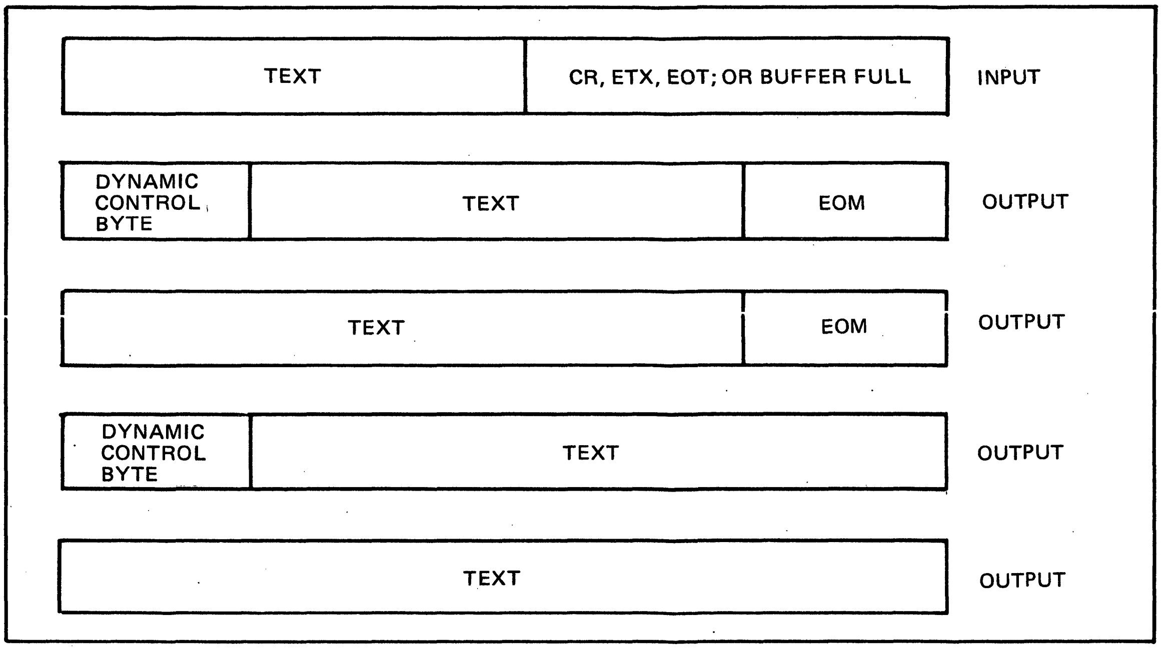


Figure 7-1. TTY Message Formats

TTY Character Mode and Buffered Mode Transmission

TTY CHARACTER MODE

Transmission for all TTY terminals is usually in character mode (one character at a time), a characteristic of the hardware that provides that:

- o The TTY line protocol handler does all editing of data before any transmission.
- o Multiple input lines are not allowed at the same time.

TTY BUFFERED MODE (VIP 7200 AND 7800)

For VIPs 7200 and 7800 only, the buffered mode, available as a hardware option, permits:

- o The TTY line protocol handler to process multiple lines of input at the same time.
- o The operator to do local editing of data before it is transmitted.
- o The application to instruct the TTY line protocol handler not to edit input data.

Buffered mode permits the TTY line protocol handler to process a write order while a read order is pending. A "quasi full duplex" operation gives the line protocol handler the ability to have the application send to the terminal, sequences that cause the terminal to send information back to the application's buffer.

Buffered quasi full duplex operates as follows:

1. When the channel control program (CCP) of the multiline communications processor (MLCP) is currently processing a write order to the terminal, a subsequent read or write operation is not given to the CCP until the current write order completes.
2. When the CCP is processing a read order and the next following order is a write order, that write order is processed while the read order is active.
3. When the write order (2 above) completes and the read order has not yet completed, a subsequent read or write order will not be processed until the read is completed. When the read order is completed before the write order, actions in 1 above take effect.
4. When the read order is completed, the line protocol handler returns to its original state, i.e., no orders pending. The line protocol handler can initiate read or write orders to the CCP.

VIP 7200 AND 7800 HARDWARE SWITCH OPTIONS WITH CHARACTER OR BUFFERED MODE

The TTY line protocol handler supports the following VIP 7200/7800 hardware switch options for character mode or buffered mode as follows:

Character Mode	Buffered Mode
Character/buffered mode switch on character mode.	Character/buffered mode switch on buffered mode.
Parity switch on even.	Parity switch on even.
Full/half duplex switch on full.	Full/half duplex switch on full. Page/line switch as necessary. End-of-message (EOM) character internal switch set to ETX or EOT (not to CR).

VIP 7200 AND 7800 FUNCTION AND CONTROL KEYS

Function and control keys on the VIP 7200 and 7800 are supported only in buffered mode.

When issuing a write request that will cause an automatic response by the terminal, the application must first issue an asynchronous read request, then issue a write request that contains a control message to the terminal.

TTY Line Protocol Handler Time-Out Intervals

Table 7-1 lists the TTY line protocol handler's time-out intervals for the LPH functions.

Table 7-1. TTY Line Protocol Handler Time-Out Intervals

Line Protocol Handler Function	Time-Out Interval
Connect	Five minutes
Read	Character mode: five minutes after receipt of the first character of the message; Buffered mode: five minutes after the line protocol handler receives the request.
Write	Thirty seconds

USING THE TTY LINE PROTOCOL HANDLER

TTY-Specific IORB Values

The TTY-specific IORB item I_CT2, device-specific word I_DVS, and software status word I_ST are shown and defined in Tables 7-2, 7-3, and 7-4, respectively. Section 6 describes the general form of the IORB.

Table 7-2. Function Codes in I_CT2 of the IORB

Function Code	Definition	Use
0	Wait online	Used by the line protocol handler to complete the description of the requested I/O function
1	Write	
2	Read	
A	Connect	
B	Disconnect	

Table 7-3. TTY Device-Specific Word I_DVS in the IORB

Bit Number	Bit Setting	Meaning of Bit Setting
0	0	Must be zero.
1	0	Must be zero.
For connect call only (function code A)		
2	0	Do not use Auto Call Unit.
	1	Use Auto Call Unit.
3	0	Must be zero.
4	0	First byte in buffer on output is a control byte.
	1	First byte in buffer on output is a data byte.
For read call only (function code 2)		
5	0	Input data is in nontransparent mode.
	1	Input data is in transparent mode.
6	0	Must be zero.

Table 7-3 (cont). TTY Device-Specific Word I_DVS in the IORB

Bit Number	Bit Setting	Meaning of Bit Setting
For write call only (function code 1)		
7	0	Stop output immediately on detecting a BRK received from the terminal.
	1	Continue output when BRK detected.
8	0	Must be zero.
9	0	Must be zero.
For read call only (function code 2)		
A	0	Do not echo keyboard input.
	1	Echo keyboard input.
For read and write calls (function codes 2, 1)		
B	0	No LF (line feed) at end of message.
	1	LF (line feed) at end of message.
C	0	CR (carriage return) at end of message.
		No CR (carriage return) at end of message.
For connect call only (function code A)		
D	0	Data transfer is in character mode.
	1	Data transfer is in buffered (block) mode.
For disconnect call (function code A)		
E	0	Abort (dequeue) all IORBs on the request queue.
	1	Process outstanding requests on the request queue.
F	0	Hang up phone after disconnect.
	1	Do not hang up phone after disconnect.

Table 7-4. TTY Software Status Word I_ST in the IORB

Bit	Meaning When Bit Set to 1
0	N/A
1	N/A
2	Data service rate error
3	N/A
4	Communications control block (CCB) service error
5	No stop bit in character input
6	Long record
7	N/A
8	N/A
9	N/A
A	Nonzero residual range
B	Phone hang-up
C	N/A
D	N/A
E	N/A
F	Fatal error: bus parity or memory error
Although nonexistent resource, bus parity, and uncorrectable memory errors are combined in bit F, each occurrence is noted separately in the resource control table (RCT). See Figure C-1.	

Control and Characteristics of TTY Input Data

This subsection describes user control over the characteristics of TTY input data, and applies to character-mode processing unless otherwise noted.

TTY CONTROL BYTE (INPUT)

The description of the TTY control byte for output (see "TTY Control Byte (Send)" below) applies also to the TTY line protocol handler's control byte for input.

TTY NONTRANSPARENT INPUT

TTY input is nontransparent when the application sets to 0 bit 5 of the IORB's device-specific word I_DVS (Table 7-3). Input is accepted until the end-of-range or a CR (carriage return), ETX (end of text), or EOT (end of transmission) control character, whichever is first, is reached. The line protocol handler does not transmit the CR, ETC, or EOT control character as part of the message.

TTY TRANSPARENT INPUT

TTY input text is transparent when the application sets to 1 bit 5 of the device-specific word I_DVS at read time (Table 7-3). All input data, including and control characters, is stored in the buffer until end-of-range is reached.

TTY LINE FEED (LF) AND CARRIAGE RETURN (CR) INPUT SEQUENCE

The application can specify at read time a sequence of LF and CR characters, with the B- and C-bits of the IORB's device-specific word I_DVS, as indicated in Table 7-3. When the message is received successfully, the specified character combinations are retransmitted back to the terminal.

KEYBOARD INPUT CHARACTER AND LINE CONTROL

When an input character with a parity error is received, the line protocol handler sends a BEL character back to the terminal. The user must then retype that input character if it is to be included in the text being sent to the application.

The user can correct or delete erroneous characters or lines and can declare control characters to be data characters, as described below.

To correct one or more characters in the current line, i.e., before the CR is pressed, press the @ key. This deletes the character that immediately preceded the @ character, and displays the @ symbol. Each succeeding @ entry deletes another character, moving from right to left to the beginning of the line.

To delete the current line, i.e., before the CR is entered, press and hold the CTRL (control) key and press X. This deletes the current line, displays the message *DEL* on the next line, and results in a carriage return. The user can then enter a correct line.

To declare a control character (e.g., @, CTRL X, CR, and) be accepted as a data character (transparent mode) press the back slash ( ) key before entering that control character. The system interprets the back slash as an escape character. In transparent mode, all input characters are data characters and have no editing functions.

TTY DISPLAY OF INPUT CHARACTERS

The user can cause an input character to be sent back to the terminal (displayed on the screen or typed on the console) by setting to 1 the A-bit of the device-specific word I_DVS (Table 7-3). For full duplex printers, the application need specify that characters be returned only when they are to be echoed by the system software.

TTY INPUT IN BUFFERED MODE (VIP 7200 AND 7800 ONLY)

When the application at connect time sets to 1 the D-bit of the device-specific word I_DVS, input is accepted until an ETX or EOT control character or end-of-range is encountered.

When the application sets bit 5 of I_DVS to 1 at read time, TTY input in buffered mode is transparent, i.e., there is no editing. When the bit 5 is set to 0, TTY input in buffered mode is nontransparent, i.e., control characters are edited.

As in character mode, the application can specify an LF and CR sequence, as described above under "Line Feed (LF) and Carriage Return (CR) Input Sequence."

Control and Characteristics of TTY Output Data

This subsection describes user control of the characteristics of TTY output data and is applicable to character-mode processing unless otherwise stated.

TTY CONTROL BYTE (SEND)

The TTY line protocol handler's control byte, included as the first character of the application's buffer, controls the message's head-of-form sequence. At connect time, the application specifies the control byte by setting to 0 bit 4 of the IORB's device-specific word I_DVS (Table 7-3).

Figure 7-2 shows the format and content of the TTY control byte.

0	1	2	3	4	5	6	7

BITS 0 THROUGH 2:
NOT USED

BIT 3:
0 = DO NOT GENERATE A HEAD-OF-FORM SEQUENCE
1 = GENERATE HEAD-OF-FORM SEQUENCE CONSISTING OF LF, DL ISSUED THREE TIMES

BITS 4 THROUGH 7:
NOT USED, MUST BE ZERO

Figure 7-2. Control Byte for TTY Line Protocol Handler

END-OF-MESSAGE (EOM) SEQUENCE ON TTY OUTPUT

The EOM sequence is controlled by the B- and C-bits of the IORB's device-specific word I_DVS (Table 7-3), as specified by the application at write time. The TTY line protocol handler sends an EOM sequence according to the following B- and C-bit values:

I_DVS Bits		
B	C	EOM Sequence
0	0	CR, DEL
0	1	None
1	0	LF, CR, DEL
1	1	LF, DEL

At read time, the application can specify the same B- and C-bit values in order to send an EOM sequence back to the terminal when the message is successfully received.

TTY DETECTION OF BRK CHARACTERS

When the application sets to 0 bit 7 of the device-specific word I_DVS at write time, the line protocol handler will immediately stop all output when it detects a BRK key character in the input stream from the terminal. The line protocol handler ignores the BRK character when bit 7 is set to 1, until the write order is completed.

TTY OUTPUT IN BUFFERED MODE

Control and characteristics for TTY output in buffered mode are the same as described above for character mode. However, in processing in buffered mode (VIP 7200/7800 only) the line protocol handler processes all physical I/O requests in the same sequence as they are received. If there is already an outstanding read request, only a subsequent write request can be initiated before the read request is satisfied or the time-out for that read request is elapsed.

SECTION 8

VIP LINE PROTOCOL HANDLER

The VIP line protocol handler supports synchronous VIP (visual information projection) terminals, and the synchronous receive-only printers (ROP).

The basic VIP comprises a cathode ray tube (CRT) display screen and keyboard, with a synchronous communications interface, with operating speeds as follows:

VIP	Baud Rate
7760	9600
7700R	Up to 9600
7700	Up to 4800

GENERAL VIP LINE PROTOCOL HANDLER OPERATION

Software Functional Support for the VIP

The following VIP line protocol handler software functions support the basic VIP terminal:

- o Poll and select communications procedures
- o Nonpoll communications procedures
- o Point-to-point and multipoint configuration support
- o Switched and private line operation
- o Auto-answer for switched network operation
- o Modem, direct connect, and modem bypass interconnection modes
- o Message transfer to and from a CRT (1920-character format)

- o Fully addressable CRT entry marker control
- o Pre-editing (control byte) and post-editing (I_DVS)
- o Transfer of hardware function code to and from the application
- o Error recovery procedures

The following functions support added terminal options:

- o User-controlled CRT forms mode
- o Message transfer to receive-only printer (ROP)

User-Supplied Software Functions for VIP Support

The application program must supply the following functions to support data exchange between the VIP and the application:

- o User-specified device arguments, (polling interval, and at system building, station addresses)

For messages to the VIP terminal, the application should provide:

- o Optional; hardware function codes (1, 2)
- o Complete message text
- o Optional; pre-editing and post-editing characters within message text
- o Mandatory; complete forms definition message text for forms mode

For messages received from the VIP, the application must provide:

- o Interpretation of hardware function codes (1, 2)
- o Message processing
- o Interpretation of format codes (LF, CR, HT) in the message text

VIP Time-Out Intervals

Table 8-1 lists the time-out intervals used by the line protocol handler for the connect, read, and for ROP write functions for the listed devices. The line protocol handler will try and retry the connect, read, and write functions until the indicated time-out period has elapsed.

Table 8-1. VIP Line Protocol Handler Time-Out Intervals

Function	Time-Out Interval	(Device)
Connect	5 minutes	Communications supervisor
	Tries connect one time, returns B status	Nonpolled
	Tries five times	Polled
	Tries indefinitely	Tributary station
Read	None 10 minutes Indefinite	According to the settings of bits 0 and 1 in I_DVS (see Table 8-3)
Write (ROP)	15 seconds	Screen (nonpolled)
	1 second	Screen (polled)
	21 seconds	TN1200, 7717
	95 seconds	TN300, 7714, 7716 (300 baud)
	180 seconds	TN300, 7714, 7716 (150 baud)
	190 seconds	TN300, 7714, 7716 (110 baud)
	190 seconds	TTY33, TTY35
NOTE: Based on 1920-character display screen.		

USING THE VIP LINE PROTOCOL HANDLER

VIP-Specific IORB Values

The VIP-specific input/output request block (IORB) item I_CT2, device specific word I_DVS, and software status word I_ST, are shown in Tables 8-2, 8-3, and 8-4, respectively. Section 6 describes the general form of the IORB.

Table 8-2. Function Codes in I_CT2 of the IORB

Function Code	Definition	Use
0	Wait online	Used by the line protocol handler to complete the description of the requested I/O function.
1	Write	
2	Read	
A	Connect	
B	Disconnect	

Table 8-3. VIP Device-Specific Word I_DVS in the IORB

Bit Number(s)	Bit Setting	Meaning of Bit Setting
For connect call only (function code A)		
0, 1	00	Time-out after 10-minute interval.
	01	No time-out on read requests (i.e., indefinite).
	10	Immediate time-out, no time-out interval.
	11	Reserved for later use by the system.
2	0	Do not use Auto Call Unit.
	1	Use Auto Call Unit.
3	0	Set cursor to home position on page overflow.
	1	Do not set cursor to home position on page overflow.
4	0	Include control byte in first byte of buffer.
	1	Do not include control byte in buffer.
5, 6, 7		Logical poll interval (polled lines only):
	000	Poll continuously.
	001	1-second poll interval.
	010	2-second poll interval.

Table 8-3 (cont). VIP Device-Specific Word I_DVS in the IORB

Bit Number(s)	Bit Setting	Meaning of Bit Setting
5, 6, 7 (cont)	011	3-second poll interval.
	100	4-second poll interval.
	101	5-second poll interval.
	110	15-second poll interval.
	111	30-second poll interval.
8	0	There are no hardware function codes.
	1	There are hardware function codes.
9	0	Must be zero.
A	0	Must be zero.
For write call only (function code 1)		
B	0	No LF (line feed) at end of message.
	1	Issue LF (line feed) at end of message.
C	0	Issue CR (carriage return) at end of message.
	1	Do not issue CR (carriage return) at end of message.
For disconnect call only (function code B)		
D	0	Must be zero.
E	0	Abort (dequeue) all IORBs on the request queue.
	1	Process all outstanding requests on the request queue.
F	0	Hang up phone after disconnect.
	1	Do not hang up phone after disconnect.

Table 8-4. VIP Software Status Word I_ST in the IORB

Bit	Meaning When Bit Set to 1
0	N/A
1	Read error
2	Data service rate error
3	N/A
4	Communications control block (CCB) service error
5	N/A
6	Long record
7	Poll failure
8	NAK limit reached
9	Excessive checksum/parity errors
A	Nonzero residual range
B	Phone hang-up
C	N/A
D	Uncorrectable page overflow
E	Busy received
F	Fatal error: bus parity or memory error
Although nonexistent resource, bus parity, and uncorrectable memory errors are combined in bit F, each occurrence is noted separately in the resource control table (RCT). See Figure C-1.	

VIP Polling Options

Polling (the line protocol handler's continuous request to the VIP terminal on a polled line for data) is subject to two kinds of control, a polling interval and a poll duration.

The application, at connect time, must specify the arguments for the poll interval and poll duration, by setting the appropriate bits in the IORB's device-specific word I_DVS (Table 8-3).

VIP POLL INTERVAL

The VIP poll interval specifies the minimum period of time between each successive request (poll) by the line protocol handler for data from a VIP terminal. The line protocol handler will poll the VIP once for each read request, and when the request is not satisfied, again after the specified poll period elapses.

For example, with a 1-second poll interval, the line protocol handler will issue the same read request every second. For a zero poll interval, the line protocol handler will poll the VIP continuously.

The application specifies the poll interval according to the bit settings of the bits 5, 6, and 7 in the device-specific word I_DVS of the IORB, as shown in Table 8-3.

VIP POLL DURATION (TIME-OUT)

Poll duration, or the time-out interval, is the maximum time that the line protocol handler will wait for polled data from the VIP, before discontinuing the read attempt and read request. Possible time-out intervals are immediate (i.e., after only one poll); 10 minutes; and indefinite (i.e., the VIP is polled continuously, with no time-out, until requested data is received). The application specifies the poll duration or time-out interval with the bits 0 and 1 in the device-specific word I_DVS, according to the bit values shown in Table 8-3.

VIP LINE PROTOCOL HANDLER POLL FUNCTIONS

Within the controls specified in the poll argument values by the application, the line protocol handler provides all necessary polling functions, e.g., how terminals share a common line, or which terminal is processed next, etc. When the application bypasses these line protocol handler poll functions, i.e., by specifying immediate time-out after only one poll, the application must then provide for proper operation and coordination among all terminals on the line.

Control and Characteristics of VIP Input (Keyboard/Screen)

VIP INPUT MESSAGE HEADER

The line protocol handler strips the message header from the input data, except for the hardware function codes, and does not include the header in the application's buffer.

VIP HARDWARE FUNCTION CODES

VIP hardware function codes are listed in the appropriate hardware device manuals.

These codes, provide a special message labeling capability to be used by the application.

The application can include two function codes in the message header of each text message to or from a terminal by setting to 1 bit 8 of the IORB's device-specific word I_DVS (see Table 8-3) at connect time. The line protocol handler then inserts the two user-specified hardware function codes at read time into the IORB's item I_FCS (see Figure 6-1 and Table 6-2).

VIP INPUT DATA

The line protocol handler places into the application's buffer all data, between the STX and ETX control characters, received from the VIP terminal. The data is inserted into the buffer in 7-bit ASCII, with the most significant bit always zero. The LPH strips the ETX and LRC (longitudinal redundancy check character, see Appendix A) from the data and does not include them in the buffer.

Control and Characteristics of VIP Output

This subsection pertains to VIP output and is applicable to the keyboard, display screen, or read-only printer (ROP) as indicated.

VIP OUTPUT MESSAGE HEADER

The VIP line protocol handler supplies the output message header, but not the hardware function codes. Those may be supplied by the application as described above under "VIP Hardware Function Codes."

At write time, when the hardware codes are specified, they are placed in the I_FCS item of the IORB. When they are not specified, i.e., the bit 8 of I_DVS set to 0 at connect time, the line protocol handler will insert two spaces, instead of function codes 1 and 2, into the I_FCS item (see Figure 6-1 and Table 6-2).

VIP CONTROL BYTE (SEND)

The VIP control byte is specified when the application sets to 0 the bit 4 of the device-specific word I_DVS at connect time. The line protocol handler then places the control byte as the first character of the application's buffer.

The control byte controls the output form feed sequence according to its bit settings as shown in Figure 8-1. The line protocol handler provides the output ETX control character and the LRC (longitudinal redundancy check) character.

0	1	2	3	4	5	6	7
R	1	0	Y	Z	Z	Z	Z

R
RESERVED (NOT EXAMINED)
Y=0
DO NOT ISSUE FORM FEED SEQUENCE
Y=1
ISSUE FORM FEED SEQUENCE
ZZZZ
NUMBER OF LINES TO SKIP BEFORE PRINTING (BINARY)
(E.G., IF ZZZZ=0100, VIP LPH WILL PERFORM
4 FF SEQUENCES)

Figure 8-1. VIP Control Byte (Send)

VIP OUTPUT DATA

The application's output data must be 7-bit ASCII (the eighth bit is ignored). Any ASCII control characters, if included in the application's data, are not transmitted.

For keyboard/display screens, the line protocol handler sends a CR, LF sequence when the application's buffer contains the hexadecimal character X'05' (NL).

For the read-only printer (ROP) the line protocol handler sends a CR, LF sequence (according to the type of ROP) shown below, when the application's buffer contains the X'05' character (NL).

ROP Type	Line Sequence
TN1200, 7717	CR, LF, 36 DELs
7714, 7716, TN300, TTY 35	CR, LF, 9 DELs
TTY 33	CR, LF

VIP KEYBOARD/SCREEN OUTPUT EDITING CONTROL

The line protocol handler sends LF and CR editing characters for VIP keyboard/screen devices according to the values of the B- and C-bits of the device-specific word I_DVS (Table 8-3). The application specifies these bit values at write time to send the CR and LF characters, as follows:

I_DVS Bits B	C	Editing Characters Sent
0	0	CR
0	1	None
1	0	LF, CR
1	1	LF

VIP RECEIVE-ONLY PRINTER EDITING SEQUENCE

The line protocol handler sends an output editing character sequence for the receive-only printer (ROP) according to the values of the B- and C-bits of the device-specific word I_DVS (Table 8-3). The application specifies these bit values at write time to send the ROP output editing sequence, according to the ROP type, as shown in Table 8-5.

Table 8-5. VIP Receive-Only Printer Editing Sequence

ROP Types	I_DVS Bits B	C	Output Editing Sequence
TN1200, 7717	0	0	CR, 36 DELs
7714, 7716, TN300, TTY 35	0	0	CR, 9 DELs
TTY 33	0	0	CR
All	0	1	None
TN1200, 7717	1	0	LF, CR, 36 DELs
7714, 7716, TN300, TTY 35	1	0	LF, CR, 9 DELs
TTY 33	1	0	LF, CR
TN1200, 7717	1	1	LF, 36 DELs
7714, 7716, TN300, TTY 35	1	1	LF, 9 DELs
TTY 35	1	1	LF

VIP RECEIVE-ONLY PRINTER FORM FEED SEQUENCE

The VIP line protocol handler sends an output form feed sequence according to the ROP type and whether the ROP has the hardware form feed option, as shown in Table 8-6.

Table 8-6. VIP Receive-Only Printer Form Feed Sequence

ROP Type	Output Form Feed Sequence
Without form feed feature	
TN1200, 7717	LF, 36 DELs (both three times)
7714, 7716, TN300	LF, 9 DELs (both three times)
TTY 35	LF, 9 DELs (both three times)
TTY 33	LF, three times
With form feed feature	
7717, TN1200	FF, 240 DELs
7714, 7716, TN300	FF, 65 DELs
TTY 35	FF, 65 DELs

ERROR PROCESSING BY VIP LINE PROTOCOL HANDLER

Table 8-7 lists the errors reported by the VIP line protocol handler for any VIP configuration. It also lists corresponding return status error codes (see Table 6-1), corresponding bits in the VIP software status word I_ST (see Table 8-4), and possible recovery actions.

Table 8-8 lists the MLCP-specific error condition according to particular VIP configurations, the corresponding error codes, and bits in the I_ST.

Table 8-7. Errors Reported by VIP Line Protocol Handler

Error Condition	Posted Error Return Status[a]	I_ST Bit[b]	Possible Recovery	Comments
Error during open	B	As reported	Retry nine times	
"Not available" message received	7	E	None	
Page overflow not corrected	7	D	None, or retry once	
Invalid range in IORB	4	None	None	
Read time-out	7	3	10-minute retry	
NAK limit reached	7	8		
Busy received	7	E		
Purged due to immediate close	B	None		
Station disabled	B	None		
Fatal error at interrupt level	B	None		
Data service rate error	0 (transmit) 7 (receive)	2 2, 8	Not applicable Retry nine times	Not fatal
Communication control block service rate error	7	4, 8		
Long record	0	6	None (ACK sent to VIP)	Not fatal
Illegal character	0 (transmit)	7	Replace illegal character with delete characters	Bad character in application's buffer
Sequence error, or Poll failure	7 (receive)	7, 8		
Phone hang up	B	B	None	
Nonexistent resource, or Bus parity error, or Unrecoverable memory error	B	None	No retry possible	
Excessive checksum or parity error	B	8, 9	Retry nine times	

[a] See Table 6-1.

[b] See Table 8-4.

Table 8-8. MLCP Error Condition Reported by VIP Line Protocol Handler

Error Condition	VIP Configuration[a]	Posted Error Return Status[b]	I_ST Bit[c]	Possible Recovery Action	Comments
No interrupt from MLCP	P, C (except open)	B	7	Retry five times	Poll failure
	P, C (open only)	B	7	None	CCP/MLCP failure
	NP, C	B	7	None	VIP lockup
	NP,	B	7	None	VIP inaccessible

[a] VIP configuration codes are:

P - polled; NP - not polled; C - control station

[b] See Table 6-1.

[c] See Table 8-4.

PROCESSING NONPOLLED VIP ERRORS

When the VIP does not send a Q-frame within 15 seconds after the data connection is made (i.e., DSR (data set ready) on), the line protocol handler posts the connect IORB with a return status of 6 (see Table 6-1) and with all I_ST bits set to 0.

When the VIP sends a message within 15 seconds after the data connection is made (i.e., DSR on), and the message is erroneous (missed EOT character, parity error), the line protocol handler posts the connect IORB with a return status of B and with all I_ST bits set to 0.

In either case, the application can reissue the connect request without first issuing a disconnect directive.

When, after a successful connect, the application loses communication with the VIP and there are no outstanding requests on the VIP queue, the application will not be notified until the VIP line protocol handler receives the next read or write request.

SECTION 9

POLLED VIP EMULATOR (PVE) LINE PROTOCOL HANDLER

The PVE line protocol handler allows a Level 6 system to be connected to a communications link that operates according to the polled VIP protocol. The line can be half or full duplex, may be dedicated or switched, and operates at up to 9600 baud.

The computer that controls the communications link is known as the control station (CS), which may be any Honeywell host system that supports the VIP protocol.

GENERAL PVE OPERATION

The PVE appears to the control station as a VIP terminal, and is the tributary station. Each PVE supports up to 32 tributary stations per line, as designated at system building.

To the control station, each PVE tributary station is known externally by a poll address, and internally to a Level 6 control station, by a logical resource number (LRN). There is a one-to-one relationship between the poll address and the LRN.

An application program in a Level 6 system communicates with the control station by issuing read and write requests to the appropriate LRN. Similarly, the control station sends and receives as though it is communicating with a polled VIP that has the appropriate poll address.

Figure 9-1 illustrates a typical PVE configuration.

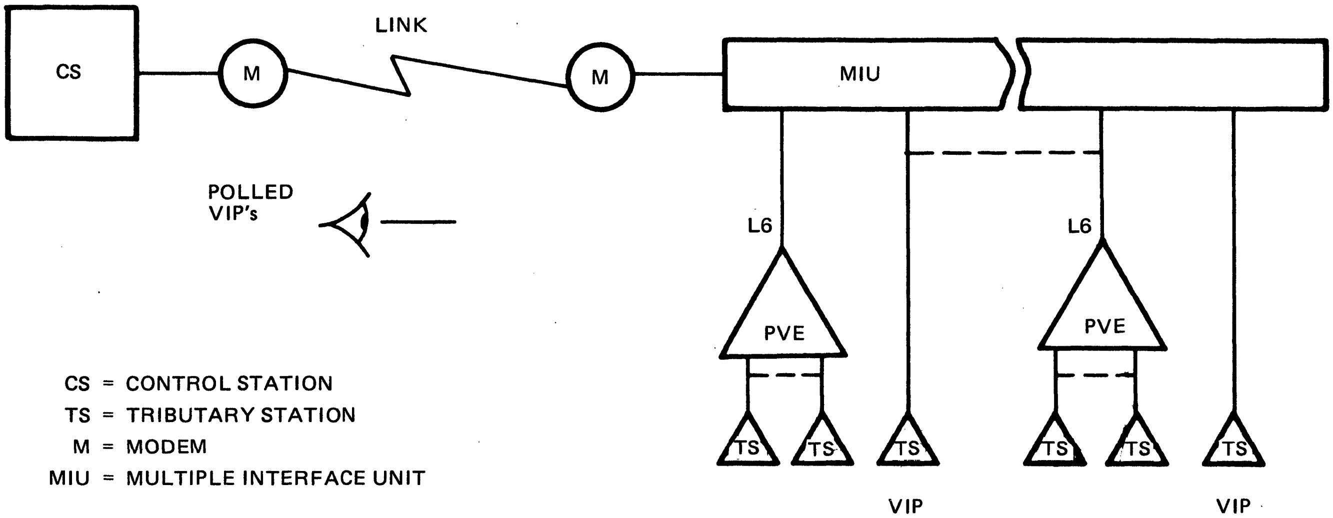


Figure 9-1. Typical PVE Configuration

When the PVE receives a select request with the LRN-associated poll address, it forwards the message to the control station to satisfy the application's read request. When the PVE receives a poll request for the LRN-associated poll address, it forwards the message to the control station to satisfy the application's write request. Thus the application provides the equivalent of the screen and keyboard, with read and write requests, respectively.

The PVE line protocol handler supports only the screen and keyboard features of the VIP.

USING THE PVE LINE PROTOCOL HANDLER

PVE-Specific IORB Values

The PVE-specific IORB item I_CT2, device-specific word I_DVS, and software status word I_ST are shown in Tables 9-1, 9-2, and 9-3, respectively. Section 6 describes the general form of the IORB.

Table 9-1. Function Codes in I_CT2 in IORB

Function Code	Definition	Use
0	Wait online	Used by the line protocol handler to complete the description of the requested I/O function
1	Write	
2	Read	
A	Connect	
B	Disconnect	

Table 9-2. PVE Device-Specific Word I_DVS in the IORB

Bit Number	Bit Setting	Meaning of Bit Setting
0	0	Must be zero.
1	0	Must be zero.
For connect call only (function code A)		
2	0	Do not use Auto Call Unit
	1	Use Auto Call Unit
3	0	Must be zero.
4	0	
5	0	
6	0	
7	0	
8	0	Does not support VIP function codes.
	1	Supports VIP function codes.
9	0	Must be zero.
A	0	Include received DEL characters in buffer.
	1	Strip received DEL characters.
B	0	Must be zero.

Table 9-2 (cont). PVE Device-Specific Word I_DVS in the IORB

Bit Number	Bit Setting	Meaning of Bit Setting
C	0	Must be zero.
D	0	Must be zero.
E, F		LPH response to application when LPH receives data but no read IORB available
	00	Send NAK.
	01	Send ACK. (VIP Status Codes)
	10	Return busy status. (VIP Status Codes)
	11	Send NAK (same as 00).
For disconnect call only (function code B)		
E	0	Abort (dequeue) all IORB's on request queue.
	1	Process all outstanding requests on request queue.
F	0	Hang up phone after disconnect.
	1	Do not hang up phone after disconnect.

Table 9-3. PVE Software Status Word I_ST in the IORB

Bit	Meaning When Bit Set to 1
0	N/A
1	N/A
2	Data service rate error
3	N/A
4	Communications control block (CCB) service error
5	N/A
6	Long record
7	0 = ETX character received 1 = ETB character received
8	NAK limit reached
9	Excessive checksum/parity errors
A	Nonzero residual range
B	Phone hang-up
C	N/A
D	N/A
E	N/A
F	Fatal error: bus parity or memory error

Although nonexistent resource, bus parity, and uncorrectable memory errors are combined in bit F, each occurrence is noted separately in the resource control table (RCT). See Figure C-1.

VIP Protocol Message Structure for PVE

Figure 9-2 shows two VIP protocol message structures for PVE.

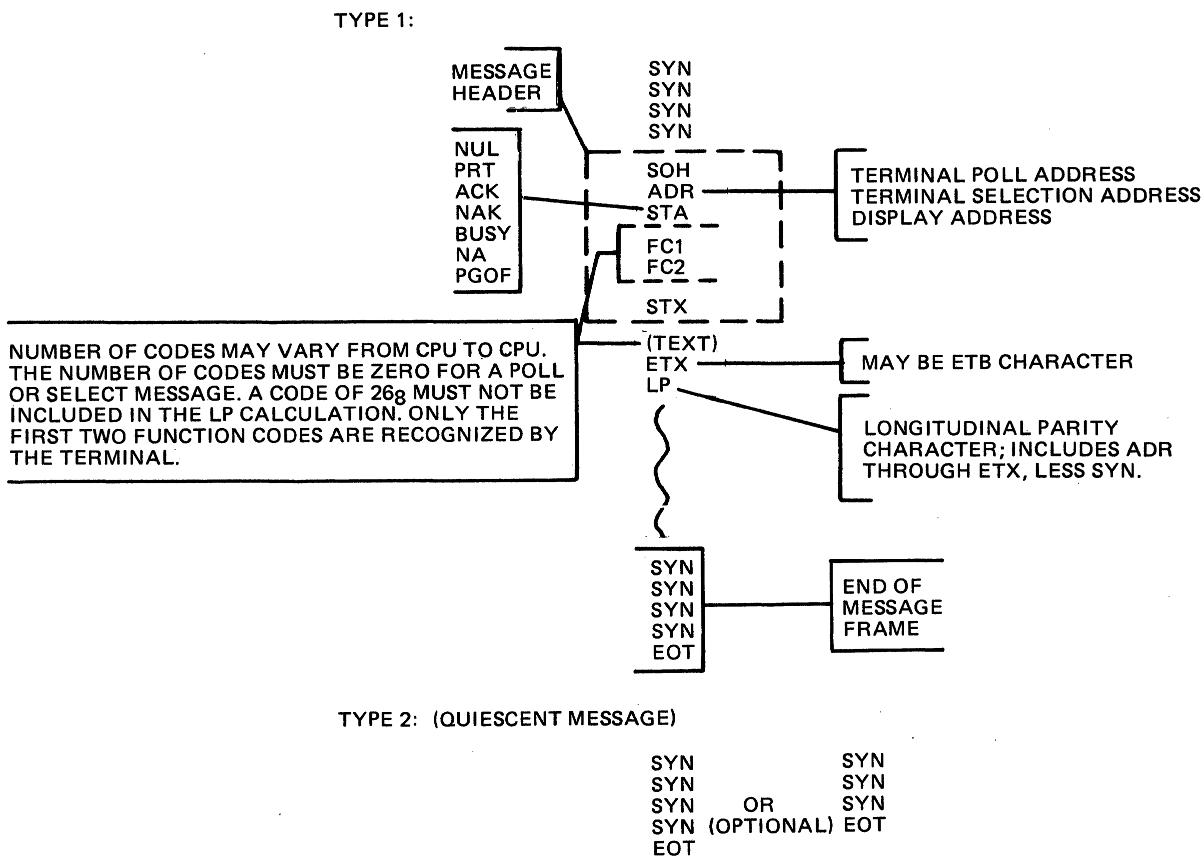


Figure 9-2. VIP Protocol Message Structure for PVE

Control and Characteristics of PVE Input

PVE INPUT MESSAGE HEADER

The PVE line protocol handler strips the message header, between the SOH and STX control characters, and does not include it in the application's buffer.

PVE HARDWARE FUNCTION CODES

PVE hardware function codes are listed in the appropriate hardware device manuals.

These codes provide a special message-labeling capability to be used by the application.

The application can include two function codes in the message header of each text message by setting to 1 the bit 8 of the IORB's device-specific word I_DVS (see Table 9-2) at connect time. The line protocol handler then inserts the two user-specified hardware function codes at read time into the IORB's item I_FCS (see Figure 6-1 and Table 6-2).

PVE INPUT DATA

The line protocol handler places into the application's buffer all data between the STX and ETX control characters. The data is inserted into the buffer in 7-bit ASCII, with the most significant bit always zero. The LPH strips the ETX and LRC (longitudinal redundancy check character, see Appendix A) from the data and does not include them in the buffer.

It also strips DEL characters when the application, at connect time, sets to 1 the A-bit of the device-specific word I_DVS (Table 9-2).

By setting the E- and F-bits of I_DVS as shown in Table 9-2, the application can control the response that the line protocol handler sends when it receives data from the application, but no read IORB is available.

Control and Characteristics of PVE Output

PVE OUTPUT MESSAGE HEADER

The PVE line protocol handler normally supplies the output header, between the SOH and STX control characters. The applica-cation may specify hardware function codes (1, 2) as described above under "PVE Hardware Function Codes." At write time, when specified, the codes are extracted from the I_FCS item of the IORB. When the codes are not specified, (bit 8 of I_DVS set to 0 at connect time), the line protocol handler will supply two spaces, instead of the codes, into I_FCS.

PVE TERMINAL ADDRESS (ADR) AND MESSAGE STATUS (STA)

The PVE line protocol handler supplies an ADR (terminal address) of X'60' (keyboard/screen) and an STA (message status) of NUL to the application.

PVE OUTPUT DATA

The application's output data must be 7-bit ASCII. The most significant bit is used by the line protocol handler during transmission of odd parity.

Output data must not include the ASCII control characters SOH, STX, ETB, ETX, EOT, or SYN.

The line protocol handler supplies output ETX control characters and longitudinal redundancy check characters (LRCs) (described in Appendix A).

PVE LINE PROTOCOL HANDLER TIME-OUT INTERVALS

Table 9-4 lists the time-out intervals used by the line protocol handler for the connect, read, and write functions. The line protocol handler will attempt or reattempt the functions until the indicated time-out period has elapsed.

In addition to the interval in the table, there is also a gross time-out of one minute, which expires when the control station ceases to poll or select any tributary station.

Table 9-4. PVE Time-Out Intervals

Function	Time-Out Interval
Connect	Five minutes
Read	Indefinite
Write	Indefinite

ERROR REPORTING BY PVE LINE PROTOCOL HANDLER

Table 9-5 lists the errors reported by the PVE line protocol handler. It also lists corresponding return status error codes (see Table 6-1) and corresponding bits in the software status word I_ST (see Table 9-3).

Table 9-5. Errors Reported by PVE Line Protocol Handler

Error Condition	Posted Error Return Status	I_ST Bit	Comments
No interrupt from MLCP	6	7	Poll failure or CCP/MLCP failure
NAK limit reached	7	8	Write failure
Purged due to immediate close	B	None	
Station disabled	B	None	
Fatal error interrupt level	B	None	
Data service rate error	0 (send) 7 (receive)	2 2, 8	Not fatal
Communication control block service rate error	7	4, 8	
Long record	0	6	Not fatal
Phone hang-up	B	B	
Nonexistent resource, or Bus parity error, or Unrecoverable memory error	B	None	

SECTION 10

BSC 2780/3780 LINE PROTOCOL HANDLER

The BSC (binary synchronous transmission) 2780/3780 line protocol handler (LPH) supports BSC 2780 and BSC 3780 point-to-point, nontransparent or transparent EBCDIC, or nontransparent ASCII transmission between a Level 6 system and another host system (subject to certain restrictions).

The 3780 protocol is very similar to the standard 2780 protocol and unless specifically stated otherwise, the rest of this section and the term BSC pertain to both.

GENERAL BSC LINE PROTOCOL HANDLER OPERATION

When a station (device or computer) at either end of a communication line has a message to send, it requests use of the line by sending a ENQ bid message. (See Appendix E for definition of ENQ and other control characters.) The receiving station must respond with an ACK/0 sequence before the sending station can transmit a data message.

BSC Transmit and Receive Operations

A station that has control of the line, i.e., the right to transmit, is known as the master (primary)[1] station. The station that relinquishes control, i.e., will receive, is the slave (secondary) station.

When the first data message from the master station is successfully received, the slave station responds with an ACK/1 sequence. Acknowledgments for subsequent remaining messages alternate between ACK/0 and ACK/1. The master/slave status for each respective station remains in effect until the master station gives up control by sending an EOT (end-of-transmission) character (which is not acknowledged by the slave station).

---

[1]Primary and secondary are arguments of the CLM BSC directive used when the system is being built.

When a bidding station does not receive an ACK/0 response within a specified interval (time-out period), it sends another ENQ message. At the same time, or at nearly the same time, the other station may be sending an ENQ message, bidding for the line. Thus both stations may be bidding with neither receiving an ACK response. This is known as line contention. Line contention can be avoided by designating one station as the primary and and the other as secondary when the system is built. Then when the designated primary station receives an ENQ response to its bid message, it retransmits the ENQ message to the secondary station, which in turn ignores its own bid request and responds to the primary station with an ACK or NAK.

The BSC line protocol handler allows a receiving station to reply to a data message with an RVI (reverse interrupt) message if it has an urgent requirement to transmit data.

Figure 10-1 illustrates bids and other interactions between a master and slave station.

BSC Data Transmission Modes

BSC operates in either basic data transmission mode or in advanced data transmission mode, according to whether a control byte is included in the data being transmitted. (See "BSC Control Byte (Receive)" and "BSC Control Byte (Send)" later in this section.)

BSC BASIC DATA TRANSMISSION MODE

In basic data transmission mode, there is no control byte included in the data being transmitted along the communications line.

BSC ADVANCED DATA TRANSMISSION MODE

In advanced data transmission mode, the application includes a control byte (that is not part of the data). The control byte indirectly controls the operation of the line protocol handler (e.g., sending an ETB or ETX), or conveys information about a data transfer (e.g., whether transparent text was received).

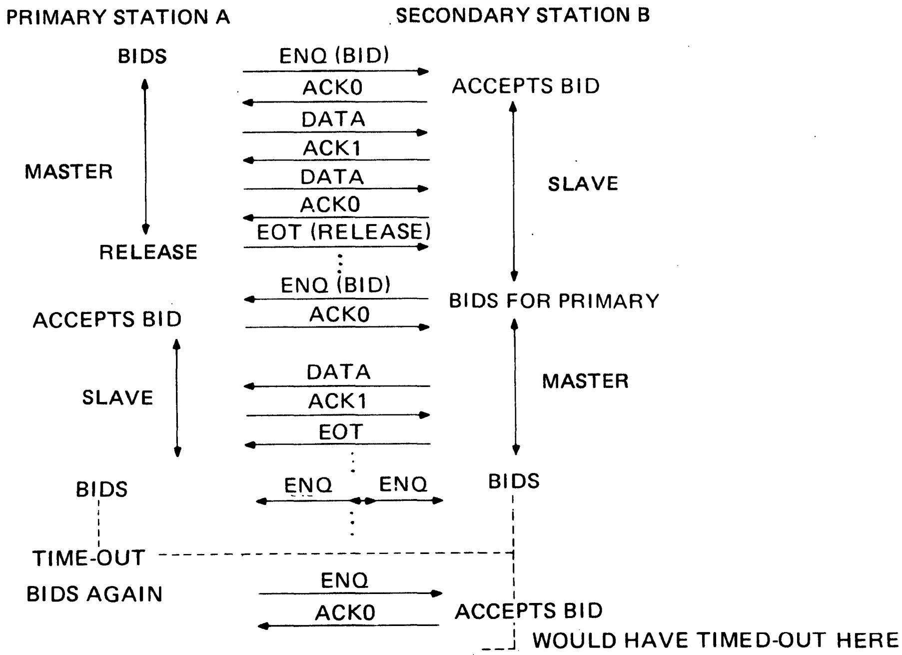


Figure 10-1. Example of BSC Communication

BSC 2780 AND BSC 3780 DIFFERENCES

The 3780 protocol differs from the 2780 protocol in that the 3780 protocol has a set of extensions that provide:

- The ability to receive a conversational reply.
- The ability to receive two records and to transmit a single record, when the two-buffer option is selected at connect time.
- The ability to receive and transmit selected BSC control characters in nontransparent mode.

BSC 2780/3780 FEATURES

The following discussions in this subsection include references to BSC-specific fields in the input/output request block IORB (see Table 6-2) and to control bytes, and precede their descriptions. See Tables 10-2 and 10-3 later in this section for descriptions of the device-specific word I_DVS and software status word I_ST, respectively. Control bytes are described under "Control Byte (Receive)" and "Control Byte (Transmit)."

BSC Two-Buffer Feature

With the two-buffer feature, the use of the second buffer reduces line turnaround time, i.e., two records can be transmitted with only one acknowledgment. However, there are these disadvantages:

- o When a line (parity) error occurs, both records must be retransmitted.

- o One transmission requires two writes be issued, which are not posted until an acknowledgment is received.

- o Four buffers are necessary to operate the line efficiently.

Figure 10-2 shows record transmissions with and without the two-buffer feature.

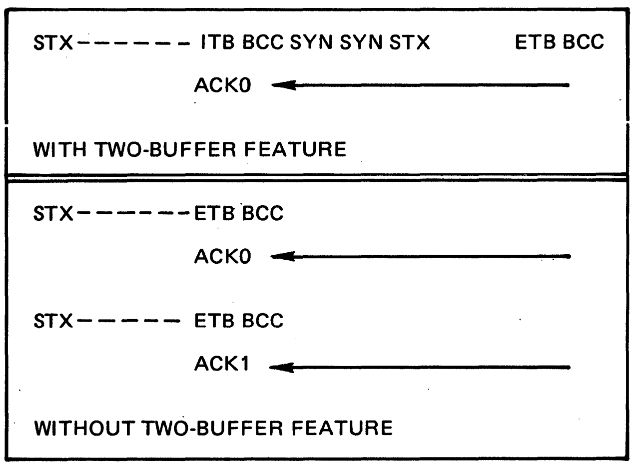


Figure 10-2. BSC Two-Buffer Feature in Record Transmission

Before selecting the two-buffer feature, compare the advantage of better line utilization against the disadvantages of a more complex program and increased buffer usage, and consider the following:

1. In BSC 2780 with the two-buffer option, two records can be received or transmitted (using an ITB (intermediate text block) sequence).

2. In BSC 3780, with the two-buffer option two records can be received, using an ITB sequence, and single records can be transmitted. This implies that an application using BSC 3780 must be able to receive up to two records at any one time, but can only initiate single-record transmission.

3. The two-buffer feature cannot be used with synchronous reads, because the intermediate files being received may be terminated by an ETX record. If the ETX record is the first of the two records being read, the second read (synchronous) would not be posted to the system.

For example:

```
┌──►READ (asynchronous)  )
│   .                    )
│   . process            )  Assumes always two records
│   .                    )  per transmission.
│   READ (synchronous)   )
│   .
└── . process
    .
```

The following sequence is better:

```
    READ (asynchronous)
┌──►READ (asynchronous)
│   WAIT (1)
│   .
│   . process
│   .
│   READ (asynchronous)
│   WAIT (2)
│   .
│   . process
└── .
```

BSC Temporary Text Delay (TTD) Feature

The following describes the sequence of the temporary text delay (TTD) feature.

1. When a master station receives an ACK, and no output request block (IORBs) are queued, that station waits two seconds for one IORB (or two IORBs when there are two buffers) to be queued.

2. The master station then sends the temporary text delay (TTD) control character sequence (STX, ENQ) to the slave station.

3. When the slave station responds with a NAK, the master station checks whether the application has queued the appropriate write requests. If the write requests are not queued, the master station continues the TTD sequence until the application issues the necessary write requests.

4. If the EOT or ETX bit (A-bit or D-bit) in the I_DVS word of the IORB is set (Table 10-2), one write request will effect transmission.

Figure 10-3 is an example of the temporary text display sequence.

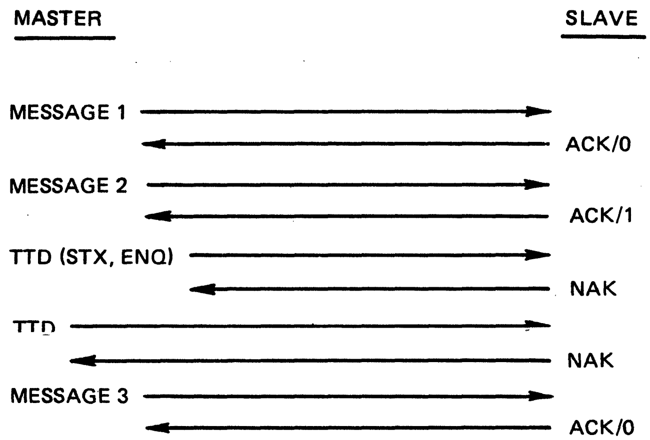


Figure 10-3. BSC Temporary Text Delay (TTD) Sequence Example

BSC Wait Before Acknowledge (WACK) Feature

A BSC slave station will send ACK/0 and ACK/1 responses to messages satisfactorily received, provided there is at least one outstanding read request (two with the two-buffer feature), in addition to the request being processed.

1. When no read request is queued, the slave station posts the current read, waits two seconds for read requests to be queued, then sends a WACK response (DLE; DLE,), indicating to the master station that the last message was received, but the slave station cannot accept more data.

2. The master station waits (time-out), then sends an ENQ message.

3. If a read request was queued during the time-out, the slave station responds with an ACK, and the master station can send its next data message.

4. If no read request was queued during the time-out, the slave station waits another two seconds, and when necessary sends another WACK sequence.

Figure 10-4 is an example of the wait before acknowledge (WACK) sequence.

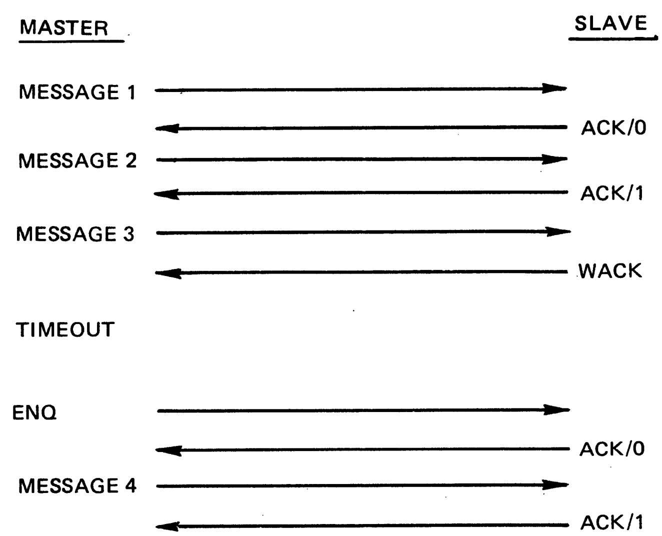


Figure 10-4. BSC Wait Before Acknowledge (WACK) Sequence Example

BSC Reverse Interrupt (RVI) Feature

When a slave station is processing read requests, and must unexpectedly transmit an urgent message, that station must issue a reverse interrupt (RVI) message, which informs the master station that the slave station is requesting control of the line.

On receiving an RVI character, the master station should empty its buffers and give up control of the line. However, the master station does not have to acknowledge the RVI by giving up control.

The application program can request the BSC line protocol handler to send an RVI character, by either of the following methods:

1. Use the control byte. The application issuing read requests issues a transmit request with bit 5 of the control byte set to 1 (see Figure 10-10), and with the urgent message in the application's buffer.

2. Use the device-specific word I_DVS of the IORB. The application issuing read requests issues a transmit request with the B-bit of I_DVS set to 1 and with the urgent message in the application's buffer.

The application issuing write requests can detect an RVI character by any of these methods:

1. Test bit 3 of the control byte after a successful write request is posted. A bit setting of 1 indicates that the RVI for that IORB was received.

2. Test bit 3 of the IORB's software status word I_ST. A bit setting of 1 indicates the RVI was received.

Figure 10-5 is an example of a reverse interrupt (RVI) sequence.

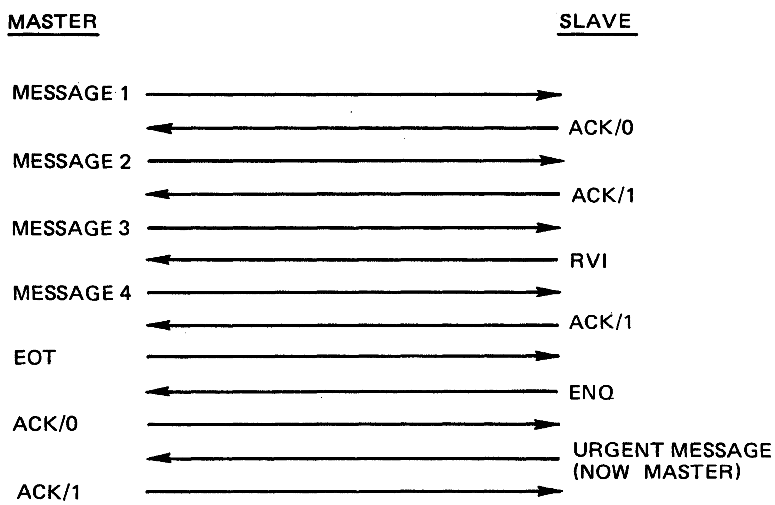


Figure 10-5. BSC Reverse Interrupt (RVI) Sequence Example

BSC End of Transmission (EOT) Feature

The appliation program, by any of the following methods (1, 2, or 3), can cause the BSC line protocol handler to send an end-of-transmission (EOT) message:

1a. At connect time, specify use of the control byte by setting to 0 bit 4 of the IORB's device-specific word I_DVS.

b. When bit 4 of the first byte of the application's buffer (control byte, specified at write time) is set to 1, the BSC line protocol handler will send an EOT control character after the data in the application's buffer is successfully transmitted.

2a. When the control byte is not specified at connect time, set to 1 the A-bit of the IORB's device-specific word I_DVS at write time.

b. The BSC line protocol handler will send an EOT control character after the data in the application's buffer is successfully transmitted.

3a. After successful completion of a write request, issue a disconnect with or without a queue abort, and no physical disconnect.

b. The master station will send an EOT character and give up its master status.

c. However, when another IORB is queued for write, that station will again request its master status.

The application can detect receipt of an EOT control character in either of the following ways:

1. If the control byte was specified at connect time, bit 4 of the control byte, of the read request on which the EOT was received, will be set to 1.

2. If the control byte was not specified at connect time, bit 12 of the software status word I_ST, of the request on which the EOT character was received, will be set to 1.

With either method, the line protocol handler does not post any read requests that were queued before the EOT character was detected. To remove read requests from the queue, the application must issue a disconnect with a queue abort. The line protocol handler always posts the IORB with a device unavailable (B) return status (Table 6-1). The BSC line may or may not be available for further use, depending on whether or not an EOT character was sent abnormally.

BSC Line Protocol Handler Time-Out Interval

On a read, the time-out interval in waiting for a line-request bid is 10 minutes.

For a read or write request, when no response is received, the time-out interval is 12 seconds.

Once a station has successfully bid for a line, the time-out interval for subsequent reads (from the slave station) or writes (from the master station) is 12 seconds.

BSC Features Specific to 3780

BSC 3780 CONVERSATIONAL REPLY FEATURE

The conversational reply feature permits a 3780 application, after transmission of an entire message (whose last record is denoted by an ETX rather than an ETB), to selectively receive a message from a host computer without a preliminary line bid sequence.

The conversational reply sequence serves as the affirmative reply to the last message transmission block, and as a break or interrupt to later transmissions. The line protocol handler indicates to the application receipt of a conversational reply sequence in bit 5 of the IORB software status word I_ST, and/or in bit 2 of the control byte of the ETX write order.

In the following example, a 3780 application attempts to transmit three 2-record messages to a remote host computer. The transmission sequence is interrupted by the receipt of a conversational reply, which occurs after transmission of the second message. After the complete conversational reply (containing one or more records) is received, transmission of the third message can resume, following completion of a successful line bid sequence. Figure 10-6 illustrates the example sequence.

The application's use of the conversational reply feature requires that the application issue the requisite number of read orders (dependent on one- or two-buffer mode) before the transmission of a text block that terminates with an ETX sequence. If the application does not issue the required read(s), the last text block is not transmitted, and the line protocol handler will initiate a temporary text delay (TTD) sequence until the necessary read orders are issued. If the application does not transmit an ETX sequence, it need not issue supporting read order(s).

BSC 3780 TWO-BUFFER FEATURE

The discussion under "BSC Two-Buffer Feature" earlier in this section applies also to BSC 3780 operation.

BSC 3780 TRANSMISSION/RECEIPT OF BSC CONTROL CHARACTERS

In BSC 2780 nontransparent mode, detection of any BSC control characters within a message would abort the transmission or reception of that message.

In 3780 nontransparent mode, selected, noncritical BSC control characters, i.e., STX, SOH, DLE, NAK, and EOT, can be successfully transmitted and received.

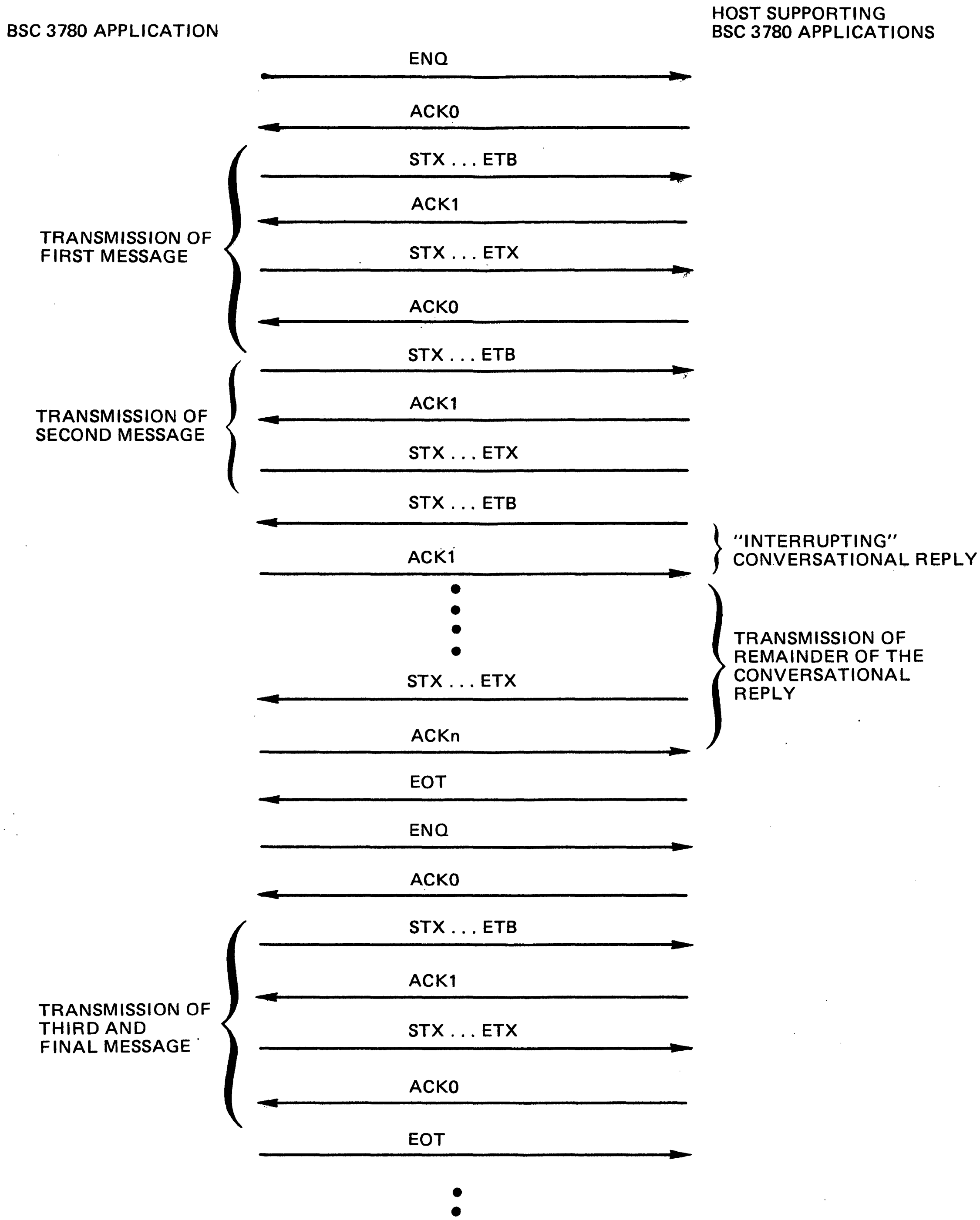


Figure 10-6. Example of Conversational Reply in BSC 3780 Transmission Sequence

USING THE BSC 2780/3780 LINE PROTOCOL HANDLER

BSC-Specific IORB Values

The BSC-specific IORB item I_CT2, device-specific word I_DVS, and software status word I_ST, are shown and defined in Tables 10-1, 10-2, and 10-3, respectively. Section 6 has a general description of the IORB.

Table 10-1. Function Codes in I_CT2 Field in the IORB

Function Code	Definition	Use
0	Wait online	Used by the line protocol handler to complete the description of the requested I/O function.
1	Write	
2	Read	
A	Connect	
B	Disconnect	

Table 10-2. BSC Device-Specific Word I_DVS in the IORB

Bit Number	Bit Setting	Meaning of Bit Setting
0	0	Must be zero.
1	0	Must be zero.
For connect call only (function code A)		
2	0	Do not use Auto Call Unit.
	1	Use Auto Call Unit.
3	0	Must be zero.
4	0	Use control byte.
	1	Do not use control byte.
5	0	Must be zero.
6	0	Must be zero.
7	0	Must be zero.

Table 10-2 (cont). BSC Device-Specific Word I_DVS in the IORB

Bit Number	Bit Setting	Meaning of Bit Setting
	For connect call only (function code A) (cont)	
8	0	Use single buffer per transfer.
	1	For 2780: use two buffers per send/receive.
		For 3780: use two buffers per receive.
9	0	Use BSC 2780 protocol.
	1	Use BSC 3780 protocol.
For write call only (function code 1)		
A	0	Do not send EOT after this transmission.
	1	Send EOT after this transmission.
B	0	Do not send RVI if station in slave status.
	1	Send RVI if station in slave status.
C	0	Send data in nontransparent mode.
	1	Send data in EBCDIC transparent mode.
D	0	Send ITB or ETB characters following the data.
	1	Send ETX characters following the data.
For disconnect call only (function code B)		
E	0	Abort (dequeue) all IORBs on request queue.
	1	Process outstanding requests on request queue.
F	0	Disconnect line on completion.
	1	Do not disconnect line on completion.

Specifying Use of BSC 2780 and/or 3780 to the System

The inclusion of BSC 2780 and/or 3780 in the system is done at system building. The application can select and use either 2780 or 3780 according to the setting of bit 9 in the device-specific word I_DVS in the IORB (see Table 10-2).

Table 10-3. BSC Software Status Word I_ST in the IORB

Bit	Meaning When Bit Set to 1
0	N/A
1	N/A
2	Data service rate error
3	Lost line bid; RVI received
4	Communications control block service error
5	Conversational reply received (3780 only)
6	Long record
7	0 - ITB and/or ETB characters received 1 = ETX character received
8	N/A
9	N/A
A	Nonzero residual range
B	Phone hang-up
C	EOT character received
D	Transparent message received
E	NAK limit reached
F	Fatal error: bus parity or memory error

Although nonexistent resource, bus parity, and uncorrectable memory errors are combined in bit F, each occurrence is noted separately in the resource control table (RCT). See Figure C-1.

Formats and Characteristics of BSC Input Data

The formats and characteristics of BSC input data for both ASCII and EBCDIC are described and illustrated below.

Figure 10-7 shows the format and contents of BSC input data received from another computer.

SOM	(CONTROL BYTE) DATA	EOM	BCC

SOM (START OF MESSAGE)
: A ONE- OR TWO-CHARACTER SEQUENCE THAT IS STRIPPED BY THE BSC LPH.

CONTROL BYTE
: THE CONTROL BYTE, IF SPECIFIED, IS THE FIRST BYTE OF THE APPLICATION'S DATA.

DATA
: INFORMATION STORED IN THE APPLICATION'S BUFFER AND SPECIFIED AT READ TIME.

EOM (END OF MESSAGE)
: A ONE- OR TWO-CHARACTER SEQUENCE THAT IS STRIPPED BY THE BSC LPH.

BCC
: AN LRC CHARACTER OR CRC CHARACTER THAT IS INSERTED BY THE BSC LPH.

Figure 10-7. BSC Input Data Format and Contents

BSC CONTROL BYTE (RECEIVE)

When bit 4 of the IORB's device-specific word I_DVS is set to 0 at connect time (see Table 10-2), the BSC line protocol handler uses the first byte of the application's buffer as the control byte. Figure 10-8 shows the control byte's format and content.

0	1	2	3	4	5	6	7

BITS 0 THROUGH 3
: NOT APPLICABLE; NOT EXAMINED

BIT 4=0
: DATA STORED IN APPLICATION'S BUFFER

BIT 4=1
: EOT RECEIVED; NO DATA STORED IN APPLICATION'S BUFFER

BIT 5
: NOT APPLICABLE; NOT EXAMINED

BIT 6=0
: DATA RECEIVED IN NONTRANSPARENT MODE

BIT 6=1
: DATA RECEIVED IN TRANSPARENT MODE

BIT 7=0
: ITB OR ETB RECEIVED

BIT 7=1
: ETX RECEIVED

Figure 10-8. Control Byte (Receive) for BSC Line Protocol Handler

ASCII INPUT FOR BSC

ASCII input characteristics and format (Figure 10-7) are as follows:

1. SOM (start-of-message) consists of the STX control character only.

2. The control byte (if specified at connect time) is stored in the first byte of the applications' buffer, and indicates the end-of-message (EOM) sequence. When bit 7 is 0, it indicates detection of an ITB or ETB control character; when 1, it indicates detection of an ETX character. Note that bit 7 of both the control byte and of I_ST are specified.

3. Data must be 7-bit ASCII with odd parity. The BSC line protocol handler strips the parity bit and resets it to zero when it stores it in the application's buffer.

4. The EOM sequence, one of the three control chracters ITB, ETB, or ETX, is indicated by bit 7 of the IORB software status word I_ST after a successful read is posted. See Table 10-3 for bit 7 indicators.

5. The BCC (block check character) is described in Appendix A.

EBCDIC INPUT FOR BSC

EBCDIC input format and characteristics are as follows:

1. SOM (start-of-message) consists of the STX control character only.

2. The control byte (if specified at connect time) is stored in the first byte of the application's buffer, and indicates the end-of-message (EOM) sequence, as follows:

Bit 4 = 1	End of transmission (EOT) detected.
Bit 7 = 0	ITB or ETB character detected.
Bit 7 = 1	ETX character detected.

3. Data must be 8-bit EBCDIC; it will not have any BSC control characters.

4. The EOM sequence, one of the control characters ITB, ETB, or ETX, is indicated by bit 7 of the IORB software status word I_ST after a successful read is posted. See Table 10-3 for bit 7 indicators.

5. The BCC (block check character) is described in Appendix A.

TRANSPARENT EBCDIC INPUT FOR BSC

Transparent EBCDIC input format and characteristics are as follows:

1. SOM (start-of-message) consists of the two-character sequence DLE, STX.

2. The control byte, if specified at connect time, is stored in the first byte of the application's buffer, and indicates the EOM (end-of-message) sequence according to the bit 7 setting (Figure 10-8).

3. Data may be any EBCDIC character, including BSC control characters.

4. EOM (end-of-message) sequence may be one of the following, indicated by bit settings of the IORB software status word I_ST, after a successful read has been posted:

I_ST Bits

D	7	Resulting EOM Sequence
1	0	DLE, ITB
1	0	DLE, ETB
1	1	DLE, ETX

5. The block check character (BCC) is described in Appendix A.

Formats and Characteristics of BSC Output Data

Formats and characteristics of BSC output data (both ASCII and EBCDIC) are described and illustrated below.

Figure 10-9 shows the format and content of BSC data transmitted to another computer.

SOM	(CONTROL BYTE) DATA		EOM	BCC

SOM
A ONE- OR TWO-CHARACTER SEQUENCE THAT IS INSERTED IN FRONT OF THE DATA BY THE BSC LPH.

CONTROL BYTE
THE CONTROL BYTE, IF SPECIFIED, IS STORED IN THE FIRST BYTE OF THE APPLICATION'S BUFFER.

EOM
A ONE- OR TWO-CHARACTER SEQUENCE THAT IS INSERTED BY THE BSC LPH.

BCC
AN LRC CHARACTER OR CRC CHARACTER THAT IS INSERTED BY THE BSC LPH.

DATA
INFORMATION THAT IS TRANSMITTED FROM THE APPLICATION'S BUFFER BY THE BSC LPH.

Figure 10-9. Format and Content of BSC Output

BSC CONTROL BYTE (SEND)

When bit 4 of the IORB's device-specific word I_DVS is set to 0 at connect time (see Table 10-2), the BSC line control handler uses the first byte of the application's buffer as the control byte. Figure 10-10 shows the format and content of the BSC line protocol handler's control byte for sending data.

0	1	2	3	4	5	6	7

BITS 0, 1
NOT APPLICABLE, NOT USED
BIT 2=1
CONVERSATIONAL REPLY RECEIVED
BIT 3=1
RVI RECEIVED (RETURN STATUS ONLY)
BIT 4=1
SEND THE DATA THAT IS IN YOUR BUFFER AND, AFTER IT HAS BEEN ACKNOWLEDGED, SEND EOT
BIT 5=1
SEND AN RVI RESPONSE ON THE NEXT ACKNOWLEDGMENT OF A READ
BIT 6=0
SEND NONTRANSPARENT EBCDIC
BIT 6=1
SEND TRANSPARENT EBCDIC OR ASCII
BIT 7=0
SEND ITB OR ETB
BIT 7=1
SEND ETX

Figure 10-10. Control Byte (Send) for BSC Line Protocol Handler

BSC ASCII OUTPUT

ASCII output characteristics and format are as follows:

1. SOM (start-of-message) consists of only the STX character.

2. The control byte, when specified, is assumed to be the first byte of the application's buffer, and indicates the EOM (end-of-message) sequence, which is either ITB, ETB, or ETX, designated as follows:

   a. Bit 6 must be 0.

   b. Bit 7 = 0. Send ITB or ETB. ITB is sent when the record is odd numbered (1, 3, 5, etc.) and the two-buffer feature is used.

      Bit 7 = 1. Send ETX.

   If the control byte is not specified, the EOM sequence is defined by I_DVS as described in 4 below.

3. Data must be 7-bit ASCII; it cannot have any BSC control characters.

4. EOM, which is either ITB, ETB, or ETX, can be indicated by the control byte (see 2 above) or by the C- and D-bits of the IORB device-specific word I_DVS (Table 10-2) as follows:

   a. C-bit must be zero.

   b. D-bit = 0. Send ITB or ETB. ITB is sent when the record is odd-numbered (1, 3, 5, etc.) and the two-buffer feature is used.

      D-bit = 1. Send ETX.

5. BCC (block check character) is described in Appendix A.

BSC EBCDIC OUTPUT

EBCDIC output characteristics and format are as follows:

1. SOM (start-of-message) consists of only the STX character.

2. The control byte, when specified, is assumed to be the first byte of the application's buffer, and indicates the EOM (end-of-message) sequence, which is either ITB, ETB, or ETX, designated as follows:

a. Bit 6 must be 0.

b. Bit 7 = 0. Send ITB or ETB. ITB is sent when the record is odd-numbered (1, 3, 5, etc.) and the two-buffer feature is used.

Bit 7 = 1. Send ETX.

If the control byte is not specified, the EOM sequence is defined by I_DVS as described in 4 below.

3. Data may be 8-bit EBCDIC; it cannot have any BSC control characters.

4. EOM (end-of-message), which is either ITB, ETB, or ETX, can be indicated by the control byte (see 2 above) or by the C- and D-bits of the IORB device-specifid word I_DVS (Table 10-2) as follows:

a. C-bit must be zero.

b. D-bit = 0. Send ITB or ETB. ITB is sent when the record is odd-numbered (1, 3, 5, etc.) and the two-buffer feature is used.

D-bit = 1. Send ETX.

5. BCC (block check character) is described in Appendix A.

BSC TRANSPARENT EBCDIC OUTPUT

Transparent EBCDIC output characteristics and format are as follows:

1. SOM (start-of-message) consists of the <u>two-character</u> sequence DLE, STX.

2. The control byte, when specified, is assumed to be the first byte of the application's buffer, and indicates the EOM (end-of-message) sequence, which is either DLE ITB; DLE ETB; or DLE ETX, designated as follows:

a. Bit 6 must be 0.

b. Bit 7 = 0. Send DLE ITB or DLE ETB. DLE ITB is sent when the record is odd-numbered (1, 3, 5, etc.) and the two-buffer feature is used.

Bit 7 = 1. Send DLE ETX.

If the control byte is not specified, the EOM sequence is defined by I_DVS as described in 4 below.

3. Data may be any EBCDIC character, including any BSC control characters.

4. EOM, which can be either DLE ITB; DLE ETB; or DLE ETX, can be indicated by the control byte (see 2 above) or by bit 4 and bit D of the IORB device-specific word I_DVS (Table 10-2) as follows:

   a. Bit 4 must be 1.

   b. D-bit = 0. Send DLE ITB or DLE ETB. DLE ITB is sent when the record is odd-numbered (1, 3, 5, etc.) and the two-buffer feature is used.

      D-bit = 1. Send DLE ETX.

5. BCC (block check character) is described in Appendix A.

APPENDIX A

COMMUNICATIONS SUBSYSTEM

Communications software, as discussed in this manual, is a functional package referred to as the communications subsystem, and which comprises:

- o Communications supervisor
- o Line protocol handlers (LPHs)
- o Multiline communications processor (MLCP)
- o Multiline communications processor driver

COMMUNICATIONS SUPERVISOR

The communications supervisor is the physical I/O interface between a communications application program and the device/files it uses. It provides the following services, similar to those provided by the Monitor, to an application:

- o Validates and queues, on a first-in/first-out basis, an application's requests for services, then activates the appropriate line protocol handler.

- o Dequeues requests for services, and through system software, interacts with the application when the requested I/O service is completed or an unexpected event occurs.

- o Services time-outs for the line protocol handlers.

- o Controls modems in detecting phone connects and disconnects.

- o Disconnects phones when requested by the application.

LINE PROTOCOL HANDLERS (LPHs)

The line protocol handlers transfer data between a communications device and the application that uses it.

The communications subsystem and its line protocol handlers do the following:

- o When the system is bootstrapped:
  - Validate specifications for device types by reading the device's identification sequence
  - Initialize the device by sending to it the priority level at which it is to operate
- o Validate the application's input/output request block (IORB) fields
- o Convert user-supplied functions into device-specific instructions, initiating the I/O operation
- o Modify channel numbers to even or odd values, according to whether the function is input or output
- o Set a timer in order to detect a device fault
- o Detect and process ATTENTION signals
- o Read return status indicators from a device to ascertain result of an I/O operation
- o Process error recovery, when possible
- o Process unsolicited interrupts
- o Build the return status word indicating logical result of the I/O request, and through the Monitor, passing that value to the application program
- o Pass a value indicating the logical conclusion of the I/O request, through the Monitor, to the application program. (Table 6-1 lists the return status codes).
- o Report the following errors and statuses:
  - Convert hardware return status into the standard software status and insert it into the software status word I_ST of the application's IORB (see Table 6-3).
  - Place the residual range value (see Table 6-2) into the I_RSR entry of the IORB.

MULTILINE COMMUNICATIONS PROCESSOR (MLCP)

The MLCP includes a channel control program (CCP) that is associated with each line protocol handler (see Figure A-1).

Through the appropriate hardware device-pac, the channel control program controls transmission of data over communication lines. Its functions are:

- Process characters by storing them in, then extracting them from, a buffer
- Insert and delete (or strip) headers and trailers
- Insert and delete control characters preceding or following a message to or from a remote terminal or host computer.

The MLCP Programmer's Reference Manual describes the MLCP and related programming information.

MULTILINE COMMUNICATIONS PROCESSOR DRIVER

The MLCP driver receives MLCP orders from the line protocol handler and activates the appropriate channel control program (see above and Figure A-1) to process the orders. The driver also:

- Processes a line protocol handler's requests for control functions or for data
- Services interrupts from the MLCP and passes them to the line protocol handler

MODEM SUPPORT

For asynchronous devices, the communications subsystem supports the direct-connect feature, and provides the following modem support:

- Bell System Data Sets, Types 103A, 113F, or 202
- Honeywell modem bypass
- Any user-defined (at system building) modem type

For synchronous communications, the communications subsystem supports the direct-connect feature, and provides the following modem support:

- Bell System Data Sets, Types 201A, 201B, 201C, 203, or 208A
- Honeywell modem bypass

o User-defined (at system building) modem types

AUTO CALL UNIT

When included in the system (at system building) an Auto Call Unit (autodial feature) performs the following to initiate a line connection with a remote device:

1. The system attempts to dial a line, using a list of telephone numbers supplied at system building, the first entry on the list being zero. The first number to be dialed can then be specified with a set dial ($SDL) macro call or with the set ACU telephone number (SDL) command. If the first number on the list is not specified (by the macro call or command), the system skips to the next number on the list.

2. Dials each number on the list three times at 40-second intervals until the list is exhausted or a connection made, whichever comes first.

3. Checks that a connection to a modem is made.

4. Passes control to the application.

The Auto Call Unit supports Data Auxiliary Set Automatic Calling Units 801A and 801C.

Two data set options are required to use the Auto Call Unit:

o The option that terminates the call, through the data set, after the DSS (data set status change) goes on.

o The option that stops the ACR timer when the DSS goes on.

COMMUNICATIONS SUBSYSTEM OPERATION EXAMPLE

The following example, and Figure A-1, broadly indicate the interaction of the communications subsystem's components in the processing of a connect, write and then disconnect request. The operations described apply to either the file system or physical I/O interface, without reference to a specific device or line protocol.

Example:

1. The communications supervisor takes the application's connect request through the file system or physical I/O interface, then passes it to the phone monitor within the multiline communications processor.

2. The phone monitor makes a line connection to the device.

3. The appropriate line protocol handler processes the logical connection.

4. The communications supervisor passes the application's subsequent write request to the line protocol handler, which translates the request into MLCP driver orders.

5. The line protocol handler calls the MLCP driver, which issues the orders to the MLCP.

6. The channel control program in the MLCP processes the write order, transmitting the data to the device, during which the line protocol handler suspends itself.

7. When the MLCP senses completion of the data transfer, the channel control program returns an interrupt that is initially processed by the communications supervisor and the MLCP driver.

8. The MLCP driver reactivates the line protocol handler (at the interrupt level) to minimally process the interrrupt.

9. When processing is completed, control passes to the MLCP driver.

10. If additional processing is necessary, the line protocol handler can schedule itself, on a noninterrupt basis, to do postinterrupt processing of the interrupt.

11. The application's disconnect request is processed the same as a connect request:

    a. As requested by the communications processor, the channel control program disconnects the physical connection.

    b. The line protocol handler does the necessary logical disconnect processing.

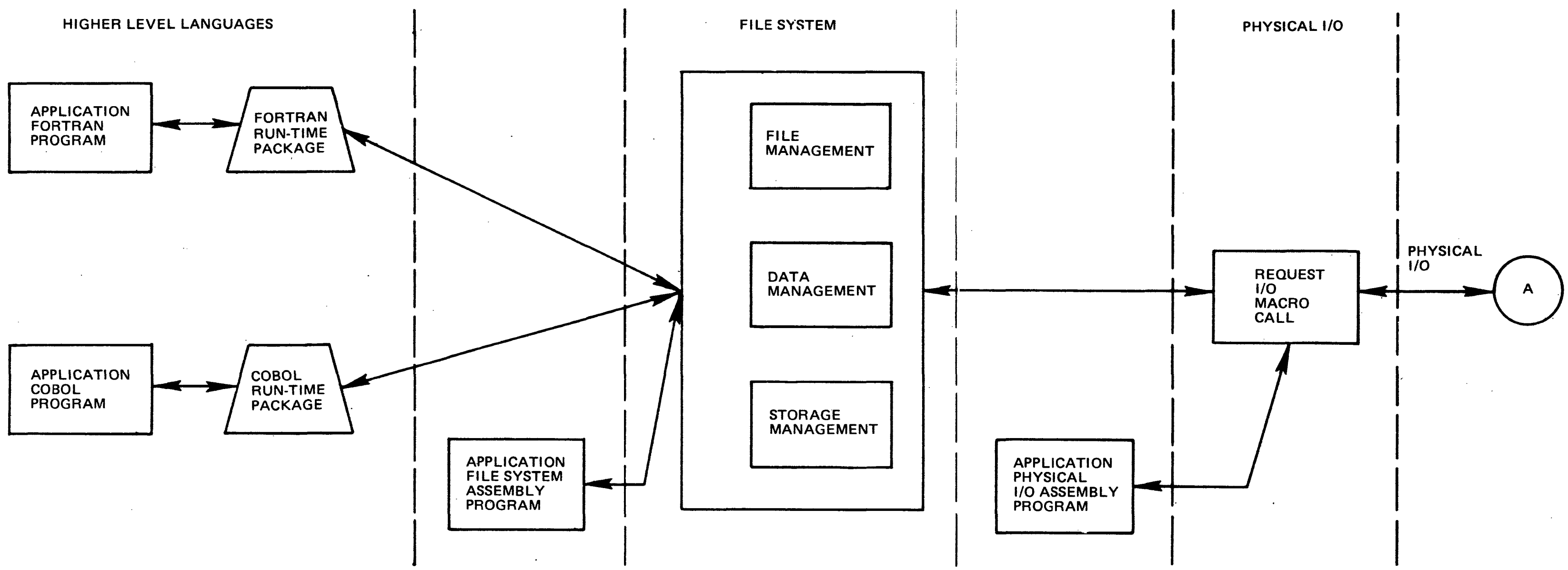
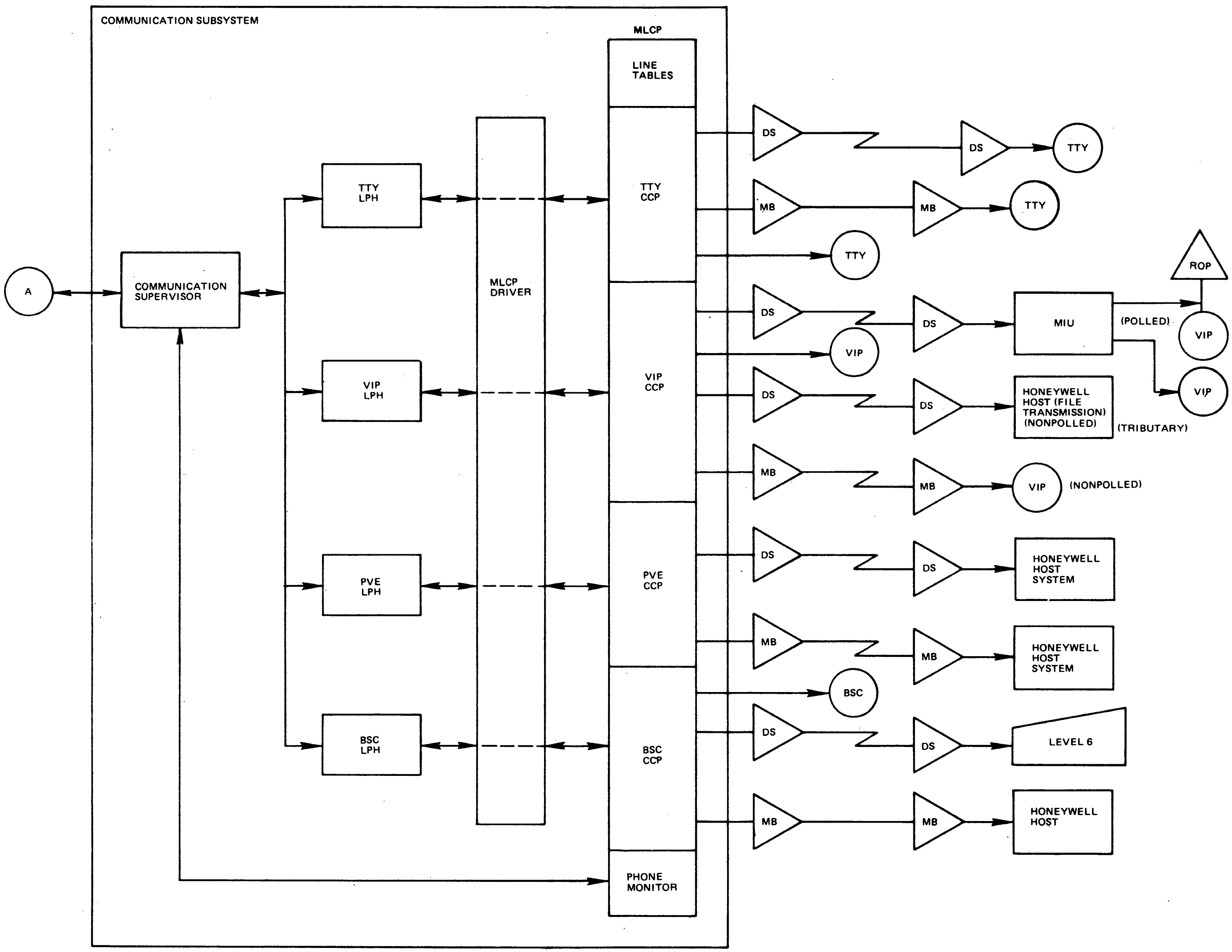


Figure A-1. Simplified Flow - Communications Subsystem



LEGEND

DS = DATASET
MB = MODEM BYPASS

Figure A-1 (cont). Simplified Flow - Communications Subsystem

COMMUNICATIONS SUBSYSTEM ERROR AND CORRECTION PROCEDURES

GCOS uses the following procedures including parity checking, block checking, and time-out, to detect errors occurring over communication lines.

Parity Error Check

The system sends a parity (check) bit with each transmitted character. The parity bit, plus the number of character bits set to 1, will always be an odd- or even-numbered total for every character, according to whether transmission is synchronous or asynchronous. The standard for synchronous transmission is odd parity (total is an odd number); for asynchronous transmission it is even parity (total is an even number).

Block Error Check

GCOS uses two kinds of block error checking, the longitudinal redundancy check (LRC) and the cyclic redundancy check (CRC). Their check characters are known as block check characters (BCC), and the checking calculation result is a block checksum.

LONGITUDINAL REDUNDANCY CHECK (LRC)

The LRC is a form of parity check that is applied to the entire message. The system appends an LRC character, which is an exclusive OR of the message characters, to every message.

The VIP and PVE line protocol handlers use the LRC method.

CYCLIC REDUNDANCY CHECK (CRC)

The CRC method is block oriented. The system transmits data without appending a parity bit on every character. The system computes the CRC character(s) with special algorithms applied to the data to be checked, then appends these characters to the message.

Only the BSC line protocol handler uses the CRC method.

BSC BLOCK CHECK CHARACTER (BCC)

In ASCII transmission, the 8-bit block check character BCC is the result of an exclusive OR operation on all bits received, beginning with the first character following the STX, and ending with the ITB, ETB, or ETX control character. It is based on the polynomial $X^8 + 1$.

In EBCDIC transmission the block check character (BCC) is 16 bits, and is calculated by the system with the checking polynomial $1 + X^2 + X^{15} + X^{16}$.

Time-Out Check

After sending a message, the transmitting device/computer waits for an acknowledgment from the receiving device. When there is no acknowledgment after a specific interval, the sender retransmits the message.

When there is no acknowledgment after a specified number of transmissions, the sender takes whatever action is programmed into the system.

Some procedures provide that the receiving device, on receipt of erroneous data, request retransmission from the sender, using the ACK/NAK response method. (See Appendix E for ACK/NAK definitions.) In this procedure, the sending device waits for an ACK or NAK response (or elapse of the time-out interval) before continuing the communication.

APPENDIX B

CHANGING TERMINAL'S FILE CHARACTERISTICS

Before an application is executed, the user can change the file characteristics of a terminal, e.g., line length or record size, detabbing, device type (input, output, etc.), with the system command STTY (set terminal characteristics) or with the $STTY macro call.

This permits the user to modify those terminal characteristics established at system building.

Table B-1 shows examples of possible values for the device-specific word and file-indicator word arguments of the STTY command and the $STTY macro call (described in the Commands and System Service Macro Calls manuals, respectively).

The table indicates the following:

Column 1 - Device/file operational mode; for BSC, whether advanced or basic data transmission mode.

Column 2 - Input/output operations specified by the corresponding argument values; defined at the bottom of the table.

Column 3 - Argument values for the device-specific word (I_DVS) for the named device, in hexadecimal. See the appropriate device-specific IORB field value tables in Sections 7 through 10.

Column 4 - File-indicator word argument values, in hexadecimal.

NOTE: For BSC, the leading control byte allows EOT, ETB/ETX, and RVI control characters, and transparent mode, to be sent.

Table B-1. Possible Argument Values for STTY Command and $STTY Macro Call

Device/File Operational Mode	Input/Output Operations (See Below)	Device-Specific Argument Value	File Indicator Argument Value
	For TTY		
Interactive	CR, LF, E, CB, PH, QA	0030	3180
Interactive	CR, LF, E, CB, QA	0031	3180
Interactive	CR, LF, E, PH, QA	0830	3180
Interactive	CR, LF, E, QA	0831	3180
Forms	PH, QA, PG	0C0C	3180
Forms	QA, PG	0C0D	3180
Printer Emulation	CR, E, CB, PH, QA	0020	5180
Printer Emulation	CR, E, CB, QA	0021	5180
Data Entry	PH, QA, TR	0C08	3180
Data Entry	QA, TR	0C09	3180
	For VIP		
Interactive	CR, LF, PO, CB, PH, QA, TM, PL	0110	3180
Interactive	CR, LF, PO, CB, QA, TM, PL	0111	3180
Interactive	CR, LF, PO, PH, QA, TM, PL	0910	3180
Interactive	CR, LF, PO, QA, TM, PL	0911	3180
Forms	QA, PL	1909	5180
Forms	PH, QA, PL	1908	3180
Forms	QA, PL	1909	3180
Printer Emulation	CR, CB, PH, QA	0000	5180
Printer Emulation	CR, CB, QA	0001	5180
Receive-only printer	CR, CB, PH, QA	0000	5180
Receive-only printer	CR, CB, QA	0001	5180

Table B-1 (cont). Possible Argument Values for STTY Command and $STTY Macro Call

Device/File Operational Mode	Input/Output Operations (See Below)	Device-Specific Argument Value	File Indicator Argument Value
For PVE (polled VIP emulator)			
	CR, CB, QA	0001	3180
	CR, CB, PH, QA, FC	0080	3180
	For BSC		
Advanced	CB, PH, QA	0000	2980
Advanced	CB, QA	0001	2980
Basic	PH, QA, ETB	0800	2980
Basic	QA, ETB	0801	2980
Basic	PH, QA, TR, ETB	0808	2980
Basic	QA, TR, ETB	0809	2980

CR - Carriage return

LF - Line feed

E - Echo input characters

CB - Control byte

PH - Physical disconnect (hang up)

QA - Queue abort

PG - Page transfer (forms mode)

TR - Transparent text

FC - Hardware function codes present

PO - Page overflow recovery (home cursor)

TM - Time-out on read

PL - 1-second poll interval (ignored if nonpolled line)

ETB - Send ETB/ETX characters

APPENDIX C

RESOURCE CONTROL TABLE (RCT)

The resource control table (RCT) is the interface between the line protocol handler and its devices. For each line protocol handler and device, the system builds an RCT that contains the characteristics uniquely describing that device.

The RCT contains the physical data that the line protocol handler needs to interface with the device. The RCT also includes a work area where every line protocol handler can save whatever values, pointers, etc., that it needs.

Figure C-1 shows the format of an RCT for communications devices. Table C-1 defines the communications-specific items in the RCT. Table C-2 defines the terminal attributes and status field (R_STS).

	0 – 9	A – F
	CHANNEL	LEVEL
R_TYP	LINE ADAPTER TYPE	
R_FLGS	FLAGS (0)	
R_STTS	DEVICE STATUS (0)	
R_STS	TERMINAL ATTRIBUTES AND STATUS	
R_ATTN	ADDRESS OF ATTENTION SUBROUTINE	
R_MSG	MESSAGE COUNT	NAK COUNT

BIT	MEANING IF SET
0 – 5	(RESERVED)
6	CONNECT/DISCONNECT FUNCTIONS SUPPORTED
7	DEVICE RESERVED
8	ATTENTION INTERRUPT HAS OCCURRED
9	DISABLE DEVICE ON ATTENTION INTERRUPT
A	DEVICE DISABLED
B	LPH-CORRECTED ERROR OCCURRED
C	CORRECTED MEMORY ERROR OCCURRED
D	NONEXISTENT RESOURCE ERROR
E	BUS PARITY ERROR
F	UNCORRECTED MEMORY ERROR OCCURRED

NOTE: THE WORD R_FLGS WILL HAVE BIT 6 SET. INDICATING THAT THE CONNECT/DISCONNECT FUNCTIONS ARE ALLOWED.

Figure C-1. Format of Communications Resource Control Table (RCT)

Table C-1. Communications-Specific Items in the RCT

Item	Description	Use
R_STTS	Hardware status	Device hardware status; mapped into software status word I_ST of the (IORB) (see Table 6-3).
R_STS	Terminal attributes and status	See Table C-2 below.
R_MSG		Bits 0 through 7: Count of messages sent to and received from the terminal (maximum 256). For VIP devices, count includes certain control messages exchanged on the line, thus does not represent the number of text messages. Bits 8 through F: Count of NAKs sent to and received from the terminal (maximum 256).

Table C-2. Terminal Attributes and Status Word R_STS of the RCT

Bit	Meaning When Bit Set On
0-9	Reserved for system and later use
A	Device disabled by the system
B	Input possible
C	Output possible
D	Device connected
E	Device physically enabled
F	Device logically enabled

APPENDIX D

SAMPLE APPLICATION PROGRAMS

COBOL PROGRAM EXAMPLES

COBOL TTY or VIP Application Example

The COBOL source program listing in Figure D-1 is an example of an interactive application that involves either VIP or TTY devices.

This program (named CARCOM) processes commands entered from the operator terminal, and includes input/output operations to two communications terminals (either TTY or VIP). An input and output file is assigned to each device. The program uses the operator terminal for entering commands and for receiving error messages. Input/output processing messages are displayed on the line printer.

COMMANDS IN THE COBOL EXAMPLE

The program processes the following interactive commands received from the operator terminal. The command COMND is entered from either terminal 1 or terminal 2 (see "File Assignments" below).

Command	Program Action
OPEN filename	Opens the file
CLOSE filename	Closes the file
ROUTE	Routes terminal output to other terminals as input
GO	Exits command mode, looks for input from terminals
COMND (entered from terminal 1 or 2)	Exits terminal input mode; returns to operator terminal in command mode
STOP	Stops execution

FILE ASSIGNMENTS IN COBOL EXAMPLE

The program CARCOM uses the following file names and corresponding logical file numbers (LFNs):

File Name	LFN	Device
COM1IN	3	Input terminal 1
COM1OT	4	Output terminal 1
COM2IN	5	Input terminal 2
COM2OT	6	Output terminal 2
PRINTER	1	Printer

ERROR MESSAGES IN COBOL EXAMPLE

When appropriate, the COBOL example CARCOM displays these messages, in the formats:

$$\left\{\begin{array}{l}\text{OPEN}\\\text{CLOSE}\\\text{READ}\\\text{WRITE}\end{array}\right\}\ \text{ERROR FILE}\ \left\{\begin{array}{l}\text{COM1IN}\\\text{COM1OT}\\\text{COM2IN}\\\text{COM2OT}\end{array}\right\}\ \text{zz - FILE STATUS}$$

zz = File status code

Program actions resulting from these messages are:

OPEN or CLOSE message:

Returns control to the operator terminal

READ or WRITE message:

Tries the I/O operation four times; then close the file and return control to the operator terminal

STATUS CODES IN COBOL EXAMPLE

The program CARCOM includes checks that verify operation of COBOL error returns and information status returns. The check codes are:

9I - For a read operation, indicates there is no data. For a write operation, indicates that the device is busy.

95 - Record length error.

EXECUTION OF COBOL TTY OR VIP PROGRAM EXAMPLE

When the program begins to execute, the operator terminal displays the message:

TYPE COMMANDS, THEN GO.

At least two files on the same device must be open to proceed to the next level of command input. At this level, the program displays the message:

COMMANDS?

The operator may then enter commands to: (1) open files; (2) close files; (3) route (message switch); (4) activate the read/write loop; or (5) stop.

NOTE: Activating the read/write loop deactivates command input from the console and causes the application to check open terminals for input.

To return to the command level, the operator types COMND from an active terminal.

A typein from a remote terminal is echoed back to that terminal and displayed on the second terminal.

```
GCOS6     COBOL
 SOURCE   PROGRAM

           IDENTIFICATION DIVISION.
           PROGRAM-ID. CARCOM.
          *        COBOL COMMUNICATIONS
           ENVIRONMENT DIVISION.
           CONFIGURATION SECTION.
           SOURCE-COMPUTER. HIS-SERIES-60 LEVEL-6.
           OBJECT-COMPUTER. HIS-SERIES-60 LEVEL-6.
          *
           INPUT-OUTPUT SECTION.
           FILE-CONTROL.
               SELECT COM1IN
                   ASSIGN TO OC-MSD,
                   ORGANIZATION IS SEQUENTIAL WITH VLR,
                   ACCESS MODE IS SEQUENTIAL,
                   FILE STATUS IS IN1-STAT.
               SELECT COM1OT
                   ASSIGN TO OD-MSD,
                   ORGANIZATION IS SEQUENTIAL,
                   ACCESS MODE IS SEQUENTIAL,
                   FILE STATUS IS OT1-STAT.
               SELECT COM2IN
                   ASSIGN TO OE-MSD,
                   ORGANIZATION IS SEQUENTIAL WITH VLR,
                   ACCESS MODE IS SEQUENTIAL,
                   FILE STATUS IS IN2-STAT.
               SELECT COM2OT
                   ASSIGN TO OF-MSD,
                   ORGANIZATION IS SEQUENTIAL,
                   ACCESS MODE IS SEQUENTIAL,
                   FILE STATUS IS OT2-STAT.
               SELECT PRINTFILE
                   ASSIGN TO OA-PRINTER,
                   ORGANIZATION IS SEQUENTIAL,
                   ACCESS MODE IS SEQUENTIAL,
                   FILE STATUS IS PRT-STAT.
          *
           DATA DIVISION.
          *
           FILE SECTION.
           FD  COM1IN
               BLOCK CONTAINS 1 RECORDS,
               LABEL RECORDS ARE OMITTED.
          *
           01  IN1-REC    PIC    X(80).
          *
           FD  COM1OT
               BLOCK CONTAINS 1 RECORDS,
               LABEL RECORDS ARE OMITTED.
          *
           01  OUTCOM1-REC.
               02 CTL1   PIC   X.
               02  OT1-REC   PIC   X(80).
          *
           FD  COM2IN
               BLOCK CONTAINS 1 RECORDS,
               LABEL RECORDS ARE OMITTED.
           01  IN2-REC    PIC    X(80).
          *
           FD  COM2OT
               BLOCK CONTAINS 1 RECORDS,
               LABEL RECORDS ARE OMITTED.
```

Figure D-1. COBOL TTY or VIP Application Example

```
       01  OUTCOM2-REC.
           02  CTL2  PIC  X.
           02  OT2-REC  PIC  X(80).
      *
       FD  PRINTFILE
           BLOCK CONTAINS 1 RECORDS,
           LABEL RECORDS ARE OMITTED.
       01  PRT-REC  PIC  X(120).
      *
       WORKING-STORAGE SECTION.
       01  CTITLE.
           02  FILLER  PIC  XX  VALUE SPACES.
           02  FILLER  PIC  X(15)  VALUE "COBOL COMM TEST".
       01  CCMND1.
           02  FILLER  PIC  XX  VALUE SPACES.
           02  FILLER  PIC  X(27)  VALUE "TYPE FILE COMMANDS, THEN GO".
           02  FILLER  PIC  XX  VALUE SPACES.
       01  CCMND2.
           02  FILLER  PIC  XX  VALUE SPACES.
           02  FILLER  PIC  X(8)  VALUE "COMMAND?".
       01  HEAD1.
           02  FILLER  PIC  X(52)  VALUE SPACES.
           02  FILLER  PIC  X(15)  VALUE "COBOL COMM TEST".
           02  FILLER  PIC  X(53)  VALUE SPACES.
           02  FILLER  PIC  X(53)  VALUE SPACES.
       01  HDR2.
           02  FILLER   PIC   X(6)    VALUE SPACES.
           02  FILLER   PIC   X(27)   VALUE "****  INPUT MSG FILE:  ".
           02  HDR2FIL   PIC   X(6)   VALUE SPACES.
       01  HDR3.
           02  FILLER   PIC   X(6)    VALUE SPACES.
           02  FILLER   PIC   X(28)   VALUE "****  OUTPUT MSG FILE:  ".
           02  HDR3FIL   PIC   X(6)   VALUE SPACES.
       01  LOADCOMP.
           02  FILLER  PIC  XX  VALUE SPACES.
           02  FILLER  PIC  X(13)  VALUE "LOAD COMPLETE".
       01  CONIN.
           02  CMDFLD.
               03  GOLFLD  PIC  X(2)  VALUE SPACES.
               03  FILLER  PIC  X(3)  VALUE SPACES.
           02  FILLER  PIC  X  VALUE SPACES.
           02  FILFLD  PIC  X(6)  VALUE SPACES.
       01  CONIN1 REDEFINES CONIN.
           02  FILLER  PIC X(5).
           02  FILFLD1 PIC X(6).
       01  DSK-REC.
           02  ITEMNUM PIC  XXX  VALUE SPACES.
           02  FILLER  PIC  XX  VALUE SPACES.
           02  DESCFLD PIC  X(20)  VALUE SPACES.
           02  FILLER  PIC  XX  VALUE SPACES.
           02  QTYRED  PIC  9999  VALUE ZERO.
           02  FILLER  PIC  X(30) VALUE SPACES.
       01  IN1-STAT   PIC   XX   VALUE SPACES.
       01  OT1-STAT   PIC   XX   VALUE SPACES.
       01  IN2-STAT   PIC   XX   VALUE SPACES.
       01  OT2-STAT   PIC   XX   VALUE SPACES.
       01  PRT-STAT   PIC   XX   VALUE SPACES.
       01  RDR-STAT   PIC   XX   VALUE SPACES.
       01  INVF-STAT  PIC  XX  VALUE SPACES.
      *
       77  RKEY-I    PIC  999  VALUE ZERO.
       77  DO-IT     PIC  XX  VALUE "GO".
       77  OPNFIL     PIC  X(4)  VALUE "OPEN".
       77  CLSFIL     PIC  X(5)  VALUE "CLOSE".
       77  LOADF  PIC  X(4)  VALUE "LOAD".
```

Figure D-1 (cont). COBOL TTY or VIP Application Example

```
          77  ENDER   PIC  X(4)   VALUE "EOF".
          77  IN1     PIC  X(6)   VALUE "COM1IN".
          77  OT1     PIC  X(6)   VALUE "COM1OT".
          77  IN2     PIC  X(6)   VALUE "COM2IN".
          77  OT2     PIC  X(6)   VALUE "COM2OT".
          77  RDRF   PIC  X(6)  VALUE "CARDIN".
          77  INVF   PIC  X(6)  VALUE "INVFIL".
          77  WHO-ORD   PIC  9  VALUE ZERO.
          77  WHO-ERR   PIC  9  VALUE ZERO.
          77  FILCOUNT      PIC  99  VALUE ZERO.
          77  RTEFLG     PIC  99  VALUE ZERO.
          77  ROUTE      PIC  X(5)  VALUE "ROUTE".
          77  COMDNM     PIC  X(5)  VALUE "COMND".
          77  KEYEQ      PIC  X(13)  VALUE "RELATIVE KEY ".
          77  BDKYNM     PIC  X(13)  VALUE "INVALID KEY= ".
          77  ORDERCMD  PIC  XX  VALUE "O ".
          77  UPDATCMD  PIC  XX  VALUE "U ".
          77  DISPITM   PIC  XX  VALUE "D ".
          77  CCCHAR     PIC  X  VALUE "A".
          77  NOTIFY     PIC  9999  VALUE 9999.
          77  SWITCH1     PIC  99  VALUE ZERO.
          77  SWITCH2     PIC  99  VALUE ZERO.
          77  INVSWTCH  PIC  99  VALUE ZERO.
          77  TRNSWTCH  PIC  99  VALUE ZERO.
          77  STATIN1     PIC  99  VALUE ZERO.
          77  STATOT1     PIC  99  VALUE ZERO.
          77  STATIN2     PIC  99  VALUE ZERO.
          77  STATOT2     PIC  99  VALUE ZERO.
          77  FRSUM1IN  PIC  99  VALUE ZERO.
          77  FRSUM1OT  PIC  99  VALUE ZERO.
          77  FRSUM2IN  PIC  99  VALUE ZERO.
          77  FRSUM2OT  PIC  99  VALUE ZERO.
          77  SUM9T1     PIC  9(4)  VALUE ZERO.
          77  SUM9T2     PIC  9(4)  VALUE ZERO.
          77  QTYSUB  PIC  S9999  VALUE ZERO.
          77  NMCKRSLT  PIC  9  VALUE ZERO.
          77  MAXNUM  PIC  9999  VALUE ZERO.
          77  MAXITMNO  PIC  999  VALUE 200.
          77  MAXQTY  PIC  9999  VALUE 1000.
          77  CHKNUM  PIC  9999  VALUE ZERO.
         *
          01  INSPECTI.
              02  INCMD  PIC  X(5)  VALUE SPACES.
              02  FILLER  PIC  X(75)  VALUE SPACES.
          01  OPDSPL.
              02  FILLER  PIC  XX  VALUE SPACES.
              02  OFLNAM  PIC  X(6)  VALUE SPACES.
              02  FILLER  PIC  XX  VALUE SPACES.
              02  FILLER  PIC  X(6)  VALUE "OPENED".
          01  OPERDSPL.
              02  FILLER  PIC  XX  VALUE SPACES.
              02  FILLER  PIC  X(19)  VALUE "OPEN ERROR   FILE: ".
              02  OFLNER  PIC  X(6)  VALUE SPACES.
              02  FILLER  PIC  X(6)  VALUE SPACES.
              02  FILLER  PIC  X(8)  VALUE "STATUS= ".
              02  KEYERR  PIC  XX  VALUE SPACES.
          01  RDERMSG.
              02  FILLER  PIC  XX  VALUE SPACES.
              02  FILLER  PIC  X(19)  VALUE "READ ERROR   FILE: ".
              02  RDERFIL  PIC  X(6)  VALUE SPACES.
              02  FILLER  PIC  X(6)  VALUE SPACES.
              02  FILLER  PIC  X(8)  VALUE "STATUS= ".
              02  RDERSTAT  PIC  XX  VALUE SPACES.
          01  WRERMSG.
```

Figure D-1 (cont). COBOL TTY or VIP Application Example

```
            02  FILLER   PIC   XX  VALUE SPACES.
            02  FILLER   PIC   X(19)  VALUE "WRITE ERROR  FILE: ".
            02  WRERFIL   PIC   X(6)   VALUE SPACES.
            02  FILLER   PIC   X(6)   VALUE SPACES.
            02  FILLER   PIC   X(8)   VALUE "STATUS= ".
            02  WRERSTAT   PIC   XX   VALUE SPACES.
        01  CLOSPL.
            02  FILLER   PIC   XX  VALUE SPACES.
            02  CFLNAM   PIC   X(6)   VALUE SPACES.
            02  FILLER   PIC   XX  VALUE SPACES.
            02  FILLER   PIC   X(6)   VALUE "CLOSED".
        01  CLERMSG.
            02  FILLER   PIC   XX  VALUE SPACES.
            02  FILLER   PIC   X(19)  VALUE "CLOSE ERROR  FILE: "
            02  CFLNER   PIC   X(6)   VALUE SPACES.
            02  FILLER   PIC   X(6)   VALUE SPACES.
            02  FILLER   PIC   X(8)   VALUE "STATUS= ".
            02  CKEYERR   PIC   XX   VALUE SPACES.
        01  BADFIL.
            02  FILLER   PIC   XX  VALUE SPACES.
            02  FILLER   PIC   X(16)  VALUE "ILLEGAL FILENAME".
        01  BADCMD.
            02  FILLER   PIC   XX  VALUE SPACES.
            02  FILLER   PIC   X(15)  VALUE "ILLEGAL COMMAND".
        01  NOTESUM.
            02  FILLER   PIC   XX  VALUE SPACES.
            02  FILLER   PIC   X(6)   VALUE "FILE: ".
            02  ERR9T    PIC    X(6)   VALUE SPACES.
            02  FILLER   PIC   X(6)   VALUE SPACES.
            02  FILLER   PIC   X(10)  VALUE "STATUS= 9T".
        01  STOPCOB.
            02  FILLER   PIC   XX  VALUE SPACES.
            02  FILLER   PIC   X(10)  VALUE "STOP COBOL".
        01  KEY-MSG.
            02  FILLER   PIC X(16)  VALUE "FILE KEY STATUS ".
            02  BAD-KEY PIC XX   VALUE SPACES.
            02  FILLER   PIC X(12)  VALUE " TEST FAILED".
       *
        PROCEDURE DIVISION.
       *
        PHEADS.
            MOVE CCCHAR TO CTL1.
            MOVE CCCHAR TO CTL2.
            DISPLAY CTITLE.
            OPEN OUTPUT PRINTFILE.
            MOVE HEAD1 TO PRT-REC.
            WRITE PRT-REC AFTER ADVANCING PAGE.
        PCMD1.
            DISPLAY CCMND1.
            MOVE SPACES TO CONTN.
            ACCEPT CONTN.
            IF CMDFLD IS EQUAL TO OPNFIL GO TO OPENIT.
            IF CMDFLD IS EQUAL TO CLSFIL GO TO CLOSIT.
            DISPLAY BADCMD.
            GO TO PCMD1.
        PCMD2.
            DISPLAY CCMND2.
            MOVE SPACES TO CONTN.
            ACCEPT CONTN.
            IF CMDFLD IS EQUAL TO OPNFIL GO TO OPENIT.
            IF CMDFLD IS EQUAL TO CLSFIL GO TO CLOSIT.
            IF CMDFLD IS EQUAL TO ROUTE GO TO SETROUTE.
            IF CMDFLD IS EQUAL TO DO-IT GO TO READ1.
```

Figure D-1 (cont). COBOL TTY or VIP Application Example

```
           DISPLAY BADCMD.
           GO TO PCMD2.
       OPENIT.
           IF FILFLD1 IS EQUAL TO IN1 GO TO OPIN1.
           IF FILFLD1 IS EQUAL TO OT1 GO TO OPOT1.
           IF FILFLD1 IS EQUAL TO IN2 GO TO OPIN2.
           IF FILFLD1 IS EQUAL TO OT2 GO TO OPOT2.
           DISPLAY BADFIL.
           IF FILCOUNT GREATER THAN 1 GO TO PCMD2.
           GO TO PCMD1.
       OPIN1.
           OPEN INPUT COM1IN.
           IF IN1-STAT = "00" OR IN1-STAT = "95";
               MOVE 1 TO STATIN1;
               MOVE 1 TO SWITCH1;
               MOVE IN1 TO OFLNAM;
               GO TO OPMSG.
           MOVE IN1 TO OFLNER.
           MOVE IN1-STAT TO KEYERR.
           GO TO OPERMG.
       OPOT1.
           OPEN OUTPUT COM1OT.
           IF OT1-STAT = "00" OR OT1-STAT = "95";
               MOVE 1 TO STATOT1;
               MOVE OT1 TO OFLNAM;
               GO TO OPMSG.
           MOVE OT1 TO OFLNER.
           MOVE OT1-STAT TO KEYERR.
           GO TO OPERMG.
       OPIN2.
           OPEN INPUT COM2IN.
           IF IN2-STAT = "00" OR IN2-STAT = "95";
               MOVE 1 TO STATIN2;
               MOVE 1 TO SWITCH2;
               MOVE IN2 TO OFLNAM;
               GO TO OPMSG.
           MOVE IN2 TO OFLNER.
           MOVE IN2-STAT TO KEYERR.
           GO TO OPERMG.
       OPOT2.
           OPEN OUTPUT COM2OT.
           IF OT2-STAT = "00" OR OT2-STAT = "95";
               MOVE 1 TO STATOT2;
               MOVE OT2 TO OFLNAM;
               GO TO OPMSG.
           MOVE OT2 TO OFLNER.
           MOVE OT2-STAT TO KEYERR.
           GO TO OPERMG.
       OPMSG.
           DISPLAY OPDSPL.
           ADD 1 TO FILCOUNT.
           IF FILCOUNT GREATER THAN 1 GO TO PCMD2.
           GO TO PCMD1.
       OPERMG.
           DISPLAY OPERDSPL.
           IF FILCOUNT GREATER THAN 1 GO TO PCMD2.
           GO TO PCMD1.
       CLOSIT.
           IF FILFLD IS EQUAL TO IN1 GO TO CLIN1.
           IF FILFLD IS EQUAL TO OT1 GO TO CLOT1.
           IF FILFLD IS EQUAL TO IN2 GO TO CLIN2.
           IF FILFLD IS EQUAL TO OT2 GO TO CLOT2.
           DISPLAY BADCMD.
```

Figure D-1 (cont). COBOL TTY or VIP Application Example

```
               IF FILCOUNT GREATER THAN 1 GO TO PCMD2.
               GO TO PCMD1.
           CLIN1.
               CLOSE COM1IN.
               IF IN1-STAT = "00":
                   MOVE ZERO TO SWITCH1:
                   MOVE ZERO TO STATIN1:
                   MOVE IN1 TO CFLNAM:
                   GO TO CLOPMSG.
               MOVE IN1 TO CFLNER.
               MOVE IN1-STAT TO CKEYERR.
               GO TO COPERMG.
           CLOT1.
               CLOSE COM1OT.
               IF OT1-STAT = "00":
                   MOVE ZERO TO STATOT1:
                   MOVE OT1 TO CFLNAM:
                   GO TO CLOPMSG.
               MOVE OT1 TO CFLNER.
               MOVE OT1-STAT TO CKEYERR.
               GO TO COPERMG.
           CLIN2.
               CLOSE COM2IN.
               IF IN2-STAT = "00":
                   MOVE ZERO TO SWITCH2:
                   MOVE ZERO TO STATIN2:
                   MOVE IN2 TO CFLNAM:
                   GO TO CLOPMSG.
               MOVE IN2 TO CFLNER.
               MOVE IN2-STAT TO CKEYERR.
               GO TO COPERMG.
           CLOT2.
               CLOSE COM2OT.
               IF OT2-STAT = "00":
                   MOVE ZERO TO STATOT2:
                   MOVE OT2 TO CFLNAM:
                   GO TO CLOPMSG.
               MOVE OT2 TO CFLNER.
               MOVE OT2-STAT TO CKEYERR.
               GO TO COPERMG.
           CLOPMSG.
               DISPLAY CLOSPL.
               SUBTRACT 1 FROM FILCOUNT.
               IF FILCOUNT GREATER THAN 1 GO TO PCMD2.
               GO TO PCMD1.
           COPERMG.
               DISPLAY CLERMSG.
               IF FILCOUNT GREATER THAN 1 GO TO PCMD2.
               GO TO PCMD1.
           SETROUTE.
               IF STATIN1 = 1 AND STATOT2 = 1 GO TO OKSET.
               IF STATIN2 = 1 AND STATOT1 = 1 GO TO OKSET.
               DISPLAY BADCMD.
               GO TO PCMD2.
           OKSET.
               MOVE 1 TO RTEFLG.
               GO TO PCMD2.
           READ1.
               IF FILCOUNT = ZERO GO TO PCMD1.
               IF SWITCH1 = ZERO GO TO READ2.
               MOVE SPACES TO IN1-REC.
               READ COM1IN AT END GO TO DONEIT.
               IF IN1-STAT = "00" GO TO GOODR1.
               IF IN1-STAT = "9T":
```

Figure D-1 (cont). COBOL TTY or VIP Application Example

```
                GO TO READ2.
           MOVE ZERO TO SUM9I1.
           MOVE IN1-STAT TO RDERSTAT.
           MOVE IN1 TO RDERFIL.
           DISPLAY RDERMSG.
           ADD 1 TO ERSUM1IN.
           IF ERSUM1IN NOT LESS THAN 4 GO TO CLIN1.
       READ2.
           IF SWITCH2 = ZERO GO TO READ1.
           MOVE SPACES TO IN2-REC.
           READ COM2IN AT END GO TO DONEIT.
           IF IN2-STAT = "00" GO TO GOODR2.
           IF IN2-STAT = "9I";
                GO TO READ1.
           MOVE ZERO TO SUM9I2.
           MOVE IN2-STAT TO RDERSTAT.
           MOVE IN2 TO RDERFIL.
           DISPLAY RDERMSG.
           ADD 1 TO ERSUM2IN.
           IF ERSUM2IN NOT LESS THAN 4 GO TO CLIN2.
           GO TO READ1.
       GOODR1.
           MOVE ZERO TO ERSUM1IN.
           MOVE ZERO TO SUM9I1.
           PERFORM PRTIN1 THRU CHK9IPT1.
           MOVE IN1-REC TO INSPECTI.
           IF INCMD IS EQUAL TO COMDNM GO TO RCMD2.
           IF RTEFLG IS NOT EQUAL TO ZERO;
                MOVE IN1-REC TO OT2-REC;
                GO TO WRITE2.
           MOVE IN1-REC TO OT1-REC.
           GO TO WRITE1.
       PRTIN1.
           MOVE IN1 TO HDR2FIL.
           MOVE HDR2 TO PRT-REC.
           WRITE PRT-REC.
           MOVE SPACES TO PRT-REC.
           MOVE IN1-REC TO PRT-REC.
       CHK9IPT1.
           WRITE PRT-REC.
           IF PRT-STAT = "9I" GO TO CHK9IPT1.
       WRITE1.
           WRITE OUTCOM1-REC.
           IF OT1-STAT = "00" GO TO WRT1OK.
           IF OT1-STAT = "9I" GO TO WRITE1.
           MOVE OT1-STAT TO WRERSTAT.
           MOVE OT1 TO WRERFIL.
           DISPLAY WRERMSG.
           ADD 1 TO ERSUM1OT.
           IF ERSUM1OT NOT LESS THAN 4 GO TO CLOT1.
           GO TO READ2.
       WRT1OK.
           MOVE ZERO TO ERSUM1OT.
           PERFORM PRTOT1 THRU CHK9IPO1.
           GO TO READ2.
       PRTOT1.
           MOVE OT1 TO HDR3FIL.
           MOVE HDR3 TO PRT-REC.
           WRITE PRT-REC.
           MOVE SPACES TO PRT-REC.
           MOVE OT1-REC TO PRT-REC.
       CHK9IPO1.
           WRITE PRT-REC.
           IF PRT-STAT = "9I" GO TO CHK9IPO1.
```

Figure D-1 (cont). COBOL TTY or VIP Application Program

```
          GOODR2.
              MOVE ZERO TO ERSUM2IN.
              MOVE ZERO TO SUM9I2.
              PERFORM PRTIN2 THRU CHK9IPI2.
              MOVE IN2-REC TO INSPECTI.
              IF INCMD IS EQUAL TO COMDNM GO TO PCMD2.
              IF RTEFLG IS NOT EQUAL TO ZERO:
                  MOVE IN2-REC TO OT1-REC;
                  GO TO WRITE1.
              MOVE IN2-REC TO OT2-REC.
              GO TO WRITE2.
          PRTIN2.
              MOVE IN2 TO HDR2FIL.
              MOVE HDR2 TO PRT-REC.
              WRITE PRT-REC.
              MOVE SPACES TO PRT-REC.
              MOVE IN2-REC TO PRT-REC.
          CHK9IPI2.
              WRITE PRT-REC.
              IF PRT-STAT = "9I" GO TO CHK9IPI2.
          WRITE2.
              WRITE OUTCOM2-REC.
              IF OT2-STAT = "00" GO TO WRT2OK.
              IF OT2-STAT = "9I" GO TO WRITE2.
              MOVE OT2-STAT TO WRERSTAT.
              MOVE OT2 TO WRERFIL.
              DISPLAY WRERMSG.
              ADD 1 TO ERSUM2OT.
              IF ERSUM2OT NOT LESS THAN 4 GO TO CLOT2.
              GO TO READ1.
          WRT2OK.
              MOVE ZERO TO ERSUM2OT.
              PERFORM PRTOT2 THRU CHK9IPO2.
              GO TO READ1.
          PRTOT2.
              MOVE OT2 TO HDR3FIL.
              MOVE HDR3 TO PRT-REC.
              WRITE PRT-REC.
              MOVE SPACES TO PRT-REC.
              MOVE OT2-REC TO PRT-REC.
          CHK9IPO2.
              WRITE PRT-REC.
              IF PRT-STAT = "9I" GO TO CHK9IPO2.
          DONEIT.
              DISPLAY STOPCOR.
              STOP RUN.
          END COBOL
NO DIAGNOSTICS

GCOS6     COBOL
FILE  MAP
LINE  LFN IFN

   11  03  0C-MSD          COM1IN                      0103     80
   16  04  0D-MSD          COM1OT                      01FB     81
   21  05  0E-MSD          COM2IN                      0224     80
   26  06  0F-MSD          COM2OT                      024C     81
   31  01  0A-PRINTER      PRINTFILE                   0276    120
```

Figure D-1 (cont). COBOL TTY or VIP Application Example

COBOL BSC Application Example

The source program listing in Figure D-2 is an example of a COBOL communications program to test BSC file transmission by:

1. Generating records

2. Transmitting the records over one communication line

3. Reading them back over another communication line for comparison

The program name is BSCTST. When executed, it displays the following error messages, as appropriate:

Error format 1:

```
BSC TEST FILE- { INPUT  } PROBLEM- { OPEN  } STATUS - zz
               { OUTPUT }          { CLOSE }
                                   { READ  }
                                   { WRITE }
```

zz=9I - Device busy
zz=00 - Program may read or write

Program action: Issues reads and writes four times; then the file is closed and the program terminated.

Error format 2:

BSC - TEST - NO MATCH RECORD nnnn

Program action: Reading application does not receive the expected record; records out of sequence or garbled.

File is closed and the program terminated.

GCOS6   COBOL
SOURCE   PROGRAM

```
          IDENTIFICATION DIVISION.
          PROGRAM-ID.  BSCTST.
         * THIS IS A PROGRAM WHICH TESTS BSC FILE TRANSMISSION -
         * IT DOES SO BY GENERATING RECORDS , SENDING THEM OUT
         * AND BRINGING THEM BACK IN FOR COMPARISON
         * FOR A MORE DETAILED DESCRIPTION REFER TO THE GCOS 6.1
         * TEST SPECIFICAITON FOR COBOL COMMUNICATIONS
          ENVIRONMENT DIVISION.
          CONFIGURAITON SECITON.
          SOURCE-COMPUTER. HIS-SERIES-60    LEVEL-6.
          OBJECT-COMPUTER. HIS-SERIES-60 LEVEL-6.
         *
          INPUT-OUTPUT SECTION.
          FILE-CONTROL.
         *
              SELECT T-OUTPUT
              ASSIGN TO OI,
              ORGANIZATION IS SEQUENTIAL WITH VLR,
              ACCESS IS SEQUENTIAL,
              FILE STATUS IS OUT-STAT.
              SELECT T-INPUT
              ASSIGN TO AO,
              ORGANIZATION IS SEQUENTIAL WITH VLR,
              ACCESS IS SEQUENTIAL,
              FILE STATUS IS IN-STAT.
         *
          DATA DIVISION.
         *
          FILE SECTION.
          FD  T-OUTPUT
              BLOCK CONTAINS 1 RECORDS,
              LABEL RECORDS ARE STANDARD.
          01  OUT-REC     PIC X(80).
         *
          FD  T-INPUT
              BLOCK CONTAINS 1 RECORDS,
              LABEL RECORDS ARE STANDARD.
          01  IN-REC    PIC X(80).
         *
          WORKING-STORAGE SECITON.
         *
          77  IN-STAT    PIC XX     VALUE SPACES.
          77  OUT-STAT   PIC XX            VALUE SPACES.
          77  MAX-CNT           PIC 9999          VALUE 1001.
          77  W-INPUT           PIC X(6)          VALUE "INPUT ".
          77  W-OUTPUT            PIC X(6)          VALUE "OUTPUT".
          77  W-OPEN          PIC X(5)          VALUE "OPEN ".
          77  W-CLOSE   PIC X(5)  VALUE "CLOSE".
          77  W-READ          PIC X(5)          VALUE "READ ".
          77  W-WRITE   PIC X(5)           VALUE "WRITE".
          01  TEST-REC.
              02  FILLER  PIC X(12)  VALUE "TEST RECORD ".
              02  TR-CNT  PIC 9999  VALUE ZERO.
              02  FILLER  PIC X(54)  VALUE SPACES.
              02  FILLER PIC X(10)   VALUE "**********".
          01  EOF-REC.
              02 FILLER   PIC X(3)   VALUE "EOF".
              02  FILLER  PIC X(77)    VALUE SPACES.
          01  ER-MSG1.
```

Figure D-2. COBOL BSC Application Example

```
           02  FILLER    PIC X(16)   VALUE "BSC TEST- FILE- ".
           02  E-FILE    PIC X(6)    VALUE SPACES.
           02  FILLER    PIC X(10)   VALUE " PROBLEM- ".
           02 E-TYPE  PIC X(5)   VALUE SPACES.
           02  FILLER     PIC X(9)    VALUE " STATUS- ".
           02  E-STAT     PIC XX      VALUE SPACES.
       01  ER-MSG2.
           02  FILLER  PIC X(28)  VALUE "BSC TEST- NO MATCH, RECORD- ".
           02  BAD-REC    PIC 9(4)     VALUE ZEROES.
       01  EOT-MSG.
           02 FILLER PIC X(9) VALUE "BSC EOT- ".
           02 FINAL-CNT PIC 9(4) VALUE ZEROES.
           02 FILLER PIC X(20) VALUE " RECORDS TRANSMITTED".
      *
       PROCEDURE DIVISION.
       HSEKEEP.
           MOVE ZEROES TO TR-CNT.
       OPEN-UP.
           OPEN INPUT T-INPUT.
           IF IN-STAT NOT EQUAL "00"; MOVE W-OPEN TO E-TYPE;
               GO TO IN-ERR.
           OPEN OUTPUT T-OUTPUT.
           IF OUT-STAT NOT EQUAL "00"; MOVE W-OPEN TO E-TYPE;
               GO TO OUT-ERR.
      *
      *
       MASTER.
           ADD 1 TO TR-CNT.
           MOVE TEST-REC TO OUT-REC.
       READ1.
           READ T-INPUT AT END; MOVE TR-CNT TO FINAL-CNT;
               DISPLAY EOT-MSG; GO TO CLOSE-UP.
           IF IN-STAT = "00"; GO TO COMPARE.
           IF IN-STAT = "9I"; GO TO WRITE1.
           MOVE W-READ TO E-TYPE.
           GO TO IN-ERR.
       WRITE1.
           WRITE OUT-REC.
           IF OUT-STAT = "00"; GO TO COMPARE.
           IF OUT-STAT = "9I"; GO TO WRITE1.
           MOVE W-WRITE TO E-TYPE.
           GO TO OUT-ERR.
       COMPARE.
           IF IN-REC IS EQUAL TO TEST-REC; GO TO MASTER.
           IF OUT-REC = EOF-REC; GO TO CLOSE2.
           MOVE TR-CNT TO BAD-REC.
           DISPLAY ER-MSG2.
           GO TO STOP-PG.
      *
      *
       IN-ERR.
           MOVE W-INPUT TO E-FILE.
           MOVE IN-STAT TO E-STAT.
           GO TO OP-MSG.
       OUT-ERR.
           MOVE W-OUTPUT TO E-FILE.
           MOVE OUT-STAT TO E-STAT.
       OP-MSG.
           DISPLAY ER-MSG1.
           GO TO STOP-PG.
      *
        CLOSE-UP.
           CLOSE T-INPUT.
```

Figure D-2 (cont). COBOL BSC Application Example

```
                    IF IN-STAT IS NOT EQUAL "00": MOVE W-CLOSE TO E-TYPE;
                        GO TO IN-ERR.
                    GO TO STOP-PG.
               CLOSE2.
                    CLOSE T-OUTPUT.
                    IF OUT-STAT IS NOT EQUAL "00"; MOVE W-CLOSE TO E-TYPE;
                        GO TO OUT-ERR.
                    GO TO MASTER.

               STOP-PG.
                    STOP RUN.
               END COBOL.
NO DIAGNOSTICS

 GCOS6     COBOL
 FILE   MAP
 LINE   LFN IFN

   16  09  OT-MSD          T-OUTPUT                              009F       80
   21  10  AO-MSD          T-INPUT                               00C7       80
```

Figure D-2 (cont). COBOL BSC Application Example

FORTRAN Application Example for TTY

The FORTRAN source program (program name FORCL4) listing shown in Figure D-3 is an example of a FORTRAN application program involving a TTY remote device.

The program processes eight message groups before terminating. It first issues four data messages to the remote terminal and to the operator terminal. It issues the write requests from alternate data buffers to ascertain the status of the interfaces among the file system, FORTRAN Compiler, and the communications subsystem. When the four initial message groups are complete, the program requests input data from the operator terminal.

After the operator enters a message, the operator terminal displays the message and an acknowledgment message. When the fourth message is received, the application program terminates.

Every input message, which is preceded by a blank or NUL character that is not displayed, may have up to 59 ASCII characters.

The system continually monitors the status register, displaying error condition codes or status messages on the operator terminal. For example, a condition indicating no data available (buffer busy) at the remote device, lasting more than 20 seconds, causes a status return code of 516_{10}. The program continues the read attempt since that status is not an error condition. The read statement is issued only after a status code 0_{10} is returned to indicate that data is available (buffer not busy).

```
FORCL4       GCOS6-1 FORTRAN  REV: 0101   D     1977/04/20 1540:00.3    PAGE: 02
C
C   OUTPUT MESSAGES TO REMOTE DEVICE (LFN 9)
C   4 MESSAGES ISSUED TO DEVICE AND LFN4
C   FROM ALTERNATING BUFFERS
C
 70   WRITE(9,80)CW3,N
 80   FORMAT(1X,A48,I2)
      WRITE(4,80)CW3,N
      GO TO 20
 90   WRITE(9,80)CW4,N
      WRITE(4,80)CW4,N
      IF(N .EQ. 4) GO TO 15
      GO TO 20
C
C   INPUT FROM REMOTE DEVICE (LFN 8)
C   4 MESSAGES ALLOWED
C
C        SPACE 1 CHARACTER AND TYPE UP TO 59 CHARACTERS
C        FOLLOWED BY A CARRIAGE RETURN
C        TYPE SECOND MESSAGE WHEN DEVICE   TYPES
C        "MESSAGE X RECD"
C
 100  READ(8,110)CR1
 110  FORMAT(1X,60A1)
      WRITE(4,110)CR1
 112  CALL ZFSTOT(9,ISTAT)
      IF(ISTAT .EQ. 0)GO TO 114
      GO TO 112
 114  WRITE(9,115)N
 115  FORMAT(1X,'MESSAGE ',I2,' RECD')
      IF(N .NE. 8)GO TO 20
      GO TO 130
 120  READ(8,110)CR2
      WRITE(4,110)CR2
 121  CALL ZFSTOT(9,ISTAT)
      IF(ISTAT .EQ. 0)GO TO 125
      GO TO 121
 125  WRITE(9,115)N
      IF(N .NE. 8)GO TO 20
C
C CLOSE UNITS AND EXIT
C
 130  CALL ZFSTOT(9,ISTAT)
      IF(ISTAT .EQ. 0) GO TO 140
      GO TO 130
 140  CLOSE(UNIT=8)
      CLOSE(UNIT=9)
      STOP
      END
  0  DIAGNOSTICS
```

Figure D-3. FORTRAN Application Example for TTY

```
FORCL4       GCOS6-1 FORTRAN  REV: 0101    D      1977/04/20 1540:00.3     PAGE: 01

 C       FORTRAN COMMUNICATION PROGRAM - FORCL4
 C
 C          ILLUSTRATES USE OF ZFSTIN AND ZFSTOT
 C
 C          WRITES 4 MESSAGES TO THE OPERATOR'S TERMINAL (LFN 4)
 C          AND SEND TO A REMOTE DEVICE (IE. TTY) ON LFN 9 VIA MLCP
 C          FOLLOWED BY A READ OF 4 MESSAGES FROM THE SAME REMOTE
 C          DEVICE (IE. TTY) ON LFN 8. ALL MESSAGES ARE DISPLAYED
 C          ON THE OPERATOR'S CONSOLE, AND RECEIVED MESSAGES ARE
 C          ACKNOWLEDGED ON THE REMOTE DEVICE
 C          DEVICE STATUS IS REPORTED USING,
 C               CALL ZFSTIN(I,J) FOR INPUT, AND
 C               CALL ZFSTOT(I,J) FOR OUTPUT.
 C
         PROGRAM FORCL4
         CHARACTER *48 CW3,CW4
         CHARACTER CR1(60),CR2(60)
         DATA CW3/'THIS IS COMM. OUTPUT TO THE TTY - MESSAGE NUMBER'/
 C
         J = 0
         N = 0
         K = 9
         CW4 = CW3
         OPEN(UNIT=8)
         OPEN(UNIT=9)
         GO TO 20
   15    K = 8
 C
 C       CHECK COMMUNICATION DEVICE STATUS
 C       USING ZFSTIN OR ZFSTOT ROUTINE
 C
   20    N = N + 1
   25    J = 0
   30    IF(K.EQ.8)CALL ZFSTIN(K,ISTAT)
         IF(K.EQ.9)CALL ZFSTOT(K,ISTAT)
         IF(ISTAT .EQ. 0) GO TO (70,90,70,90,100,120,100,120),N
         IF(ISTAT - 516)50,40,50
   40    J = J + 1
         IF(J .LT. 10000) GO TO 30
   50    WRITE(4,60)N,ISTAT
   60    FORMAT(1X,'STATUS RTN MESSAGE NO.',I2,' STATUS TYPE',I4)
         IF(ISTAT .EQ. 516) GO TO 25
         GO TO 140
```

Figure D-3 (cont). FORTRAN Application Example for TTY

Assembly Language Example for TTY or VIP Using Physical I/O

Figure D-4 shows an assembly language source program (SENDER), using Physical I/O, that tests TTY or VIP terminals by sending character strings to the terminals.

The user enters SENDER 07 to test a TTY terminal, or SENDER 0A to test a VIP terminal. The values 07 and 0A are the logical resource numbers (LRNs) of the TTY and VIP, respectively.

The program will halt on the first instruction, and will continue when the Execute button is pressed.

```
         title     sender
*
         libm      exec_lib
         xdef      sender
*
sender   hlt
         ldv       $r3,0                $r3 <- default lrn
         ldr       $r7,+$b7             $r7 <- parameter count
         cmv       $r7,2                test parameter count < 2
         bl        >+$a
         ldb       $b6,+$b7             $b6 <- a(p1 char count)
         ldr       $r6,+$b6             $r6 <- p1 char count
         ldb       $b5,+$b7             $b5 <- a(p2 char count)
         ldr       $r5,+$b5             $r5 <- p2 char count
         ldv       $r1,2                $r1 <- 2 = invalid lrn
         cmv       $r5,2                test char count > 2
         bg        exit
         ldv       $r1,0                $r1 <- 0
         llh       $r1,$b5.$r1          $r1 <- 1st char (ascii)
         ldh       $r3,<tab.$r1         $r3 <- 1st char (hex)
         blz       $r3,exit             test for bad char
         ldv       $r1,1                $r1 <- 1
         llh       $r1,$b5.$r1          $r1 <- 2nd char (ascii)
         ldh       $r1,<tab.$r1         $r1 <- 2nd char (hex)
         blz       $r1,exit             test for bad char
         sol       $r3,4                $r3 <- $r3*16
         or        $r3,=$r1             $r3 <- hex lrn
$a       ldv       $r4,-14              $r4 <- iorb count
         lab       $b4,iorb00           $b4 <- a(1st iorb)
$b       sth       $r3,$b4.$af+1        $r3 -> lrn
         $RQIO,
         nop       >$+2                 trace
         bnez      $r1,>exit            test for error
         lab       $b4,$b4.$af*2+6      $b4 <- a(next iorb)
         binc      $r4,>-$b             test iorb count = o
         ldv       $r1,0                $r1 <- 0 = success
exit     ldr       $r2,=$r1             $r2 <- error code
         $TRMRQ,
*
iorb00   resv      $af,0
         dc        x'01'
         dc        x'0a'
         resv      $af,0
         dc        0
         dc        0
         dc        0
         dc        0
iorb20   resv      $af,0
         dc        x'41'
         dc        x'41'
         dc        <msg20
         dc        43
         dc        x'20'
         dc        0
         dc        0
iorb28   resv      $af,0
         dc        x'41'
         dc        x'41'
```

Figure D-4. Assembly Language Example for TTY or VIP Using Physical I/O

```
        dc      <msg28
        dc      43
        dc      x'20'
        dc      0
        dc      0
iorb30  resv    $af,0
        dc      x'01'
        dc      x'41'
        dc      <msg30
        dc      43
        dc      x'20'
        dc      0
        dc      0
iorb38  resv    $af,0
        dc      x'41'
        dc      x'41'
        dc      <msg38
        dc      43
        dc      x'20'
        dc      0
        dc      0
iorb40  resv    $af,0
        dc      x'41'
        dc      x'41'
        dc      <msg40
        dc      43
        dc      x'20'
        dc      0
        dc      0
iorb48  resv    $af,0
        dc      x'01'
        dc      x'41'
        dc      <msg48
        dc      43
        dc      x'20'
        dc      0
        dc      0
iorb50  resv    $af,0
        dc      x'41'
        dc      x'41'
        dc      <msg50
        dc      43
        dc      x'20'
        dc      0
        dc      0
iorb58  resv    $af,0
        dc      x'41'
        dc      x'41'
        dc      <msg58
        dc      43
        dc      x'20'
        dc      0
        dc      0
iorb60  resv    $af,0
        dc      x'01'
        dc      x'41'
        dc      <msg60
        dc      43
        dc      x'20'
        dc      0
        dc      0
```

Figure D-4 (cont). Assembly Language Example for TTY or VIP Using Physical I/O

```
iorb68    resv    $af,0
          dc      x'41'
          dc      x'41'
          dc      <msg68
          dc      43
          dc      x'20'
          dc      0
          dc      0
iorb70    resv    $af,0
          dc      x'01'
          dc      x'41'
          dc      <msg70
          dc      43
          dc      x'20'
          dc      0
          dc      0
iorb78    resv    $af,0
          dc      x'41'
          dc      x'41'
          dc      <msg78
          dc      43
          dc      x'20'
          dc      0
          dc      0
iorb99    resv    $af,0
          dc      x'01'
          dc      x'0b'
          resv    $af,0
          dc      0
          dc      x'03'
          dc      0
          dc      0
msg20     dc      x'42'
          text    '20 21 22 23 24 25 26 27 '
          dc      z'202020202120222023202420252026202720'
msg28     dc      x'41'
          text    '28 29 2A 2B 2C 2D 2E 2F '
          dc      z'2020282029202a202b202c202d202e202f20'
msg30     dc      x'41'
          text    '30 31 32 33 34 35 36 37 '
          dc      z'202030203120322033203420352036203720'
msg38     dc      x'41'
          text    '38 39 3A 3B 3C 3D 3E 3F '
          dc      z'2020382039203a203b203c203d203e203f20'
msg40     dc      x'41'
          text    '40 41 42 43 44 45 46 47 '
          dc      z'202040204120422043204420452046204720'
msg48     dc      x'41'
          text    '48 49 4A 4B 4C 4D 4E 4F '
          dc      z'2020482049204a204b204c204d204e204f20'
msg50     dc      x'41'
          text    '50 51 52 53 54 55 56 57 '
          dc      z'202050205120522053205420552056205720'
msg58     dc      x'41'
          text    '58 59 5A 5B 5C 5D 5E 5F '
          dc      z'2020582059205a205b205c205d205e205f20'
msg60     dc      x'41'
          text    '60 61 62 63 64 65 66 67 '
          dc      z'202060206120622063206420652066206720'
msg68     dc      x'41'
          text    '68 69 6A 6B 6C 6D 6E 6F '
```

Figure D-4 (cont). Assembly Language Example for TTY or VIP Using Physical I/O

```
         dc       z'2020682069206a206b206c206d206e206f20'
msg70    dc       x'41'
         text     '70 71 72 73 74 75 76 77 '
         dc       z'2020702071207220732074207520762077 20'
msg78    dc       x'41'
         text     '78 79 7A 7B 7C 7D 7E 7F '
         dc       z'2020782079207a207b207c207d207e207f20'
*
tab      dc       z'80808080'   00 01 02 03
         dc       z'80808080'   04 05 06 07
         dc       z'80808080'   08 09 0A 0B
         dc       z'80808080'   0C 0D 0E 0F
         dc       z'80808080'   10 11 12 13
         dc       z'80808080'   14 15 16 17
         dc       z'80808080'   18 19 1A 1B
         dc       z'80808080'   1C 1D 1E 1F
         dc       z'80808080'   20 21 22 23
         dc       z'80808080'   24 25 26 27
         dc       z'80808080'   28 29 2A 2B
         dc       z'80808080'   2C 2D 2E 2F
         dc       z'00010230'   30 31 32 33
         dc       z'04050607'   34 35 36 37
         dc       z'08098080'   38 39 3A 3B
         dc       z'80808080'   3C 3D 3E 3F
         dc       z'800a0b0c'   40 41 42 43
         dc       z'0d0e0f80'   44 45 46 47
         dc       z'80808080'   48 49 4A 4B
         dc       z'80808080'   4C 4D 4E 4F
         dc       z'80808080'   50 51 52 53
         dc       z'80808080'   54 55 56 57
         dc       z'80808080'   58 59 5A 5B
         dc       z'80808080'   5C 5D 5E 5F
         dc       z'800a0b0c'   60 61 62 63
         dc       z'0d0e0f80'   64 65 66 67
         dc       z'80808080'   68 69 6A 6B
         dc       z'80808080'   6C 6D 6E 6F
         dc       z'80808080'   70 71 72 73
         dc       z'80808080'   74 75 76 77
         dc       z'80808080'   78 79 7A 7B
         dc       z'80808080'   7C 7D 7E 7F

         end      sender,sender
```

Figure D-4 (cont). Assembly Language Example for TTY or VIP Using Physical I/O

APPENDIX E

ASCII AND EBCDIC CONTROL CHARACTERS AND CHARACTER SETS

Tables E-1 and E-2 illustrate the ASCII and EBCDIC character sets, respectively. In addition to the ASCII characters, Table E-1 shows the hexadecimal equivalents; Table E-2 shows the binary and hexadecimal equivalents of the EBCDIC character set.

Following are lists of the control characters and special graphic characters that appear in the two tables:

<u>CONTROL CHARACTERS</u>

ACK	Acknowledge
BEL	Bell
BS	Backspace
BYP	Bypass
CAN	Cancel
CC	Cursor Control
CR	Carriage Return
CU1	Customer Use 1
CU2	Customer Use 2
CU3	Customer Use 3
DC1	Device Control 1
DC2	Device Control 2
DC3	Device Control 3
DC4	Device Control 4
DEL	Delete
DLE	Data Link Escape
DS	Digit Select
EM	End of Medium
ENQ	Enquiry
EO	Eight Ones
EOT	End of Transmission
ESC	Escape
ETB	End of Transmission Block
ETX	End of Text
FF	Form Feed
FS	Field Separator
GE	Graphic Escape
GS	Group Separator
HT	Horizontal Tab
IFS	Interchange File Separator
IGS	Interchange Group Separator
IL	Idle
IRS	Interchange Record Separator
IUS	Interchange Unit Separator
LC	Lowercase
LF	Line Feed
NAK	Negative Acknowledgment
NL	New Line
NUL	Null
PF	Punch Off
PN	Punch On
RES	Restore
RLF	Reverse Line Feed
RS	Reader Stop
SI	Shift In
SM	Set Mode
SMM	Start of Manual Message
SO	Shift Out
SOH	Start of Heading
SOS	Start of Significance
SP	Space
STX	Start of Text
SUB	Substitute
SYN	Synchronous Idle
TM	Tape Mark
UC	Uppercase
US	Unit Separator
VT	Vertical Tab

SPECIAL GRAPHIC CHARACTERS

¢ Cent Sign
. Period, Decimal Point
< Less-than Sign
(Left Parenthesis
+ Plus Sign
¦ Logical OR
& Ampersand
! Exclamation Point
$ Dollar Sign
* Asterisk
) Right Parenthesis
; Semicolon
¬ Logical NOT
- Minus Sign
/ Slash
| Vertical Line
, Comma
% Percent
_ Underscore
^ Circumflex

> Greater-than Sign
? Question Mark
\ Grave Accent
: Colon
Number Sign
@ At Sign
' Prime, Apostrophe
= Equal Sign
" Quotation Mark
~ Tilde
{ Opening Brace
⑀ Hook
⑂ Fork
} Closing Brace
\ Reverse Slant
⑁ Chair
¦ Long Vertical Mark
[Opening Bracket
] Closing Bracket

Table E-1. ASCII/Hexadecimal Character Equivalents

	H1							
H2	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	○	P		p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	r	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	¦
D	CR	GS	–	=	M	]	m	}
E	SO	RS	.	>	N	∧	n	~
F	SI	US	/	?	O	—	o	DEL

Table E-2. EBCDIC/Hexadecimal/Binary Character Equivalents

Bit Positions 4, 5, 6, 7	Second Hexadecimal Digit	00				01				10				11				Bit Positions 0,1
		00	01	10	11	00	01	10	11	00	01	10	11	00	01	10	11	Bit Positions 2,3
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	First Hexadecimal Digit
0000	0	NUL	DLE	DS		SP	&	-						{[a]	}[a]	\[a]	0	
0001	1	SOH	DC1	SOS				/		a	i	~[a]		A	J		1	
0010	2	STX	DC2	FS	SYN					b	k	s		B	K	S	2	
0011	3	ETX	TM							c	l	t		C	L	T	3	
0100	4	PF	RES	BYP	PN					d	m	u		D	M	U	4	
0101	5	HT	NL	LF	RS					e	n	v		E	N	V	5	
0110	6	LC	BS	ETB	UC					f	o	w		F	O	W	6	
0111	7	DEL	IL	ESC	EOT					g	p	x		G	P	X	7	
1000	8	GE[a]	CAN							h	q	y		H	Q	Y	8	
1001	9	RLF[a]	EM						\[a]	i	r	z		I	R	Z	9	
1010	A	SMM	CC	SM		¢	!	;[a]	:								|[a]	
1011	B	VT	CU1[a]	CU2[a]	CU3[a]	.	$	,	#									
1100	C	FF	IFS		DC4	<	*	%	@					⑀[a]		⑁[a]		
1101	D	CR	IGS	ENQ	NAK	(	)	–	'									
1110	E	SO	IRS	ACK		+	;	>	=					⑂[a]				
1111	F	SI	IUS	BEL	SUB	'	¬	?	"								EO[a]	

[a]This character is not supported in the 2780 character set.

APPENDIX F

DEVICE-SPECIFIC CONTROL CHARACTERS

This appendix lists the nonalphanumeric control characters for devices supported by the communications subsystem.

NOTE: A slash between two characters indicates that both keys are pressed simultaneously, e.g., CTRL/H indicates that the CTRL key and H key are passed at the same time.

Table F-1. TTY Nonalphanumeric Control Characters

Character	Hexadecimal Value	Function	Key Strokes
ENQ	05	Answer back	CTRL/E
BEL	07	Ring Bell	CTRL/G
BS	08	Backspace (nondestructive cursor backward)	CTRL/H
LF	0A	Line feed	CTRL/J
FF	0C	Form feed (clear screen)	CTRL/L
CR	0D	Carriage return	CTRL/M
DC2	12	Nondestructive cursor forward	CTRL/R
SP	20	Space	CTRL/P or space bar

NOTES:
1. In a terminal with lowercase capability, uppercase characters require the use of the shift.
2. DC2 is an option for VIP 7100/7200 only.

Table F-2. VIP Nonalphanumeric Control Characters

Character	Hexadecimal Value	Function	Key Strokes
BS	08	Backspace.	CTRL/H
HT	09	Horizontal tab.	CTRL/I
LF	0A	Line feed.	CTRL/J or LINE FEED
FF	0C	Form feed.	CTRL/L
CR	0D	Carriage return.	CTRL/M or RETURN
DC1	11	Reverse line feed.	CTRL/Q
DC2	12	Forward space (nondestructive cursor forward).	CTRL/R
DC3	13	Defines next two characters as line character position.	CTRL/S
DC4	14	Page return.	CTRL/T
ESC	1B	First of several 2-character sequences used for VIP control.	[
FS	1C	First character of a 2-character sequence to define beginning of a fixed field.	\
GS	1D	Defines start of variable field.	]
SP	20	Space.	CTRL/P or space bar
	5E	Defines start of blank field.	

Table F-3. BSC Nonalphanumeric Control Characters

Character	Hexadecimal Value	Function	Key Strokes
NUL	00	Nontransparent data	CTRL/@
SOH	01	Nontransparent data; last record of file	CTRL/A
STX	02	Transparent data	CTRL/B
ETX	03	Transparent data; last record of the file	CTRL/C

NOTE: Table applies only to advanced data transmission mode, and describes control byte for line control. The control byte is neither sent nor received over the line.

APPENDIX G

DUMP ROUTINE (DUMCP) FOR MULTILINE COMMUNICATIONS PROCESSOR (MLCP)

The Honeywell program DUMCP, which is provided in source and object format, dumps the contents of memory (all or part) of the multiline communications processor (MLCP). DUMCP has the following functions:

- o In the dump, shows formatted lists of line control tables, communications control blocks, and communications channel programs.
- o Can print the dump on the operator terminal, line printer, or serial printer.
- o Can be used by the programmer for:
  - Aid in debugging application programs
  - Documenting problems
  - Pinpointing hardware, software, or firmware difficulties

DUMCP cannot run in the batch task group ($B).

DUMCP uses one MLCP channel to transfer dump data from the MLCP to main memory (in block-mode read). The user must therefore specify that MLCP channel and the channel of the output device that will produce the dump.

LINKING THE BOUND UNIT CONTAINING DUMCP

The bound unit that contains DUMCP can be invoked in either of two ways:

- o It may be loaded and activated as a self-contained unit, by the operating system.

o It may be activated by the application program, at one of three starting locations, when the application is linked with DUMCP.

<u>Linking DUMCP as a Self-Contained Bound Unit</u>

To execute the bound unit that contains only DUMCP, the user must load the Linker (with the LINKER command), specifying the following Linker directives (see <u>Program Execution and Checkout</u> manual):

SYS

(Optional) Designates that the bound unit can be a system task in the system task group.

LINKN DUMCP

Requests that the object bound unit DUMCP be linked.

VDEF RDMLCP, X'nnnn'

Designates nnnn as the MLCP channel for block-mode read.

VDEF DMPOUT, X'nnnn'

Designates nnnn as the channel number of the device where the dump is to be printed, which must be an operator terminal, line printer, or serial printer.

MAP

Requests a link map.

QUIT

Terminates execution of the Linker when the bound unit has been created.

NOTES:

1. More than one bound unit may be linked, each with its own unique name, depending on the type of system and on the MLCP channel to be used for the dump routine.

2. When the purpose of the dump is to diagnose a channel error, that channel (value nnnn) should not be designated to be used by the dump routine.

Example:

In this example, a linked version of DUMCP is placed on the volume Z10107. First the working directory is changed to one that contains the object module DUMCP.O; then the Linker is called, according to the Linker directives shown below:

```
CWD     ^Z10107>SOURCE
LINKER  DUMCP -COUT >SPD>LPT00 -SZ 8
```

The user need not specify a relocation base or start address. The bound unit can then be executed.

Any error will result in an error message, and/or error code, issued at execution time to the operator terminal. The System Messages manual describes DUMCP error messages.

Linking DUMCP With the Application Program

Either of the following methods can be used to specify values for the dump output device and for the block-mode read channel that will transfer dump data from the MLCP to main memory:

1. Add the following assembly language XDEF external label definition statements to the source module DUMCP.P:

   XDEF (DMPOUT,Z'nnnn')

   nnnn designates the channel of the output device

   XDEF (RDMLCP,Z'nnnn')

   nnnn designates the block-mode read channel,

or

2. During linking, specify the following VDEF directives:

   VDEF DMPOUT,X'nnnn'

   The value nnnn designates the channel of the output device.

   VDEF RDMLCP,X'nnnn'

   The value nnnn designates the block-mode read channel.

When Linker directives are specified to create the bound unit, enter LINKN DUMCP to request that the object unit DUMCP be linked.

After DUMCP is linked to the application, the dump routine can be entered in any of three ways (described below) according to whether the entry point is specified as STRTD0, STRTD1 or STRTD2.

In any case, the application must include an XLOC (define external locations) instruction; i.e., XLOC STRTD0, XLOC STRTD1 XLOC STRTD2.

STRTD0 ENTRY POINT IN USING DUMCP

When entry point STRTD0 is used, DUMCP will halt at first entry. The user must then set certain register (see below) through the control panel before execution of DUMCP is resumed. These register values override the channel numbers specified in the source program or when DUMCP was linked with the application.

NOTE: Register values for dumping the DLCP (dual line communications processor) of the Model 23 Central Processor are shown separately.

Register	Value to be Entered
$R4	Channel number of dump output device
$R5	Channel number used for block-mode read
$R6	0000; or first memory address of area to be dumped
$R7	0FFF (13FF for Model 23); or the last memory address of area to be dumped
$B5	Return address. If no value is entered, default is that the current address is returned to the system.

The values in the registers control the contents of the dump, as shown in Table G-1.

The format of the entry to specify entry point STRTD0 is:

JMP <STRTD0

The dump routine dumps the MLCP (DLCP) memory to the specified device. Register $R2 (Table G-2) indicates results of the dump. When the dump is completed, control returns to the application at the instruction pointed to by register $B5.

STRTD1 ENTRY POINT IN USING DUMCP

When using entry point STRTD1, the user must set certain registers (see below) before starting to execute the dump. These register values override the channel numbers specified in the source program or when DUMCP was linked with the application.

NOTE: Register values for dumping the DLCP of the Model 23 Central Processor are shown separately.

Register	Value to be Entered
$R4	Channel number of output device for the dump
$R5	Channel number used for the block-mode read
$R6	0000; or the first memory address of area to be dumped
$R7	0FFF (13FF for Model 23); or the last memory address of area to be dumped

The values in the registers control the contents of the dump, as shown in Table G-1.

See Figure G-1 for detailed example of dump formats and contents.

Table G-1. Register Values and DUMCP Dump Contents

Register and Contents	Resulting Dump Contents
$R6 0000 $R7 0FFF 13FF (Model 23)	Fully formatted dump, comprising line control tables, communications control programs, and communications control blocks
$R6 0000 $R7 01FF	Line control tables only
$R6 0E00 $R7 0FFF (Model 23) $R6 1200 $R7 13FF	Communications control blocks only
$R6 Other than: 0000, or 0E00 1200 (Model 23) $R7 Less than: 0FFF 13FF (Model 23)	Unformatted dump of MLCP area within the addresses (byte addresses) specified in $R6 and $R7

The format of the entry specifying entry point STRTD1 is:

LNJ $B5,<STRTD1

The dump routine immediately dumps MLCP (or DLCP) memory to the specified device. The contents of $R2 (see Table G-2) will indicate a successful dump or an error condition. When the dump is completed, control returns to the application program at the instruction pointed to by register $B5.

Table G-2. Register $R2 at Dump Execution - DUMCP Linked to Application

Register $R2 Contents	Meaning
0	Dump successfully completed; no errors.
1	Invalid MLCP channel numbers.
2	Device other than operator terminal or serial/line printer specified as the output device.

STRTD2 ENTRY POINT IN USING DUMCP

STRTD2 should be used when the block-mode read channel (RDMLCP) and the output-device channel number (DMPOUT) values, specified in XDEF statements or in Linker VDEF directives (see above) are to be used without change. Registers need not be changed prior to the dump request.

The format of the entry specifying entry point STRTD2 is:

```
LNJ    $B5,<STRTD2
```

The contents of register $R2 (see Table G-2) will indicate successful dump or an error condition.

When the dump is completed, control returns to the application program at the instruction pointed to by $B5.

DUMCP DUMP FORMATS

Formatted dumps of the MLCP comprise the following areas, whose formats are shown in Figure G-1 below.

- o Line control table (LCT) area, byte locations 0000 through 01FF. The LCT has 64 bytes, each shown in eight groups (four for Model 23) for easier reading.

- o Channel control program (CCP), byte locations 0200 through 0DFF (11FF for Model 23). The format shows 16 bytes per line for easier reading.

- o Communication control block (CCB) area, byte locations 0E00 through 0FFF (1200 through 13FF for Model 23). There are four CCBs per channel. CCBs 0 through 3 are for the receive channel, CCBs 4 through 7 for the send channel. The dump shows the address, range, control byte, and status for each CCB. An R following an address indicates that the address field refers to the right byte of a word. When there is no R following the address, the the address refers to the left byte.

  NOTE: CCBs are used in the following order: For the receive channel, CCB 1 is used first, CCB 0 used last. For the send channel, CCB 5 is used first, CCB 4 used last.

DUMCP PROGRAMMING

The following DUMCP programming considerations apply:

1. The application source program contains a macro call, making it necessary to preprocess the source through EXEC_LIB when reassembly is required.

2. When possible, use an inactive MLCP channel for the block-mode read channel, because the channel specified will be initialized and corresponding channel control block list reset.

3. To allow variations of RDMLCP and DMPOUT values, it may be convenient to line more than one iteration of the dump, with different names.

4. When a printer whose channel number was designated is not ready or is disabled, the DUMCP program loops until the printer's READY button is pressed.

5. DUMCP does not provide trap handling.

6. DUMCP executes at interrupt level 3. Therefore, its execution preempts all system activities including clock functions.

GC086 MCP DUMP REV 3

RAM READ FROM CHAN. FC00

LCT	LIN0	LIN1	LIN2	LIN3	LIN4	LIN5	LIN6	LIN7
0000	FC	FC	00	00	00	00	00	00
0001	00	00	00	00	00	00	00	00
0002	00	E6	00	00	00	00	00	00
0003	00	00	00	00	00	00	00	00
0004	00	00	00	00	00	00	00	00
0005	00	02	00	00	00	00	00	00
0006	00	53	00	00	00	00	00	00
0007	00	B1	00	00	00	00	00	00
0008	00	00	00	00	00	00	00	00
0009	01	00	00	00	00	00	00	00
0010	00	30	00	00	00	00	00	00
0011	00	00	00	00	00	00	00	00
0012	00	00	00	00	00	00	00	00
0013	00	0E	00	00	00	00	00	00
0014	00	00	00	00	00	00	00	00
0015	00	00	00	00	00	00	00	00
0016	00	00	00	00	00	00	00	00
0017	00	00	00	00	00	00	00	00
0018	00	F5	00	00	00	00	00	00
0019	00	5B	00	00	00	00	00	00
0020	00	82	00	00	00	00	00	00
0021	00	00	00	00	00	00	00	00
0022	00	D0	00	00	00	00	00	00
0023	00	03	00	00	00	00	00	00
0024	00	06	00	00	00	00	00	00
0025	00	82	00	00	00	00	00	00
0026	00	06	00	00	00	00	00	00
0027	00	86	00	00	00	00	00	00
0028	00	00	00	00	00	00	00	00
0029	00	00	00	00	00	00	00	00
0030	00	00	00	00	00	00	00	00
0031	00	17	00	00	00	00	00	00
0032	FC	FC	00	00	00	00	00	00
0033	00	00	00	00	00	00	00	00
0034	E6	E6	00	00	00	00	00	00
0035	00	00	00	00	00	00	00	00
0036	00	00	00	00	00	00	00	00
0037	02	00	00	00	00	00	00	00
0038	82	82	00	00	00	00	00	00
0039	2C	2C	00	00	00	00	00	00
0040	00	00	00	00	00	00	00	00
0041	00	00	00	00	00	00	00	00
0042	30	00	00	00	00	00	00	00
0043	31	00	00	00	00	00	00	00
0044	00	00	00	00	00	00	00	00
0045	0E	0E	00	00	00	00	00	00
0046	00	00	00	00	00	00	00	00
0047	00	00	00	00	00	00	00	00
0048	00	A0	00	00	00	00	00	00
0049	00	00	00	00	00	00	00	00
0050	33	43	00	00	00	00	00	00
0051	03	44	00	00	00	00	00	00
0052	00	00	00	00	00	00	00	00
0053	82	82	00	00	00	00	00	00
0054	A0	60	00	00	00	00	00	00
0055	03	0A	00	00	00	00	00	00
0056	06	06	00	00	00	00	00	00
0057	76	76	00	00	00	00	00	00

Figure G-1. DUMCP Dump Example

```
0550   51  10  90  00  51  1E  02  E0  A5  F0  61  E0  97  E0  5B  E0
0560   09  E0  95  E0  95  E0  95  E0  95  51  1F  50  1C  92  FF  F1
0570   07  50  1D  92  FF  F1  EB  F0  43  03  50  3A  92  FF  E1  09
0580   01  A2  52  3D  F1  0B  E0  F9  01  A0  52  3D  F1  03  E0  F9
0590   51  1F  50  1C  92  FF  E1  10  50  1F  92  82  E1  07  92  02
05A0   E1  03  E0  BE  04  E0  BD  50  1F  92  10  F1  B9  01  A0  E0
05B0   EA  E0  AF  E0  B1  E0  AD  E0  AF  E0  2A  50  1F  04  50  3A
05C0   92  00  E1  0F  01  A0  55  03  F2  13  01  A0  55  04  F2  0D
05D0   E0  09  01  A0  55  03  93  7F  F2  03  02  06  50  11  94  40
05E0   51  11  02  06  51  1F  50  1E  92  FF  F1  25  50  1C  92  FF
05F0   E1  03  E0  BE  50  1D  92  FF  E1  11  90  FF  51  1D  E0  B6
0600   E0  09  E0  AE  E0  B0  E0  AC  E0  AE  90  00  51  1D  E0  F3
0610   01  50  3A  92  FF  F1  1E  A0  92  70  E1  36  92  61  E1  36
0620   92  7C  E1  36  51  1F  50  1C  92  00  E1  DB  50  1F  92  02
0630   E1  3A  E0  D3  A2  92  B0  E1  19  92  31  E1  19  92  BC  E1
0640   19  51  1F  50  1C  92  00  E1  BE  50  1F  92  82  E1  1D  E0
0650   B6  90  05  E0  07  90  06  E0  03  90  04  11  50  10  94  02
0660   51  10  90  00  51  1E  51  1D  02  E0  9E  90  FF  51  1E  90
0670   00  51  1D  E0  90  00  00  80  81  82  84  84  84  84  84  84
0680   84  82  00  80  81  00  00  00  81  83  85  85  00  00  00  00
0690   00  00  00  00  00  00  80  00  00  00  00  84  86  83  00  00
06A0   00  00  00  00  00  82  00  00  00  00  00  00  00  00  00  00
** ALL ZEROS **
06C0   00  00  00  00  00  00  00  00  00  00  00  15  05  04  3C  30
06D0   31  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
** ALL ZEROS **

CCB AREA

CCB     ADDRESS    RANGE    CONTROL    STATUS
LINE    0
0000     000000     0000       00       0000
0001     0057A7     01F4       0C       000E
0002     003E71R    0000       80       1000
0003     000000     0000       00       0000
0004     0045E9     0000       86       1000
0005     0045C3     0000       C6       1000
0006     0045F9     0000       D2       1000
0007     006FF7     0000       80       F000
LINE    1
0000     000000     0000       00       0000
0001     003E98R    0000       80       5200
0002     006DF3R    0001       80       50A0
0003     000000     0000       00       0000
0004     000000     0000       00       0000
0005     000000     0000       00       0000
0006     000000     0000       00       0000
0007     000000     0000       00       0000
LINE    2
0000     000000     0000       00       0000
0001     000000     0000       00       0000
0002     000000     0000       00       0000
0003     000000     0000       00       0000
0004     000000     0000       00       0000
0005     000000     0000       00       0000
0006     000000     0000       00       0000
0007     000000     0000       00       0000
LINE    3
0000     000000     0000       00       0000
0001     000000     0000       00       0000
0002     000000     0000       00       0000
```

Figure G-1 (cont). DUMCP Dump Example

```
0058      00      00      00      00      00      00      00      00
0059      00      00      00      00      00      00      00      00
0060      FF      00      00      00      00      00      00      00
0061      16      16      00      00      00      00      00      00
0062      06      06      00      00      00      00      00      00
0063      C6      C6      00      00      00      00      00      00

CCP
0200  00  00  90  00  51  08  50  34  34  50  02  36  50  14  32  01
0210  E0  FE  E0  7C  E0  17  50  3D  34  90  C1  51  14  32  01  06
0220  90  00  51  23  51  24  90  82  51  14  32  01  F0  E9  50  37
0230  56  38  E0  0B  E0  09  E0  07  E0  0C  E0  03  E0  56  90  04
0240  54  30  02  E0  E7  E0  51  50  1C  92  FF  F1  05  90  10  60
0250  01  50  3A  92  FF  E1  11  50  3B  E2  07  90  02  63  01  E0
0260  15  90  02  62  01  E0  0F  50  3B  E2  07  90  02  61  01  E0
0270  05  90  02  60  01  50  3A  92  FF  F1  21  50  1C  92  FF  F1
0280  08  10  92  10  E1  1D  E0  02  10  61  01  F3  EF  E0  18  E0
0290  77  E0  9A  E0  71  E0  8A  E0  69  E0  AD  10  63  01  F3  FC
02A0  E0  05  60  01  E0  E4  50  1C  92  FF  F1  05  90  10  60  01
02B0  50  3B  05  52  3C  E1  13  50  3A  92  FF  E1  07  90  1F  63
02C0  01  E0  47  90  1F  61  01  E0  41  90  00  51  3B  50  30  94
02D0  80  51  30  50  37  92  0B  E1  13  50  3A  92  FF  F1  07  90
02E0  26  61  01  E0  25  90  17  63  01  E0  1F  50  3A  92  FF  E1
02F0  07  90  03  63  01  E0  13  90  03  61  01  E0  0D  E0  97  E0
0300  91  F0  28  E0  95  E0  3C  E0  7A  50  23  60  01  50  3A  92
0310  00  E1  05  50  24  60  01  90  FF  60  01  60  01  60  01  50
0320  30  93  80  92  80  02  F1  D8  E0  D4  50  3A  92  FF  E1  0A
0330  50  3D  62  01  62  01  62  01  06  50  3D  60  01  60  01  60
0340  01  06  F0  E7  50  37  93  08  92  08  E1  25  50  3A  92  FF
0350  E1  17  50  37  56  3E  62  01  90  FF  60  01  60  01  60  01
0360  50  30  94  80  51  30  E0  96  50  37  56  3E  60  01  E0  E9
0370  50  3A  92  FF  E1  07  90  10  62  01  E0  D1  90  10  60  01
0380  E0  E7  E0  2E  50  3D  34  90  00  51  1E  90  00  51  17  94
0390  82  51  14  32  E0  18  50  10  94  82  51  10  02  90  00  51
03A0  1E  E0  03  E0  F2  90  80  51  14  32  90  00  51  03  51  04
03B0  01  50  17  56  18  E0  0F  E0  CC  E0  D1  E0  09  01  A0  52
03C0  3D  E1  FB  E0  0D  50  3A  92  FF  E1  F3  01  A2  52  3D  E1
03D0  F5  51  1F  50  3A  92  FF  E1  07  50  1F  93  7F  51  1F  50
03E0  1F  93  C0  F2  69  50  1F  56  1A  E0  63  E0  65  E0  6D  E0
03F0  5D  E0  5B  E0  5F  E0  5F  E0  55  01  A0  51  1F  93  C0  F2
0400  15  50  1F  56  1A  F0  0F  E0  49  E0  0B  E0  4D  E0  49  E0
0410  05  E0  43  E0  63  50  1F  04  E3  8A  11  E0  DD  E0  A7  E0
0420  85  E0  81  01  A2  51  1F  93  40  F2  17  50  1F  93  7F  56
0430  1A  E0  0F  E0  67  E0  0B  E0  61  E0  71  E0  05  E0  61  E0
0440  37  50  1F  04  93  7F  E3  DA  11  E0  D9  E0  D3  E0  39  E0
0450  CD  E0  49  E0  49  E0  49  E0  53  F0  3F  90  FF  51  1E  E0
0460  0F  50  3A  92  FF  F1  05  50  1F  E0  AD  50  1F  E0  D5  50
0470  3A  92  FF  F1  AF  E0  83  50  1C  92  FF  E1  03  E0  F1  50
0480  1D  92  FF  F1  DD  E0  E9  50  10  94  02  51  10  90  07  11
0490  90  00  51  1E  51  1D  02  E0  83  E0  67  E0  63  E0  3B  E0
04A0  65  E0  CD  E0  E3  E0  BB  E0  A7  E0  A1  51  1F  50  1C  92
04B0  FF  F1  09  50  1D  92  00  F1  03  E0  EB  50  1F  92  03  E1
04C0  09  50  11  94  80  51  11  E0  07  50  10  94  01  51  10  90
04D0  00  51  1E  51  1D  E0  2D  E0  D1  51  1F  50  1E  92  FF  F1
04E0  03  E0  C3  90  01  11  50  10  94  02  51  10  02  90  00  51
04F0  1E  E0  0B  E0  E3  E0  B1  E0  AD  F0  A7  E0  A7  E0  AB  E0
0500  5D  E0  5D  E0  55  51  1F  50  1E  92  FF  F1  2F  50  1C  92
0510  00  E1  09  50  1D  92  FF  E1  03  E0  8B  90  00  51  1D  50
0520  1F  92  05  E1  0E  92  2D  E1  0A  50  10  94  04  51  10  02
0530  E0  CC  50  10  94  80  51  10  02  E0  C3  50  1F  92  05  E1
0540  09  92  2D  E1  05  90  03  E0  03  90  02  11  50  10  94  02
```

Figure G-1 (cont). DUMCP Dump Example

```
0003     000000    0000     00     0000
0004     000000    0000     00     0000
0005     000000    0000     00     0000
0006     000000    0000     00     0000
0007     000000    0000     00     0000
LINE    4
0000     000000    0000     00     0000
0001     000000    0000     00     0000
0002     000000    0000     00     0000
0003     000000    0000     00     0000
0004     000000    0000     00     0000
0005     000000    0000     00     0000
0006     000000    0000     00     0000
0007     000000    0000     00     0000
LINE    5
0000     000000    0000     00     0000
0001     000000    0000     00     0000
0002     000000    0000     00     0000
0003     000000    0000     00     0000
0004     000000    0000     00     0000
0005     000000    0000     00     0000
0006     000000    0000     00     0000
0007     000000    0000     00     0000

LINE    6
0000     000000    0000     00     0000
0001     000000    0000     00     0000
0002     000000    0000     00     0000
0003     000000    0000     00     0000
0004     000000    0000     00     0000
0005     000000    0000     00     0000
0006     000000    0000     00     0000
0007     000000    0000     00     0000
LINE    7
0000     000000    0000     00     0000
0001     000000    0000     00     0000
0002     000000    0000     00     0000
0003     000000    0000     00     0000
0004     000000    0000     00     0000
0005     000000    0000     00     0000
0006     000000    0000     00     0000
0007     000000    0000     00     0000
END OF MCP DUMP
```

Figure G-1 (cont). DUMCP Dump Example

CUT ALONG LINE

HONEYWELL INFORMATION SYSTEMS

Technical Publications Remarks Form

TITLE	SERIES 60 (LEVEL 6) COMMUNICATIONS PROCESSING	ORDER NO.	CB03, REV. 1
		DATED	JULY 1978

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be promptly investigated by appropriate technical personnel and action will be taken as required. If you require a written reply, check here and furnish complete mailing address below. ☐

FROM: NAME ______________________ DATE ____________

TITLE ______________________

COMPANY ______________________

ADDRESS ______________________

____

PLEASE FOLD AND TAPE –
NOTE: U. S. Postal Service will not deliver stapled forms

CUT ALONG LINE

FOLD ALONG LINE

FIRST CLASS
PERMIT NO. 39531
WALTHAM, MA
02154

Business Reply Mail
Postage Stamp Not Necessary if Mailed in the United States

Postage Will Be Paid By:

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTENTION: PUBLICATIONS, MS 486

FOLD ALONG LINE

Honeywell

Honeywell

Honeywell Information Systems
In the U.S.A.: 200 Smith Street, MS 486, Waltham, Massachusetts 02154
In Canada: 2025 Sheppard Avenue East, Willowdale, Ontario M2J 1W5
In Mexico: Avenida Nuevo Leon 250, Mexico 11, D.F.

21378, 2.5778, Printed in U.S.A.

CB03, Rev. 1