PERKIN-ELMER

OS/32 BASIC DATA COMMUNICATIONS

Reference Manual

48-077 F00 R00

The Perkin-Elmer Corporation, Data Systems Group, 2 Crescent Place, Oceanport, New Jersey 07757

Printed in the United States of America

TABLE OF CONTENTS

PREFACE xi

CHAPTERS

1 BASIC DATA COMMUNICATIONS

1.1 INTRODUCTION 1-1

1.2 DATA COMMUNICATIONS NETWORKS 1-2

1.3 A BASIC DATA COMMUNICATIONS SYSTEM 1-3
1.3.1 Terminals 1-4
1.3.1.1 Terminal Communications Modes 1-4
1.3.1.2 Terminal Speeds 1-6
1.3.1.3 Data Transmission Modes 1-6
1.3.1.4 Data Codes 1-8
1.3.2 Modems 1-9
1.3.3 Transmission Lines 1-10
1.3.4 Multiplexors (MUXs) and Concentrators 1-11

2 OS/32 DATA COMMUNICATIONS SYSTEM

2.1 INTRODUCTION 2-1

2.2 ADAPTERS 2-2
2.2.1 Asynchronous Adapters 2-2
2.2.2 Bisynchronous Adapters 2-3
2.2.3 Zero-Bit Insertion/Deletion (ZBID) Adapters 2-4

2.3 LINE DRIVERS 2-5

2.4 DEVICE-INDEPENDENT ACCESS 2-5
2.4.1 Terminal Managers 2-6
2.4.2 Protocols 2-8

2.5 DEVICE-DEPENDENT ACCESS 2-10

CHAPTERS (Continued)

3 DEVICE HANDLING

3.1	INTRODUCTION	3-1
3.2	SUPERVISOR CALL 7 (SVC7)	3-1
3.2.1	Supervisor Call 7 (SVC7) Parameter Block	3-1
3.2.1.1	Function Code Field	3-3
3.2.1.2	Error Status Field	3-3
3.2.1.3	Logical Unit (lu) Field	3-5
3.2.1.4	Read and Write Key Fields	3-5
3.2.1.5	Logical Record Length Field	3-5
3.2.1.6	Device Mnemonic Field	3-5
3.2.1.7	Filename Field	3-5
3.2.1.8	Extension Field	3-5
3.2.1.9	File Size Field	3-6
3.3	COMMAND FUNCTIONS	3-6
3.3.1	Allocate Function	3-6
3.3.2	Assign Function	3-6
3.3.3.	Change Access Privileges Function	3-7
3.3.4	Close Function	3-8
3.3.5	Delete Function	3-8
3.3.6	Checkpoint Function	3-9
3.3.7	Fetch Attributes Function	3-9
3.3.8	Vertical Forms Control (VFC) Function	3-9
3.3.10	Rename Function	3-10
3.3.10	Reprotect Function	3-10
3.4	OPERATOR COMMANDS	3-10
3.4.1	ALLOCATE Command	3-11
3.4.2	ASSIGN Command	3-13
3.4.3	CLOSE Command	3-15
3.4.4.	DELETE Command	3-16
3.4.5	XALLOCATE Command	3-17
3.4.6	XDELETE Command	3-19

4 DEVICE-INDEPENDENT ACCESS

4.1	INTRODUCTION	4-1
4.2	TERMINAL MANAGER ACCESS	4-1
4.3	SEQUENCE OF OPERATION	4-2
4.4	SUPERVISOR CALL 1 (SVC1)	4-4
4.4.1	Supervisor Call 1 (SVC1) Parameter Block	4-4

5 DEVICE-DEPENDENT ACCESS

5.1	INTRODUCTION	5-1

5.2	LINE DRIVER ACCESS	5-1
5.2.1	Sequence of Operations	5-2
5.2.2	Supervisor Call 15 (SVC15) and the Task Environment	5-3
5.2.2.1	Supervisor Call 15 (SVC15) Trap Handling	5-4
5.3	SUPERVISOR CALL 15 (SVC15) PARAMETER BLOCK	5-5
5.3.1	Function Code Field	5-6
5.3.2	Logical Unit (lu) Field	5-7
5.3.3	Status Information Field	5-8
5.3.4	Command Number Field	5-12
5.3.5	Driver Command Word (DCW) Pointer	5-12
5.3.6	Length of Last Read Field	5-12
5.3.7	Length of Last Write Field	5-12
5.3.8	Data Fields	5-13
5.3.9	Data Field Chain	5-13
5.4	BUFFER TYPES	5-14
5.4.1	Direct Buffers	5-15
5.4.2	Indirect Buffers	5-15
5.4.3	Chained Buffers	5-16
5.4.4	Queued Buffers	5-19
5.4.4.1	Coding a Queued Buffer Request	5-20
5.5	DRIVER COMMAND WORD (DCW)	5-21
5.6	LINE DRIVER COMMAND TYPES	5-23
5.6.1	Null-Type Commands	5-24
5.6.1.1	NO OPERATION (NOP) Command	5-24
5.6.1.2	WAIT Command	5-24
5.6.1.3	TRANSFER IN (XFER) Command	5-24
5.6.1.4	CONDITIONAL TRANSFER (CXFER) Command	5-24
5.6.2	Control-Type Commands	5-25
5.6.2.1	EXAMINE Command	5-25
5.6.2.2	RING WAIT Command	5-25
5.6.2.3	ANSWER Command	5-26
5.6.2.4	DISCONNECT Command	5-26
5.6.3	Read-Type Commands	5-26
5.6.3.1	READ BUFFER Command	5-26
5.6.3.2	READ1 Command	5-26
5.6.3.3	READ2 Command	5-27
5.6.4	Prepare-Type Commands	5-27
5.6.4.1	PREPARE Command	5-27
5.6.4.2	PREPARE3 Command	5-27
5.6.5	Write-Type Commands	5-27
5.6.5.1	WRITE BUFFER Command	5-28
5.6.5.2	WRITE1 Command	5-28
5.6.5.3	WRITE2 Command	5-28
5.6.6	Hold-Type Commands	5-28
5.6.6.1	HOLD SPACE (Line Break) Command	5-28
5.6.7	Mode-Type Commands	5-29
5.6.7.1	MODE TOUT (Time-out Interval) Command	5-29
5.6.7.2	MODE CMD2 (Adapter) Command	5-29

5.6.7.3	MODE RCMD (Read) and MODE WCMD (Write) Commands	5-30
5.6.7.4	MODE RDIS (Read Disable) and MODE WDIS (Write Disable) Commands	5-30
5.6.7.5	MODE DISC (Disconnect) Command	5-30
5.6.7.6	MODE SYNCNT (SYNC Character Count) Command	5-30
5.6.7.7	MODE SPCHAR (Special Character Enable Masks Command	5-30
5.6.7.8	MODE TRANSL (Translation Options) Command	5-30
5.6.8	Test-Type Commands	5-30

6 DATA COMMUNICATIONS STRUCTURES

6.1	INTRODUCTION	6-1
6.2	DATA COMMUNICATIONS LINE DRIVERS	6-1
6.3	CONTROL BLOCK FORMATS	6-2
6.3.1	Data Communications Device Control Block (DCB)	6-2
6.3.1.1	Device Control Block (DCB) Device-Independent Portion (Standard DCB)	6-3
6.3.1.2	Device Control Block (DCB) Data Communications-Related Portion	6-7
6.3.1.3	Device Control Block (DCB) Device-Dependent Portion	6-16
6.3.2	Line Control Block (LCB)	6-16
6.3.2.1	Line Control Block (LCB) Device-Independent Portion	6-16
6.3.2.2	Line Control Block (LCB) Device-Dependent Portion	6-22
6.3.2.3	Line Control Block (LCB) Data Block Descriptor Portion	6-25
6.3.3	Channel Control Block (CCB)	6-26
6.3.3.1	Channel Control Block (CCB) Device-Independent Portion	6-26
6.3.3.2	Channel Control Block (CCB) Device-Dependent Portion	6-27
6.3.4	Drop Control Table (DCT) for Zero-Bit Insertion/Deletion Data Link Control ZDLC Communications	6-30
6.3.5	Drop Definition Table (DDT) for Zero-Bit Insertion/Deletion Data Link Control (ZDLC) Communications	6-34
6.3.6	Drop Control Table (DCT) for Asynchronous Multidrop Communications	6-36
6.3.7	Drop Access Table (DAT) for Asynchronous Multidrop Communications	6-38
6.3.8	Input/Output Block (IOB) for Asynchronous Multidrop Communications	6-40
6.3.9	Station Description Table (SDT) for 3270 Emulator	6-43

6.3.10	Device Definition Table (DDT) for 3270 Emulator	6-44
6.3.11	Input/Output Handler (IOH)	6-47
6.3.12	File Manager Handler (FMH)	6-50
6.4	DEVICE CONTROL BLOCK (DCB) POINTER FOR LINE DRIVER COMMAND INTERPRETATION	6-52
6.4.1	DCB.TERM Pointer	6-53
6.4.2	DCB.DOCR, DCB.DOCW, DCB.MOCR and DCB.MOCW Pointers	6-53
6.4.3	DCB.AOC Pointer	6-54
6.4.4	DCB.INIT Pointer	6-54
6.4.5	DCB.RDN and DCB.WDN Pointers	6-54
6.4.6	DCB.ITV and DCB.OTV Pointers	6-54
6.5	EVENT SERVICE ROUTINE (ESR) SCHEDULING	6-54
6.6	SUPERVISOR CALL 15 (SVC15) STRUCTURE AND FLOW	6-56
6.7	COMMON DATA COMMUNICATIONS SUBROUTINES	6-59
6.7.1	Supervisor Call 15 (SVC15) Subroutine	6-59
6.7.2	ITSRABS Subroutine	6-60
6.7.3	CMTERM Subroutine	6-60
6.7.4	CMEXIT Subroutine	6-61
6.7.5	ISSEXEC Subroutine	6-62
6.7.6	ITSETREA Subroutine	6-64
6.7.7	ITXFRISR Subroutine	6-64
6.7.8	ITISSTOP Subroutine	6-65
6.7.9	IT..STOP Subroutine	6-65
6.7.10	ITIMLINK Subroutine	6-66
6.7.11	ITIMUNLK Subroutine	6-66
6.7.12	ITISTOTC Subroutine	6-67
6.7.13	ITISPOTC Subroutine	6-67
6.7.14	ICMDINT Subroutine	6-67
6.7.15	ITGETMOD2 Subroutine	6-68
6.7.16	ITGETMOD Subroutine	6-68
6.7.17	ITGETDAT Subroutine	6-69
6.7.18	ITGETBUF Subroutine	6-70
6.8	SUPERVISOR CALL 1 (SVC1) PROCESSING	6-70
6.9	ADDITIONAL EXECUTIVE FUNCTIONS	6-70
6.9.1	Cancellation of Input/Output (I/O)	6-71
6.9.2	Add to Task Queue	6-71
6.9.3	System Initialization	6-71
6.9.4	Timer Management	6-71
6.10	SUPERVISOR CALL 7 (SVC7) PROCESSING	6-72
6.10.1	Allocate	6-72
6.10.2	Delete	6-72
6.10.3	Assign	6-73
6.10.4	Close	6-73

6.10.5	Checkpoint	6-74
6.10.6	Fetch Attributes	6-74
6.10.7	Change Access Privileges	6-74
6.10.8	Rename	6-74
6.10.9	Reprotect	6-74

7 HOW TO WRITE AND USE A TERMINAL MANAGER

7.1	INTRODUCTION	7-1
7.2	TERMINAL MANAGER MODIFICATION	7-1
7.3	BACKGROUND INFORMATION	7-1
7.4	TERMINAL MANAGER STRUCTURE	7-2
7.4.1	Nonbuffered Terminal Manager	7-2
7.5.2	Buffered Terminal Manager (Input)	7-5
7.4.3	Buffered Terminal Manager (Output)	7-5
7.5	TERMINAL MANAGER FUNCTIONS	7-6
7.5.1	Special Terminal Manager Functions	7-11
7.5.1.1	Format Control	7-11
7.5.1.2	Time-out Control	7-11
7.5.1.3	Buffer Control	7-12
7.6	SYSTEM GENERATION (SYSGEN) CONVENTIONS	7-12
7.6.1	Register Conventions	7-12
7.6.2	Device Control Block/Line Control Block (DCB/LCB) Reference	7-13
7.6.3	EXTRN/ENTRY References	7-13
7.6.4	System Generation (Sysgen)	7-15
7.7	WRITING TERMINAL MANAGERS SUMMARY	7-15
7.8	HOW TO USE DATA COMMUNICATIONS TERMINAL MANAGERS	7-15

8 HOW TO WRITE AND USE DATA COMMUNICATIONS LINE DRIVERS

8.1	INTRODUCTION	8-1
8.2	MODIFYING A LINE DRIVER	8-1
8.3	LINE DRIVER USE OF THE DEVICE CONTROL BLOCK (DCB)	8-4
8.4	LINE DRIVER STRUCTURE	8-5
8.4.1	Driver Initiation Routine	8-5
8.4.2	Translation Tables	8-6

CHAPTERS (Continued)

8.5	DATA COMMUNICATIONS LINE DRIVER EXAMPLE	8-6
8.5.1	Command Table	8-6
8.5.2	Command Fetch	8-7
8.5.3	Modifier Fetch	8-8
8.5.4	Command/Modifer Routines	8-8
8.5.5	Entering Interrupt Service Routines (ISRs)	8-9
8.5.6	Special Character Routines	8-11
8.5.7	Read After Write (RAW) Turnaround	8-12
8.5.8	Driver Termination Phase	8-13
8.6	USING DATA COMMUNICATIONS LINE DRIVERS	8-13
8.6.1	Buffer Management	8-14
8.6.1.1	Chained Buffers	8-18
8.6.1.2	Line Driver Data Communications Device Interface	8-20

9 GENERATING AN OPERATING SYSTEM WITH DATA COMMUNICATIONS DEVICES

9.1	INTRODUCTION	9-1
9.2	DATA COMMUNICATIONS CONFIGURATION STATEMENT	9-1
9.3	SYSTEM LIBRARIES	9-2
9.3.1	The Driver Library	9-2
9.3.2	Including User-Written Drivers	9-3
9.3.2.1	Creating the DCBxxx Macro	9-3

APPENDIXES

A	LINE DRIVER COMMAND SUMMARY	A-1
B	INTERFACE SIGNAL DEFINITIONS	B-1

FIGURES

1-1	Point-to-Point and Multipoint Networks	1-2
1-2	A Simplistic Data Communications Network	1-3
1-3	Terminal Communications Modes	1-5
1-4	Asynchronous Transmission	1-6
1-5	Synchronous Transmission	1-7
2-1	A Data Communications Subsystem	2-1
2-2	OS/32 Terminal Managers	2-7

3-1	SVC7 Parameter Block Format and Coding	3-2
3-2	SVC7 Function Code Field	3-3
4-1	SVC1 Parameter Block Format and Coding	4-5
4-2	SVC1 Function Code Field	4-7
5-1	SVC15 Access to a Line Driver	5-3
5-2	SVC15 Parameter Block	5-5
5-3	SVC15 Function Code Format	5-6
5-4	SVC15 Status Field Format	5-8
5-5	SVC15 Data Field Format	5-13
5-6	Direct Buffer	5-15
5-7	Indirect Buffer	5-16
5-8	Chained/Queued Buffer Format	5-16
5-9	Chained/Queued Buffer Link Word Flag Byte	5-17
5-10	Buffer Ring	5-18
5-11	Conceptual Circular List and Format	5-19
5-12	DCW Format	5-22
6-1	DCB Sections	6-3
6-2	Basic DCB Fields	6-4
6-3	Data Communications DCB Fields	6-8
6-4	Basic LCB Fields	6-17
6-5	Device-Dependent LCB Fields	6-22
6-6	CCB Device-Independent Portion	6-26
6-7	Data Communications CCB Format	6-28
6-8	DCT (ZDLC) Format	6-30
6-9	DDT (ZDLC) Format	6-35
6-10	DCT (Asynchronous Multidrop) Format	6-37
6-11	DAT (Asynchronous Multidrop) Format	6-39
6-12	IOB Format	6-40
6-13	SDT Format	6-43
6-14	DDT (3270 Emulator) Format	6-45
6-15	IOH Format	6-48
6-16	FMH Format	6-51
6-17	SVC15 Line Driver Modules - Data Communications Operation System Interface	6-58
7-1	Nonbuffered Terminal Manager	7-3
7-2	Buffered Terminal Manager (Input)	7-4
7-3	Buffered Terminal Manager (Output)	7-5
8-1	SVC15 Driver Structure	8-3
8-2	SVC15 Using Direct Buffers	8-15
8-3	SVC15 Using Indirect Buffers	8-15
8-4	SVC16 Using Chained Buffers	8-16
8-5	SVC15 Using Queued Buffers	8-17
8-6	Example of an SVC15 Parameter Block and Associated Data	8-23
8-7	Parameter Block and Associated Fields After SVC15 Termination	8-24
8-8	SVC15 Parameter Block After Termination	8-25

TABLES

3-1	SVC7 ERROR STATUS CODE BIT SETTINGS	3-4
4-1	SVC1 DATA TRANSFER FUNCTION CODE	4-7
4-2	SVC1 EXTENDED OPTIONS	4-10
5-1	SVC15 FUNCTION CODE BIT SETTINGS	5-6
5-2	SVC15 STATUS BIT SETTINGS	5-8
5-3	SVC15 ENCODED TERMINATION CODES	5-9
5-4	DATA CODE BIT SETTINGS	5-14
5-5	CHANNEL/QUEUED BUFFER LINK WORD FLAG BYTE	5-17
5-6	QUEUED BUFFER DATA FIELD FORMAT	5-21
5-7	DCW BIT SETTINGS	5-22
6-1	DCB.LNST BIT DEFINITIONS	6-13
6-2	BLOCK DESCRIPTOR FLAG BIT DEFINITIONS	6-25
6-3	DATA COMMUNICATIONS SUBROUTINE REQUEST BITS	6-56

INDEX IND-1

PREFACE

This manual describes the concepts necessary to use the Perkin-Elmer OS/32 Basic Data Communications software. Included in this manual is a description of the areas of application, program interface, OS/32 support features and internal operations.

Chapter 1 introduces basic data communications facilities. Chapter 2 introduces the OS/32 Basic Data Communications Subsystem. The differences between device-independent and device-dependent access are detailed and brief descriptions of Perkin-Elmer adapters, line drivers and terminal managers are included. Chapter 3 describes device handling through supervisor call 7 (SVC7) or the OS/32 command language. The SVC7 parameter block and related functions are discussed as well as the OS/32 commands pertinent to data communications. Chapter 4 discusses device-independent access of data communications facilities, which is accomplished by using the SVC1 parameter block described in this chapter. Chapter 5 details device-dependent access of data communications facilities through the use of line drivers, which is accomplished by using the SVC15 parameter block described in this chapter. SVC15 buffer types and the driver command word (DCW) are also detailed in Chapter 5. Chapter 6 describes the various structures and routines used in the OS/32 Data Communications Subsystem. Chapters 7, 8 and 9 describe the internal operation of basic data communications and discuss the process of modifying or adding a line driver or terminal manager. These three chapters should be read by users whose requirements are not satisfied by basic data communications support.

Throughout this manual, there are numerous references to the Integrated Telecommunications Access Method (ITAM). Prior to the R05.1 version of OS/32, all Perkin-Elmer data communications software was packaged separately and was referred to as ITAM.

This manual is intended for use with the OS/32 R07.2 software release and higher.

For information on the contents of all Perkin-Elmer 32-bit manuals, see the 32-Bit Systems User Documentation Summary.

CHAPTER 1
BASIC DATA COMMUNICATIONS

1.1 INTRODUCTION

Put simply, the purpose of data communications is to transfer information. In the context of data processing, data communications refers to the exchange of information between computers or peripherals.

All transmissions between locations require three things:

- a transmitter or message source,
- a transmission medium, and
- a receiver.

A simple telephone call is a good example of a data transmission. You, the caller, use a transmitting medium, the telephone and the telephone lines, to transfer information to the person you are calling. The same is true of a digital data communications network with one small addition. In a digital data communications system, the transmitter must not only be able to transmit the data, it must also be able to translate the source message from its original form into a form that can travel over the available transmission path. The receiver must then be able to translate the transmitted message back into a form that can be understood by people or machines. An analogy can be made with an earlier data communications system, the telegraph. Using a telegraph, the words of a message could not be sent directly, but had to be encoded into a system of dots and dashes that could be transmitted over a wire. The dots and dashes received would then have to be decoded into a language that the people receiving the message could understand.

During the first half of this century, most of data communications involved voice transmissions. With the advent and increased use of digital computers, the importance of digital communications quickly became evident. Many users wanted their computer to be able to communicate with their customer's computer across town, or with another company's computer across the nation. Initially, this was not possible, because most conventional telecommunications facilities were developed before the birth of the computer and were, therefore, for voice rather than digital communications.

These analog (voice) systems were not fast enough to keep up with the speeds computers were becoming capable of. In addition, the earlier computers were not designed to be connected to an existing communications network. With the increasing demand for long distance computer communications, the field of digital data communications became an important part of the computer industry.

1.2 DATA COMMUNICATIONS NETWORKS

Data processing facilities are frequently joined together to form data communications networks. Such networks connect remote terminals and computer systems to each other and/or to a host computer. The earlier systems were connected via public telephone and telegraph lines, but large private networks using high-speed digital facilities and leased transmission lines soon evolved. Data can also be transferred by radio signals and, with the increased use of microwaves and the advent of space travel, by microwave relays to satellites.

If a single transmitting device is connected to a single receiving device, the data communications network is considered to have a point-to-point line (see Section A of Figure 1-1). To decrease the costs of such a network, more than one terminal can be connected over a single dedicated line. This type of line is referred to as a multidrop (or multipoint) line (see Section B of Figure 1-1). When using a multidrop line, the processor communicates with a terminal by one of two methods:

- polling and selection, or
- collision avoidance/collision detection.

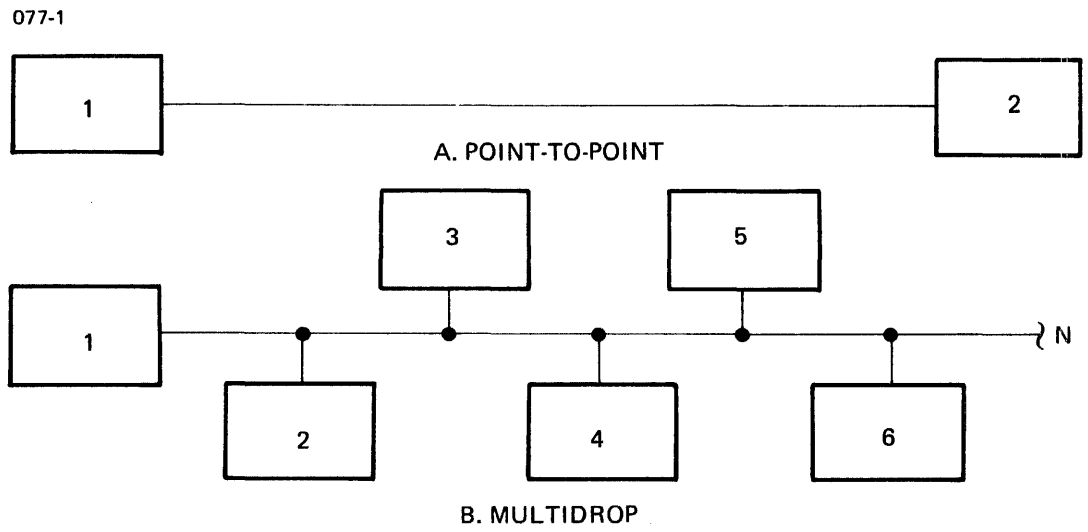


Figure 1-1 Point-to-Point and Multidrop Networks

Polling is the process by which the computer queries each individual terminal to see whether or not that terminal has any data to transfer. The processor accomplishes this by sending out a unique bit sequence (ID) that can be recognized by a particular terminal as its address. The terminal then responds either positively or negatively. A negative response causes the processor to query the next terminal. A positive response causes the processor to take the necessary steps for a data transfer to occur. Polling is useful in large networks where tight control over line usage is desired; it is also applicable in cost-conscious facilities or in applications where transmitted messages have different levels of priority.

Selection is the mechanism by which the processor itself specifies the terminal to which it wants to transmit data. This is accomplished by using the terminal's ID. Multidrop networks are able to broadcast a message to all of the terminals on the multidrop line.

In collision avoidance/collision detection, a terminal first "listens" to see if another terminal is sending data. Only if no other terminal is transmitting will it attempt to send data. If more than one terminal happens to start transmitting at the same time, both cease transmissions and wait a specified and different amount of time before they attempt to transmit again. The advantage of this method is that the processor is not needed to control the actual process as in the polling method. The main disadvantage is that there is no limit on the amount of time a low-priority terminal might have to wait before transmitting.

A special type of data communications network is the distributed processing network. Such networks divide the data processing among several smaller computers. This arrangement improves the overall performance of the network, since a failure at one node does not affect the other processors and peripherals. Such an arrangement also provides increased reliability because distributed networks provide alternate paths to other processors in the event of a nodal failure.

1.3 A BASIC DATA COMMUNICATIONS SYSTEM

Figure 1-2 depicts a simplistic data communications network.

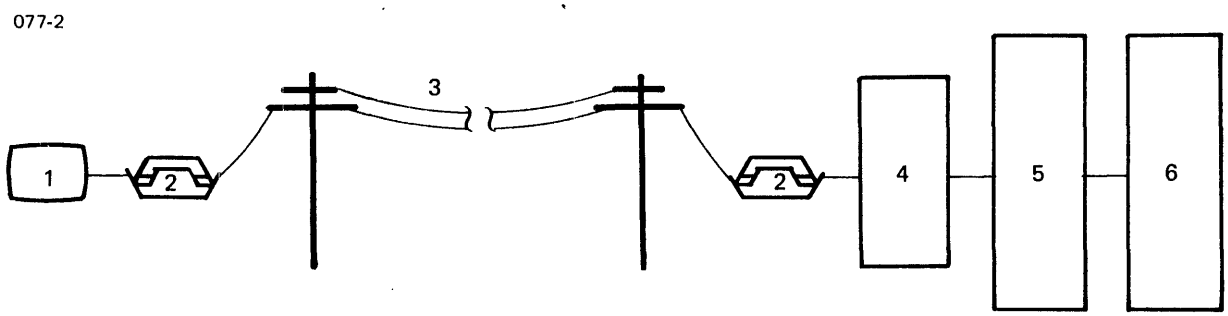


Figure 1-2 A Simplistic Data Communications Network

The components of the system in Figure 1-2 are numbered and represent the hardware usually needed in any data communications system. These devices are the:

1. Terminal
2. Modems
3. Transmission lines
4. Multiplexors and concentrators
5. OS/32 Data Communications Subsystem
6. Processor

Components 1 through 4 and 6 are not actually part of the OS/32 Data Communications Subsystem. They are, however, essential to the operation of a data communications system. The remainder of this chapter is, therefore, dedicated to a general discussion of these components so that a better understanding of their functions in a data communications system can be achieved. This discussion is intended for the reader who does not have an extensive background in data communications. The more experienced reader may skip this discussion. Chapter 2 contains a detailed discussion of the OS/32 Data Communications Subsystem.

1.3.1 Terminals

Communication between people and computers generally requires a terminal with a keyboard and some type of display device. Clearly, the CRT terminal provides the fastest and the most convenient access to the data stored or manipulated by a computer. While the CRT is the most popular type of data communications terminal, the characteristics of the many CRTs currently manufactured vary a great deal. In general, data communications terminals can be classified according to their communications mode, data transmission speed, data transfer mode and data code format. Sections 1.3.1.1 through 1.3.1.4 discuss these classifications.

1.3.1.1 Terminal Communications Modes

Data communications terminals can be divided into three modes of operation:

- Simplex
- Half-duplex
- Full-duplex

Figure 1-3 depicts these three modes.

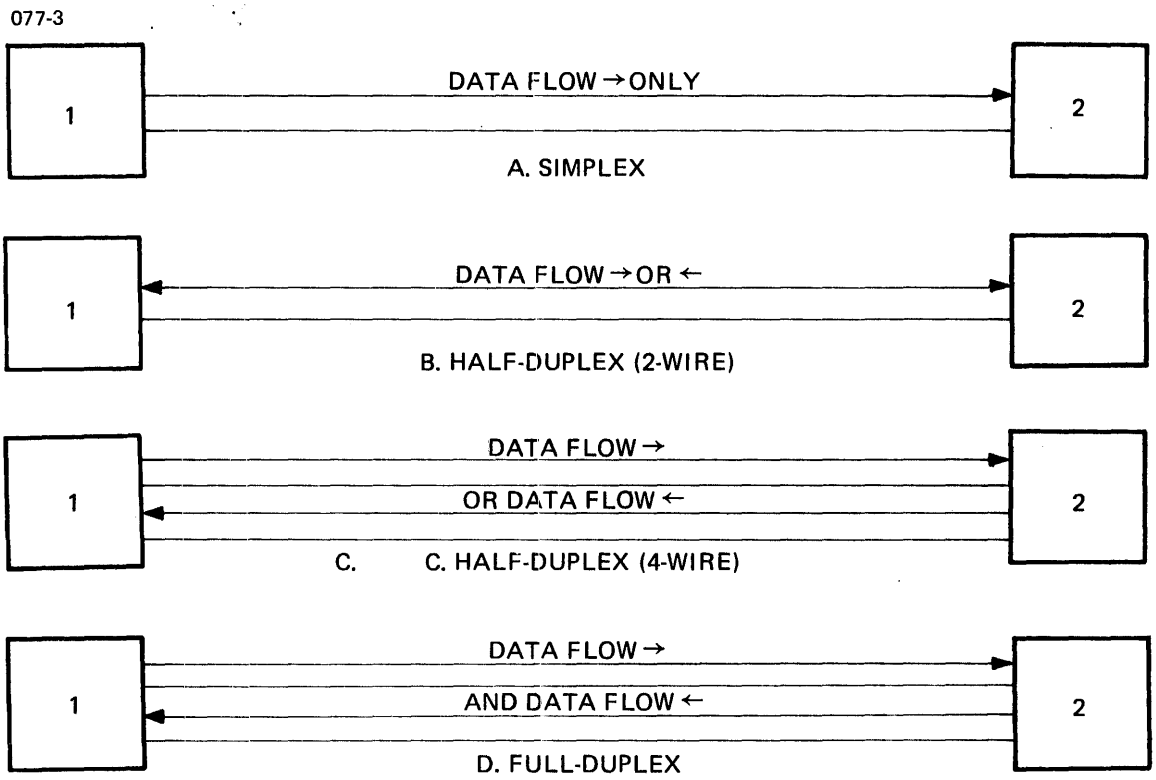


Figure 1-3 Terminal Communications Modes

A simplex terminal uses a simple 1-wire link with its receiving counterpart. In other words, a simplex terminal has a one-way only transmission path. Data cannot reverse directions.

A half-duplex terminal is a 2- or 4-wire terminal link that allows two-way communications, but transmissions can only occur in one direction at a time. The installation of special equipment that reverses the receive or transmit condition is required. Half-duplex terminals are often connected to 4-wire links to avoid turnaround delays (i.e., the modem switching itself from transmitting to receiving mode and vice-versa.)

In a full-duplex mode, the terminal is connected via a 4-wire link, which allows two-way simultaneous transmissions. Data can be sent and received at the same time.

An analogy can be made between these terminal modes and city streets. A simplex line is analogous to a one-way street in that traffic can only travel in one direction. The half-duplex mode is similiar to a narrow two-way street; traffic can travel in both directions, but only in one direction at a time. Full-duplex mode is the same as a standard two-way street; traffic can travel in both directions simultaneously.

1.3.1.2 Terminal Speeds

Although it is being replaced as a unit for measuring signaling speed, the baud is the basis of all other units of signaling speed. A baud is defined as the number of signal events per second. This definition is the basis for the two main units of signaling speed in use today: bits per second (bps) and characters per second (cps). If each bit transferred represents one signal event, the speed is expressed in bps. If a character represents one signal event, then the speed is expressed in cps.

Data terminals are divided into three basic speed catagories.

- Standard - up to 30cps
- Medium - 30 to 480cps
- High - over 480cps

1.3.1.3 Data Transmission Modes

Data can be transmitted using one of two possible modes: asynchronous or synchronous. Most low- and medium-speed terminals transmit data in the asynchronous mode. These terminals generate a coded character each time a key is depressed. Figure 1-4 illustrates the asynchronous transmission mode.

077-4

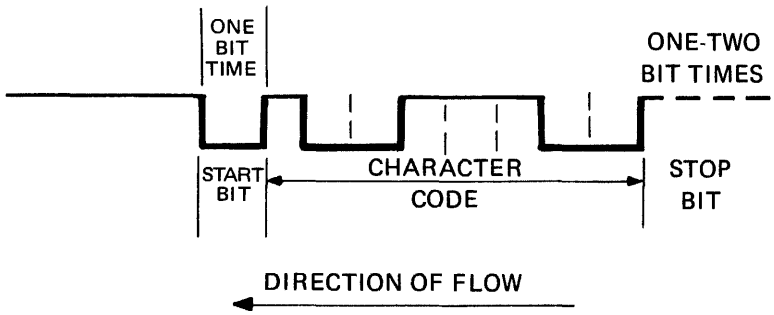


Figure 1-4 Asynchronous Transmission

In asynchronous transmission mode, each character is prefixed by a start bit and suffixed by one or more stop bits. In this mode, the performance of one operation is initiated by the signals that indicate the completion of the previous operation. In other words, a new character cannot be transmitted until the previous characters have been received. All bits within a character are sent at prescribed time intervals, but the data can have periods of inactivity while the terminal is waiting for the operator to input more data. In the asynchronous mode, the timing of the terminal and the central system are established independently of each other.

Most high-speed terminals transmit their data in the synchronous mode. In synchronous transmissions, the data is transmitted in long blocks with only a single framing pattern at the beginning of each block. These framing characters are referred to as 'sync' characters. The advantage of this mode of transmission is that each character transmitted consists only of data bits. Transmission facilities, therefore, are used more efficiently because there are no signal elements being wasted as start and stop bits. Figure 1-5 illustrates the synchronous transmission mode.

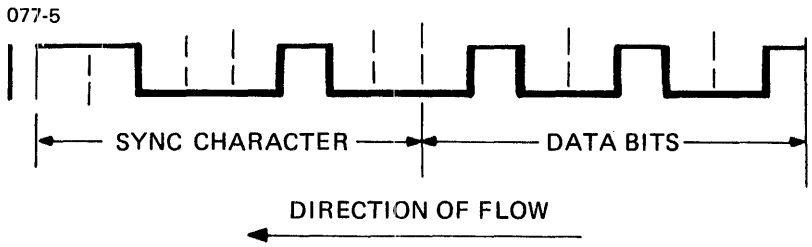


Figure 1-5 Synchronous Transmission

Unlike the asynchronous transmission mode, timing in the synchronous mode is established and maintained by the transmitting and receiving modems. These devices are synchronized so that transmissions occur at a fixed rate. No gaps are permitted between characters in the data block.

1.3.1.4 Data Codes

Data is represented in a computer by a collection or series of binary digits arranged in a particular order or grouping. These groupings are known as data codes and the three basic data codes used for data communications are Baudot, ASCII and EBCDIC.

- The Baudot code is a 5-bit code which makes only 32 unique characters possible. Because of this, two of the characters must be used as shift characters so that the code can handle the variety of letters, numbers and special characters necessary. The Baudot code has no provision for error checking. Terminal control is acheived by a line break or a special character sequence. Baudot was used principally on early teletype (TTY) machines; it is not in much use today.

- ASCII (American Standard Code for Information Interchange) is a 7-bit code with an additional bit to check for parity. It has unique code assignments for both alphanumeric and control functions. ASCII is the most widely used code today and is found on most micro and minicomputers. All Perkin-Elmer computers use the ASCII data code.

- EBCDIC (Extended Binary Coded Decimal Interchange Code) is an 8-bit code similiar to, but not compatible with, ASCII. It also uses a parity bit to check for errors. EBCDIC was created by IBM® for use on their large computer systems.

In parity checking, the processor verifies that all characters have either an even number (even parity) or an odd number (odd parity) of bits. After the transmission is completed, the receiving computer checks the parity to verify that all of the data has been transferred completely. If all of the characters do not have the correct number of odd or even bits, the computer notifies the transmitter that an error has occurred in transmission.

It is essential that all components of a data communications system can "converse" with each other in the same data code. If this is not the case, none of the data transferred will make sense to the receiving station. The solution is an emulator, a device or piece of software that makes a computer system supporting one data code behave like another system supporting a different code. Using emulators, it is possible for an ASCII terminal to "converse" with an EBCDIC processor.

IBM® is a registered trademark of International Business Machines Corporation.

1.3.2 Modems

Even though there are many other media available, the voice frequency channel or phone line is still the most popular medium for transmitting data because of its high availability and low cost. When using phone lines, a device known as a modem (modulator/demodulator) is required to interface between the processor or terminal and the communications line. The function of a modem (sometimes called a "data set") is to facilitate digital data communications over a telephone network by converting the digital (square wave) signal to an analog (sine wave) signal. It is this analog or "voice" signal that can be transmitted over the phone lines. At the other end of the line is a second modem that demodulates the analog signals back into digital signals. In our example of the telegraph, the modem is analogous to the telegraph operator who codes and decodes the telegrams. The modem can actually be a part of the computer or terminal (integrated) or it can be a stand-alone model. Modems can be divided into three types:

- Voice grade
- Wideband
- Hard-wired

Voice grade modems can be further divided into two speed catagories: low and medium. Low-speed modems are used on switched networks and interface with low-speed asynchronous terminals. These modem types generally operate at speeds of up to 300bps. Medium-speed modems are used with terminal controllers and high-speed terminals and can operate at speeds of 9,600bps and higher.

Wideband modems have very high operating speeds of 19,200 to 460,800bps. They are interfaced with special wideband leased lines and are used primarily for processor-to-processor transmissions.

Hard-wired modems operate on dedicated lines at speeds of up to one million bits per second. These modems are generally limited-distance devices that are very useful for short distance data communications.

Modems are not required when short distance direct lines can be used. Only when the transmission is to take place over telephone lines is a modem needed.

Another function of a modem is referred to as handshaking. Handshaking is the procedure that occurs when two modems are connected for the first time; it ensures that the communications link is available and functional. This is accomplished through the use of control signals sent down the line and returned. A handshaking sequence is unique to the particular type of modem.

Modems must operate in the same modes as the terminals to which they are connected. A half-duplex modem sends or receives data in only one direction at a time while a full-duplex modem can send and receive information concurrently. Each modem can transmit or receive either asynchronous or synchronous data.

1.3.3 Transmission Lines

Transmission lines are classified as one of three types according to the speed of transmission. The transmission types are:

- Low-speed (subvoice)
- Medium-speed (voice-grade)
- High-speed (wideband)

Low-speed lines are commonly used by Telex, TWX and other TTY transmissions. Both leased and switched networks can be used and can transmit data at up to 300bps. Telex and TWX are used primarily for transmitting business communications. Connections are established by dialing into these networks, but intercommunication is made via TTY rather than by voice.

Medium-speed networks are made up of voice-grade telephone lines. These telephone lines can be grouped into three catagories.

- Switched, or public, lines make up the standard telephone system and provide a dial-up service between the terminal and the processor. Switched lines can transmit data asynchronously up to 1,200bps or synchronously at rates of up to 4,800bps. This type of connection is less costly if the line's usage is low and infrequent.

- Leased lines are private point-to-point lines that are dedicated to a particular customer. The advantages of this type of line are that the connection between points is constant; it does not have to be established each time and the private line allows the user more bandwidth. These advantages make higher data transmission rates possible and make this type of connection less costly if the line is to be used frequently or constantly.

  In contrast to switched lines, leased lines can transfer data up to twice as fast and can achieve a speed up to 9,600bps.

- Special lines are 4-wire links that are available for two-way transmissions. These links usually have a higher speed and a lower error rate than switched lines, due to conditioning. Conditioning refers to the process of adjusting the electrical characteristics of the transmission path to control certain types and levels of distortion.

High-speed or wideband lines have transmission rates of 19,200bps or more. They are used primarily for intercomputer communication links.

1.3.4 Multiplexors (MUXs) and Concentrators

MUXs and concentrators are data communications devices that improve system efficiency by increasing the speeds of data transmissions.

A MUX is a device that receives and combines signals from many low-speed lines and transmits them together over the same high-speed communications channel.

A concentrator is a device that is used to interface many different terminals to a single host computer. Like a MUX, a concentrator combines the many low-speed lines into a single high-speed line.

In using data multiplexing, the instantaneous rate of data entering or exiting the terminal cannot exceed the data rate of the communications channel. A concentrator, however, uses a buffered resource sharing scheme so the instantaneous rate of data entering or exiting the terminal can exceed the data rate of the communications channel. Data concentration can, therefore, take advantage of idle terminal time, while data multiplexing cannot.

CHAPTER 2
OS/32 DATA COMMUNICATIONS SUBSYSTEM

2.1 INTRODUCTION

One of the most important components of any data communications system is the interface between the communications lines and the processor. This interface must be able to transform the signal levels and data rates produced in the communications circuit into levels and rates that the processor can understand. This is done with a unique combination of hardware and software that makes up the Perkin-Elmer OS/32 Data Communications Subsystem. Figure 2-1 shows the general breakdown of the subsystem.

Sections 2.2 through 2.5 discuss the components of the subsystem.

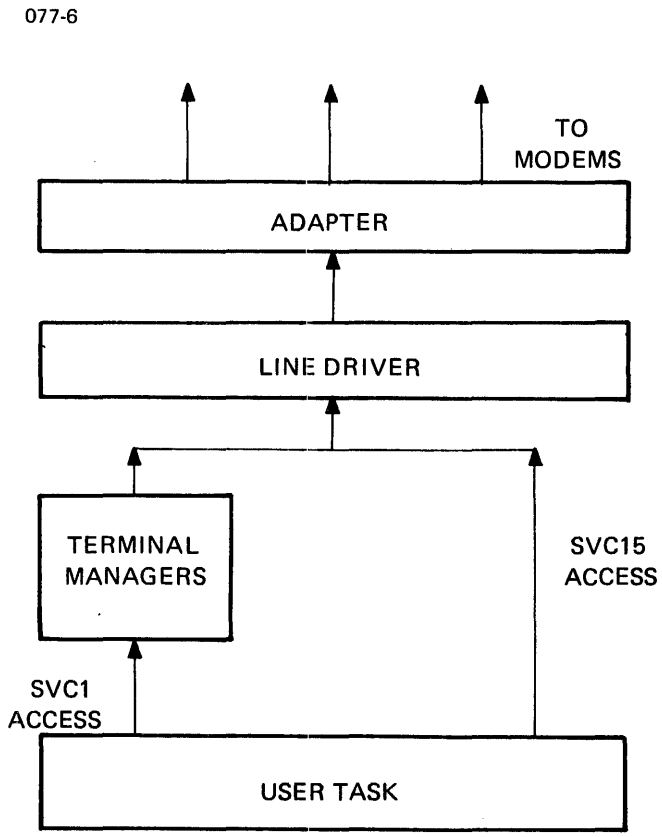


Figure 2-1 A Data Communications Subsystem

Perkin-Elmer supports two basic network types:

- PENnet is a distributed data communications network for long distance digital transmissions. It uses either leased or switched phone lines and cyclic redundancy checking (CRC) for error detection. PENnet is easy to install and use since no specialized knowledge of data communications networks is required to exploit its full facilities.

- Ethernet is a local area network (LAN) that uses a peer-to-peer protocol known as carrier-sense multiple access with collision detection (CSMA/CD). Components of an Ethernet network are connected by coaxial cable over a limited distance of about 1.5 miles with the use of repeater stations. This system transmits data at a rate of 10Mbits per second. For further information see the OS/32 Network Drivers Programming Reference Manual.

2.2 ADAPTERS

Communication adapters are hardware boards that interface the processor with a transmission facility and enable remote devices to communicate with the host computer. In general, an adapter contains circuits to generate and detect the control signals required to set up, take down and supervise the data communications channel and to provide proper status and interrupt information to the processor. There are two basic types of adapters: serial and parallel. Serial adapters interface with modems and usually use the Electrical Industries Association (EIA) RS-232 standard signals. Parallel adapters can connect indirectly to the computer's input/output (I/O) channels.

Perkin-Elmer's OS/32 Data Communications Subsystem supports three basic types of serial adapters:

- asynchronous
- bisynchronous
- zero-bit insertion and deletion (ZBID)

2.2.1 Asynchronous Adapters

In the asynchronous mode, data is transferred character-by-character. Each character is preceded by a start bit and is followed by one or two stop bits. The disadvantage of this type of transmission is that a substantial percentage of the bits transmitted are used simply for separating characters; thus, this method of transmission is more costly since the transmissions are longer.

The Perkin-Elmer OS/32 Data Communications Subsystem supports the following asynchronous adapters:

- The programmable asynchronous single line adapter (PASLA) provides an interface between 103/202-type modems over either a switched or leased network. It can accommodate local terminals that match the EIA RS-232 standard. This system has a high degree of flexibility in that it can be programmed for a variety of baud rates, character formats and line control functions. PASLA can also interface with either a half-duplex or full-duplex line. The major disadvantage is that each device must have its own PASLA.

- The 2-line MUX is a halfboard that is equivalent to two PASLAs.

- The 8-line MUX is a fullboard that is equivalent to eight PASLAs.

- The multiperipheral controller (MPC) contains the 8-channel data communications multiplexor (COMM MUX) consisting of four serial communication controllers. Each controller contains two independent, full-duplex channels that can be asynchronous, bisynchronous or ZBID interfaces, depending on the needs of the user.

2.2.2 Bisynchronous Adapters

Bisynchronous (binary synchronous or BISYNC) adapters use the BISYNC mode of data transmission. In bisynchronous communications, data is transmitted in a synchronous mode with special communications control characters that are specified for formatting text, indicating status, synchronizing functions and error control.

The OS/32 Data Communications Subsystem supports the following bisynchronous adapters:

- The synchronous 201 data set adapter contains the circuits necessary to generate and detect control signals that are needed to establish, maintain and terminate the data communications channel and provide status and interrupt information to the computer. The data transfer between a 201 modem and a 201 data set adapter occurs in the bit serial mode with a synchronizing bit clock supplied by the modem for both transmitting and receiving. Special character recognition (other than the SYNC characters), block character checking and generation, and code translation are accomplished by the processor under program control.

- The quad-synchronous adapter (QSA) is designed to operate with synchronous modems and provides an interface between a selector channel (SELCH) bus and four 2-wire or 4-wire synchronous modems. Like the 201 data set adapter, data transfer between the QSA and a modem is in the bit serial mode with a bit clock supplied by the data set. The QSA also has the logic necessary to generate and detect control signals which are needed to establish, maintain and terminate a data communications channel, and provide proper status and interrupt information to the processor.

- The single-synchronous adapter (SSA) is a synchronous adapter that, like the 201 and the QSA adapters, contains the logic necessary to generate and read the control characters which are needed to set up, supervise and terminate a data communications channel, and provide the proper status and interrupt information to the processor.

- The MPC can be used as a bisynchronous adapter as well as an asynchronous adapter. See Section 2.2.1 for further details.

2.2.3 Zero-Bit Insertion/Deletion (ZBID) Adapters

ZBID adapters are synchronous adapters different from BISYNC adapters. ZBID devices send data in frames, each of which starts and ends with a particular bit sequence called a flag. ZBID gets its name from the need to prevent the data being transmitted from resembling the ZBID flag sequence. To accomplish this, the sending station inserts a zero bit, whenever necessary, to prevent the data from appearing as a flag sequence. These zero bits are removed by the receiving station. Hence its name, zero bit insertion and deletion. ZBID is synonymous with synchronous data set link control (SDLC) transmission modes.

The OS/32 Data Communications Subsystem supports the following ZBID adapters.

- QSA
- SSA
- MPC

The Ethernet data link controller (EDLC) provides processor-to-processor serial communications at a rate of 10Mbits per second over a common coaxial cable. Communication sequences over the half-duplex channel are divided into packets with frame sizes ranging from 64 to 1,518 bytes. Each frame has two 48-bit address fields, one 16-bit type field and a 32-bit cyclic redundancy check (CRC) frame check sequence for error detection.

The QSA, SSA and MPC support ZBID devices as well as bisynchronous devices. See Section 2.2.2 for further details.

2.3 LINE DRIVERS

A line driver provides a standard software interface between the particular communications adapter and the user task (u-task) or special support programs in the processor. In general, a line driver allows the user to specify the control sequence and the data necessary to send or receive a transmission over a data communications line. More specifically, a line driver:

- interfaces with the communications hardware (adapter),
- controls reads and writes to or from the communications lines,
- performs certain basic timing functions, and
- controls some modem signals.

A line driver is essentially "dumb"; it does not know what is at the end of the communications line with which it is interfaced. The line driver, however, does know how to control the adapter in order to pass data to or from the communications line. The protocols and procedures necessary to initiate, maintain and terminate the communications link must be supplied by either special support software or the u-task with which the driver is interfaced.

The OS/32 Data Communications Subsystem supports four line drivers.

- DASY is an asynchronous line driver that interfaces with OS/32 asynchronous adapters (e.g., PASLA).
- DCSY is a character synchronous line driver that interfaces with bisynchronous adapters (e.g., Q/A).
- DZBD is a ZBID line driver that interfaces with ZBID adapters (e.g., MPC). DZBD controls line synchronization, data transparency and data blocking for medium- to high-speed (1,200 to 19,200 baud) communications lines.
- DETH is the Ethernet line driver, which is a ZBID line driver that interfaces with the EDLC.

2.4 DEVICE-INDEPENDENT ACCESS

The OS/32 Data Communications Subsystem supports two levels of data communications access:

- Device-independent
- Device-dependent

Device-dependent access is discussed briefly in Section 2.5 and more fully in Chapter 5.

The device-independent access level is ideal for the noncommunications-oriented user because it allows the user to extend the range of the system beyond the computer room. Device-independent access permits the user to communicate with remote terminals over communications lines with normal OS/32 supervisor call 1 (SVC1) I/O conventions just as if the data communications terminals were local peripherals. Device-independent access is discussed more fully in Chapter 4.

2.4.1 Terminal Managers

Device-independent support of communications devices is provided by OS/32 terminal managers. In data communications, a logical device, such as a CRT, processor or printer connected over some type of communications line, is known as a terminal. A data communications terminal needs a coordination protocol and certain buffer management procedures in order to be supported over a particular type of communications facility. Data communications terminals can be treated as local devices since the details of hardware requirements are handled by terminal managers.

A terminal manager is actually software support that contains the logic to initiate, maintain and terminate transmissions to a data communications terminal. The user accesses the terminal manager through SVC1. The terminal manager, in turn, calls a line driver and, using the line driver features, supports the terminal in a device-independent manner. The concept of a communications device with device-independent access by a terminal manager is analogous to the concept of a direct access file with file manager support. The user program presents data to or requests data from the terminal just as it would for any local device or file. The terminal manager performs the necessary physical I/O, communications line handshaking and/or data formatting.

The OS/32 Basic Data Communications Subsystem supports four different terminal managers; they are depicted in Figure 2-2.

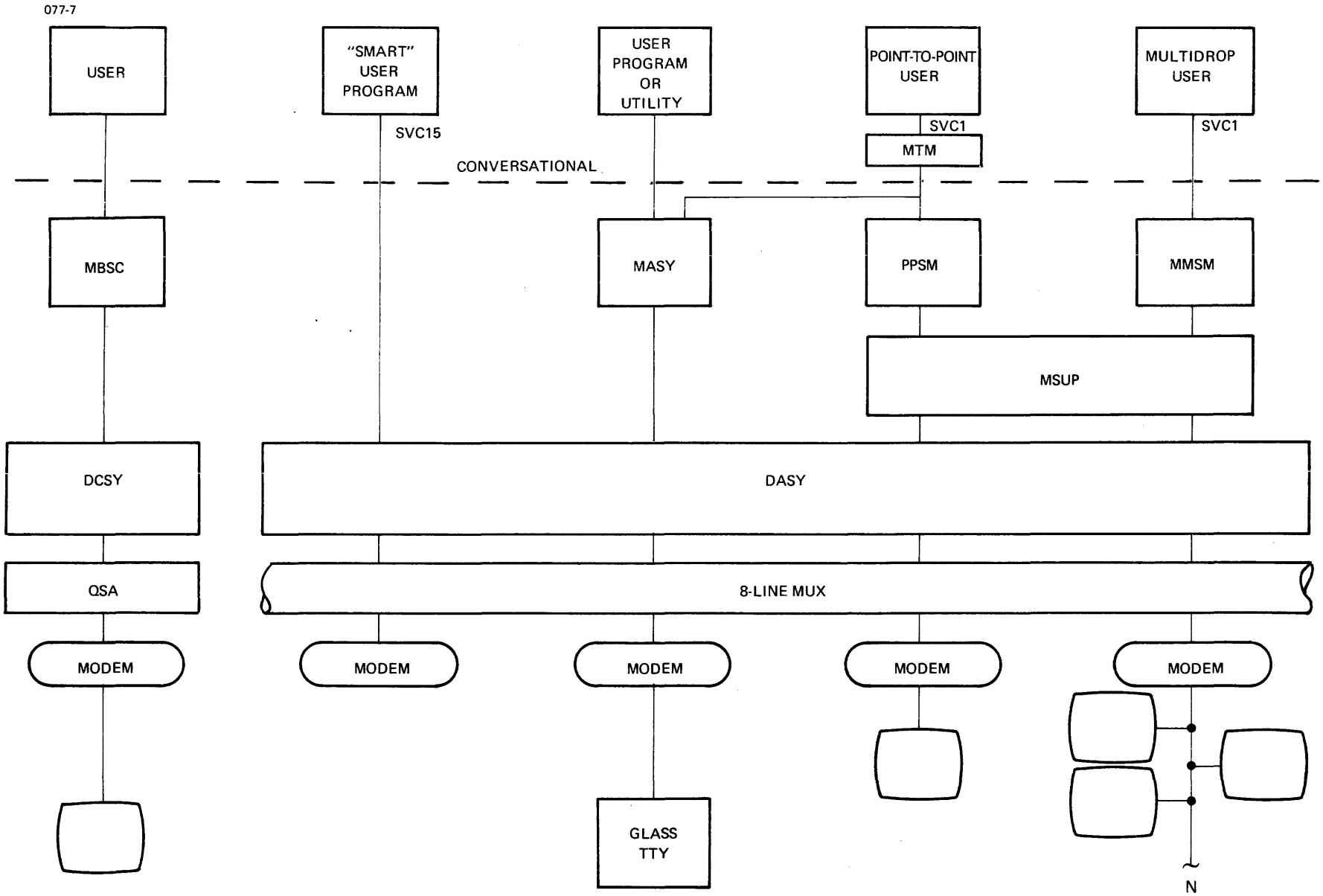


Figure 2-2 OS/32 Terminal Managers

- MASY is an unbuffered terminal manager that interfaces the DASY driver with "dumb" terminals or printers (device codes 147). See the OS/32 System Generation (Sysgen) Reference Manual for further information on device codes. MASY works in a conversational mode only and performs some very basic formatting functions (e.g., carriage returns (CRs), line feeds and trailing space truncations).

- MBSC is a buffered terminal manager that interfaces with the DCSY line driver and intelligent terminals. MBSC works with binary synchronous transmissions using the 2780 or 3780 protocols.

- PPSM is a nonbuffered terminal manager that interfaces between DASY and intelligent terminals in a point-to-point environment. It is used with devices having device codes 156 and 157. PPSM supports conversational data modes, as does MASY, but PPSM also supports a block transmission mode. This driver is used by OS/32 Reliance.

- MMSM is a nonbuffered terminal manager that interfaces between DASY and intelligent terminals in a multidrop environment. It is used with devices having the device code 158. MMSM has the same block support as PPSM, but is used for a multidrop environment (i.e., many terminals connected to a single data communications line).

Both PPSM and MMSM use an additional software module known as MSUP. MSUP contains code that is used by both PPSM and MMSM in conjunction with their own differing software. MSUP's use is analogous to two user tasks, both different, but both needing to access the same library subroutine. Because of this, PPSM and MMSM could be more correctly called PPSM-MSUP and MMSM-MSUP.

2.4.2 Protocols

A protocol is a set of conventions for the establishment and maintenance of a data transmission line. In other words, a protocol is a set of rules that the computer and/or terminal follow when using data communications facilities. Military radio communications are a good example of communications protocol. A communications link is requested with "This is XYZ calling ABC." Acknowledgments are made with a "Roger" and the end of a transmission is signified by an "Over". These particular responses are a part of the established protocol of military communications. A data communications protocol is similiar, but much more regimented. In a data communications protocol, the conventions are precise and must be followed exactly.

Data communications protocols provide facilities for error handling, handshaking, coordination procedures for switched networks and failure recovery.

Error handling includes both error detection and error correction techniques. Error detection is based on redundant information in the transmitted message. Correction of detected errors is generally done by retransmission of a portion of the message as provided for in the protocol. Another method of error correction is the replacement of an error with a special character for manual error correction.

Character errors can be checked by a method known as vertical redundancy checking (VRC). In this technique, a parity bit is added to the character bits and the process checks for either odd or even parity. If a character with the incorrect parity is detected, it is treated as an error.

The disadvantage of VRC is that it can only detect a single or an odd number of bit errors. Another technique of error checking is needed to find multiple errors. One such technique is to place parity bits along a data block in addition to those used by VRC. These special bits are added at the end of the block so the bit parity is always odd. This technique is referred to as longitudinal redundancy checking (LRC). LRC is computed by taking the exclusive-OR of a zero character successively with each character in the block.

CRC is another, more powerful way of detecting errors. This method is essentially a combination of VRC and LRC. In CRC, a polynomial formula is used to generate a special checking sequence at both ends of the transmission network. The result of this sequence is sent by the transmitting station to the receiving station and is compared to the value of the sequence calculated by the receiving station. If the two values are not equal, errors have occurred. CRC is capable of detecting multiple bit errors and many burst-type errors.

A protocol known as the data link control protocol defines the general control framework for the data communications network. A terminal can have its own internal protocol, different from the general control protocol. However, some type of control over terminals is necessary in multiterminal networks where there are many terminals connected to a single line. The process of polling terminals uses a protocol to request and establish communications links between the processor and a terminal requesting the link.

Another type of protocol is known as a communications protocol. This protocol provides rules for transmission and reception of the different types of data streams. For example, the advanced data communications control procedure (ADCCP) is a synchronous, bit-oriented, code-independent, interactive protocol specifically designed for computer-based data communications over a full-duplex mode. Other communications protocols are designed to be character-oriented, or for asynchronous transmissions, or to be used over a half-duplex transmission mode, etc.

For information on the protocols supported by the OS/32 Data Communications Subsystem, see the OS/32 Asynchronous, Bisynchronous, Bit-Synchronous and Network Drivers Programming Reference Manuals.

2.5 DEVICE-DEPENDENT ACCESS

Device-dependent support of communications devices is often referred to as the line driver access method because the user makes use of line driver features directly through the SVC15 parameter block. Direct use of data communications line drivers provides the more primitive functions that enable a user to tailor a communications system to a particular need. The position of the line drivers in the OS/32 Data Communications Subsystem makes it easy for the user to specify the special control sequences needed to complete a transmission. For this level of support, the u-task need only be assigned a communications line by the operating system. The flexibility of this system allows straightforward tailoring of the communications system and such features as command chaining, buffer management and task interrupts (traps).

The disadvantage of device-dependent access, as opposed to the OS/32 terminal managers, is that the u-task must specify all data communications control sequences through the SVC15 parameter block. It is recommended that, when possible, device-independent access be used.

CHAPTER 3
DEVICE HANDLING

3.1 INTRODUCTION

Data communications lines and/or terminals must be assigned or closed just like other devices. Also, line control blocks (LCBs) for OS/32 Data Communications buffered terminal managers need to be allocated and deleted just like direct access files. These allocations, assignments, etc., can be accomplished by using either the supervisor call 7 (SVC7) parameter block or the specific OS/32 operator commands.

The SVC7 parameter block is covered in Sections 3.2 and 3.3. The use of system operator commands is discussed in Section 3.4.

3.2 SUPERVISOR CALL 7 (SVC7)

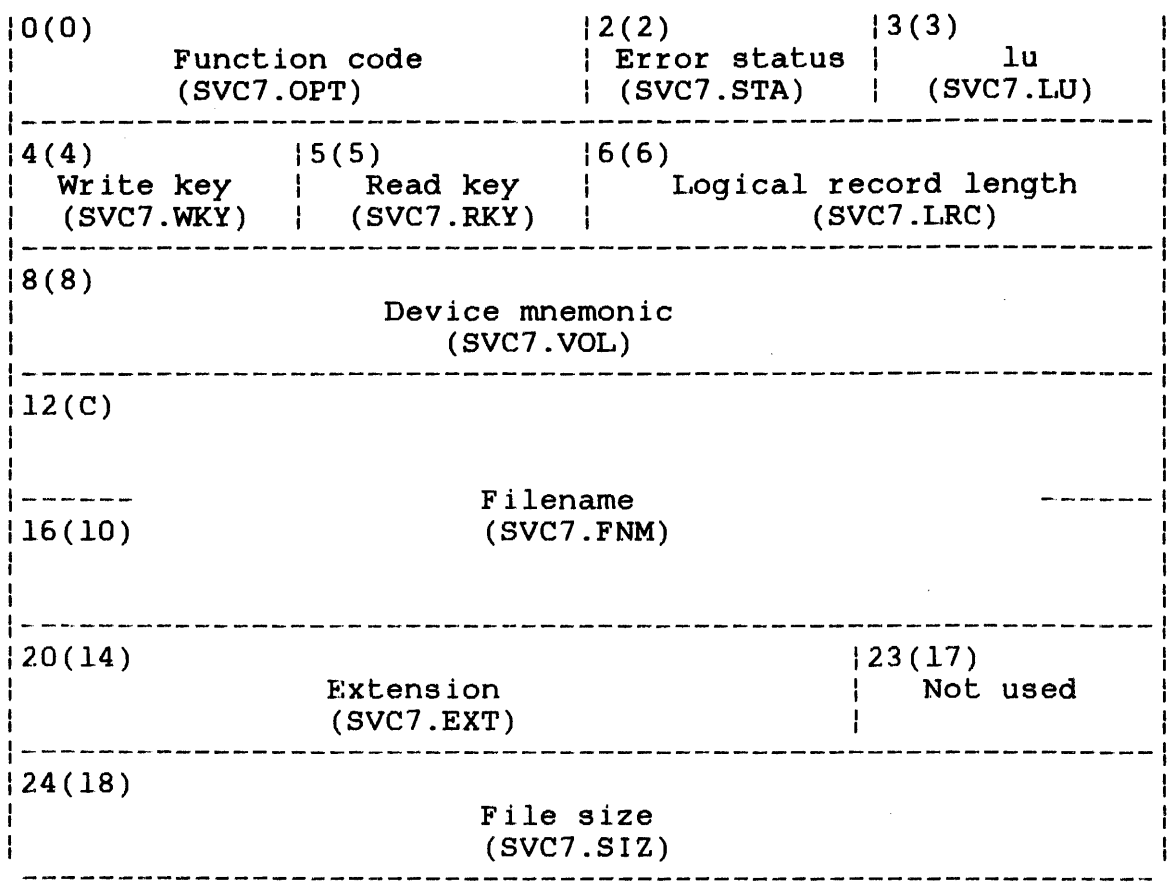
SVC7 is used in data communications to:

- allocate and delete LCBs for buffered terminal manager access (SVC1),
- assign and close logical units for line driver (SVC15) and terminal manager (SVC1) access,
- checkpoint buffered terminal manager access (SVC1),
- rename and reprotect data communications lines (SVC15 access devices) and terminals (SVC1 access devices), and
- allocate and delete drop control tables (DCTs) for channel terminal manager (SVC1) access or multidrop lines.

This discussion of SVC7 includes only information pertinent to data communications. For in-depth discussions of SVC7 support, see the OS/32 Supervisor Call (SVC) Reference Manual.

3.2.1 Supervisor Call 7 (SVC7) Parameter Block

SVC7 provides device-handling functions supported by the data communications subsystem. These functions are accomplished through the SVC7 parameter block and coding shown in Figure 3-1.




```
            SVC   7,parblk
            .
            .
            .
            ALIGN 4
parblk      DC    X'function code'
            DS    1
            DB    lu
            DB    'write key'
            DB    'read key'
            DC    H'record length'
            DC    C'4-character device mnemonic'
            DC    C'8-character filename'
            DB    C'3-character extension'
            DB    Not used for data communications
            DC    F'file size'
```

Figure 3-1 SVC7 Parameter Block Format and Coding

3.2.1.1 Function Code Field

The function code field is a 2-byte field that contains the hexadecimal number indicating the function to be performed. The format of the function code field is shown in Figure 3-2.

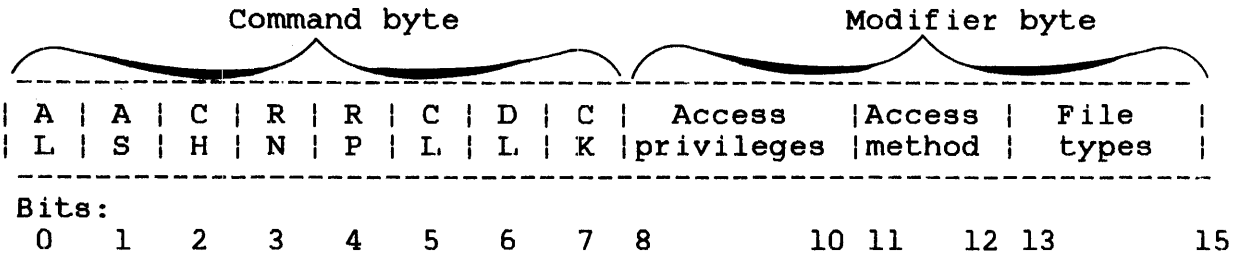


Figure 3-2 SVC7 Function Code Field

The SVC7 function code field is divided into two 1-byte sections. The first byte is referred to as the command byte and the second as the modifier byte.

The command byte, bits 0 through 7, requests one or more file management functions. If more than one command bit is set, the respective functions are processed sequentially, left to right. The modifier byte, bits 8 through 15, modifies commands specified in the command byte.

The data communications function of each bit setting in the SVC7 function code field is explained in Section 3.3.

NOTE

The modifier field is not used on a fetch attributes call. Instead, the device code is returned in this field.

3.2.1.2 Error Status Field

The error status field is a 1-byte field that receives appropriate error codes when an error occurs while executing SVC7.

The interpretation of the error status field depends on the command specified in the call. It is compatible with nondata communications status returns (see the OS/32 System Level Programmer Reference Manual). A zero status always means the desired options were performed without error. Table 3-1 details the SVC7 error status field bit settings.

TABLE 3-1 SVC7 ERROR STATUS CODE BIT SETTINGS

BIT	HEX	FUNCTION	MEANING
0	(00)	All	No error; the requested functions are complete
1	(01)	A,O	Illegal function; illegal FT or AM modifier
2	(02)	All but A,D	Logical unit (lu) error; illegal lu
3	(03)	A,O,D	Device error; no such device in the system
4	(04)	A,O,N,D	Name error; mismatch on filename.ext field (may indicate failure to allocate LCB when assigning for buffered access)
5	(05)	A	Size error; erroneous LRECL or SIZE field
6	(06)	O,D	Protect error; erroneous protection keys
7	(07)	O,H,N, P,D	Privilege error; unable to obtain requested privilege
8	(08)	O	Buffer error; unable to obtain requested privilege
9	(09)	All but A,D	Assignment error; lu not assigned or attempted to mix SVC1 and SVC15 access to same device
10	(0A)	A,N,P,D	Type error; nondirect access device or device off-line
11	(0B)	A,O,N,D	File descriptor (fd) error; illegal syntax
12	(80-FF)	O,C,D,T	Input/output (I/O) error; interpreted as SVC1 status byte

3.2.1.3 Logical Unit (lu) Field

This 1-byte field contains a hexadecimal number indicating the lu assigned to the data communications device for which the function is requested. This field is used for all SVC7 functions except allocate and delete.

3.2.1.4 Read and Write Key Fields

Protection keys for devices are specified in this halfword. These keys are required for the allocate, assign, reprotect and delete functions. It is recommended that they be reset to zero.

When executing the SVC7 fetch attributes function, the device attributes are stored in the write and read key fields of the parameter block.

3.2.1.5 Logical Record Length Field

The logical record length field is a 2-byte field containing a decimal number that indicates the physical record length when allocating the buffered logical terminal manager.

When executing a fetch attributes function, this field receives a number indicating the physical record length of the device assigned to the lu.

3.2.1.6 Device Mnemonic Field

The device mnemonic field is a 4-byte field containing ASCII code that indicates the name of the data communications line to be used when the allocate, assign, delete and fetch attributes functions are executed.

3.2.1.7 Filename Field

The filename field is an 8-byte field containing ASCII code that indicates the buffered logical terminal described by the LCB that is being allocated or assigned.

When executing a fetch attributes function, this field receives the filename from the data communications device currently assigned to the lu specified in the parameter block.

3.2.1.8 Extension Field

This 3-byte ASCII field is treated as an extension of the filename.

3.2.1.9 File Size Field

This 4-byte field contains a hexadecimal number indicating the block size established when an LCB is allocated.

3.3 COMMAND FUNCTIONS

This section briefly describes the command functions of the function code field as they relate to data communications. For a more detailed explanation of these command functions, see the OS/32 Supervisor Call (SVC) Reference Manual.

3.3.1 Allocate Function

The allocate function reserves memory space for LCBs or DCTs for SVC1 access. The space reserved must be less than the user's remaining allotment of system space.

The required parameter block fields for this function are:

- Bits 0 and 13 through 15 (file type) of the function code should be set (hex mask X'8007').
- Write key field
- Read key field
- Logical record length field
- Device mnemonic field
- Filename field
- Extension field
- File size field

For multidrop lines, a DCT must be allocated for each drop assigned.

3.3.2 Assign Function

The assign function establishes a logical connection between a line or terminal and the task. This is accomplished through a specified lu for either SVC1 or SVC15 access. For buffered terminal access, the device mnemonic, filename and extension fields specify the name of a logical terminal given to a previously allocated LCB.

The required parameter block fields for this function are:

- Bit 1 of the function code must be set; bits 8 through 10 (access privilege), 11 and 12 (access method) and 13 through 15 (file type) should be set as needed (hex mask X'40nn').
- lu field
- Write key field
- Read key field
- Device mnemonic field
- Filename field
- Extension field

Assignments of the same device to multiple logical units are governed by the specified access privileges. Multiple assignments are valid as long as assignments specify the same access method. Assignments of a device for SVC15 access when it is already assigned for SVC1 access, or vice versa, is illegal regardless of the specified access privileges. The only legal access privileges for SVC15 assignment are those specifying both read and write access (B 100 through B 111).

3.3.3 Change Access Privileges Function

This function changes the current access privileges of an assigned device to the access privileges specified in the parameter block. The new access privileges must be compatible with the existing ones; otherwise, the existing access privileges of the device remain unchanged and an error status is returned. If the device is assigned with read-only privileges, a write access privilege change is not allowed. See the OS/32 Supervisor Call (SVC) Reference Manual for a table of allowable access privilege changes.

Not all OS/32 terminal managers support change access privileges. See the appropriate terminal manager's manual for further information.

The required parameter block fields for this function are:

- Bit 2 of the function code must be set; bits 8 through 10 should be set, as needed, to indicate the desired access privileges (hex mask X'20n0').
- lu field

If an error is encountered while processing this request, the device remains assigned with its original access privilege. A device assigned for SVC15 access can only request read/write access.

3.3.4 Close Function

The close function breaks the logical connection between an lu and a data communications line and terminal by closing the currently assigned line. For logical units assigned for a buffered terminal (SVC1) write access, partially filled buffers are written to the line (i.e., checkpointed). If there are no other assignments to this device, the data terminal ready (DTR) signal to the modem is dropped, disconnecting a switched line. This occurs whether or not the device is system generated (sysgened) as a switched line.

The required parameter block fields for this function are:

- Bit 5 of the function code field must be set (hex mask X'0400').
- lu field

3.3.5 Delete Function

The delete function removes a currently unassigned LCB from memory. The required parameter block fields are:

- Bit 6 of the function code field must be set (hex mask X'0200').
- Write key field
- Read key field
- Device mnemonic field
- Filename field
- Extension field

If the logical terminal name matches the name in the LCB, the LCB is deleted.

3.3.6 Checkpoint Function

The checkpoint function ensures that terminal data buffered in memory is transmitted to the terminal. The required parameter block fields for this function are:

- Bit 7 of the function code must be set; bits 11 and 12 should be set, as needed, to specify the required access method (hex mask X'01nn').
- lu field

If the lu requested is not assigned, an error code of X'81' is returned.

3.3.7 Fetch Attributes Function

For proper operation, some programs require knowledge of the physical attributes of the device associated with a given lu. The fetch attributes function accesses and sends this information to the SVC7 parameter block. These attributes include the device mnemonic, filename, extension and buffer size. Device codes are sent to the modifier byte of the function code field and device attributes are stored in the write and read key fields. The logical record length field receives a device physical record length. The field differences for the fetch attributes function are illustrated in the OS/32 Supervisor Call (SVC) Reference Manual.

When executing this function, the modifier (device codes) field receives a hexadecimal number indicating the device type. The System Generation/32 (Sysgen/32) Reference Manual lists all device codes.

The device attributes field receives a hexadecimal number indicating certain device attributes. All supported attributes and corresponding masks are listed in the OS/32 Supervisor Call (SVC) Reference Manual. The hex mask for this function is X'0000' (no command bits set).

3.3.8 Vertical Forms Control (VFC) Function

The VFC option turns the VFC function on or off for a particular device. To execute this function, only the first four bytes of the SVC7 parameter block are required.

To use the VFC function, the command byte of the function code should be set to X'FF'. To turn on the VFC function for a particular device, set the modifier byte to X'20'. To turn the function off, set the modifier byte to X'21'. The error status and lu fields are the same for all SVC7 services.

VFC is supported by device codes 156 and 157 only. See the System Generation/32 (Sysgen/32) Reference Manual for further information.

3.3.9 Rename Function

The rename function changes the device mnemonic table (DMT) entry for the device. This routine also changes the LCB.NAME and LCB.EXT fields when it finds a buffered terminal.

The rename function may not be supported for all data communications terminal managers. See the appropriate terminal manager's manual for further information.

3.3.10 Reprotect Function

When parameters are passed by the user, the reprotect function changes the read and the write keys. Also, if a device is buffered, REP.DCB gets the device control block (DCB) address from the LCB.

3.4 OPERATOR COMMANDS

Data communications SVC7 support can be invoked via the OS/32 System operator commands. A brief description of the commands pertinent to data communications is given in this section. See the OS/32 Operator Reference Manual for a detailed description of these commands.

The term ITAM, which appears in the following commands, is synonomous with the term data communications.

3.4.1 ALLOCATE Command

The ALLOCATE command is used to allocate an LCB for data communications buffered terminal access.

Format:

```
ALLOCATE fd, ITAM[, [{lrecl}] [/[{bsize}]] [,[{keys}]]]
                    [{ 80  }]  [{  1  }]   [{0000}]
```

Parameters:

fd	is the file descriptor of the buffer or device to be allocated.
ITAM	specifies that the device to be allocated is a data communications device.
lrecl	is a decimal number specifying the logical record length of a data communications device. It cannot exceed 65,535 bytes. lrecl may optionally be followed by a slash (/), which delimits it from bsize. If no logical record length is specified, 80 is the default.
bsize	is a decimal number specifying the physical block size to be used for buffering and debuffering operations on the data communications device. When ITAM is specified, bsize represents the buffer size in bytes. For ITAM buffers, this parameter cannot exceed the maximum block size established by the sysgen procedure. If bsize is omitted, the default value for the ITAM buffer will be the "standard" buffer size for the particular device.
keys	specifies the write and read protection keys.

Functional Details:

The fd must specify the device mnemonic for the desired data communications terminal, plus a unique filename and extension for each lu to which the terminal is to be assigned. If an LCB for the specified filename and extension already exists, a NAME-ERR message is returned.

The operator DISPLAY ITAMTERM command can be used to display allocated LCBs in a manner analogous to that of the DISPLAY FILES command. See the OS/32 Operator Reference Manual for a description of these commands.

Examples:

The following allocates an LCB for a binary synchronous terminal called RJE.IN on device BSC1:.

```
AL BSC1:RJE.IN,ITAM,80/404
```

The following allocates an LCB for a binary synchronous terminal called INPUT on device BSC1. Here the default logical record buffer and block size are used.

```
ALLOCATE BSC1:INPUT,ITAM
```

3.4.2 ASSIGN Command

The ASSIGN command assigns a data communications device to a task's logical units.

Format:

Parameters:

lu	is a decimal number specifying the logical unit number to which a device or file is to be assigned.
fd	is the file descriptor of the device or file to be assigned.
access privileges	specifies the desired access privileges. Possible access privileges are:

SRO	Sharable read-only (SVC1 access)
ERO	Exclusive read-only (SVC1 access)
SWO	Sharable write-only (SVC1 access)
EWO	Exclusive write-only (SVC1 access)
SRW	Sharable read/write (default)
SREW	Sharable read, exclusive write
ERSW	Exclusive read, sharable write
ERW	Exclusive read/write

NOTE

The ASSIGN command is rejected if the specified access privileges cannot be granted. For SVC15 access, the assignments must specify both read and write access. Assignment for SVC15 access is rejected if read-only or write-only access is requested. Assignment for SVC15 access is also rejected if the device is already assigned for SVC1 access, and vice versa. See the OS/32 Operator Reference Manual.

keys	signify the write/read protection keys of the device to be assigned.
SVC1 SVC15 SVCF VFC	signify which SVC parameter block is to be used. SVC15 signifies that the specified device is to be assigned for SVC15 access. If SVC15 access is specified, VFC cannot be specified. The default value is SVC1, which specifies that the device is to be assigned for SVC1 access.

Examples:

The following example assigns an asynchronous line, PALO: to lu 9 for SVC15 access. SRW access privileges and zero keys are assumed.

```
AS 9,PALO:,,,SVC15
```

The following example assigns the bisynchronous terminal for which an LCB was allocated in the first allocate example to lu1 with SRW access.

```
AS 1,BSC1:RJE,IN
```

3.4.3 CLOSE COMMAND

The CLOSE command permits the operator to close (unassign) one or more devices assigned to the currently selected task's logical units.

Format:

$$\underline{\text{CLOSE}} \left\{ \begin{matrix} lu_1 \left[, lu_2 , \ldots , lu_n \right] \\ \underline{\text{ALL}} \end{matrix} \right\}$$

Parameters:

lu	is a decimal number specifying the logical unit assignments.
ALL	specifies that all logical units of the currently selected task are to be closed.

Functional Details:

Closing an unassigned lu does not produce an error message. A CLOSE command can be entered only if the referenced task is dormant or paused.

Examples:

The following example closes logical units 1, 3 and 5 of the currently selected task.

```
CL 1,3,5
```

The following example closes all logical units of the currently selected task.

```
CLOSE A
```

```
-----------
| DELETE  |
-----------
```

3.4.4 DELETE COMMAND

The DELETE command is used to delete currently unassigned LCBs from memory.

Format:

DELETE fd_1 [,fd_2,...,fd_n]

Parameter:

fd identifies the LCBs to be deleted.

Functional Details:

The LCB being deleted must not be currently assigned to any lu of any task.

3.4.5 XALLOCATE COMMAND

The XALLOCATE command allocates an LCB. If an LCB with the specified fd exists, it is deleted and an LCB with the same fd is allocated.

Format:

XALLOCATE fd, ITAM [,[{lrecl / 80}] / [[{bsize / 1}]][, [{keys / 0000}]]]

Parameters:

fd	is the file descriptor of the buffer or device to be allocated.
ITAM	specifies that the device to be allocated is a data communications device.
lrecl	is a decimal number specifying the logical record length of a data comunications device. It cannot exceed 65,535 bytes. lrecl may optionally be followed by a slash (/), which delimits it from bsize.
bsize	is a decimal number specifying the physical block size to be used for buffering and debuffering operations on the data communications device. When ITAM is specified, bsize represents the buffer size in bytes. For ITAM buffers, this parameter cannot exceed the maximum block size established by the sysgen procedure. If bsize is omitted, the default value for the ITAM buffer will be the "standard" buffer size for the particular device.
keys	specify the write and read protection keys.

Functional Details:

The XALLOCATE command differs from the ALLOCATE command in that if an attempt is made to allocate an existing LCB with the ALLOCATE command, an error message is given. With the XALLOCATE command, however, if the LCB to be allocated already exists, no message is generated and the LCB is deleted and reallocated. Otherwise, XALLOCATE and ALLOCATE behave in the same manner.

3.4.6 XDELETE Command

The XDELETE command is used to delete one or more LCBs. If the LCB does not exist, no error is generated.

Format:

XDELETE fd_1 [,fd_2,...,fd_n]

Parameter:

fd identifies the LCBs to be deleted.

CHAPTER 4
DEVICE-INDEPENDENT ACCESS

4.1 INTRODUCTION

Device-independent access is designed for the noncommunications-oriented user who wishes to use data communications facilities without the trouble of direct line driver control. Such control is provided for the user by the OS/32 terminal managers. A terminal manager contains the logic to initiate, maintain and terminate a data communications link, thereby freeing the user from the need to control these line driver functions through the user task (u-task). For most basic communications tasks, use of OS/32 terminal managers is sufficient to accomplish the link.

4.2 TERMINAL MANAGER ACCESS

The terminal manager provides access to remote or local devices for the following uses:

- Programs that can access local or remote devices without recompilation.
- The noncommunications-oriented user who wishes to access remote facilities without knowledge of line protocols and codes - in addition to functions common to local and remote devices, terminal managers provide functions to connect and disconnect devices from a communications line.
- The communications-oriented user who determines that a standard terminal manager provides adequate support for an application without special-purpose software.

While data communications line drivers can communicate with different types of devices, the terminal manager is designed to support only a single device type or a group of similiar device types.

In general, terminals can be accessed in either a buffered or nonbuffered mode. For nonbuffered access, data is transferred directly to or from the user buffer and every request from the user program requires at least one physical transmission over the communications line.

For buffered access, data is transferred between the user buffer and a system buffer. The terminal manager initiates data transmissions only when necessary. The system buffers needed by buffered access are contained in a system structure known as the line control block (LCB). Before accessing a terminal in the buffered mode, the LCB must be allocated in memory. At allocation time, the logical buffered terminal is given a name and the logical record length and block size are specified. After the program finishes accessing the terminal, the LCB can be deleted from memory. Supervisor call 7 (SVC7) calls or the OS/32 command language provides the means to allocate, assign, delete or rename the LCB (see Chapter 3).

In data communications, all terminal access, such as input/output (I/O) access, is accomplished via a logical unit (lu). Each terminal must be assigned to the proper lu prior to access. For nonbuffered access, the terminal is assigned to the lu using the name given to the terminal at system generation (sysgen) time. For buffered access, the logical terminal, named by a previously allocated LCB, is assigned to the desired lu. After the program has accessed the terminal, the lu should be closed. Chapter 3 discusses the OS/32 support provided via SVC7 or the command language for assigning, checkpointing and closing logical units for terminal access.

4.3 SEQUENCE OF OPERATION

For a terminal manager that supports buffered access, the user must first allocate an LCB. The terminal is then capable of accessing data in either a buffered or nonbuffered mode.

When a program is to be used with one type of buffered terminal, all required actions can be performed by SVC7 calls.

To use the device-independent facilities provided by terminal managers, the following four steps must be performed.

1. Sysgen

   Include terminal manager support by specifying the needed terminal (DCOD) at sysgen. Terminals are configured just like any local device, such as a line printer.

2. Programming

   Perform I/O requests by using SVC1 as if the requests were to a local device or direct access file. Since all I/O is performed to an lu rather than a specific device, device dependency is of no importance.

3. Execution

   A nonbuffered device is assigned to the lu just as if it were a local device. For buffered devices, an LCB must be allocated prior to assigning the lu. The terminal manager uses this LCB to control access to the terminal. The LCB contains control fields, the device name and the required buffers.

   After allocating an LCB, assign the LCB to the lu used for program access. This establishes the link between the lu and the device named by the LCB.

4. Termination

   Close the lu assigned to the device and delete the LCB from memory. The device is then available for another program.

The steps outlined for execution and termination can be performed by the program via SVC7 requests or by the system operator through the OS/32 command language.

In general, the sequence of operations necessary for terminal level access is:

1. Allocate an LCB that reserves buffer space and names a logical terminal (buffered access only).
2. Assign the terminal or logical terminal to the desired program lu.
3. Access the terminal via SVC1 calls, similar to accessing a local device.
4. Close the assigned lu when the terminal is no longer needed.
5. Delete the LCB from memory (buffered access only).

4.4 SUPERVISOR CALL 1 (SVC1)

The data communications SVC1 facility is ideal for the noncommunications-oriented user. It allows the user to easily extend the range of a system beyond the computer room. Remote terminals can be added with no impact on user programs.

All data transfers are performed at the read/write level. The OS/32 software performs the processing needed to interface with the terminal or protocol that frees the user of the problems generally associated with communications programming.

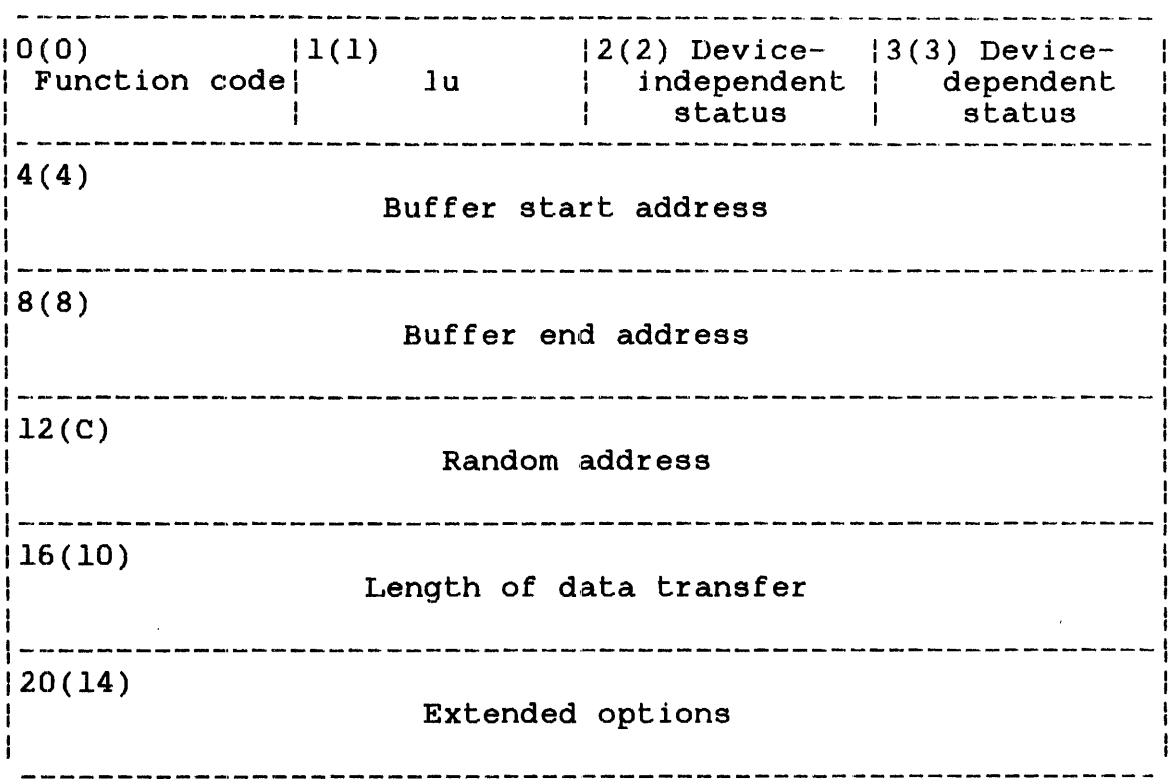
By issuing a read/write SVC1 to the OS/32 executive, the user can perform local or remote I/O. SVC1 can be used directly by a Common Assembly Language (CAL/32) program or indirectly by a high-level language run-time library (RTL) routine. OS/32 provides extended options with SVC1 to give the assembly-level user direct control over communication functions that have no parallel in local devices, such as disconnecting from a telephone line.

Section 4.4.1 describes the SVC1 functions for terminal access. The individual terminal manager description should be consulted for exact interpretation of function code and status. To write user programs that operate with either remote or local devices, see the SVC1 description in the OS/32 Supervisor Call (SVC) Reference Manual.

4.4.1 Supervisor Call 1 (SVC1) Parameter Block

SVC1 is used to initiate I/O for communications devices as well as local devices. The extended options field of the SVC1 parameter block shown in Figure 4-1 is used for terminal manager access. The function code byte, interpreted for data transfer requests, is defined in Table 4-1.

The SVC1 parameter block must be 24 bytes long, fullword boundary-aligned and located in a task-writable segment. Location within a writable segment is necessary so that the status of an I/O request can be returned to the status fields of the parameter block. All fields in the parameter block are not required for every I/O request, but must be reserved (see Figure 4-1).




```
          SVC   1,parblk
          .
          .
          .
          ALIGN 4
parblk    DB    X'function code'
          DB    X'lu'
          DS    2 bytes for status
          DC    A(buffer start)
          DC    A(buffer end)
          DC    4 bytes for random address
          DS    4 bytes for length of data transfer
          DC    Y'extended options'
```

Figure 4-1 SVC1 Parameter Block Format and Coding

Fields:

Function code — is a 1-byte field indicating whether a request is a data transfer or a command function, and the specific operation to be performed. Bit settings for data transfer requests are described in Table 4-1. Hexadecimal function codes for command function requests are defined in Table 4-2.

lu — is a 1-byte field containing the logical unit currently assigned to the device to which an I/O request is directed.

Device-independent status — is a 1-byte field receiving the execution status of an I/O request after completion. The status received is not directly related to the type of device used.

Device-dependent status — is a 1-byte field receiving the execution status of an I/O request after completion. The status received contains information unique to the type of device used.

NOTE

The device-dependent status byte, into which general purpose drivers return the low-order eight bits of the device address, is used differently for terminal managers. When an error occurs on a data communications device, the byte is used to differentiate between the possible specific errors within the general category given by the device-independent status.

Buffer start address — is a 4-byte field used only for data transfer requests and must contain the starting address of the I/O buffer that receives or sends the data being transferred.

Buffer end address — is a 4-byte field used only for data transfer requests and must contain the ending address of the I/O buffer that receives or sends the data being transferred.

Random address — is a 4-byte field containing the address of the logical record to be accessed for a data transfer request; a legal hexadecimal number must be specified in this field if bit 5 of the function code is set to 1.

Length of data transfer — is a 4-byte field used only for data transfer requests. It receives the number of bytes actually transferred as a result of a data transfer request. If an error occurs during data transfer, this field is modified with indeterminate data.

Extended options — is a 4-byte field specifying device-dependent and device-independent extended functions that must be executed by the device when it is servicing a data transfer request.

If bit 7 of the function code is zero (Format), the interpretation is identical to the interpretation for other devices described in the OS/32 System Level Programmer Reference Manual. The terminal manager will make assumptions using defaults where necessary. If bit 7 is set to 1 (Image), the extended options field is required.

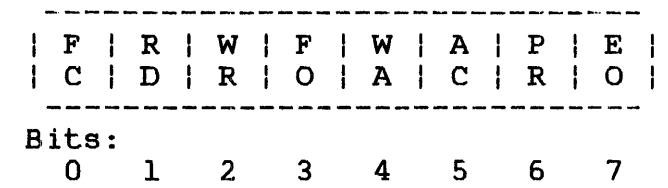


Figure 4-2 SVC1 Function Code Field

TABLE 4-1 SVC1 DATA TRANSFER FUNCTION CODE

BIT	MEANING
0	Function code type This bit indicates the I/O function to be performed. 0 Indicates a standard data transfer. 1 Indicates an SVC1 command function. If bit 7 is also set, no echoplex is desired for the next image I/O.
1,2	Read/write bits The meaning of these two request bits is modified by bits 3 to 7 to control the transfer. Basically the four values are: 10 Read 01 Write 11 Test and set 00 Wait only or test I/O complete

TABLE 4-1 SVC1 DATA TRANSFER FUNCTION CODE (Continued)

BIT	MEANING
3	ASCII/binary bit This bit indicates the type of formatting requested if bit 7 is set. 0 Indicates ASCII formatting. The internal data is in the 7-bit ASCII character set and is translated to an equivalent character set appropriate for the external device. If image I/O extended option is specified, the data is translated and the appropriate parity is added. 1 Indicates binary formatting. The internal data is 8-bit binary and will not be translated. Binary image is a straight 8-bit data transfer, no parity. If bit 3 is set and an image I/O extended option is specified, the internal data byte (eight bits) is transferred without translation.
4	Proceed/wait bit This bit indicates the action to be taken after an I/O is initiated. 0 Proceed - indicates that control is to be returned to the task after I/O initiation. 1 Wait - indicates that a task is to be put into I/O wait until the data transfer is complete.
5	Sequential/random bit 0 Sequential - indicates the next logical record is to be accessed. 1 Random - indicates the logical record specified by the hexadecimal value in the random access field is to be accessed.
6	Unconditional proceed bit 0 Indicates the task is to be put into wait until the requested device/file is free. At that time, the request is processed. 1 Indicates that the request is to be rejected with a condition code of X'F' if the requested device/file is not free.

TABLE 4-1 SVC1 DATA TRANSFER FUNCTION CODE (Continued)

BIT	MEANING
7	Standard/extended options bit 0 Indicates standard device-independent data formatting is to be performed. 1 Indicates that the extended options field is to be examined for additional formatting and line control options.

The extended options field of the SVC1 parameter block consisting of 32 bits, is defined in Table 4-2. Extended options provide both device-dependent and communications-dependent features with SVC1 access. This allows a limited amount of formatting to be done by the program without having to use SVC15 line driver access. The capability to connect to and disconnect from a switched communications line (i.e., the ability to answer a data telephone call and to hang up) is also provided. The extended options can be used only in conjunction with a read or write operation, as explained in Table 4-2.

If the user does not wish to use the extended options, connect/disconnect operations are controlled by the terminal manager. If an SVC1 request is made to an unconnected switched line, a connect is automatically performed. When the last assignment to a switched terminal is closed, a disconnect is performed.

For editing terminals (device codes 156 and 157), a user may specify no echoplex for an image I/O. Another option available for these device codes is 8-bit no parity data transfer. No echoplex is specified by preceding the image I/O with an SVC1 command with the function code set to X'81' and the extended options field set to Y'1000 0000'. To indicate 8-bit no parity data transfer, the user must set the binary bit in the function code (X'10').

An in-depth discussion of SVC1 data transfer functions and extended options can be found in the OS/32 Supervisor Call (SVC) Reference Manual.

TABLE 4-2 SVC1 EXTENDED OPTIONS

BIT	HEX MASK	MEANING
0	8000 0000	Connect When set, this bit instructs the terminal manager to establish a connection over a switched line before transferring data. It is ignored for nonswitched configurations. If the line has been sysgened with the XDCD bit set, then all I/Os are errors until an SVC1 with the 0 bit set is executed. See the System Generation/32 (Sysgen/32) Reference Manual for further details.
1	4000 0000	Disconnect When set, this bit instructs the terminal manager to disable the data terminal ready control signal to the adapter after the data transfer operations specified in the function code are completed. This allows program-controlled disconnect of switched lines and control of carrier on leased configurations.
2	2000 0000	Format/image bit 0 Indicates that no data formatting is to be performed by the terminal manager (image mode). 1 Indicates that the terminal manager performs all required data manipulation to convert between the terminal's required data format and one that can be used for transfers to other Perkin-Elmer peripherals (i.e., trim trailing blanks and stop I/O on the detection of the carriage return (CR)).
3	1000 0000	No echoplex When set, this bit is used in conjunction with function code X'81' to specify no echoplex.
4-6	0E00 0000	Not ITAM-related.

TABLE 4-2 SVC1 EXTENDED OPTIONS (Continued)

BIT	HEX MASK	MEANING
7		0 = reserved
8	0080 0000	Vertical forms control (VFC) When set, this bit requests the V/C option for an ASCII I/O operation.
9-15	007F 0000	Reserved for future device-independent options. All bits must be zero.
16-31	0000 FFFF	Reserved for device-dependent options. See individual terminal manager descriptions for definitions.

CHAPTER 5
DEVICE-DEPENDENT ACCESS

5.1 INTRODUCTION

Device-dependent support of communications devices is provided by OS/32 line driver access. For this support, the operating system is configured with communications lines that correspond to the hardware communications adapters present in the system. Through line driver access, different terminals can be supported at different times over the same line since the actual communications protocols and data formats are specified by the user program via supervisor call 15 (SVC15). SVC15 allows a program to specify a sequence of control commands and the data required by the control sequence. Line driver access supports a communications network with maximum efficiency and throughput by providing the capability to specify buffering techniques, monitor the progress of a control sequence, and alter control sequences and data while in progress.

SVC15 is similiar to SVC1, but because of the specialized requirements of data communications, SVC15 has more variables and options. The SVC1 function code allows a task to request a single function, read, write, rewind, etc., on a single data area for each SVC1 call. SVC15 allows a task to request a series of commands to be executed on a series of data areas.

A key feature of SVC15 access is the flexibility allowed in data formatting. Driver commands may result in a variable number of data fields being accessed from the SVC15 parameter block; the exact number of data fields is dependent upon the command type. For example, the READ or WRITE buffer commands obtain either one or two data fields. Other command types use either one or no data fields.

5.2 LINE DRIVER ACCESS

The SVC15 call and line drivers provide access to remote devices for the communications-oriented user to:

- access devices with protocols or codes not supported by a terminal manager,
- use special buffering techniques because of time, memory or line use considerations, or
- use data or command chaining to achieve the throughput necessary for the application.

5.2.1 Sequence of Operations

To use the device-dependent facilities provided by line drivers, perform the following steps:

1. System generation (sysgen)

   Include line driver support by specifying the desired lines at sysgen. Lines are configured just as any local device, such as a line printer.

2. Programming

   Perform communications line access by providing the control sequences and data necessary to support the remote device attached to the communications line. The control sequences and data are passed to the line driver through SVC15. Access to the line driver is by logical unit (lu), so different communications facilities of the same type can be accessed without recompiling the program. Further control is possible through the OS/32 Task Trap Facility, invoked when necessary by SVC15.

3. Execution

   Assign the desired communications line to the lu accessed by the program.

4. Termination

   Close the lu assigned to the communications line. The program can assign and close the lu via SVC7 or the OS/32 command language.

The steps outlined for execution and termination can be performed by the program via SVC7 requests or by the system operator through the OS/32 command language.

Access to a remote terminal connected to a communications line, like access to a local device or terminal, is through an lu. The general procedure for SVC15 line access is:

1. Assign the communications line to the desired program lu.

2. Issue SVC15 to specify the initial control sequence and data.

3. Use a trap-handling routine to monitor the progress of the control sequence.

4. Modify the control sequence and/or data, or issue a subsequent SVC15 specifying new control sequences and/or data to continue communications.

5. Close the lu assigned to the communications line.

5.2.2 Supervisor Call 15 (SVC15) and the Task Environment

Execution of SVC15 affects the environment of the calling task differently than the execution of SVC1. SVC15 returns control to the calling task following driver activation (no input/output (I/O) and wait). It is the task's responsibility to synchronize processing with the ongoing I/O request.

Driver control is specified by a driver command word (DCW). A DCW is actually a halfword that specifies to the driver a particular operation to be performed and certain options applicable to that operation. A DCW chain consists of consecutive DCWs with their respective chain option bits set. For further information on the DCW, see Section 5.5. The format of the DCW is shown in Figure 5-12.

The SVC15 parameter block specifies the first entry in each of two related chains used to define the following requests:

- The DCW chain that specifies the sequence of driver operations (i.e., READ, WRITE, etc.).
- The data field chain that specifies the arguments required by each driver command in the DCW chain.

Figure 5-1 shows SVC15 access to a line driver.

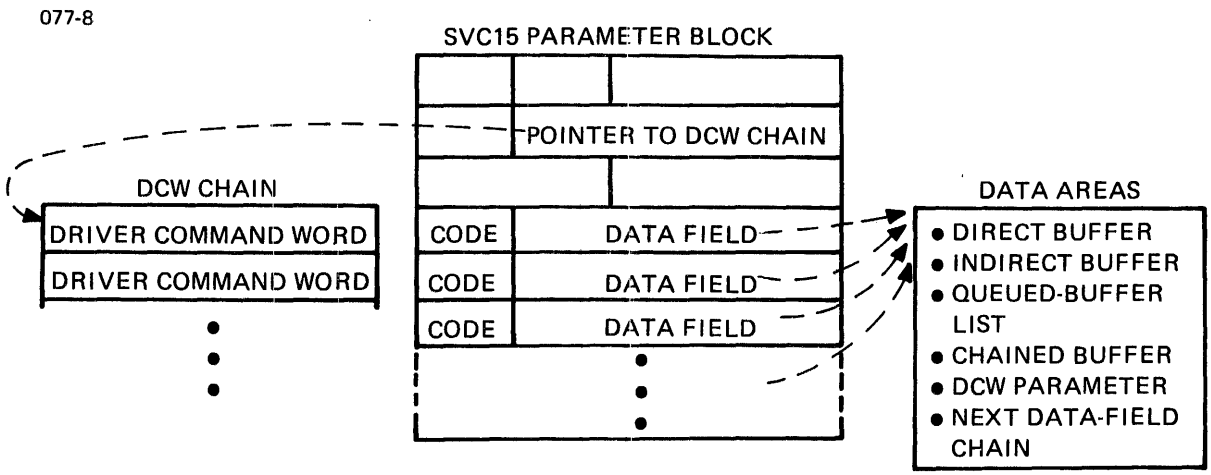


Figure 5-1 SVC15 Access to a Line Driver

SVC15 activates the line driver to fetch and execute the first DCW in the DCW chain. Once autonomous driver execution starts, control is returned to the user task (u-task) with the condition code indicating the result of the call. If there is no error in initiating the operation by the first DCW specified, the status field of the SVC15 parameter block is set to indicate that the line driver is active with the request X'4000'. For the remainder of the I/O request, as each command operation is successfully completed, the next operation is fetched from the DCW chain and executed by the line driver. This sequence of fetch and execute is repeated until the entire DCW chain is interpreted or an error condition is encountered.

5.2.2.1 Supervisor Call 15 (SVC15) Trap Handling

The progress of SVC15 execution and the facilities provided for buffer management can be monitored by the task through the use of traps. The traps that can be generated for data communications are:

- SVC15 command execution trap
- SVC15 buffer transfer trap
- SVC15 termination trap
- SVC15 halt I/O termination trap

Other types of traps can be generated for specific terminal managers such as the channel terminal manager (CTM). These traps are detailed in the appropriate manual.

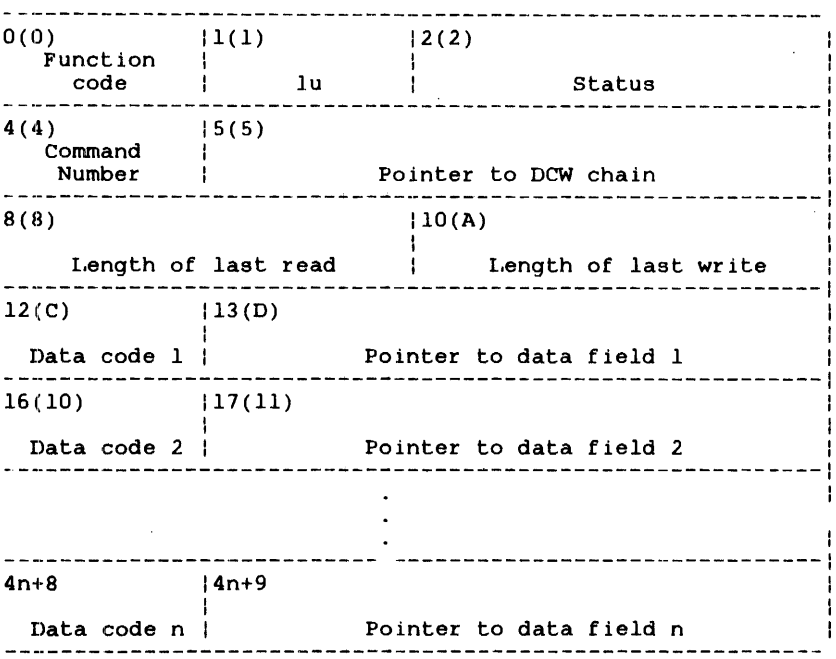
The traps listed above allow the task to synchronize execution with the concurrent processing of the SVC15 request. To enabled traps, bit 23 (TSW.ITM) of the task status word (TSW) must be set. When traps are so enabled and a trap-causing event occurs, the task trap-handling routine is given control before any subsequent task level instruction can be executed. Remember that the trap-handling routines operate at a lower priority than the line driver. Several entries can be made to the task queue before the trap-handling routine completes processing a single entry. See the OS/32 Applications Level Programmer Reference Manual for more information on trap handling.

When enabled by the appropriate bit settings in the TSW, the SVC15 function code and the DCW, a trap can be generated by adding one of the following reason codes and the address of the SVC15 parameter block to the u-task queue.

REASON CODE DECIMAL(HEX)	QUEUE ENTRY
10(0A)	Command trap
11(0B)	Buffer trap
12(0C)	Termination trap
13(0D)	Halt I/O trap

5.3 SUPERVISOR CALL 15 (SVC15) PARAMETER BLOCK

SVC15 specifies a control sequence and its associated data to the data communications line driver. The format of the SVC15 parameter block is illustrated in Figure 5-2.




```
          SVC   15, parblk
          .
          .
          .
          ALIGN 4
parblk    DB    x 'function code'
          DB    x 'lu'
          DS    2 bytes for status
          DS    1 byte for command number
          DC    3 bytes for DCW chain address
          DS    2 bytes for length of last read
          DS    2 bytes for length of last write
          DB    1 byte for data code 1
          DS    3 bytes for data field 1 address
          .
          .
          .
          DB    1 byte for data code n
          DS    3 bytes for data field n address
```

Figure 5-2 SVC15 Function Code Format

5.3.1 Function Code Field

The first byte in the parameter block is a function code provided by the u-task. This field specifies options that apply to all driver commands in the DCW chain executed by this SVC15. Figure 5-3 shows the composition of the function code byte and Table 5-1 shows the bit settings of the function code field.

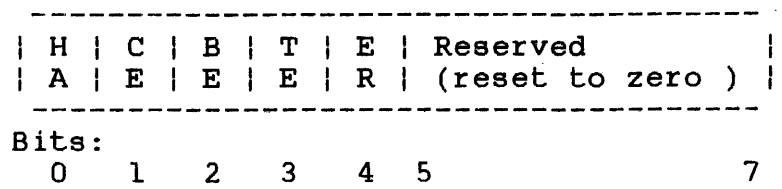


Figure 5-3 SVC15 Function Code Format

TABLE 5-1 SVC15 FUNCTION CODE BIT SETTINGS

BIT	MASK	MEANING
0	X'80'	HALT I/O (HA) An SVC15 request with this bit set indicates that a task is requesting to halt an I/O that it previously started to the indicated lu. The program status word (PSW) condition code after the request indicates the results of the halt I/O call. Traps will not be generated for the halt call itself. CC = 0 The halt I/O was accepted, the original request ends with a halt I/O status and the command number field is updated accordingly. If the original request specified termination traps, a trap is generated when the I/O terminates. If the original request was in error recovery, then the original error status may be returned at termination rather than a halt I/O status. CC = 1 The halt I/O was not accepted because the driver was not performing SVC15 I/O to the specified task lu at the time of the call. The status field is not changed.

TABLE 5-1. SVC15 FUNCTION CODE BIT SETTINGS (Continued)

BIT	MASK	MEANING
1	X'40'	Command queue entry enable (CE) This bit must be set, along with the corresponding bit in the DCW and the enable SVC15 queue entry bit in the TSW, to allow a trap at the start of each DCW execution.
2	X'20'	Buffer queue entry enable (BE) This bit must be set, along with the corresponding bit in the DCW and the enable SVC15 queue entry bit in the TSW, to allow a trap at the start of buffer use associated with the DCW.
3	X'10'	Termination queue entry enable (QE) This bit must be set, along with the enable SVC15 queue entry bit of the TSW, to allow a trap on completion, normal or abnormal, of an SVC15 call. A halt I/O call does not generate a trap. However, the call being halted does if termination queue entry enable was specified.
4	X'08'	Continuous error processing (ER) Setting this bit permits some errors to be nonterminating during continuous read operations.
5-7	X'07'	Reserved These bits are reserved for future use and must be zero.

5.3.2 Logical Unit (lu) Field

This byte identifies the lu to which the communications line is assigned for the transfer. A valid data communications device must have been previously assigned to the specified lu for SVC15 access.

5.3.3 Status Information Field

The completion status of the SVC15 operation is returned to the system via the status information field. The format of this halfword is shown in Figure 5-4; the meanings of each bit setting are listed in Table 5-2. The termination codes resulting from a terminated SVC15 command are listed in Table 5-3. While processing an SVC15 request, the completion status of each DCW is maintained internally in the same format.

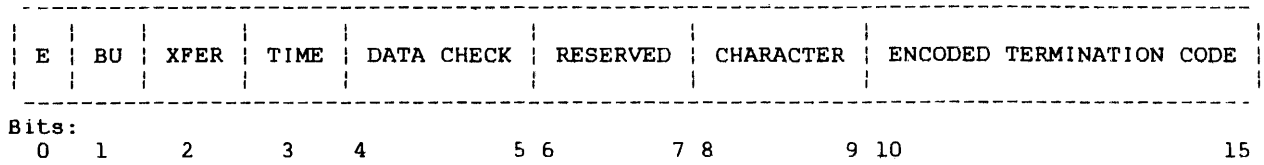


Figure 5-4 SVC15 Status Field Format

TABLE 5-2 SVC15 STATUS BIT SETTINGS

BIT	MASK	MEANING
0	X'8000'	Error Set for error condition. Bits 3, 4, 5 and any termination code greater than 2, in bits 10 to 15, are considered errors.
1	X'4000'	Busy Set when the driver is busy with an SVC request and can be cancelled via halt I/O. The SVC15 parameter block status halfword is initialized to this value and cannot terminate with this bit set.
2	X'2000'	Transfer not begun This bit is set if read operations do not receive the proper required starting control characters (e.g., for binary synchronous).
3	X'2000'	Time-out This bit is set if an entire command had not executed when the error timer expired.

TABLE 5-2 SVC15 STATUS BIT SETTINGS (Continued)

BIT	MASK	MEANING
4-5	X'0C00' X'0800' X'0400'	Data check Received data with parity, cyclic redundancy check (CRC) or longitudinal redundancy check (LRC) error.
6-7	X'0300' X'0200' X'0100'	Driver-dependent
8-9	X'00C0' X'0080' X'0040'	Special character detection Drivers can flag reception of special characters by setting these bits.
10-15	X'003F'	Encoded Termination Code Cause of driver termination; see Table 5-3.

TABLE 5-3 SVC15 ENCODED TERMINATION CODES

VALUE DEC (HEX)	STATUS	MEANING
0(00)	No errors	No errors detected
1(01)	Reserved	
2(02)	Line delete	Line delete detected during read
3(03)	Break on write	Break detected during an asynchronous write
4(04)	Break on read	Break detected during an asynchronous read
5(05)	Data check	BCC, LRC or parity error caused termination (bits 4 and 5 specify exactly)
6(06)	Buffer limit	Buffer limits reached (no proper ending sequence, binary synchronous)

TABLE 5-3 SVC15 ENCODED TERMINATION CODES (Continued)

VALUE DEC (HEX)	STATUS	MEANING
7(07)	Bad pad	PAD character (not received binary synchronous)
8(08)	Framing error	Framing or stop bit error asynchronous
9(09)	Reverse channel	Reverse channel error asynchronous
10(0A)	Loss of carrier	Lost carrier on read
11(0B)	CL2S error	Lost clear to send on write
12(0C)	Data set not ready	Data set not ready
13(0D)	Device unavailable	Adapter not present
14(0E)	Overflow	Character overflow
15(0F)	Ring	Ring signal detected during data transfer
16(10)	Buffer overrun 1	Busy and/or done bits in chained buffers not properly set
17(11)	NCE overflow	More than 255 commands executed
18(12)	Task queue error	Task queue full, invalid or nonexistent during attempt to trap (amount of data transferred questionable); error code overwrites previous error codes reported via traps
19(13)	Buffer overrun 2	Next I/O buffer not reset in time
20(14)	Time-out	Time-out
21(15)	Halt I/O	Halt I/O request; aborted I/O
22(16)	Transparent block size error	Data buffer is smaller than the transparent block size (binary synchronous only)

TABLE 5-3 SVC15 ENCODED TERMINATION CODES (Continued)

VALUE DEC (HEX)	STATUS	MEANING
23(17)	Bad character sequence	Improper binary synchronous sequence
24(18)	Illegal command	Command or modifier not valid
25(19)	Memory fault 1	Memory fault referencing data
26(1A)	Memory fault 2	Memory fault referencing buffer
27(1B)	Illegal lu	lu illegal (not SVC15, not assigned)
28(1C)	Illogical status	Device status not valid (possible strapping problem)
29(1D)	Power fail	I/O interrupted by power failure or cancel
30(1E)	Illegal software condition	Illegal condition detected
31(1F)	Illegal translate	Illegal translation table specified
32(20)	Idle line	Idle line sequence received zero-bit insertion/deletion (ZBID)
33(21)	Frame abort	Frame abort character received ZBID
34(22)	Invalid frame	Received frame of incorrect length ZBID
35(23)	Queue empty	Queued buffer list empty
36(24)	Queue overflow	Queued buffer list overflow

5.3.4 Command Number Field

The fourth byte of the parameter block indicates the number of commands executed. The number of DCWs fetched and executed is maintained by the driver and returned to the u-task via this field. In the event of an error (or halt I/O), this counter indicates the number of the commands being executed when the error occurred; i.e., it equals 1 if only one DCW of the chain was executed.

NOTE

When this byte is incremented past 255, the SVC15 request is aborted and the status indicates that the number of commands executed overflowed, X'8011'.

5.3.5 Driver Command Word (DCW) Pointer

This field must be set to the address of the first command in the DCW chain to be executed by the SVC15 call.

5.3.6 Length of Last Read Field

This halfword is updated by the driver termination to indicate the number of bytes transferred during the last READ command executed by SVC15. If no data was transferred or if no reads were executed, zero is returned.

5.3.7 Length of Last Write Field

This halfword is updated by the driver at termination to indicate the number of bytes transferred during the last WRITE command executed by SVC15. If no data was transferred, or if no reads were executed, zero is returned.

For the length of the last read and the length of the last write fields, if a DCW string contains more than one READ or WRITE command, only the length of the most recent operation is reflected in this field. However, when chained buffers are used, the total number of bytes transferred to or from the entire chained buffer by the last READ or WRITE command is reflected.

5.3.8 Data Fields

The remainder of the parameter block consists of data fields required by the DCW chain. As illustrated in Figure 5-5, each data field consists of a fullword divided into two sections. The last three bytes of the field contain a pointer to the data needed by the DCW (bits 8 through 31). This address could point to no data or to one or more data fields, depending on the particular command. The first byte (bits 0 through 7) consists of a data code indicating the type of data pointed to by the last three bytes. The data code also indicates the type of buffering desired. Definitions of the data code bit settings are detailed in Section 5.3.9.

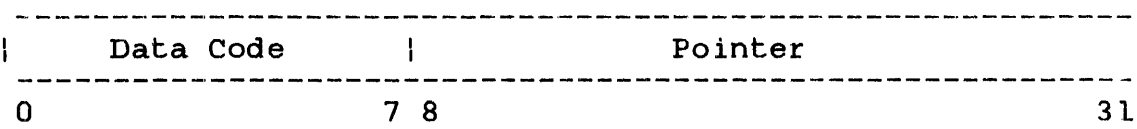


Figure 5-5 SVC15 Data Field Format

When the DCW associated with the data field is a READ or WRITE command, the data field points to a buffer. For other DCWs, the data field points to a parameter required by that DCW, such as a time value. The data field can also point to the head of another data field chain, allowing data field chains to exist in noncontiguous memory.

5.3.9 Data Field Chain

The SVC15 parameter block specifies the control sequence to be performed by pointing to a DCW chain. The first data field in the data field chain is at the fullword at offset 12(C) from the start of the parameter block.

Table 5-4 details the bit settings of the data code byte of the data field.

TABLE 5-4 DATA CODE BIT SETTINGS

DATA CODE (HEX)	CONTENTS
00	Pointer to direct buffer
01	Pointer to DCW parameter
04	Pointer to indirect buffer
08	Pointer to chained buffer
0A	Pointer to queued buffer list
80	Pointer to next data field chain

Data codes 00, 04 08 and 0A indicate that the data field contains the address of a data buffer that can be direct, indirect, chained or queued.

Data code 01 indicates that the data field contains the address of a parameter required by a driver command, such as a time value.

Data code 80 indicates that subsequent data fields are to be fetched from the specified address, allowing the data field chain to exist in noncontiguous memory.

5.4 BUFFER TYPES

SVC15 supports four buffer types:

- Direct
- Indirect
- Chained
- Queued

These buffer types are defined by the data fields in the SVC15 parameter block.

5.4.1 Direct Buffers

A direct buffer is defined by two data fields containing the starting and ending addresses of the buffer, similar to an SVC1 data buffer. The starting address points to the first data character and the end address points to the last data character (i.e., a 1-character buffer has a starting address equal to the ending address). Direct buffers can begin on any byte boundary and require two data fields in the data field chain. Figure 5-6 depicts a direct buffer.

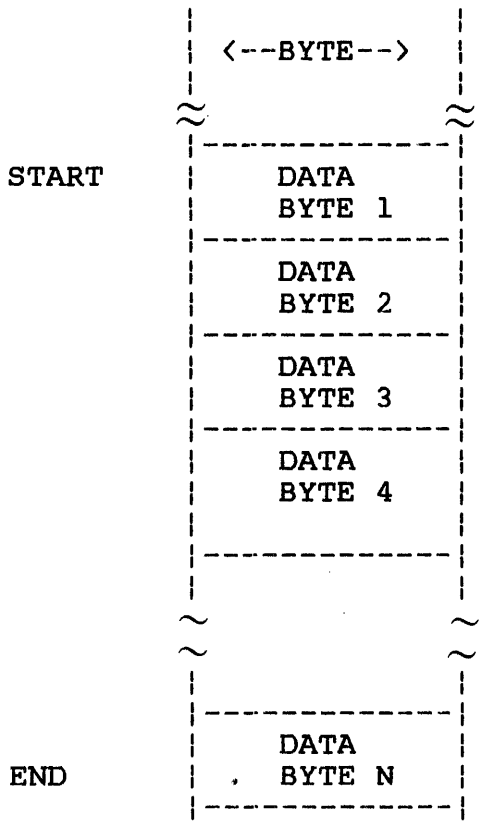


Figure 5-6 Direct Buffer

5.4.2 Indirect Buffers

An indirect buffer is specified by one data field containing its starting address. The buffer itself contains all required size information. The first halfword indicates the number of bytes available in the buffer. The second halfword of the buffer is updated by the driver and indicates how many bytes of data were actually transferred by the I/O. An indirect buffer must be aligned on a halfword boundary. Figure 5-7 depicts an indirect buffer.

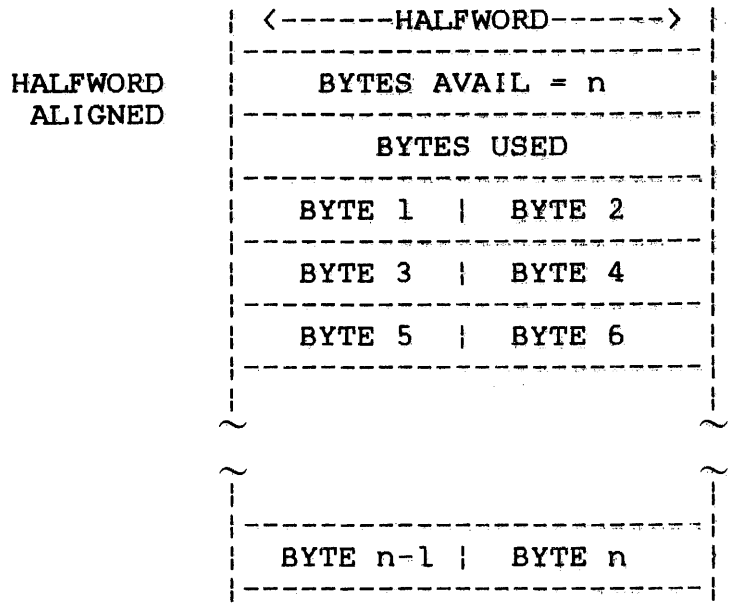


Figure 5-7 Indirect Buffer

5.4.3 Chained Buffers

Chained buffers are specified by one data field containing the address of the first buffer in the chain. Chained buffers look very much like indirect text buffers, but have an additional fullword at the beginning called the link word that might contain the address of another chained buffer. Thus, two or more buffers can be linked together into a chain. The last buffer in a chain of linked buffers contains a zero link word indicating the end of the chain. Figure 5-8 illustrates chained buffers.

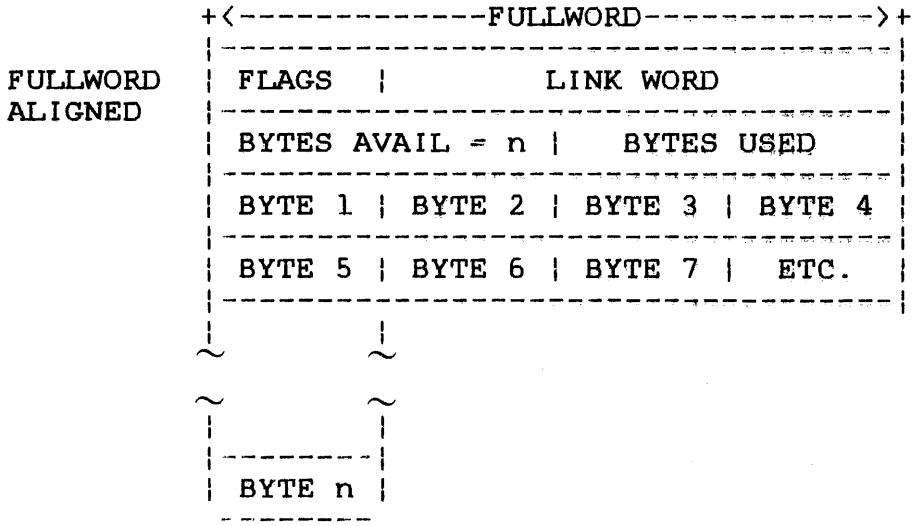


Figure 5-8 Chained/Queued Buffer Format

Chained buffers can also be configured into a closed chain (a ring) by having the last buffer link back to the first buffer.

The first byte of the link word is used for certain flags to indicate conditions or options within the buffer. Figure 5-9 shows the format of the chained/queued buffer link word flag byte and Table 5-5 details the bit settings. Chained buffers must be aligned on a fullword boundary.

A task can manipulate the links and data of chained buffers while I/O is in progress. Bits 0 through 7 of the link word (the flag byte) are used for coordination between driver and u-tasks.

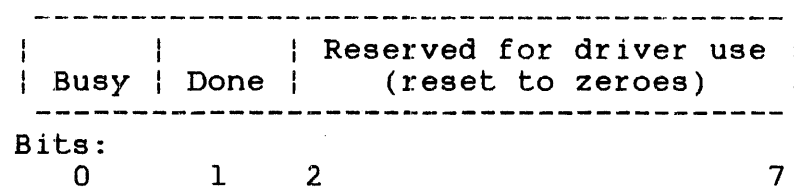


Figure 5-9 Chained/Queued Buffer Link Word Flag Byte

TABLE 5-5 CHAINED/QUEUED BUFFER LINK WORD FLAG BYTE

(BUSY) BIT 0	(DONE) BIT 1	MEANING
0	0	Buffer is available for driver use. The link word contains a valid address or zero.
1	0	The driver is currently using this buffer for I/O. The u-task must not change data, size values, link word or flags.
1	1	The driver finished using the buffer. The driver will not use this buffer again if it reoccurs in the chain; i.e., a ring. U-task can now change any value and the bytes used reflect actual transfer.
0	1	Invalid setting. Driver treats it as if busy and done = 11.

The driver attempts to set up two I/O buffers when chained buffers are specified. This means that if the task needs more than a single chained buffer for I/O, it must supply at least two linked buffers when the SVC15 call is issued. These two buffers are set up in internal buffers associated with this line. The first buffer is flagged as busy (bit 0 of link word set), a buffer trap is generated if enabled and I/O is started.

If the link word of buffer 2 is valid, the driver fills buffer 1. When buffer 1 is exhausted, the driver receives a buffer limit interrupt, finds buffer 2 is available (busy and done bits reset) and uses this buffer to continue I/O. Meanwhile, a routine is scheduled to flag buffer 1 as done and buffer 2 as busy and to attempt to set up the first I/O buffer using the current link word of buffer 2 as a pointer. If this link word is zero, the current buffer is the last buffer of the chain and the I/O must terminate within it. If this link word is nonzero, it points to buffer 3. At this time, buffer 3 does not necessarily have to be available (busy and done bits reset) indicating that buffer 1 is done and buffer 2 is now busy as long as its address is specified in the link word of buffer 2. A buffer trap is again generated to the calling task, indicating that buffer 1 is done and buffer 2 is now busy. Buffer 3 must be available before the next buffer limit interrupt. If it is not, a buffer overrun occurs, I/O is aborted and the status reflects this overrun.

For read-after-write (RAW), the driver looks ahead and sets up only one read buffer. Thus, if chained buffers are used for the read, only one buffer is set up in advance. When the driver terminates the write, one buffer is ready to perform the read. When the read uses chained buffers, a subroutine to get the next buffer is scheduled immediately after performing the write-to-read turnaround. The buffer trap for the read is performed after read I/O begins. Thus, having only one read buffer instead of two is useful when using chained buffers for both write and read. However, the buffer trap for the next to last write buffer must be identified by the task because the task must specify the second buffer in the link word of the first read buffer before the driver completes the write.

Memory is used most efficiently by linking two or more chained buffers to form a ring. See Figure 5-10.

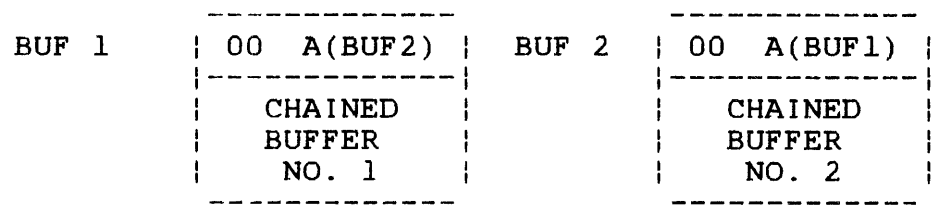


Figure 5-10 Buffer Ring

When the busy and done bits = 11, buffer 1 is finished and the driver is using buffer 2, which would be busy. The task can now process the data in buffer 1 and reset busy and done. When the driver is finished with buffer 2 (busy and done set), it chains onto buffer 1 and, finding the busy and done bits reset, uses it to continue the I/O. Thus, one SVC15 request can continuously perform I/O as long as the task keeps ahead of the driver.

5.4.4 Queued Buffers

Queued buffers are specified by two data fields, each of which contains the address of a standard Perkin-Elmer circular list. List 1 specifies a queue of buffers from which the data communications subsystem removes buffers for I/O operations. List 2 specifies a queue of buffers being returned to the applications program following I/O activity. List 1 can coincide with List 2. See Figure 5-11 for a description of the standard Perkin-Elmer circular list. Buffers are removed from the top of List 1 by a Remove from Top of List (RTL) instruction. Buffers are returned to the bottom of List 2 by an Add to Bottom of List (ABL) instruction.

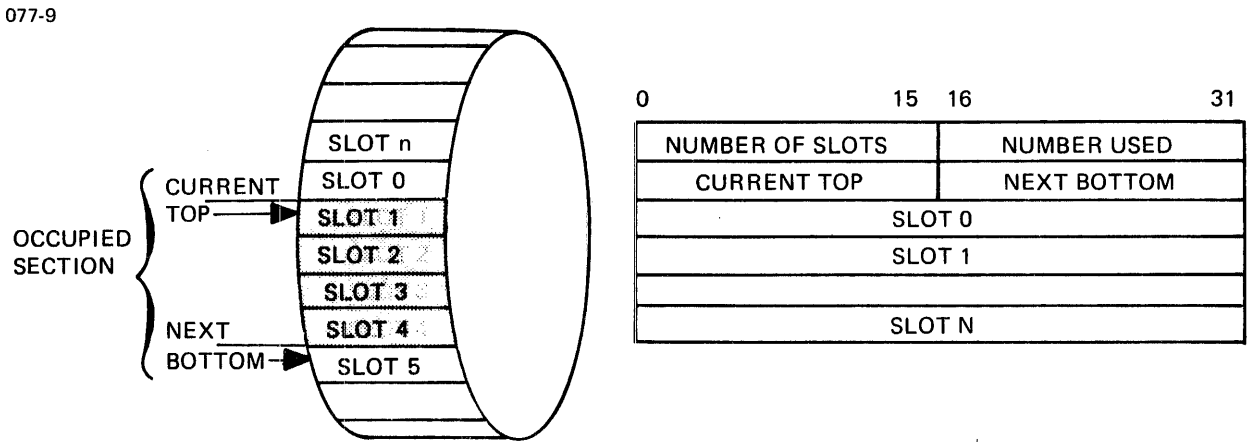


Figure 5-11 Conceptual Circular List and Format

The format of each individual queued buffer with an address in the list is identical to the format of a chained buffer. Restrictions for modifying the control fields of a buffer during I/O are the same for queued and chained buffers.

When an I/O buffer is removed from List 1, the link address field is cleared to prevent any ambiguity in error verification and the address of the buffer is maintained solely within driver control storage. The buffer is, in effect, not available to the applications program during I/O.

The busy and done bits within the flag byte are used analogous to chained buffers. When I/O is complete, the buffer is returned to the bottom of List 2. Simultaneously with I/O operation, the applications task can add new I/O buffers to the bottom of List 1 or remove completed buffers from the top of List 2. Only list processing instructions (RTL, Remove from Bottom of List (RBL), Add to Top of List (ATL), ABL) can be used by the task to modify a circular list. Any other attempt to modify circular list control fields can result in a loss of control.

If the program attempts to return a buffer to List 2 and cannot because the list is full, a queue overflow (X'24') error termination results. The addresses of any buffers currently being used for I/O are then chained to the bottom buffer in List 2 to return them to the task. As the list address field is initialized to zero at the start of I/O, the task should check the nonzero link field to detect buffers returned because of a queue overflow condition.

The buffer trap mechanism is available for queued buffers. However, to conserve processor time, a buffer trap is generated only when a buffer is added to a previously empty List 2, indicated by the status returned by the last RTL or RBL. This technique requires a program to process all buffers in List 2 whenever a trap interrupt occurs.

In Figure 5-11, the first two fullwords of a circular list contain the list parameters. Immediately following the parameter block is the list itself. The first fullword in the list is designated Slot 0, with the remaining slots designated 1, 2, 3, etc., up to a maximum slot number equal to the number in the list minus one. A maximum of 65,535 fullword slots can be specified. (Maximum slot designation is equal to X'FFFE'.)

The first parameter halfword indicates the number of slots (fullwords) in the entire list. The second parameter halfword indicates the current number of slots being used. When the second parameter halfword equals zero, the list is empty. When it equals the number of slots in the list, the list is full. Once initialized, this halfword is maintained automatically and incremented when elements are added to the list and decremented when elements are removed.

The third and fourth halfwords of the list parameter block specify the current top of the list and next bottom of the list, respectively. These pointers are also updated automatically.

5.4.4.1 Coding a Queued Buffer Request

Three separate areas must be coded with a precise format to facilitate the use of queued buffers. These areas are the data field chain, the circular list descriptor and the individual queued buffer.

The data field chain must be coded as in Table 5-6. Two fullword entries must be defined in the data field chain in order to use queued buffers. The first fullword provides the address of the circular list from which the drive can draw buffers for I/O; the second fullword provides the address of the circular list to which completed buffers are returned. Each fullword entry must contain a hexadecimal A in the high-order byte to identify the entry as a queued buffer data byte and each must be aligned on a fullword boundary.

TABLE 5-6 QUEUED BUFFER DATA FIELD FORMAT

DATA CODE BYTE 0	POINTER BYTES 1 THROUGH 3
0A	Address of circular list from which buffers are obtained
0A	Address of circular list to which buffers are returned

The circular list can be defined by a DLIST Common Assembly Language (CAL) direction or by the FORTRAN 32-bit run-time library (RTL) subroutine, DEFLST.

Each individual queued buffer must be coded as a chained buffer shown in Figure 5-8. Its address is placed into the FROM buffer list.

5.5 DRIVER COMMAND WORD (DCW)

Each DCW is a 16-bit command that specifies a primitive operation to a line driver. Figure 5-12 details the DCW format and Table 5-7 describes the DCW bit settings. Bits 0 through 3 are flag bits that indicate options in effect for the command; bits 13 through 15 indicate the general type of command; bits 8 through 12 identify the specific request; bits 4 through 7 are unused and should be reset to zeroes. Not all commands are implemented for each line driver.

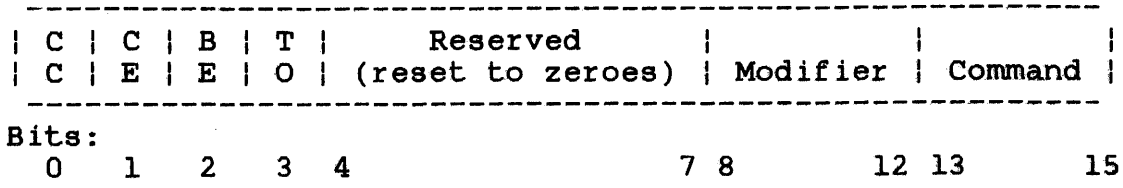


Figure 5-12 DCW Format

TABLE 5-7 DCW BIT SETTINGS

BIT	MEANING
0	CHAIN Command (CC) Set this bit to indicate that the line driver should fetch and execute the next DCW in the chain. If reset, the SVC15 request terminates after executing this DCW.
1	Command enable (CE) Set this bit to specify that a queue entry is to be generated by adding a parameter consisting of the address of the SVC15 parameter block to the task queue with reason code 10 (0A) before executing this DCW. For trap generation, the appropriate enabling bits must be set in the SVC15 function code and the TSW. If reset, no trap is generated.
2	Buffer enable (BE) Set this bit to specify that a queue entry is to be generated by adding a parameter consisting of the address of the SVC15 parameter block to the task queue with reason code (0B): o when processing the first character of any buffer used by this command o before switching to the next buffer if chained buffers are used To generate a trap, the appropriate enabling bits must be set in the SVC15 function code and TSW. If reset, no trap is generated.

TABLE 5-7 DCW BIT SETTINGS (Continued)

BIT	MEANING
3	Time-out (TO) Set this bit to specify that an error time interval is to be started when this command is fetched. If the command has not completed before this interval expires, the SVC15 request is aborted with time-out status. The interval is given a default value at sysgen time or can be modified by the MODE TOUT command. There are separate read and write time-out values. If reset, this command is not aborted because of time-out.
4-7	Reserved. Must be zero.
8-12	Command modifier These bits specify the particular command for each command type.
13-15	Driver command type These bits specify the general type or primitive requests as follows: VALUE TYPE 000 NULL 001 CONTROL 010 READ 011 PREPARE 100 WRITE 101 HOLD 110 MODE 111 TEST

5.6 LINE DRIVER COMMAND TYPES

This section describes the commands for each driver command type with the binary and hexadecimal value of bits 8 through 15 of the DCW for each. These descriptions refer to signals generated by the communications adapter hardware. For all commands, normal completion means that the next DCW is fetched. If no further DCWs exist, the SVC15 request terminates. Nonerror status conditions are noted by setting the appropriate bits in the status halfword; the status returned on termination can reflect several such cumulative conditions, i.e., special character detection.

5.6.1 Null-Type Commands

There are four null-type commands, none of which issue I/O instructions to the adapter. A CHAIN command and command trap flags are valid.

COMMAND	BINARY	HEX	NUMBER OF DATA FIELDS
NOP	00000 000	00	1
WAIT	00001 000	08	1
XFER	00010 000	10	1
CXFER	00011 000	18	2

5.6.1.1 NO OPERATION (NOP) Command

The NOP command performs no operation; one data field is fetched but not used. However, it must specify a valid program address. If the CHAIN command bit (bit 0) is set, the next command is fetched.

5.6.1.2 WAIT Command

The WAIT command suspends driver execution for a specified interval. One data field is fetched which must point to a halfword containing the value of a time interval in multiples of 100ms. The driver waits until the specified interval expires and then continues execution (if the CHAIN command is set) or terminates (if the CHAIN command is reset).

5.6.1.3 TRANSFER IN (XFER) Command

The XFER command specifies the next DCW in a chain that does not exist in contiguous memory. One data field is fetched, if a CHAIN command is set, the next command to be executed is fetched from the address contained in the data field. Thus, a branch to a DCW is performed.

5.6.1.4 CONDITIONAL TRANSFER (CXFER) Command

The CXFER command tests the internally-maintained status of the SVC15 request. Two data fields are fetched. The first data field points to two halfwords, the first of which is logically ANDed with the current state of the logical status halfword. The result is compared to the second halfword of the pair, and if equal, the next command to be executed is specified by the second data field. Otherwise, command execution continues with the next command in the current chain.

The CXFER command can be used to test for specific conditions as indicated by the logical status; the first halfword of the first data area contains a mask with a 1 in each bit portion to be tested; the second halfword of the first data area contains the value to be tested against (e.g., if one or more special characters were detected after a read, a different command sequence might be desired).

5.6.2 Control-Type Commands

A CHAIN command, command trap and time-out are valid flag bits. There are four control-type commands:

COMMAND	BINARY	HEX	NUMBER OF DATA FIELDS
EXAMINE	00000 001	01	1
RING WAIT	00001 001	09	0
ANSWER	00010 001	11	0
DISCONNECT	00011 001	19	0

5.6.2.1 EXAMINE Command

The EXAMINE command returns the device status of the specified adapter in one data field. The value obtained specifies the address of a writable byte into which the status of the device is stored. The last known physical device status is fetched from a byte in memory (DCB.DVST) maintained by the driver during I/O. When the byte is nonzero, its contents are returned and the byte is reset to zero. When the byte is zero, a sense status is performed on the device and its present status is returned. Thus, two EXAMINE commands should be suggested. The first will return the last status that caused the last error. The second will return the active device status at this time.

5.6.2.2 RING WAIT Command

The RING WAIT command suspends fetching of DCWs until a ring signal is received for the adapter. Interrupts from the adapter are enabled, but the data terminal ready lead to the modem is not. This command fetches no data fields and terminates when a ring signal is received from the adapter. When a CHAIN command is set, execution continues with the next command. Otherwise, the driver terminates. When time-out is set, the command waits as long as the value specified in the write timer halfword; when this interval expires, time-out error status is set. If time-out is not set, the command waits indefinitely for a ring signal.

5.6.2.3 ANSWER Command

The ANSWER command terminates immediately for nonswitched lines and switched lines that are already connected. For dial-in lines not connected, the data terminal ready lead to the modem is enabled, causing the modem to answer the incoming call. The command terminates when the data set indicates it is ready for I/O. TIME-OUT and CHAIN commands are handled as described in the RING WAIT command.

5.6.2.3 DISCONNECT Command

The DISCONNECT command disconnects from a switched line. The command resets the data terminal ready lead to the modem, suspends DCW fetching for one second and continues to the next command (if the CHAIN command is set) or terminates (if reset).

5.6.3 Read-Type Commands

A CHAIN command, command trap, buffer trap and time-out are valid flag bits. There are three read-type commands. When response time is critical, these commands must immediately follow a write-type or prepare-type command to use RAW or read-after-prepare-lookahead.

COMMAND	BINARY	HEX	NUMBER OF DATA FIELDS
READ BUFFER	00000 010	02	1 or 2
READ 1	00001 010	0A	1
READ 2	00010 010	12	1

5.6.3.1 READ BUFFER Command

The READ BUFFER command reads data into specified buffers. One or two data fields are fetched, depending on the buffer types. The data fields specify a buffer or buffer chain into which data is read. The first byte of the first data field fetched specifies whether the buffer type is direct text, indirect text, chained or queued buffers. If the buffer type is direct text, a second data field is fetched.

5.6.3.2 READ1 Command

The READ1 command reads one character into the specified location. One data field is fetched, containing the address of a byte in writable memory into which a character is to be read. No alignment is required. Data code must specify X'01'.

5.6.3.3 READ2 Command

The READ2 command reads two characters into the specified locations. One data field is fetched, containing the address of the first of two writable bytes into which two characters are read. No alignment is required. Data code must specify X'01'.

5.6.4 Prepare-Type Commands

A CHAIN command, command trap and time-out are valid flag bits. There are two prepare-type commands.

COMMAND	BINARY	HEX	NUMBER OF DATA FIELDS
PREPARE	00000 011	03	1
PREPARE3	00011 011	1B	1

5.6.4.1 PREPARE Command

The PREPARE command suspends DCW fetching until the specified character is received from the adapter. One data field is fetched, pointing to a byte that contains a match character. Characters are read from the line until one is detected that is equal to the match character, there the command terminates. If the PREPARE command is chained to a READ command, the READ will take effect immediately because of the read-after-prepare lookahead.

5.6.4.2 PREPARE3 Command

The PREPARE3 command is an asynchronous driver command only. The command sets a 200ms timer each time a character is received. The command ignores all line errors (parity, stop bit, etc.) and terminates if the timer expires or the match character is received.

5.6.5 Write-Type Commands

A CHAIN command, command trap, buffer trap and time-out are valid flag bits. There are three write-type commands:

COMMAND	BINARY	HEX	NUMBER OF DATA FIELDS
WRITE BUFFER	00000 100	04	1 or 2
WRITE1	00001 100	0C	1
WRITE2	00010 100	14	1

5.6.5.1 WRITE BUFFER Command

The WRITE BUFFER command writes data from specified buffers. One or two data fields are fetched, depending on the buffer type. The data field specifies a buffer or buffer chain from which data is written to the line. The left-most byte of the first data field fetched specifies whether the buffer type is direct text, indirect text, chained buffers or queued buffers. If the buffer type is direct text, a second data field is fetched.

5.6.5.2 WRITE1 Command

The WRITE1 command writes one character from the specified locations. One data field is fetched, containing the address of a byte from which a character is written to the line. No alignment is required. Data code must specify X'01'.

5.6.5.3 WRITE2 Command

The WRITE2 command is used to write two characters from specified locations. One data field is fetched, containing the address of the first of two bytes from which characters are written to the line. No alignment is required. Data code must specify X'01'.

5.6.6 Hold-Type Commands

A CHAIN command and command trap are valid flag bits. There is one hold-type command.

COMMAND	BINARY	HEX	NUMBER OF DATA FIELDS
HOLD SPACE	00000 101	05	1

5.6.6.1 HOLD SPACE (Line Break) Command

The HOLD SPACE command puts the line in a space (zero) condition for a specified interval. One data field is fetched, containing the address of a halfword with a time value in units of 100ms. The line is held in a continuous space (zero) condition for this interval. This command is valid for asynchronous communications only.

5.6.7 Mode-Type Commands

A CHAIN command and command trap are valid flag bits. Mode-type commands are used to change various default values in the DCB that are maintained by the driver. If the default value specified in the individual driver description is acceptable, a mode-type command is not necessary. Once a value is changed by a mode-type command, the only way to restore the default condition is by a mode-type command specifying the correct value. Coordinate such modifications if access is being shared by more than one program. There are ten defined mode-type commands:

COMMAND	BINARY	HEX	NUMBER OF DATA FIELDS
MODE TOUT	00000 110	06	1
MODE CMD2	00001 110	0E	1
MODE RCMD	00010 110	16	1
MODE WCMD	00011 110	1E	1
MODE RDISABL	00100 110	26	1
MODE WDISABL	00101 110	2E	1
MODE DISC	00110 110	36	1
MODE SYNCNT	00111 110	3E	1
MODE TRANSL	01000 110	46	1
MODE SPCHAR	01001 110	4E	1

5.6.7.1 MODE TOUT (Time-out Interval) Command

The MODE TOUT command sets the error time intervals for commands that enable the time-out flag. One data field is fetched containing the address of the first of two halfwords; the first halfword contains an error time-out interval for read-type operations, the second contains an interval for write-type operations. Both are in one second units. The data field pointer to the two halfword parameters should point to a fullword-aligned address.

This timer is strictly for error detection; when it expires, the SVC15 call terminates in error. Interval timing is performed via the WAIT command, which uses a separate (100ms resolution) clock. The resolution of the time is accurate to +0, -1 second. Time-out values should therefore be set to the desired number of seconds plus one.

5.6.7.2 MODE CMD2 (Adapter) Command

The MODE CMD2 command specifies the device-dependent command used to set adapter options. One data field is fetched with the address of a byte containing the device command that should be output to specify programmable adapter options such as parity, number of data bits, etc.

5.6.7.3 MODE RCMD (Read) and MODE WCMD (Write) Commands

These commands specify the device-dependent commands to set read or write mode in the adapter. One data field is fetched for these commands, with the address of a byte containing the device command to be output for read or write data transfers, respectively. This command should enable interrupts.

5.6.7.4 MODE RDIS (Read Disable) and MODE WDIS (Write Disable) Commands

These commands specify the device-dependent commands used to quiesce the read or write side of the adapter. One data field is fetched with the address of a byte containing the device command to be output to set the quiesce read or write side of the line. These commands should disable interrupts.

5.6.7.5 MODE DISC (Disconnect) Command

The MODE DISC command specifies the device-dependent command used to disconnect the adapter from the line. One data field is fetched, with the address of a byte containing the device command to be output to disconnect the communications line (reset data terminal ready).

5.6.7.6 MODE SYNCNT (SYNC Character Count) Command

The MODE SYNCNT command specifies the SYNC character count. One data field is fetched, with the address of a byte containing the number of leading SYNCs transmitted (synchronous drivers only).

5.6.7.7 MODE SPCHAR (Special Character Enable Masks) Command

The MODE SPCHAR command sets up the bit masks necessary for special character detection. One data field is fetched, containing the address of the first of two halfwords; the first has a bit mask used to enable recognition of specific special characters during read-type operations; the second contains a bit mask used to enable special character recognition during write-type operations.

5.6.7.8 MODE TRANSL (Translation Options) Command

The MODE TRANSL command specifies translation options. One data field is fetched containing the address of a byte with a series of indicators controlling translation options.

5.6.8 Test-Type Commands

Reserved for driver-dependent on-line test functions.

CHAPTER 6
DATA COMMUNICATIONS STRUCTURES

6.1 INTRODUCTION

This chapter describes the internal operation of the various structures and subroutines that must be added or modified to include the Basic Data Communications Subsystem in the operating system.

The Basic Data Communications System Support Module includes many subroutines that are called from the line drivers. These subroutines are involved with interfacing the driver with the operating system and with common buffer management and command fetching operations.

6.2 DATA COMMUNICATIONS LINE DRIVERS

The most obvious differences between data communications line drivers and the general-purpose drivers are:

- Parameter blocks
- Supervisor call 15 (SVC15) instead of SVC1
- Format of the device-dependent portion of the device control block (DCB)
- Event service routine (ESR)

  Normally, the driver schedules the ESR itself from the interrupt service (IS) state. However, the operating system executive might also schedule the ESR because of a time-out, cancel, power fail or the close of a logical unit (lu) during input/output (I/O) by the calling task or command processor.

The OS/32 Basic Data Communications System Support Module includes many subroutines that are called from data communications line drivers. These subroutines are involved with interfacing the driver with the operating system and with common buffer management and command fetching operations. Some of these subroutines are exclusively for driver use, but have been included in this module to reduce code duplication between drivers.

6.3 CONTROL BLOCK FORMATS

This section describes the control blocks and other system structures used in data communications. The following structures will be discussed:

- Device control block (DCB)
- Line control block (LCB)
- Channel control block (CCB)
- Drop control table (DCT) for zero-bit insertion/deletion data link control (ZDLC) communications
- Drop definition table (DDT) for ZDLC communications
- DCT for asynchronous multidrop communications
- Drop access table (DAT) for asynchronous multidrop communications
- Input/output block (IOB) for asynchronous multidrop communications
- Station descriptor table (SDT) for 3270 emulator
- DDT for 3270 emulator
- Input/output handler (IOH)
- File manager handler (FMH)

6.3.1 Data Communications Device Control Block (DCB)

The data communications DCB provides a table-driven mechanism for line drivers and terminal managers. It contains parameters and system information, such as addresses of line driver/terminal manager modules, addresses of task control blocks (TCBs) and user parameter blocks and specific device-related fields. The DCB structure is divided into three portions:

- Device-independent (standard DCB)
- Data communications
- Device-dependent

Figure 6-1 shows a generalized DCB and how its three sections relate to each other.

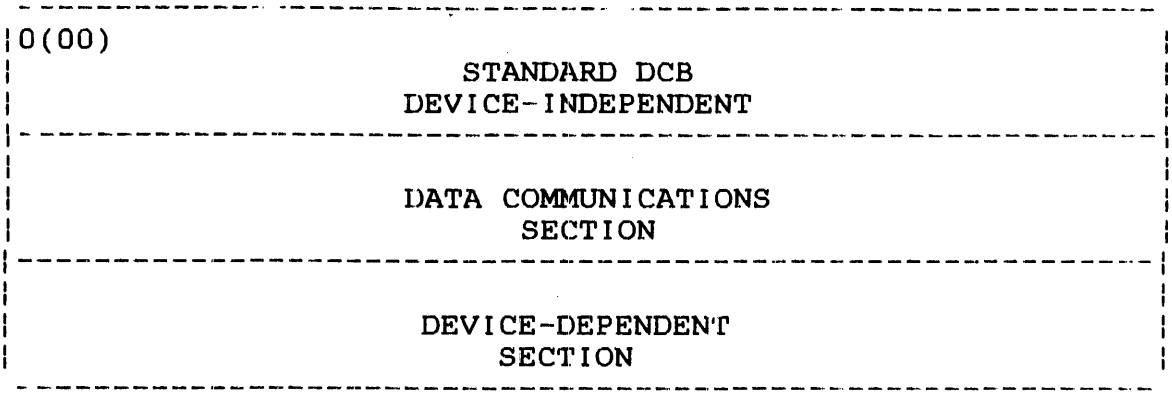


Figure 6-1 DCB Sections

The device-independent portion, also called the basic DCB, is identical to the standard OS/32 DCB. Figure 6-2 shows the fields of the basic DCB.

The data communications-related portion of the DCB, immediately following the basic DCB, has fields pertaining only to devices requiring basic data communications support. Figure 6-3 shows the data communications-related portion of the DCB.

The device-dependent portion of the DCB, immediately following the data communications-related portion, is used only if device-dependent access is requested. This section contains fields unique to individual lines or devices (e.g., asynchronous devices, binary synchronous lines and ZDLC lines).

All references to the DCB fields must use the names specified in Figures 6-2 and 6-3. These names come from the OS/32 and Basic Data Communications Internal Macro and Structure Library supplied in the standard software package.

6.3.1.1 Device Control Block (DCB) Device-Independent Portion (Standard DCB)

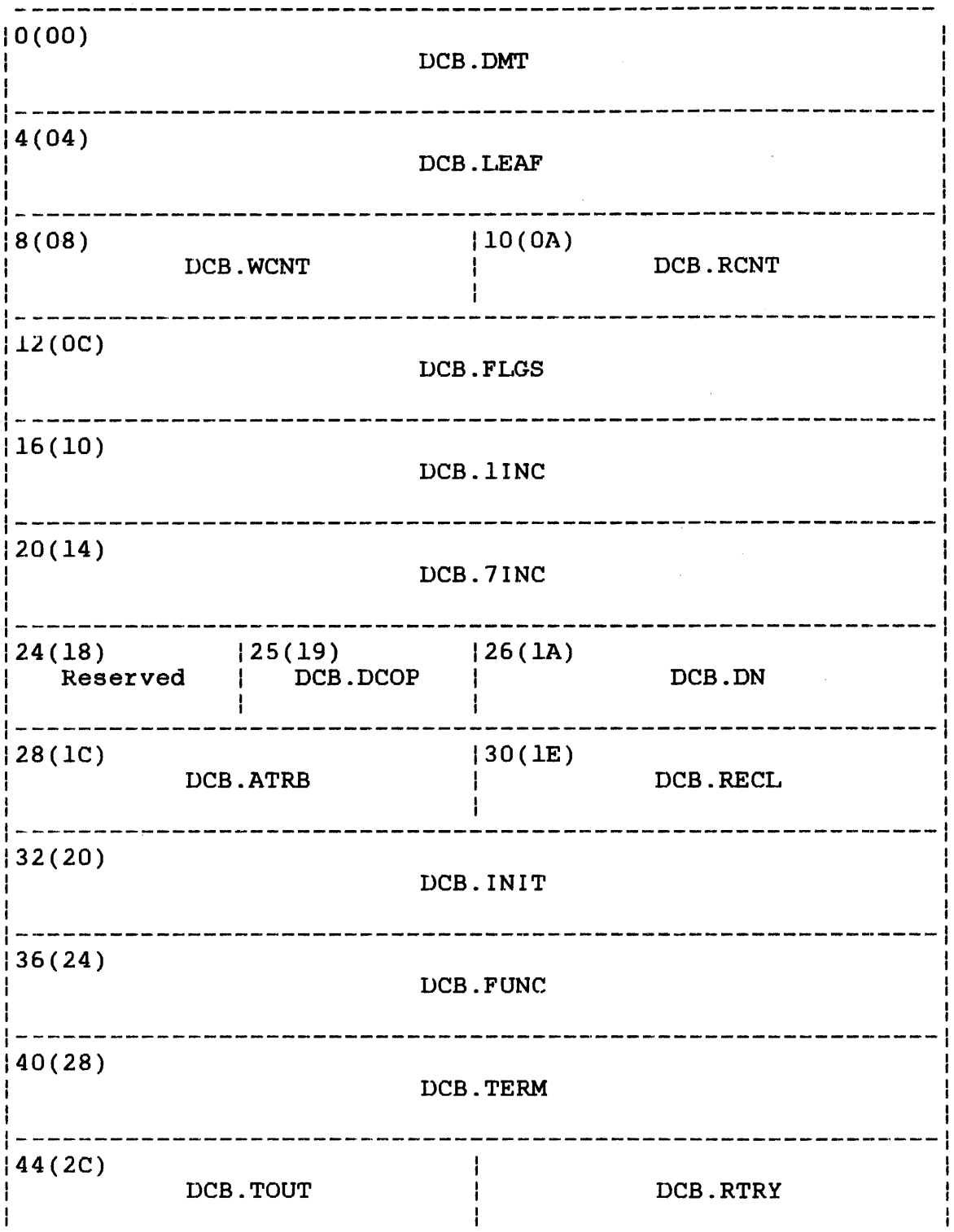


Figure 6-2 Basic DCB Fields

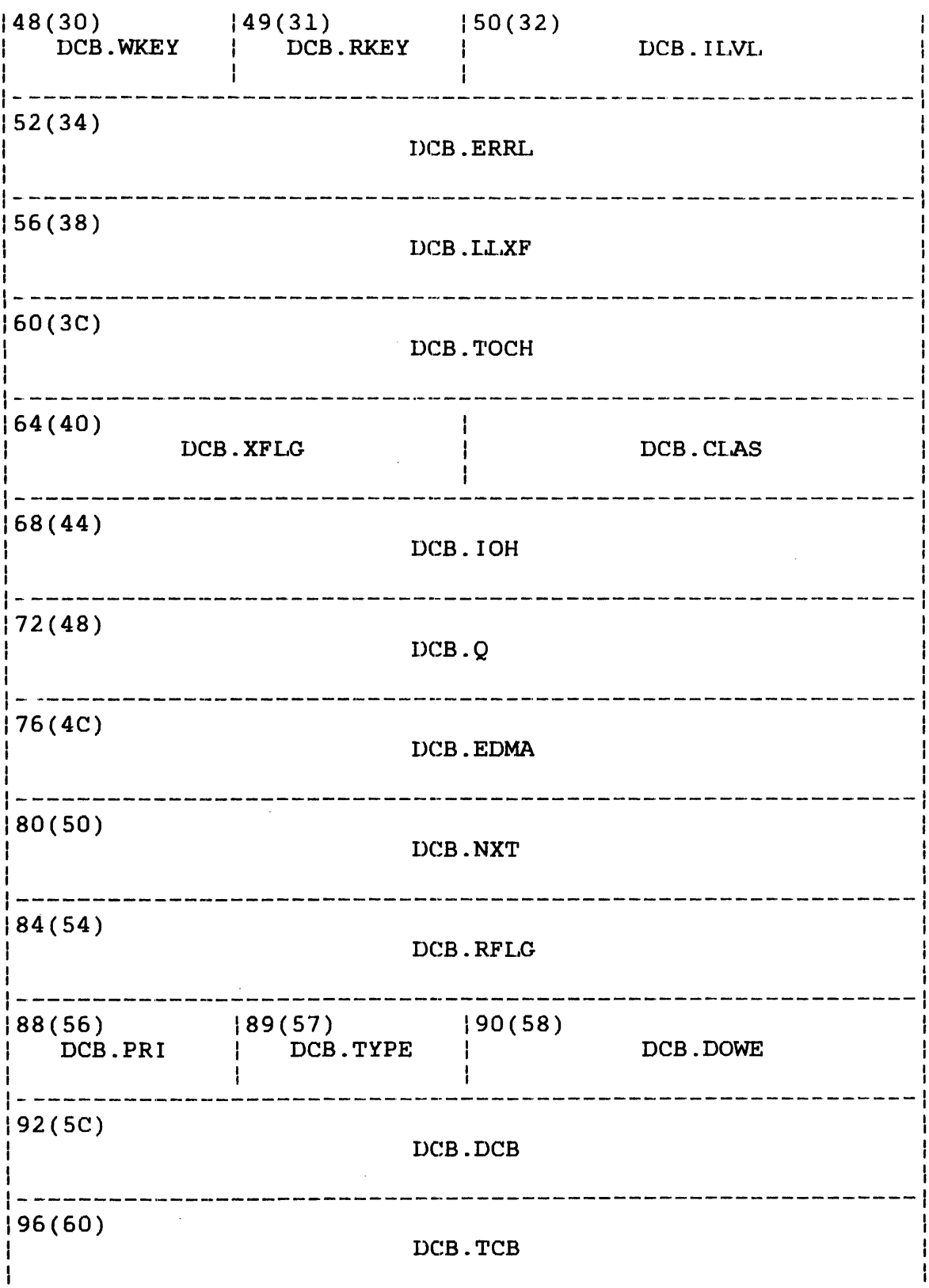


Figure 6-2 Basic DCB Fields (Continued)

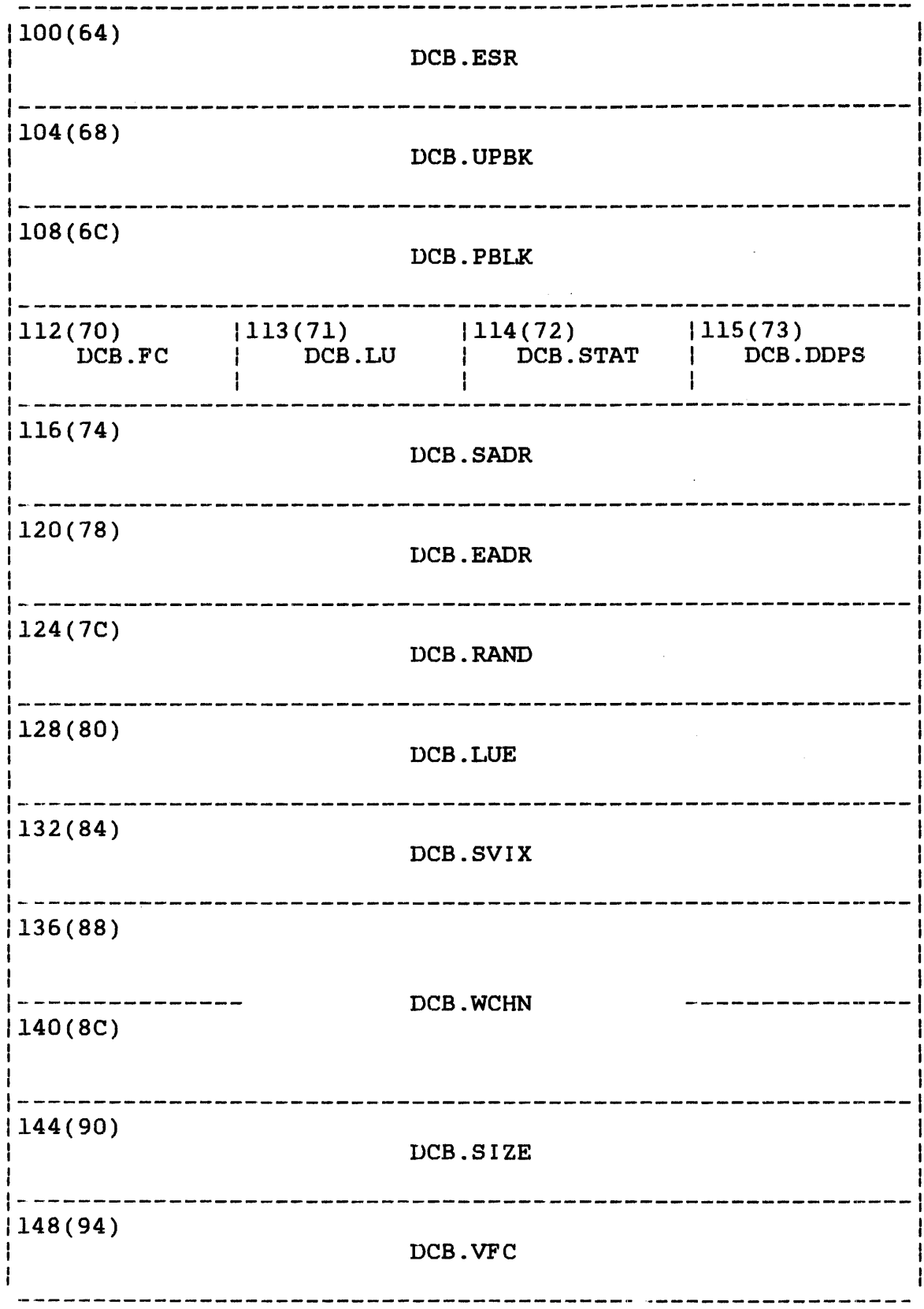


Figure 6-2 Basic DCB Fields (Continued)

Fields:

DCB.LEAF — is the fullword address of the event coordination table entry for the physical devices described by the DCB.

DCB.INIT — is the fullword starting address of the SVC1 driver/terminal manager code. This address corresponds to the INITxxxx label in the driver/terminal manager module itself.

DCB.TERM — is the fullword address of the driver termination routine scheduled as the final ESR at end of command processing.

DCB.IOH — is the fullword list address of the IOH routines. The SVC1 executor vectors from this IOH list to the specific routine for the requested function. IOH lists exist for asynchronous devices (IOHXASY), binary synchronous devices (IOHMBSC), ZBID devices (IOHCZBD) and SVC15 accessed devices (IOHSVCF).

DCB.DONE — is the fullword address of a special routine that does special device-dependent functions at I/O completion before branching to the standard IODONE routine.

DCB.ESR — is the fullword address of the next driver entry point. At system queue service (SQS) time, event servicing begins here. Normally, this address is for the ISSEXEC routine.

6.3.1.2 Device Control Blocks (DCB) Data Communications-Related Portion

The data communications DCB contains fields pertaining to the devices requiring data communications support. All the fields within this part of the DCB are described in this section. Figure 6-3 shows the format of the data communications portion of the DCB.

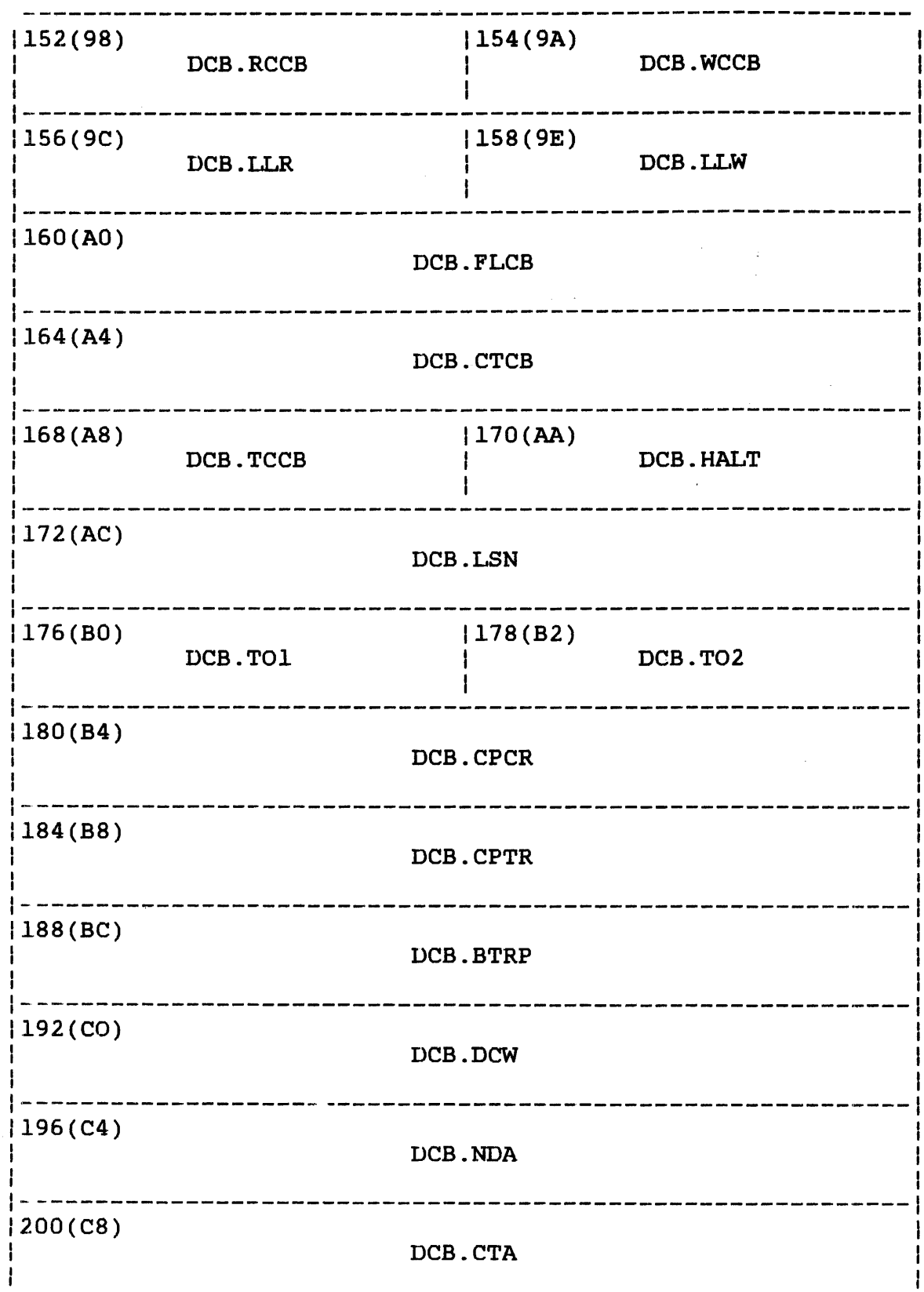


Figure 6-3 Data Communications DCB Fields

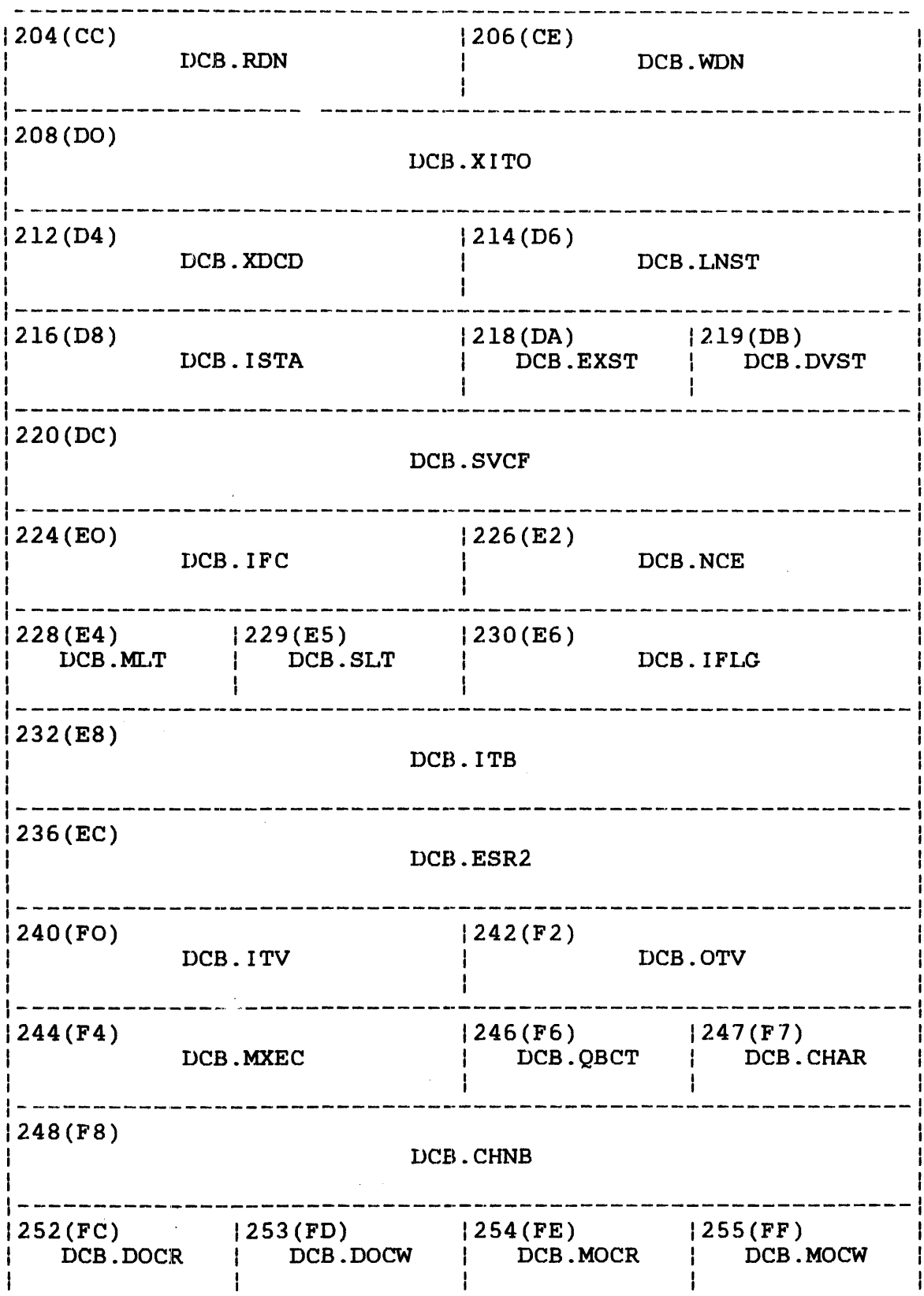


Figure 6-3 Data Communications DCB Fields (Continued)

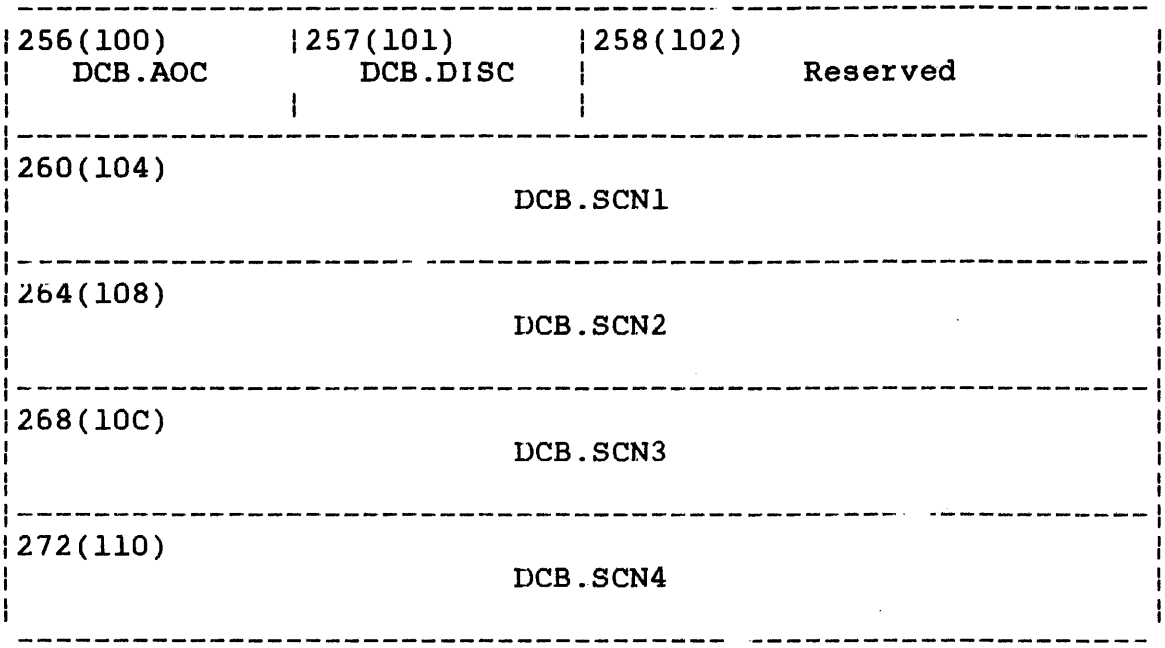


Figure 6-3 Data Communications DCB Fields (Continued)

Fields:

DCB.RCCB is the halfword with the address of the read CCB.

DCB.WCCB is the halfword with the address of the write CCB.

NOTE

The auto driver channel requires CCBs to hold buffer pointers and other information required during interrupt service routines (ISRs). Data communications require the read CCB and the write CCB to process read-after-write (RAW). A simplex device can have only one CCB and can specify zero for the unused one.

Devices geared to human response time can also use a single CCB for reads and writes. The use of one CCB saves some system memory, but also inhibits the RAW lookahead, possibly slowing read response time. (See below for further information on CCBs.)

DCB.LLR is the driver-maintained halfword to specify the length of last read.

DCB.LLW is the driver-maintained halfword to specify the length of last write.

DCB.FLCB is the fullword address of a pointer to the beginning of an LCB chain used by file manager routines GETFCB and RELEFCB. This entry point must be in the same position as DCB.FCB within the file manager DCB.

DCB.CTCB is the fullword address of the TCB for the currently executing task. The TCB selects I/O requests on behalf of the task.

DCB.TCCB is the halfword address of the timer CCB.

DCB.HALT is the halfword address of a special halt routine (pure code) called by some drivers.

DCB.LSN is the logical segment number for an address check.

DCB.TO1 is time-out 1 (screen time).

DCB.TO2 is time-out 2 (OP response limit).

DCB.CPCR is the fullword address for a channel program continuation return, referred to for line driver/terminal manager interface.

DCB.CPTR is the fullword address for a channel program termination return, referred to for line driver/terminal manager interface.

DCB.BTRP is the fullword address of the pointer to a continuous read buffer trap, referred to for line driver/terminal manager interface.

DCB.DCW is the fullword address of the relocated driver command words (DCWs).

DCB.NDA is the fullword address of the relocated next data area.

DCB.CTA — is the fullword address of the command table within the particular driver. Every driver has a command table with addresses of routines that execute specific DCW commands. When the task issues an SVC15 request to the driver, the command field in the DCW serves as an index to one of these driver routines.

Drivers can support different subsets of the standard set of driver commands. One command table might have some driver commands in common with other drivers while having other commands unique to itself. The command table must be coded as a DAC label. The label must be declared an EXTRN.

DCB.RDN — is the halfword with the read device number.

DCB.WDN — is the halfword with the write device number.

Both DCB.RDN and DCB.WDN — are adjusted at assign time based on the actual system generated (sysgened) device number (DCB.DN) and the type of line indicated in the DCB.XDCD field (i.e., 2-wire or 4-wire, simplex, etc.).

DCB.XITO — is the fullword with the SVC1 extended data communications options obtained from the user parameter block.

DCB.XDCD — is the halfword for the extended device code. To use file manager routines, the entry must be at the same offset in the DCB as LCB.XDCD is within the LCB. DCB.XDCD must be generated as DC Z(XDCD). XDCD must be declared an EXTRN.

DCB.LNST — is the halfword for the line activity status used by line drivers and terminal managers. Table 6-1 describes the status bits for this halfword. DCB.LNST is structurally identical to LCB.LNST.

TABLE 6-1 DCB.LNST BIT DEFINITIONS

BIT	HEX MASK	NAME	MEANING
0	8000	LNS.BSYM/B	DCB is being used.
1	4000	LNS.RWM/B	Line is currently performing a read.
2	2000	LNS.INTM/B	Line is currently with a read or write initiation phase.
3	1000	LNS.ACKM/B	Next ACK should be an ACK1.
4	0800	LNS.ACQM/B	An ACK is required.
5	0400	LNS.HLDM/B	Put any future I/O requests into I/O wait. A DONE return to user task (u-task) is pending.
6	0200	LNS.IOM/B	I/O is currently in progress.
7	0100	LNS.RVIM/B	Reverse interrupt received.
8	0080	LNS.ERRM/B	An outstanding unrecoverable error exists.
9	0040	LNS.EOTM/B	End of transmission received.
10	0020	LNS.IOQM/B	An image I/O write is on queue.
11		Reserved	
12	0008	LNS.CPTM/B	Line checkpoint is being performed.
13	0004	LNS.CLSM/B	Line close is being performed.
14	0002	LNS.EQM/B	ENQ sent for read time-out.
15		Reserved	

DCB.ISTA is the halfword into which the line driver stores SVC15 status.

DCB.EXST is the byte reserved for the driver.

DCB.DVST is the byte for saving the latest device status after an interrupt. This field and the DCB.ISTA field are useful for debugging purposes.

DCB.SVCF — is the fullword with a pointer to the line driver initiation routine. This address is used by the SVC15 executor and the terminal manager to enter the driver. In the DCB coding, DCB.SVCF must be coded as DAC INITxxxx and this label must be declared an EXTRN.

As for general-purpose drivers, the driver initiation routine pointer is the beginning address of the driver responsible for communicating with the attached adapter and device. Entered in the event service state from the SVC15 executor, this routine assumes DCB.DCW and DCB.NDA are valid and usually starts execution of DCW commands.

DCB.IFC — is the halfword containing the SVC15 function byte and other information bits for line drivers.

DCB.NCE — is the halfword with which the line driver keeps track of the number of commands executed.

DCB.MLT — is the byte containing the main line-type descriptor for ITFM, the file manager.

DCB.SLT — is the byte containing the subline-type descriptor for ITFM.

DCB.IFLG — is the halfword for flags.

DCB.ITB — is the fullword with bits to be used by ISSEXEC and ITSRABS when scheduling ESRs for buffer management and system support functions.

DCB.ESR2 — is the fullword address of the second ESR.

DCB.ITV — is the halfword with an error timer value in seconds for input (reads). The MODE time-out command can change this value.

DCB.OTV — is the halfword with an error timer value in seconds for output (writes). The MODE time-out command can change this value.

DCB.MXEC — is the halfword with a value for the maximum number of allowable error retries.

DCB.QBCT — is the byte with the count of queued buffers in use. With this count, the line driver keeps track of queued buffers.

DCB.CHAR — is the byte for temporarily saving characters.

DCB.CHNB is the fullword address of the first buffer in a chain.

DCB.DOCR is the byte used by the driver to disable the adapter after completing each read request. DCB.DOCR is coded as DCB 'hexadecimal value' to agree with the adapter.

DCB.DOCW is the byte used by the driver to disable the adapter after completing each write request. DCB.DOCW is coded as DCB 'hexadecimal value' to agree with the adapter.

DCB.MOCR is the byte used by the driver to enable interrupts and to place the adapter into read mode. DCB.MOCR is coded as DB X'value', with the value of the bits depending upon how the adapter is to be used.

DCB.MOCW is the byte used by the driver to enable interrupts and to place the adapter into write mode. DCB.MOCW must be coded as DB X'value', with the value of the bits dependent upon how the adapter is to be used.

DCB.AOC is the byte used to load programmable adapters with required information. DCB.AOC is coded as DB X'values', where the values might reflect such programmable information as:

- line speed,
- character size,
- parity information,
- number of stop bits,
- SYNC character,
- test function or local loop-back, and
- synchronization technique.

DCB.DISC is the byte used by the driver to disable interrupts and, for switched lines, to disconnect the line (i.e., to drop data terminal ready). DCB.DISC is coded as DC X'value'. If the read device number is zero, then this command is issued to the write device number.

DCB.SCN1 is the fullword for the SVC15 data chain area 1.

DCB.SCN2 is the fullword for the SVC15 data chain area 2.

DCB.SCN3 is the fullword for the SVC15 data chain area 3.

DCB.SCN4 is the fullword for the SVC15 data chain area 4.

Terminal managers may place values into these fullwords to provide data buffering information (e.g., buffer BEGIN and END address or TO and FROM list addresses to the line driver).

6.3.1.3 Device Control Block (DCB) Device-Dependent Portion

In order to remain reentrant, all additional storage required by any terminal managers must be in additional DCB space set aside here or in extra memory obtained from system space.

For a description of the device-dependent portion of the DCB, see the appropriate data communications manuals.

6.3.2 Line Control Block (LCB)

All buffered terminal managers use an LCB to provide required format pointers, line control and device-independent interfaces.

An LCB is usually a copy of the appropriate DCB with additional space for data blocks. The LCB is obtained from dynamic system space. As shown in Figure 6-5, the LCB consists of three segments:

- Device-independent segment (basic LCB)
- Device-dependent segment
- Data block descriptor

6.3.2.1 Line Control Block (LCB) Device-Independent Portion

The device-independent portion of the LCB is pictured in Figure 6-4. The device-independent portion of the LCB is structurally identical to the basic DCB of Figure 6-1.

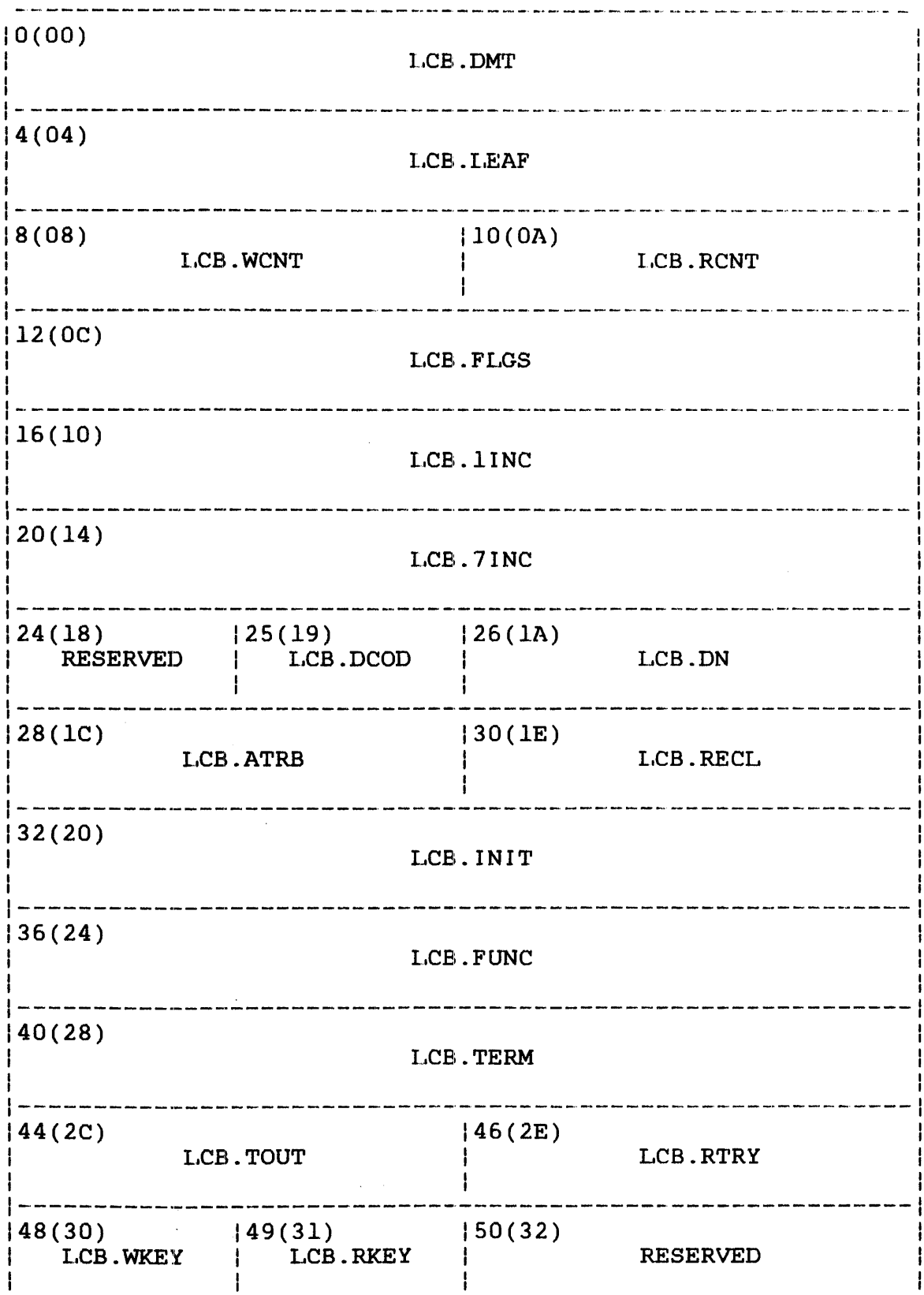


Figure 6-4 Basic LCB Fields

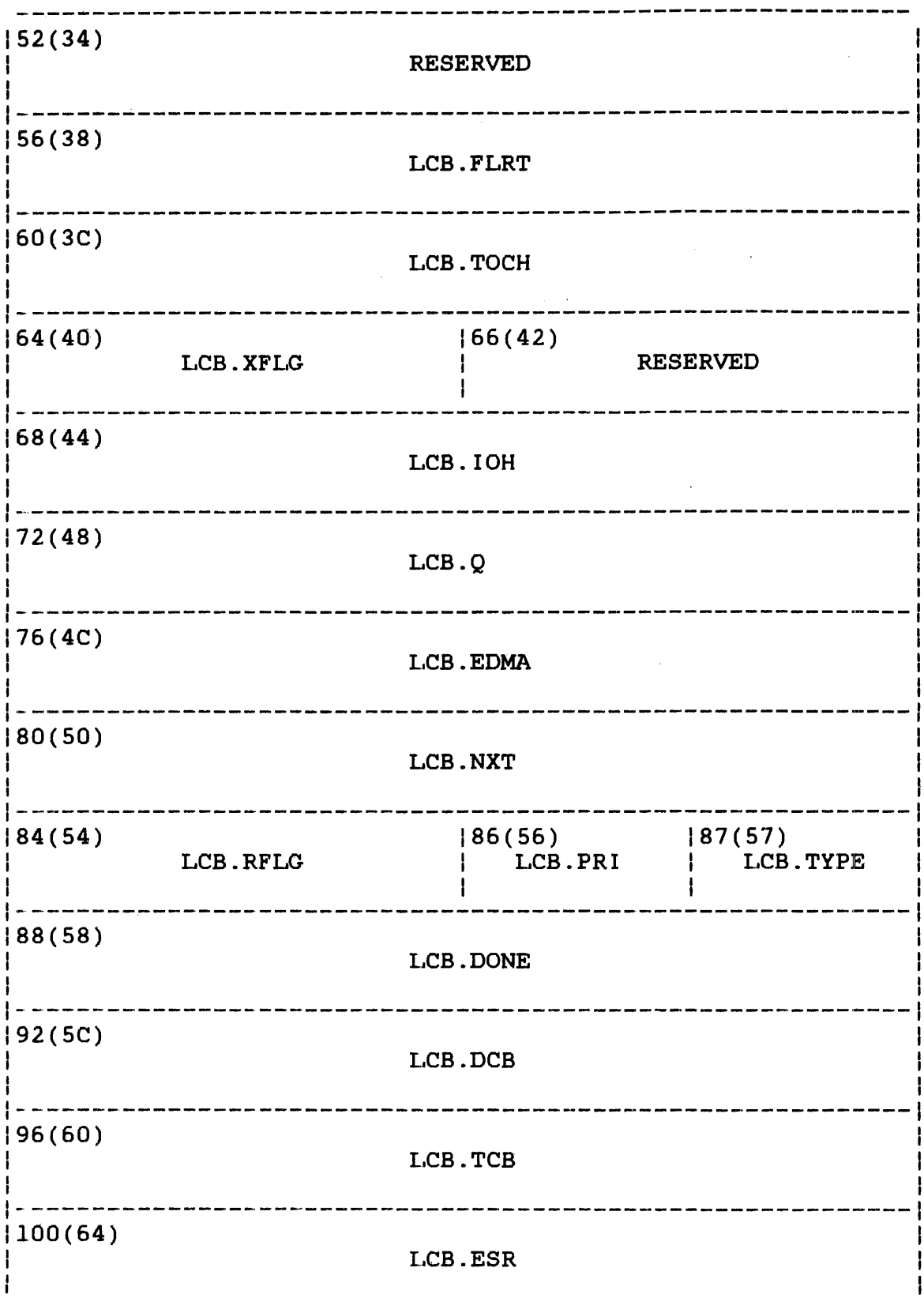


Figure 6-4 Basic LCB Fields (Continued)

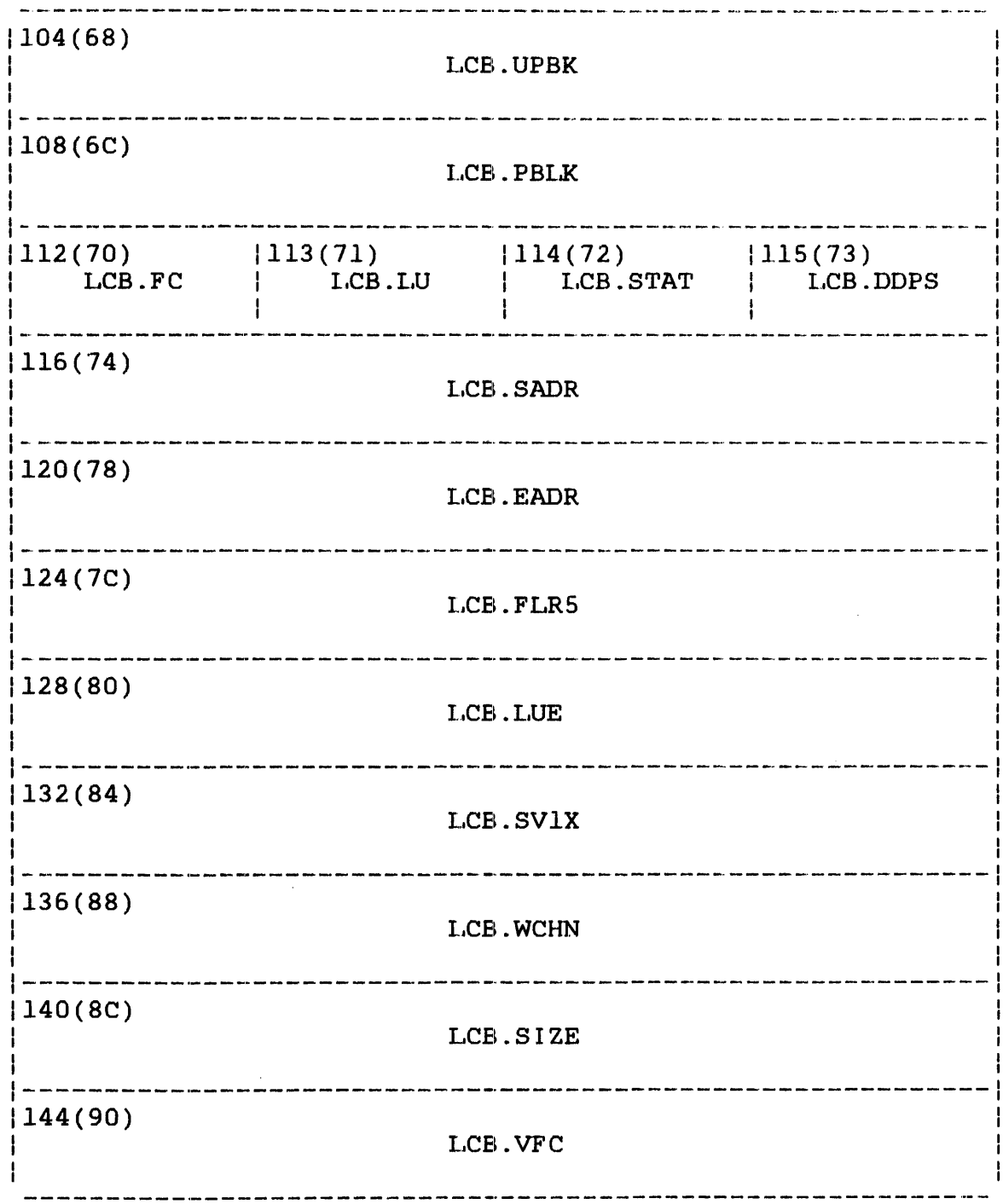


Figure 6-4 Basic LCB Fields (Continued)

Fields:

LCB.DMT	is the fullword address of the device mnemonic table (DMT) entry.
LCB.LEAF	is the fullword address of an event leaf.
LCB.WCNT	is the halfword for the write count.
LCB.RCNT	is the halfword for the read count.
LCB.FLGS	is the fullword for the flags.
LCB.1INC	is the SVC1 device intercept.
LCB.7INC	is the SVC7 device intercept.
LCB.DCOD	is the byte for the device code (DCB number).
LCB.DN	is the halfword for the device number (a physical address).
LCB.ATRB	is the halfword for the device attributes.
LCB.RECL	is the halfword for the record length.
LCB.INIT	is the fullword address of the driver initiation routine.
LCB.FUNC	is the fullword address of the driver function routine.
LCB.TERM	is the fullword address of the driver termination routine.
LCB.TOUT	is the halfword for the time-out constant.
LCB.RTRY	is the halfword for the operation retry count.
LCB.WKEY	is the halfword for the write key.
LCB.RKEY	is the halfword for the read key.
LCB.FLRT	is the fullword for the close/checkpoint save area.
LCB.TOCH	is the fullword address of the time-out chain.
LCB.XFLG	is the halfword for the device-dependent flags.
LCB.IOH	is the fullword address of the IOH list. The default is zero.
LCB.Q	is the fullword address of the queue strategy routine. The default is zero.

LCB.EDMA	is the fullword address of the extended direct memory access (EDMA) strategy routine.
LCB.NXT	is the fullword link to the next IOB.
LCB.RFLG	is the halfword for the request-dependent flags.
LCB.PRI	is the byte for the I/O priority.
LCB.TYPE	is the byte for the IOB-type code.
LCB.DONE	is the fullword address of the IODONE executor.
LCB.DCB	is the fullword address of the DCB.
LCB.TCB	is the fullword address of the TCB.
LCB.ESR	is the fullword address of the next entry into driver.
LCB.UPBK	is the fullword task-relative address of the SVC parameter block.
LCB.PBLK	is the fullword absolute address of the SVC parameter block.
LCB.FC	is the byte for the SVC function code.
LCB.LU	is the byte for the SVC logical unit (lu).
LCB.STAT	is the byte for the I/O status.
LCB.DDPS	is the byte for the device-dependent status.
LCB.SADR	is the fullword absolute starting address of the SVC1 buffer.
LCB.EADR	is the fullword absolute ending address of the SVC1 buffer.
LCB.FLR5	is the fullword checkpoint save area for register 5.
LCB.LUE	is the fullword LCB address used in the contiguous file manager.
LCB.SV1X	is the extended SVC1 word.
LCB.WCHN	is the task waiting for this I/O to complete.
LCB.SIZE	is the size in sectors or lines.
LCB.VFC	is the fullword address of VFCDCB.

6.3.2.2 Line Control Block (LCB) Device-Dependent Portion

The device-dependent portion of the LCB is pictured in Figure 6-5.

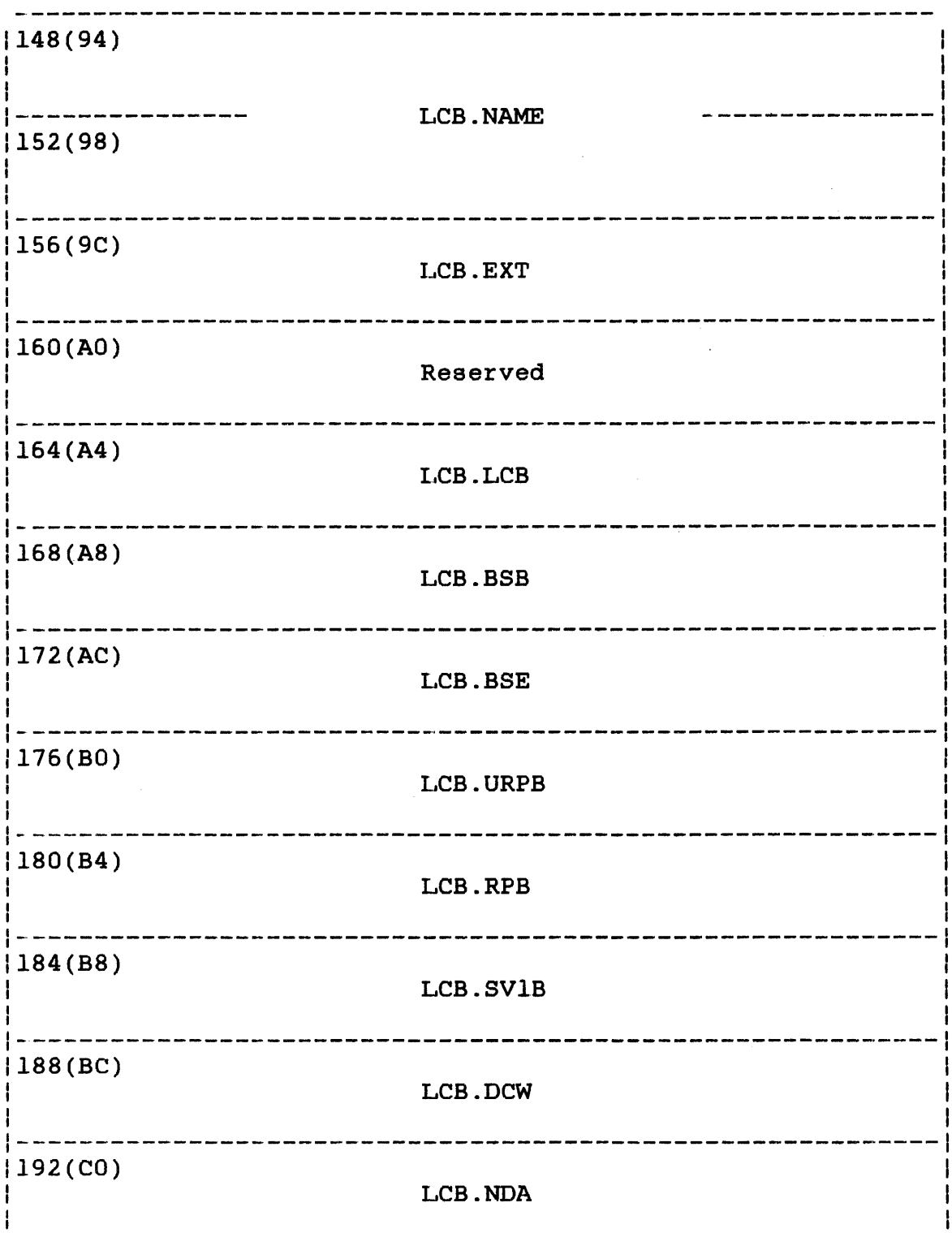


Figure 6-5 Device-Dependent LCB Fields

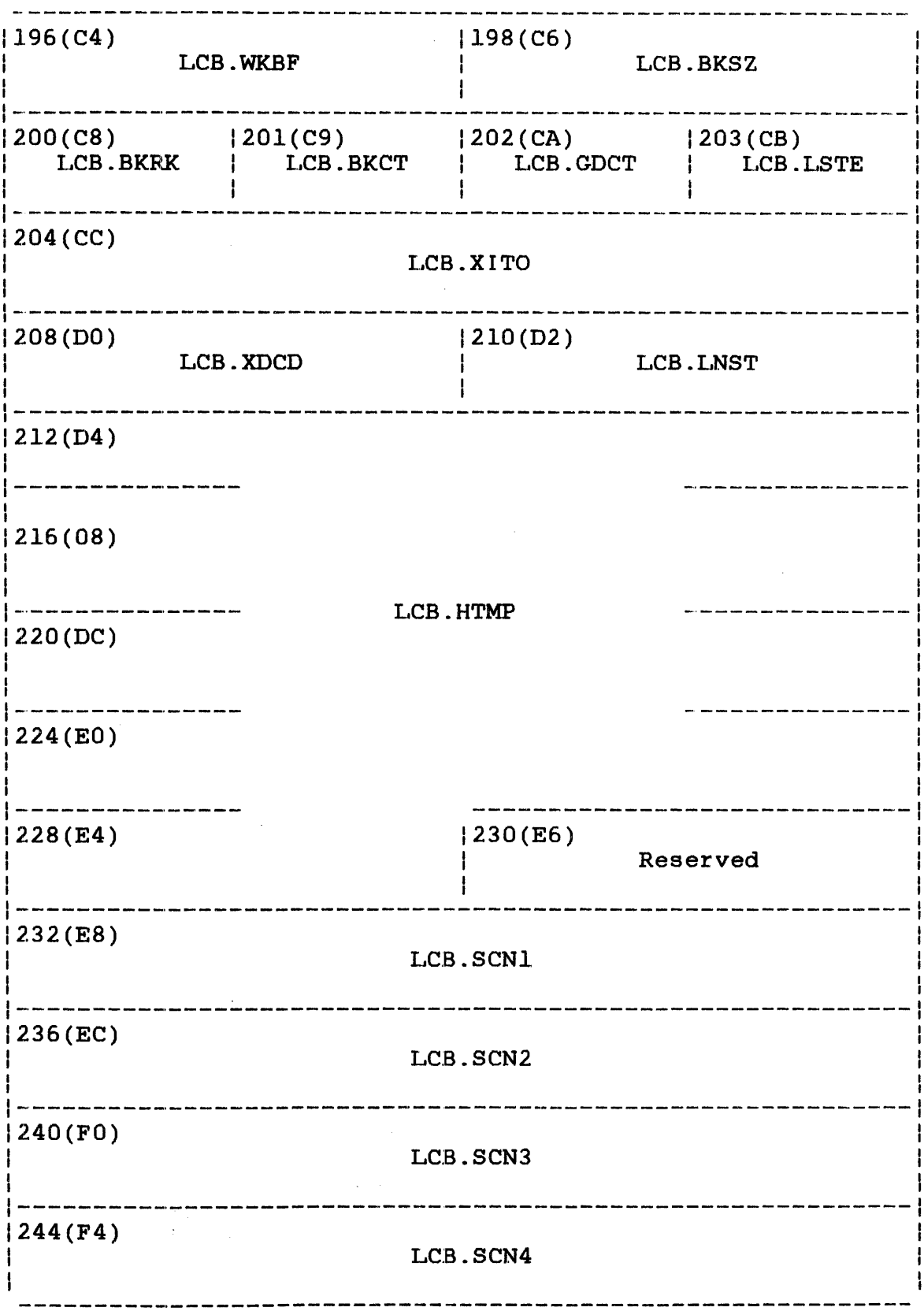


Figure 6-5 Device-Dependent LCB Fields (Continued)

It should be noted that the device-dependent portion of the LCB described in this section may vary in structure for different terminal managers.

Fields:

LCB.NAME	is the doubleword for the filename.
LCB.EXT	is the fullword for the extension.
LCB.LCB	is the fullword LCB-to-LCB linkage address for SVC7.
LCB.BSB	is the fullword absolute starting address of the segment (the buffer segment begin address).
LCB.BSE	is the fullword absolute ending address of the segment (the buffer segment end address).
LCB.URPB	is the fullword task-relative address of the SVC parameter block.
LCB.RPB	is the fullword absolute address of the SVC parameter block.
LCB.SV1B	is the fullword address of the SVC1 buffer for delays.
LCB.DCW	is the fullword for storage of the DCW address for retries.
LCB.NDA	is the fullword for storage of the next data area (NDA) address for retries.
LCB.WKBF	is the halfword address of the work buffer to receive ACK.
LCB.BKSZ	is the halfword for the size of each data buffer.
LCB.BKRK	is the byte for the maximum records permitted in each block.
LCB.BKCT	is the byte for the total number of data blocks assigned.
LCB.GDCT	is the byte for the good transmission counter.
LCB.LSTE	is the byte for the error code of the last transmission.
LCB.XITO	is the fullword for the extended options.
LCB.XDCD	is the halfword for the extended device code.

LCB.LNST is the halfword for the line activity status.

LCB.HTMP are the nine halfwords for the horizontal tab bit map.

LCB.SCN1 is the fullword for data area 1.

LCB.SCN2 is the fullword for data area 2.

LCB.SCN3 is the fullword for data area 3.

LCB.SCN4 is the fullword for data area 4.

6.3.2.3 Line Control Block (LCB) Data Block Descriptor Portion

The data block descriptor described in this section is an LCB subtable that controls the use of individual internal buffers.

BLK.RKCT is the byte for the count of records in the block.

BLK.ADR is the address of the data block.

BLK.PTR is the halfword for relative offset into a data block.

BLK.DSCR is the halfword for the data block descriptor flags. See Table 6-2.

TABLE 6-2 BLOCK DESCRIPTOR FLAG BIT DEFINITIONS

BIT	HEX MASK	NAME	MEANING
0	8000	BLK.BSM/B	Line buffer is currently being used.
1	4000	BLK.RWM/B	Line buffer is currently being used for a read.
2	2000	BLK.BKM/B	Line buffer blocking or deblocking is currently in progress.
3	1000	BLK.IOM/B	Line buffer is currently being used for I/O.
4	0800	BLK.QUM/B	Line buffer is on queue for either an output write or input deblocking.
5	0400	BLK.INM/B	Second line buffer is on queue.
6	0200	BLK.QU2M/B	Reserved
8	0080	BLK.EXM/B	Line buffer contains an ETX character.
9	0040	BLK.AKM/B	Reserved

6.3.3 Channel Control Block (CCB)

The CCB contains the address of the DCB, the fullwords and halfwords used by the line drivers and the clock halfword.

6.3.3.1 Channel Control Block (CCB) Device-Independent Portion

This section discusses the device-independent (standard) fields of the CCB. The device-independent CCB format is illustrated in Figure 6-6.

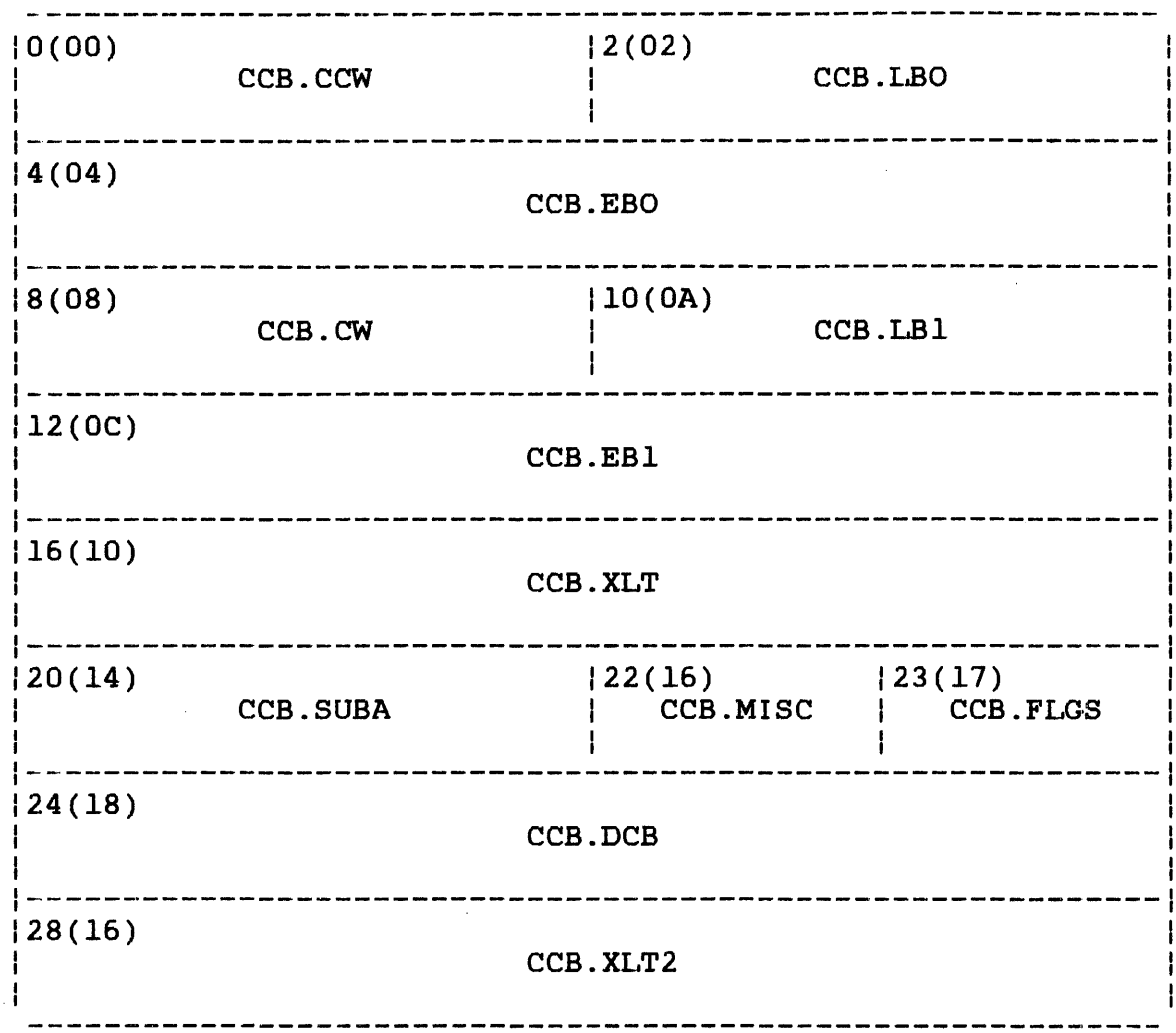


Figure 6-6 CCB Device-Independent Portion

Fields:

CCB.CCW	is the halfword containing the channel command word (CCW).
CCB.LBO	is the halfword containing the length of buffer 0.
CCB.EBO	is the fullword containing the end address of buffer 0.
CCB.CW	is the 16-bit check "word".
CCB.LB1	is the halfword containing the length of buffer 1.
CCB.EB1	is the fullword containing the end address of buffer 1.
CCB.XLT	is the fullword containing the address of the translation table.
CCB.SUBA	is the halfword address of the subroutine (pure code).
CCB.MISC	is the 1-byte temporary save area used by line drivers.
CCB.FLGS	is the 1-byte for CCB flags.
CCB.DCB	is the fullword containing the address of the associated DCB.
CCB.XLT2	is the fullword address of the secondary translation table.

6.3.3.2 Channel Control Block (CCB) Device-Dependent Portion

This section discusses the fields of the CCB used by data communications. The CCB format is illustrated in Figure 6-7.

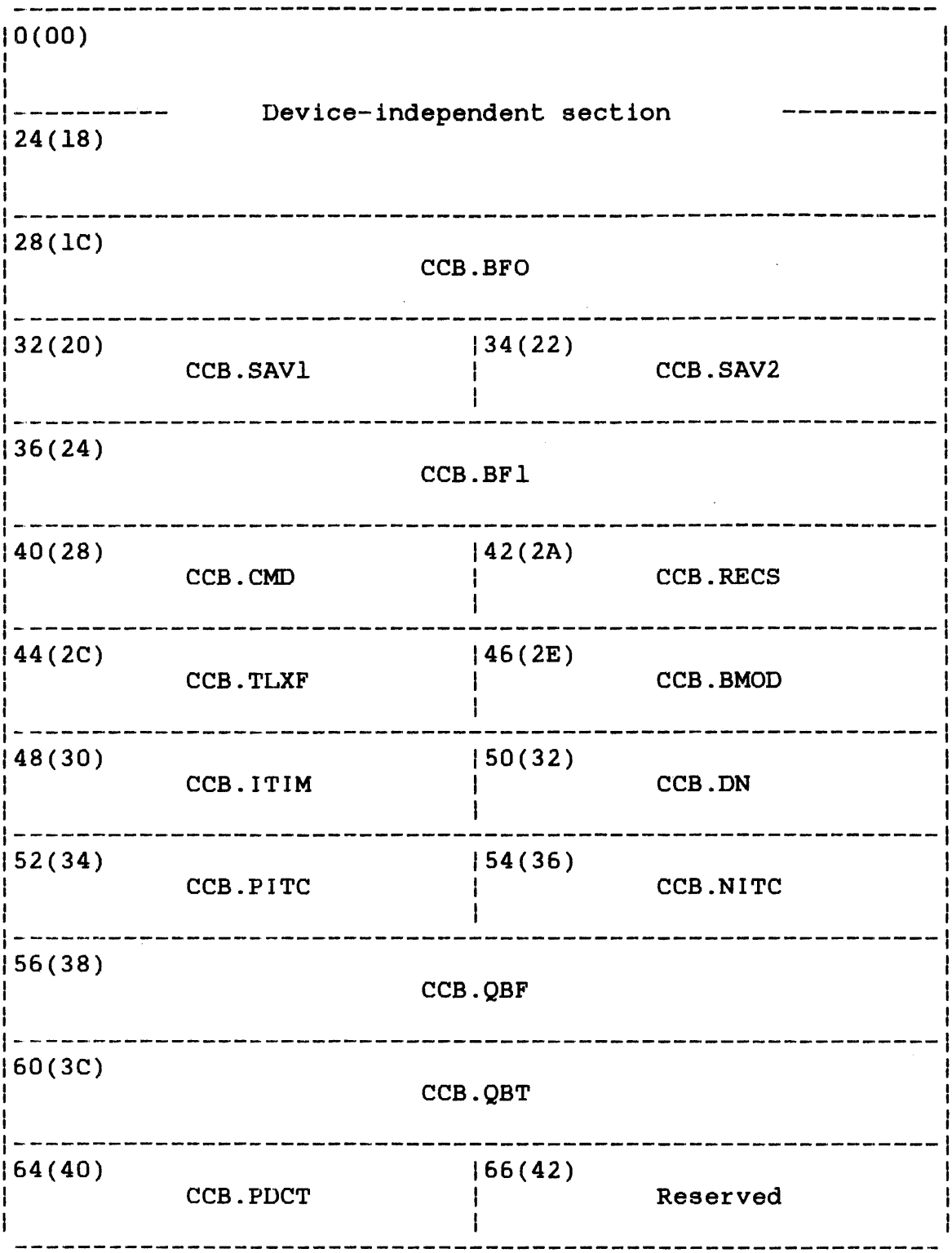


Figure 6-7 Data Communications CCB Format

Fields:

CCB.BF0	is the fullword used by the line driver during ISRs containing the beginning address of buffer zero.
CCB.SAV1	is the halfword save area 1.
CCB.SAV2	is the halfword save area 2.
CCB.BF1	is the fullword used by the line driver during ISRs containing the beginning address of buffer 1.
CCB.CMD	is the halfword storage for the DCW.
CCB.RECS	is the halfword used by the binary synchronous driver to contain CCB count equivalent to transparent record size.
CCB.TLXF	is the halfword used to total the length of transfers when using chained buffers.
CCB.BMOD	is the halfword required by drivers that must maintain their present mode; i.e., binary synchronous and future asynchronous drivers.
CCB.ITIM	is the halfword used by the clock.
CCB.DN	is the halfword used by the clock containing a device number correspondng to the CCB (read or write). It is initialized by the file manager at assign time.
CCB.PITC/ CCB.NITC	are the halfwords used by the clock containing previous and next pointers of the forward- and backward-linked time chain. It must be coded as DC H'0',H'0'.
CCB.QBF	is the address of a queued buffer from the list (queued buffer support, lines only).
CCB.QBT	is the address of a queued buffer to the list (queued buffer support, lines only).
CCB.PDCT	is the halfword pad count used by the direct I/O subsystem (DIOS). The relative position of this field in the CCB is fixed.

6.3.4 Drop Control Table (DCT) for Zero-Bit Insertion/Deletion Data Link Control (ZDLC) Communications

The DCT is a system table that is either allocated by the user or sysgened into the DCB. The DCT stores data necessary for controlling ZDLC communications with a specific drop. Included with its data are the relocated addresses (i.e., addresses relocated from task space to system space) of the:

- four circular lists (for the u-task I/O buffers),
- secondary station address (SSA) of the allocated drop, and
- logical filename.ext of the drop.

When directed by the SVC1 extended option bits, these parameters are placed into the DCT from the user's DDT. Figure 6-8 depicts the DCT.

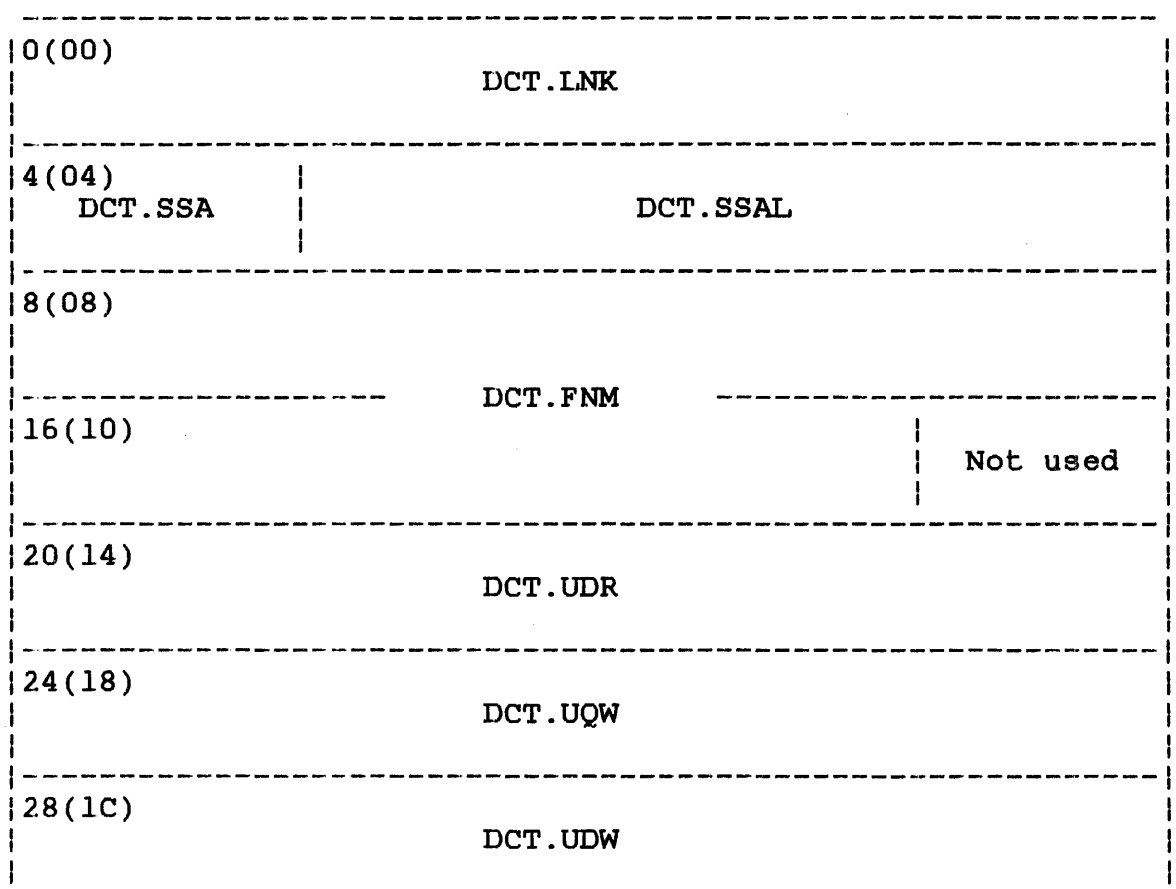


Figure 6-8 DCT (ZDLC) Format

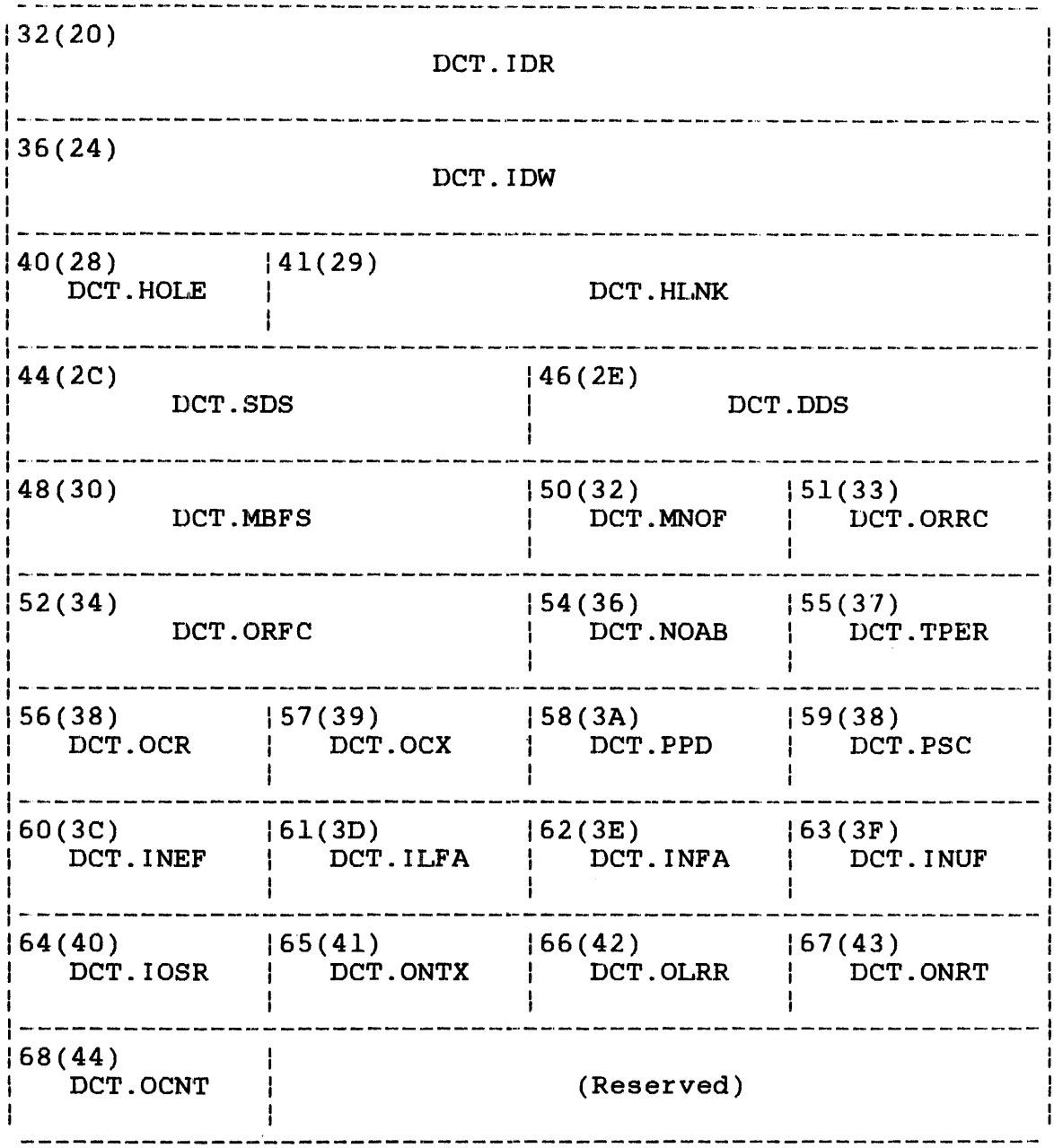


Figure 6-8 DCT (ZDLC) Format (Continued)

Fields:

DCT.LINK — is the fullword pointing to the address of the next DCT on the chain of DCTs.

DCT.SSA — is the first byte for the SSA.

DCT.SSAL — is the 3-byte link to any additional SSA bytes.

DCT.FNM — is the 12-byte field containing the 11-byte filename.ext (logical drop name) for the DCT; the last byte is unused.

DCT.UDR — is the fullword pointing to the user done with read (UDR) circular list (the read-done list) associated with the drop.

DCT.UQW — is the fullword pointing to the user queue for write (UQW) circular list (the write list) associated with the drop.

DCT.UDW — is the fullword pointing to the user done with write (UDW) circular list (the write-done list) associated with the drop.

DCT.IDR — is the fullword header pointing to the internal read-done chain of input frames. It is not yet passed to the UDR circular list.

DCT.IDW — is the fullword header pointing to the internal write-done chain of output frames that has already been transmitted, but is awaiting acknowledgement.

DCT.HOLE — is the 1-byte "frame hole" indicator having a value of X'FF', when applicable. When not used, this byte is reset to zeros. (A hole is a missing I-frame within the input data register (IDR) chain. The frame was selectively rejected by the channel terminal manager (CTM) and, consequently, not entered into the IDR chain.)

DCT.HLNK — is the 3-byte link to the address of the next frame after the hole in the IDR chain. When not used, these bytes are reset to zeros.

DCT.SDS — is the halfword static drop status describing drop criteria to the CTM.

DCT.DDS — is the halfword dynamic drop status reflecting drop activity at the given time.

DCT.MBFS — is the halfword containing the maximum size of a frame that can be transmitted for the drop.

DCT.MNOF — is the byte containing the maximum number of frames on the internal write-done (IDW) chain that can be awaiting acknowledgement.

DCT.ORRC — is the byte containing the rejection reason code for an FRMR frame (i.e., a UN-frame having a FRMR code in the control field (C-field) being output.

DCT.ORFC — is the halfword containing the rejected C-field to be included within the FRMR frame being output.

DCT.NOAB — is the byte specifying the number of address bytes in the SSA.

DCT.TPER — is the byte containing the reason code of the problem-causing trap.

DCT.OCR — is the byte containing the binary index code of the NUMBERED or UNNUMBERED C-field sequence to be output.

DCT.OCX — is the byte into which the command code requested by the above DCT.OCR is stored before the appropriate N- or UN-frame is output.

DCT.PPD — is the byte determining the poll priority of the drop.

DCT.PSC — is a byte used to defer an implied rejection. (An implied rejection occurs when a drop receives an N- or I-frame having the P/F bit set and the N(R) value less than the sequence number, N(S), of the next frame to be transmitted.)

DCT.INEF — is the byte indicating the next expected sequence number, N(S), of an incoming I-frame.

DCT.ILFA — is the byte indicating the sequence number, N(S), of the last input I-frame that was acknowledged.

DCT.INFA — is the byte indicating the sequence number, N(S), of the next expected input I-frame to be acknowledged.

DCT.INUF — is the byte indicating the sequence number, N(S), of the next input I-frame that can be passed from the IDR chain to the UDR list.

DCT.IOSR — is the byte containing the sequence number, N(S), of a selectively rejected input I-frame that is currently outstanding.

DCT.ONTX — is the byte indicating the sequence number, N(S), of the next I-frame to be transmitted.

DCT.OLRR — is the byte containing the sequence number, N(R), of the last RR (receive ready) frame received from another drop.

DCT.ONRT — is the byte containing the sequence number, N(S), of the next output I-frame needing retransmission; this I-frame must be in the IDW chain.

DCT.OCNT — 1-byte counter for I-frames transmitted or received in response to a poll.

To communicate with a drop, a DCT for that drop must be in system space. The DCT gets into system space in one of two ways:

1. The user allocates the DCT with an SVC7 call based on an extended SVC7 parameter block. Issuing the SVC7 call does the following:

   - Allocates a DCT from system space.
   - Enters the logical filename.ext and SSA from the SVC7 parameter block into the DCT.
   - Initializes DCT control fields to specific values needed by the CTM.

2. As a user convenience, static DCTs can be sysgened as permanent tables within the DCB; the user need not allocate the DCTs. At sysgen, the DCB for the ZDLC line can have one or more DCTs built in with each DCT having a defined logical filename.ext and secondary station address. To communicate with a drop defined this way, the user must know the logical filename.ext of the drop.

6.3.5 Drop Definition Table (DDT) for Zero-Bit Insertion/ Deletion Data Link Control (ZDLC) Communications

The DDT is a table the user sets up in task space as an extension to the SVC1 parameter block. It supplies additional parameters needed for ZDLC protocol support, including:

- the logical name of the drop with which ZDLC communications are wanted, and
- the addresses of the four circular lists (read pool, read-done list, write list, and write-done list) needed for I/O.

When the user issues the SVC1 call, DDT parameters are passed to the DCT and the DCB.

As defined for ZDLC protocol, a drop is a logical starting or ending point on a ZDLC communications line. A drop can be a primary station, a secondary station, or a station acting as multiple secondaries. Figure 6-9 depicts the DDT.

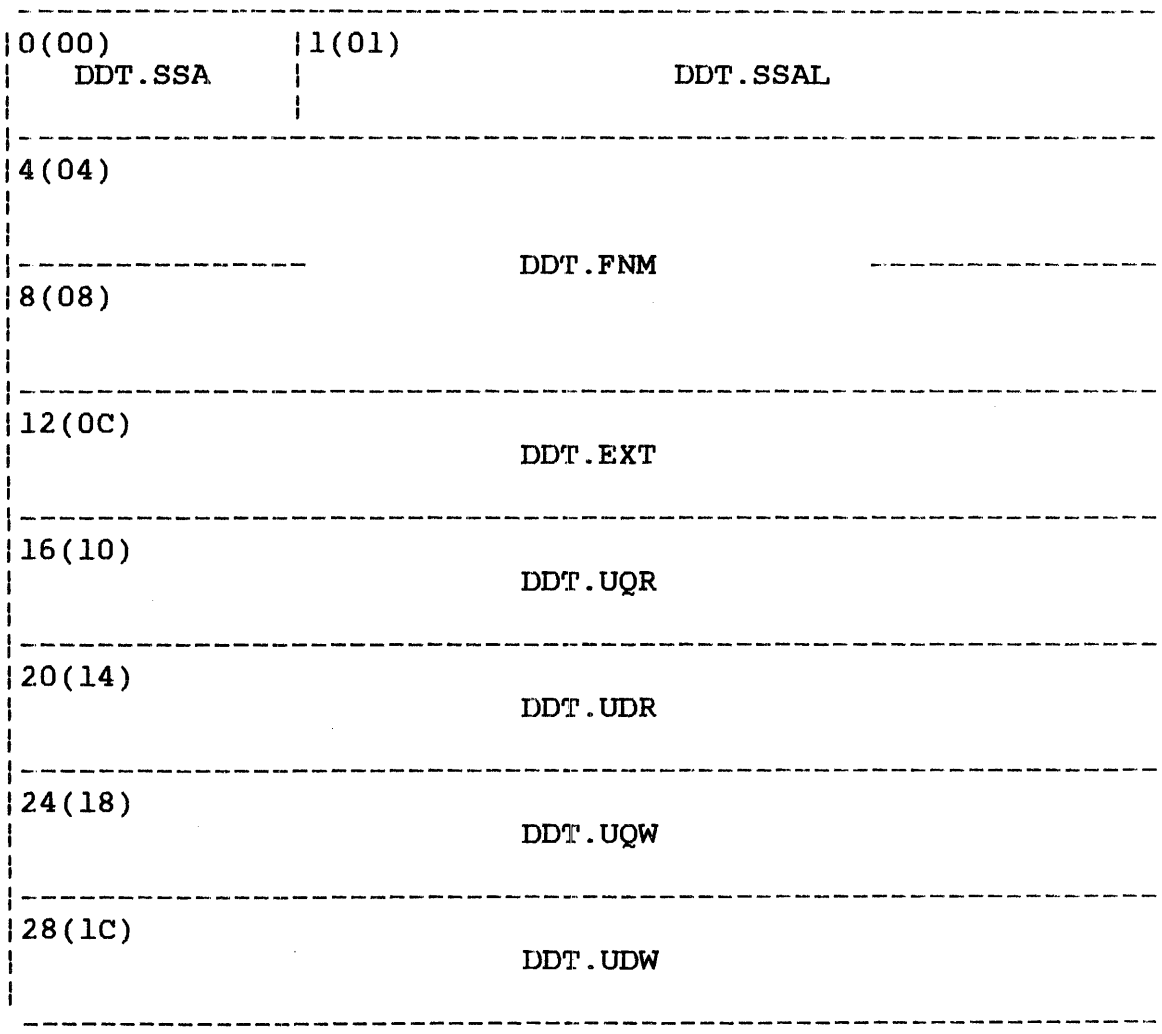


Figure 6-9 DDT (ZDLC) Format

Fields:

DDT.SSA — used when a user changes the secondary station address of a drop. Holds the replacement SSA.

DDT.SSAL — holds pointer to additional SSA bytes when a u-task changes an SSA to have more than one byte.

As seen in Figure 6-9, the first byte of this fullword is an additional SSA byte and the following three bytes point to another SSA byte, if necessary. The last SSA byte has its pointer filled set to zero.

DDT.FNM
DDT.EXT — defines the logical drop name and extension by which the drop is known to the system.

DDT.UQR — is the address of the UQR list (user's read pool).

DDT.UDR — is the address of the UDR list (user's read-done list).

DDT.UQW — is the address of the UQW list (user's write list).

DDT.UDW — is the address of the UDW list (user's write-done list).

6.3.6 Drop Control Table (DCT) for Asynchronous Multidrop Communications

The DCT for asynchronous multidrop communications is a caps system table that is allocated by the user executing the generate macro or command. One DCT per terminal is allocated to specify the drop. Fields in the DCB specific to asynchronous multidrop communications contain the DCT queue control data. The following are included with the DCTs:

- Logical filename/extension of the drop
- Device access queue control data
- Line polling/selection address of the drop

The structure of the DCT is shown in Figure 6-10.

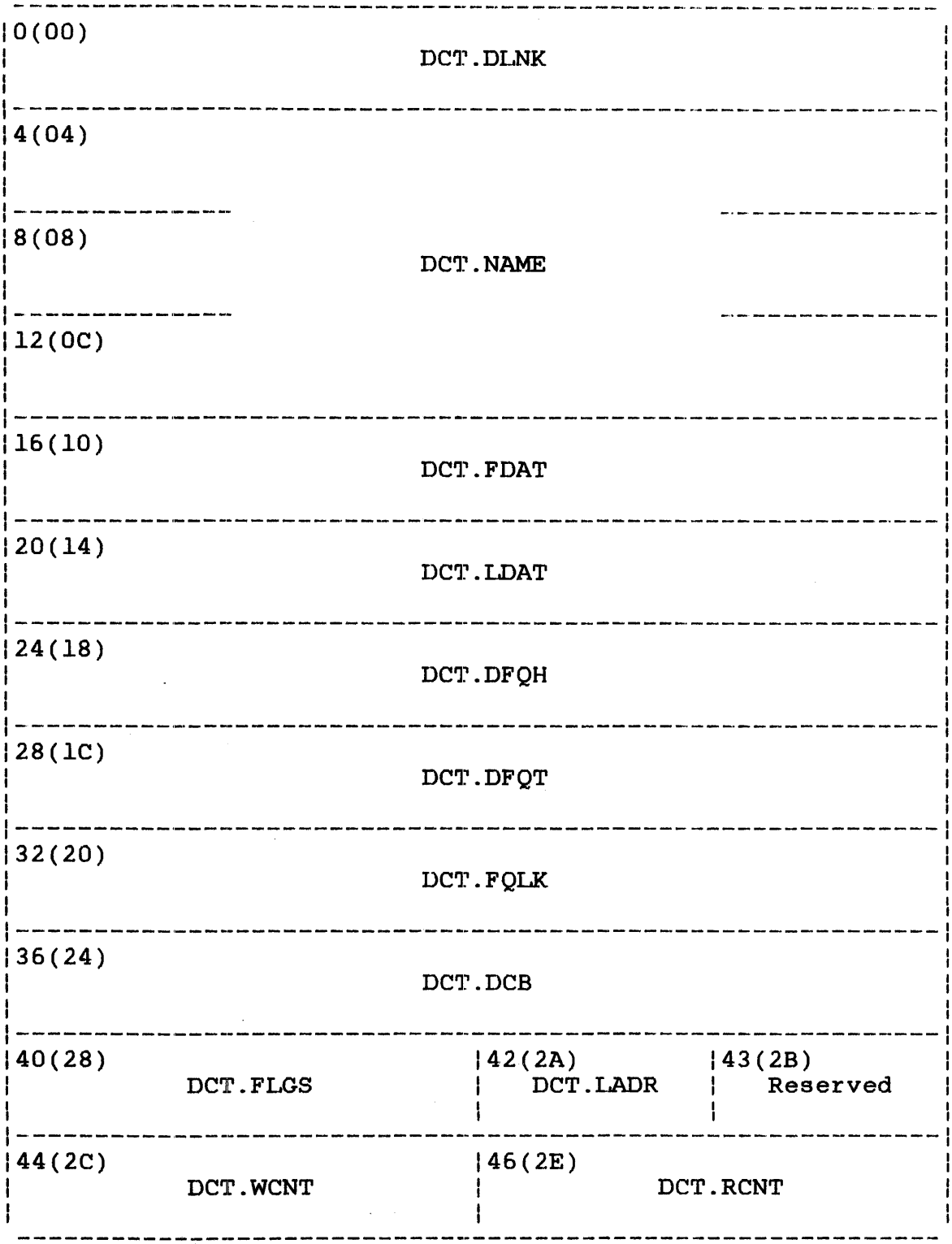


Figure 6-10 DCT (Asynchronous Multidrop) Format

Fields:

DCT.DLNK	is the link to the next DCT (from the DCB).
DCT.NAME	is the file descriptor (fd) for the particular terminal.
DCT.FDAT	is the first DAT used by this DCT.
DCT.LDAT	is the last DAT used by this DCT.
DCT.DFQH	is the first DAT on the function queue chain.
DCT.DFQT	is the last DAT on the function queue chain.
DCT.DQLK	is the link to the next DCT in the function queue chain.
DCT.DCB	is the pointer to the parent DCB.
DCT.FLGS	is a halfword reserved for flags.
DCT.LADR	is the polling/selection line address for the particular terminal.
DCT.WCNT	is the halfword for the write count.
DCT.RCNT	is the halfword for the read count.

6.3.7 Drop Access Table (DAT) for Asynchronous Multidrop Communications

The DAT is a system table that is internally allocated when an lu is assigned to a terminal. The following are included with its data:

- lu
- IOB queue control data
- Timer chain data

The structure of the DAT is shown in Figure 6-11.

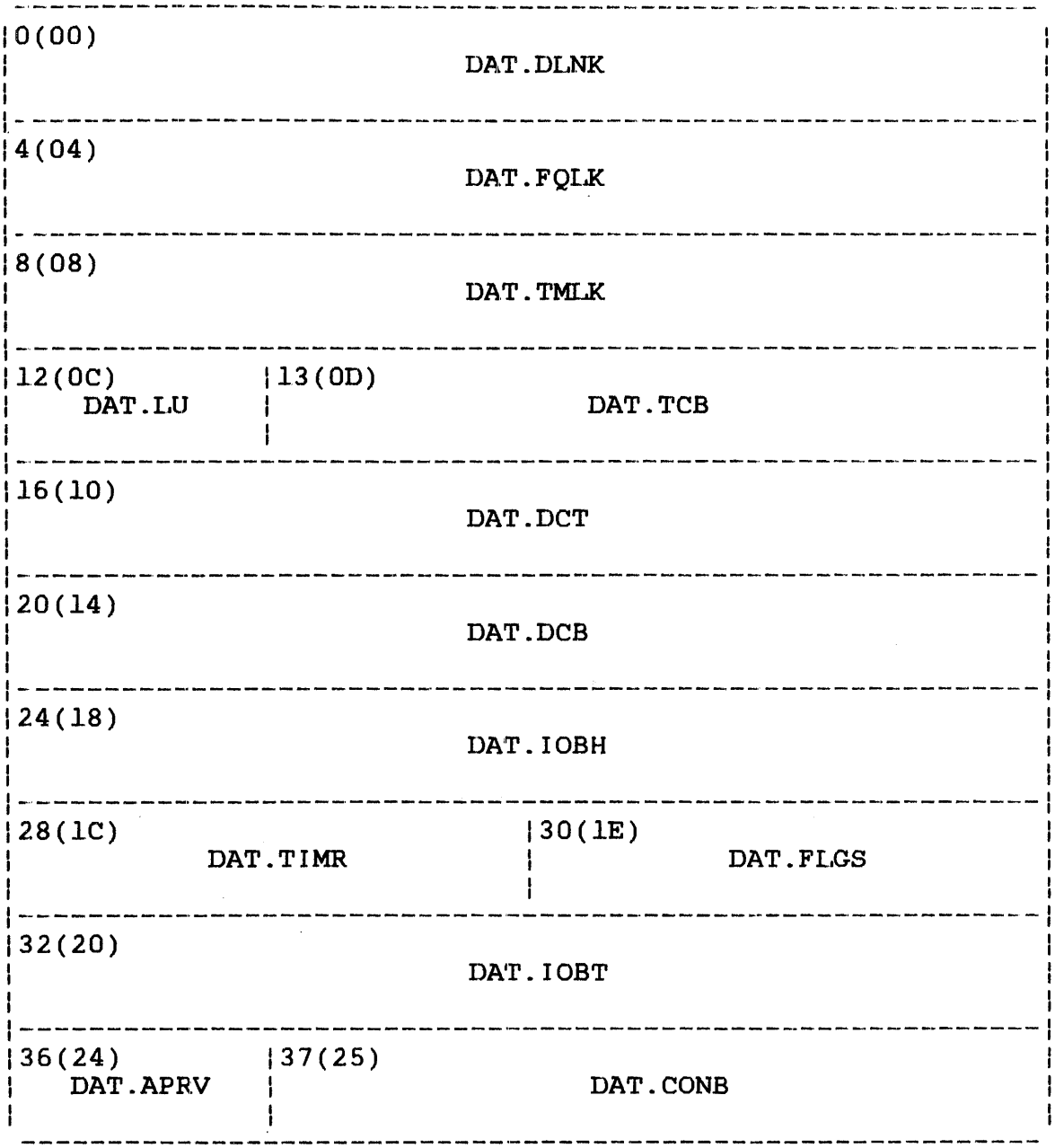


Figure 6-11 DAT (Asynchronous Multidrop) Format

Fields:

DAT.DLNK is the link to the next DAT for the current DCT.

DAT.FQLK is the link to the next DAT in the function queue chain.

DAT.TMLK	is the link for the timer chain.
DAT.LU	is the byte for the lu assigned to the DAT.
DAT.TCB	is the address of the.
DAT.DCT	is the address of the parent DCT of this DAT.
DAT.DCB	is the address of the parent DCB of this DAT.
DAT.IOBH	is the pointer to the first IOB for this DAT.
DAT.TIMR	is the timer value.
DAT.FLGS	is a halfword reserved for flags.
DAT.IOBT	is a pointer to the last IOB for this DAT.
DAT.APRV	is the access privilege byte.
DAT.CONB	is the multi-terminal monitor (MTM) TCB save area.

6.3.8 Input/Output Block (IOB) for Asynchronous Multidrop Communications

The IOB is allocated from system space and chained to the TCB. This occurs both at load time and assignment time. At load time, the number of IOBs built is dependent on the Link option IOBLOCK. If the option is not specified, a default of one IOB is contained in the TCB. At assignment time, additional IOBs are allocated and chained, thereby eliminating the problem of proceed I/O waiting for a free I/O block. The IOB contains I/O information that includes the SVC1 parameter block. The structure of the IOB is shown in Figure 6-12.

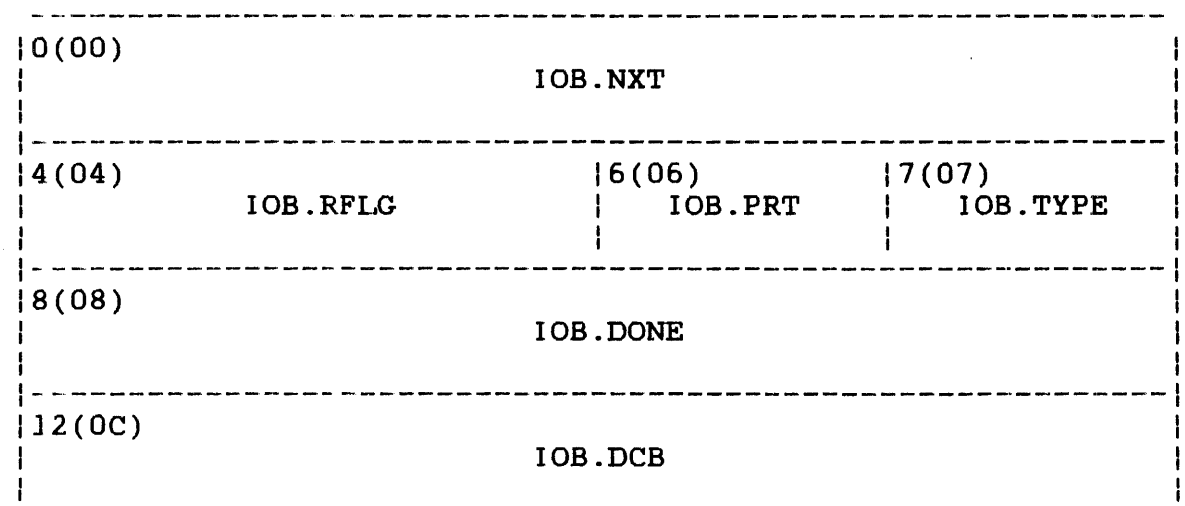


Figure 6-12 IOB Format

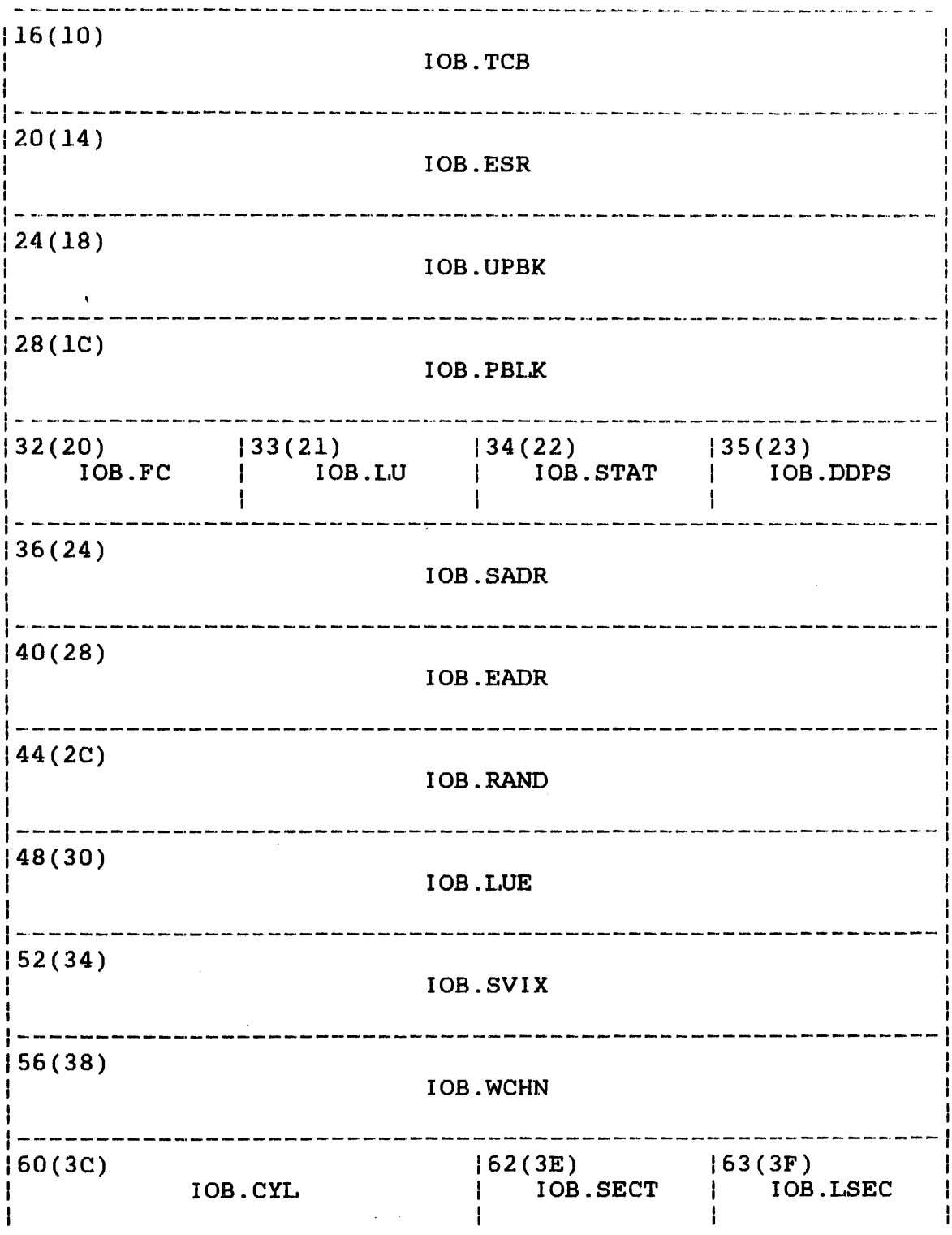


Figure 6-12 IOB Format (Continued)

Fields:

IOB.NXT	is a fullword that holds the forward pointer.
IOB.RFLG	is the request-dependent flag.
IOB.PRT	is the I/O priority.
IOB.TYPE	is the type byte.
IOB.DONE	is the address of the IODONE/SUB executor.
IOB.DCB	is the address of the parent DCB.
IOB.TCB	is the pointer to the caller TCB.
IOB.ESR	is the driver entry fullword.
IOB.UPBK	is the unrelocated parameter block address.
IOB.PBLK	is the relocated parameter block address.
IOB.FC	is the SVC1 function code.
IOB.LU	is the SVC1 I/O lu number.
IOB.STAT	is the device-independent status from the SVC1 parameter block.
IOB.DDPS	is the device-dependent status from the SVC1 parameter block.
IOB.SADR	is the buffer start address.
IOB.EADR	is the buffer ending address.
IOB.RAND	is the positional address of the logical record to be accessed for a data transfer.
IOB.LUE	is the lu entry address.
IOB.SV1X	is the extended SVC1 word.
IOB.WCHN	is the list of tasks waiting for the current I/O to finish.
IOB.CYL	is the requested cylinder *2 for a disk.
IOB.SECT	is the starting relative sector.
IOB.LSEC	is the relative position of the last sector.

6.3.9 Station Description Table (SDT) for 3270 Emulator

The SDT is attached to the DCB and is created from system space, one per control unit, when the GENERATE command is executed. The following are included with its data:

- Control units polling/selection addresses
- Pointers to the DDTs for the devices on the control unit

The structure of the SDT is shown in Figure 6-13.

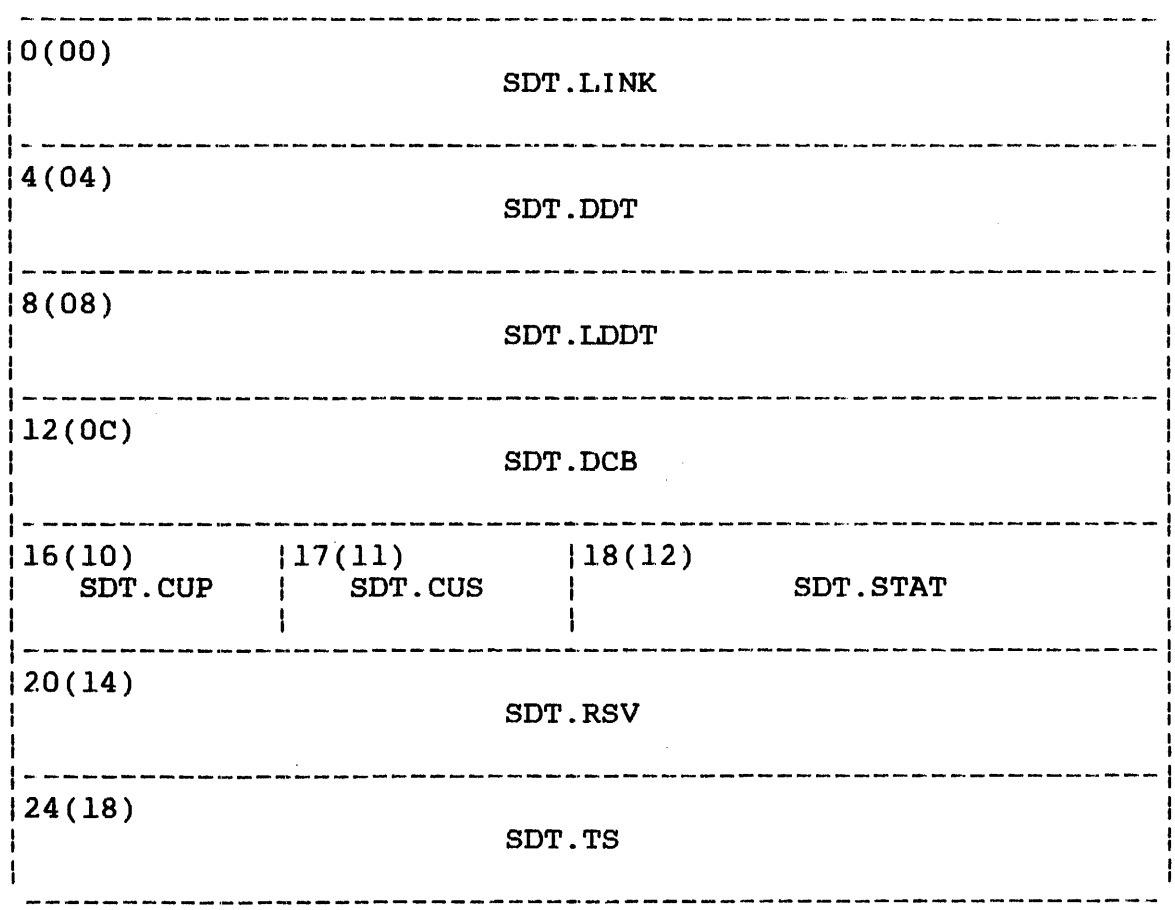


Figure 6-13 SDT Format

Fields:

SDT.LINK	is the link to the next SDT.
SDT.DDT	is the link to the first DDT on the station.
SDT.LDDT	is the link to the last DDT on the station.
SDT.DCB	is the link to the parent DCB.
SDT.CUP	is the station control unit polling address.
SDT.CUS	is the station control unit selection address.
SDT.STAT	is the status of the control unit.
SDT.TS	is the general polling time stamp.

6.3.10 Device Definition Table (DDT) for the 3270 Emulator

The DDT is a system table that is attached to the SDT. DDTs are created at generation time from system space and one DDT is created for each device to be generated. Each DDT is chained to the proper SDT according to which control unit the device is to be attached. The following are included with its data:

- Device address for the virtual terminal for which it is generated
- Pointer to the screen image storage (SIS) buffer

The structure of the DDT for the 3270 emulator is shown in Figure 6-14.

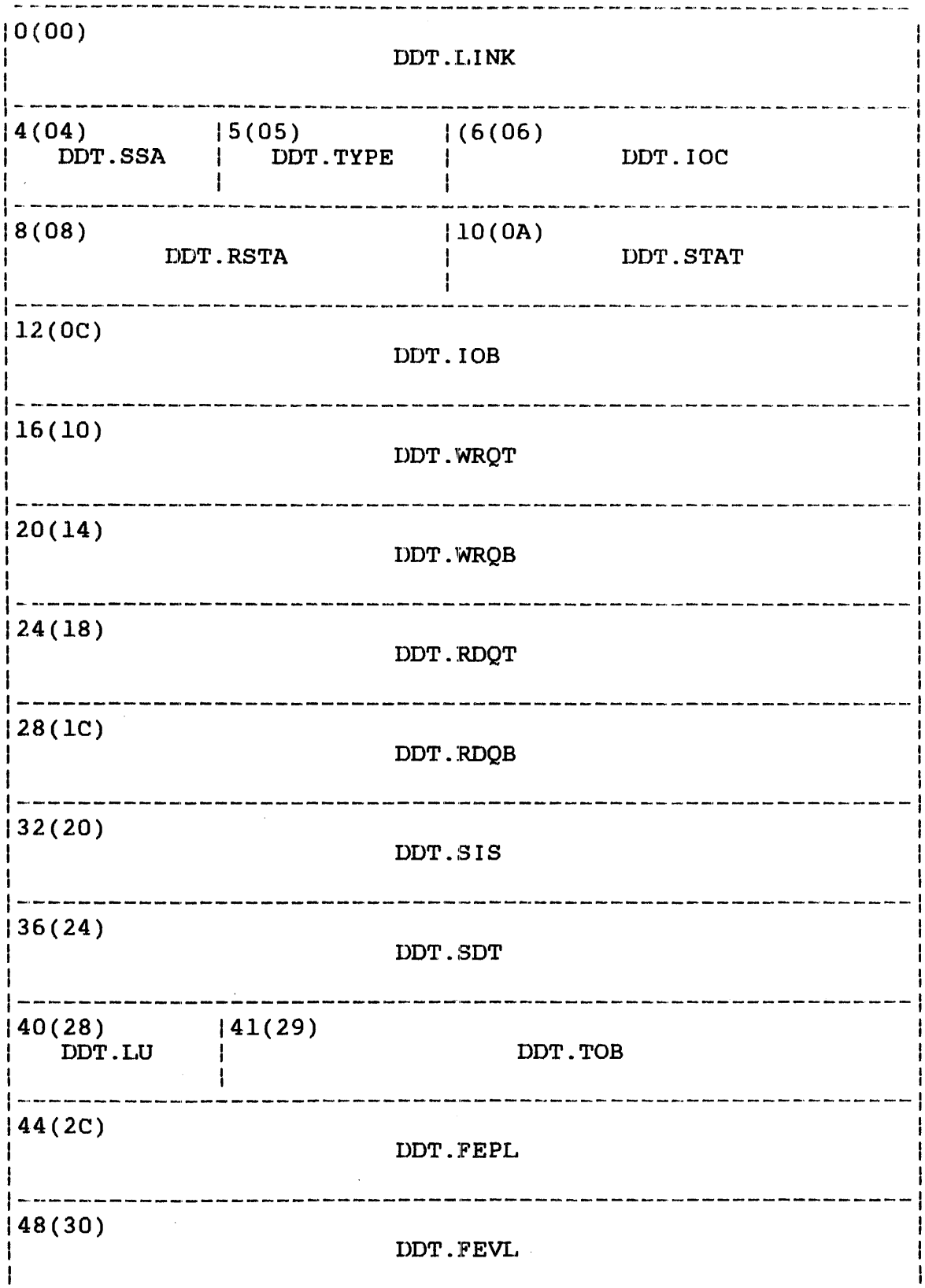


Figure 6-14 DDT (3270 Emulator) Format

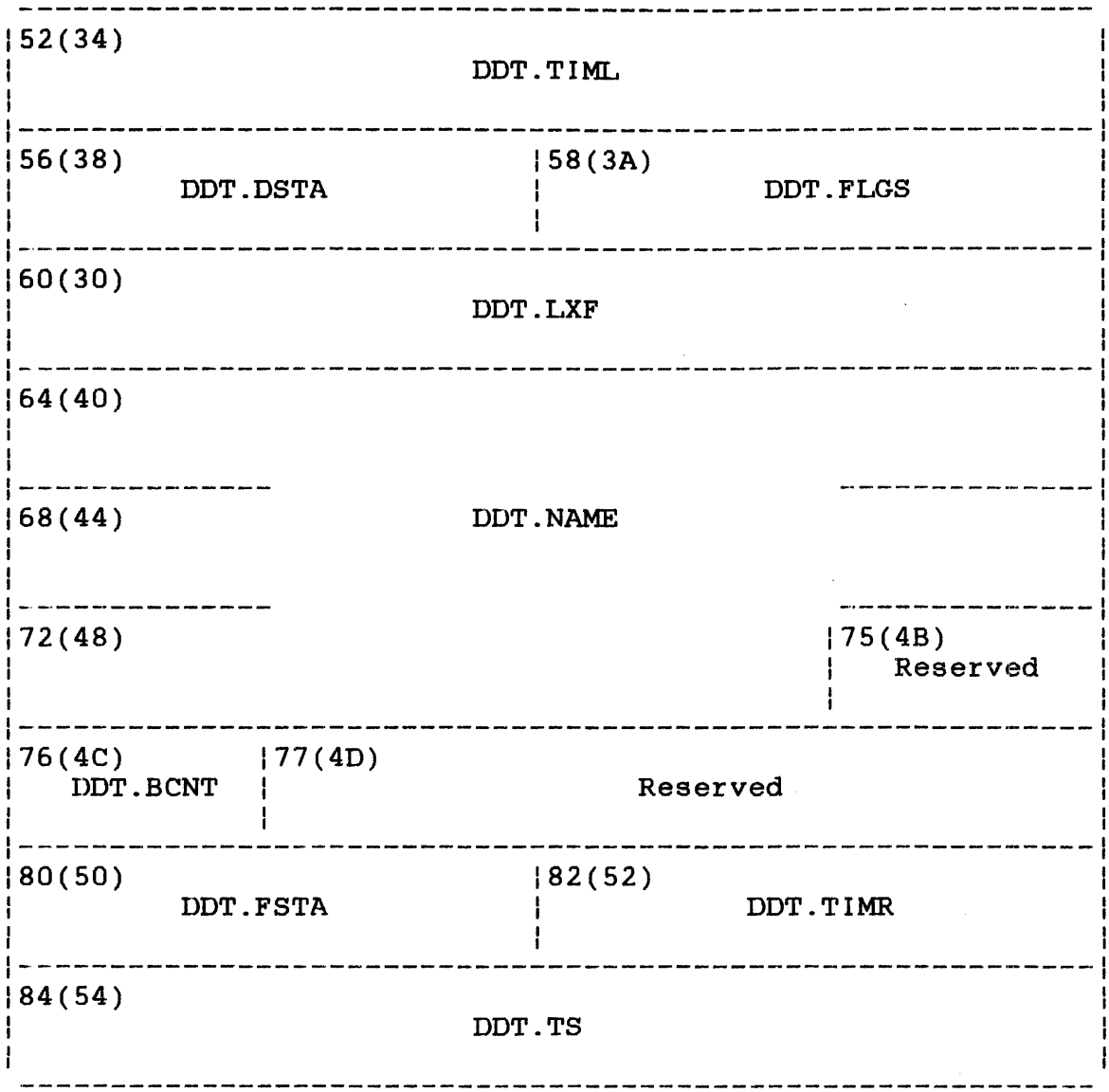


Figure 6-14 DDT (3270 Emulator) Format (Continued)

Fields:

DDT.LINK	is the link to the next DDT.
DDT.SSA	is the secondary station address.
DDT.TYPE	is the 3270 device type.
DDT.IOC	is the I/O request counter.
DDT.RSTA	is the last reported device status.

DDT.STAT	is the 3270 device status.
DDT.IOB	is the address of the retained IOB.
DDT.WRQT	is the pointer to the top of the write ready queue.
DDT.WRQB	is the pointer to the bottom of the write ready queue.
DDT.RDQT	is the pointer to the top of the read done queue.
DDT.RDQB	is the pointer to the bottom of the read done queue.
DDT.SIS	is the address of the screen image storage.
DDT.SDT	is the address of the station descriptor table.
DDT.LU	is the assigned lu.
DDT.TOB	is the address of the connector TOB.
DDT.FEPL	is the format terminal manager (FTM) entry from the protocol terminal manager (PTM) link.
DDT.FEVL	is the FTM entry from the virtual terminal monitor (VTM) link.
DDT.TIML	is the logical I/O timer link.
DDT.DSTA	is the dynamic status halfword.
DDT.FLGS	is a halfword reserved for control flags.
DDT.LXF	is the length of data transfer.
DDT.NAME	is the virtual terminal name.
DDT.BCNT	is the write queue buffer counter.
DDT.FSTA	is the FTM status field.
DDT.TIMR	is the current logical timer value.
DDT.TS	is the general polling time stamp.

6.3.11 Input/Output Handler (IOH)

The IOH is a system structure that handles I/O from the OS/32 SVC1 parameter block. The structure of the IOH is shown in Figure 6-15.

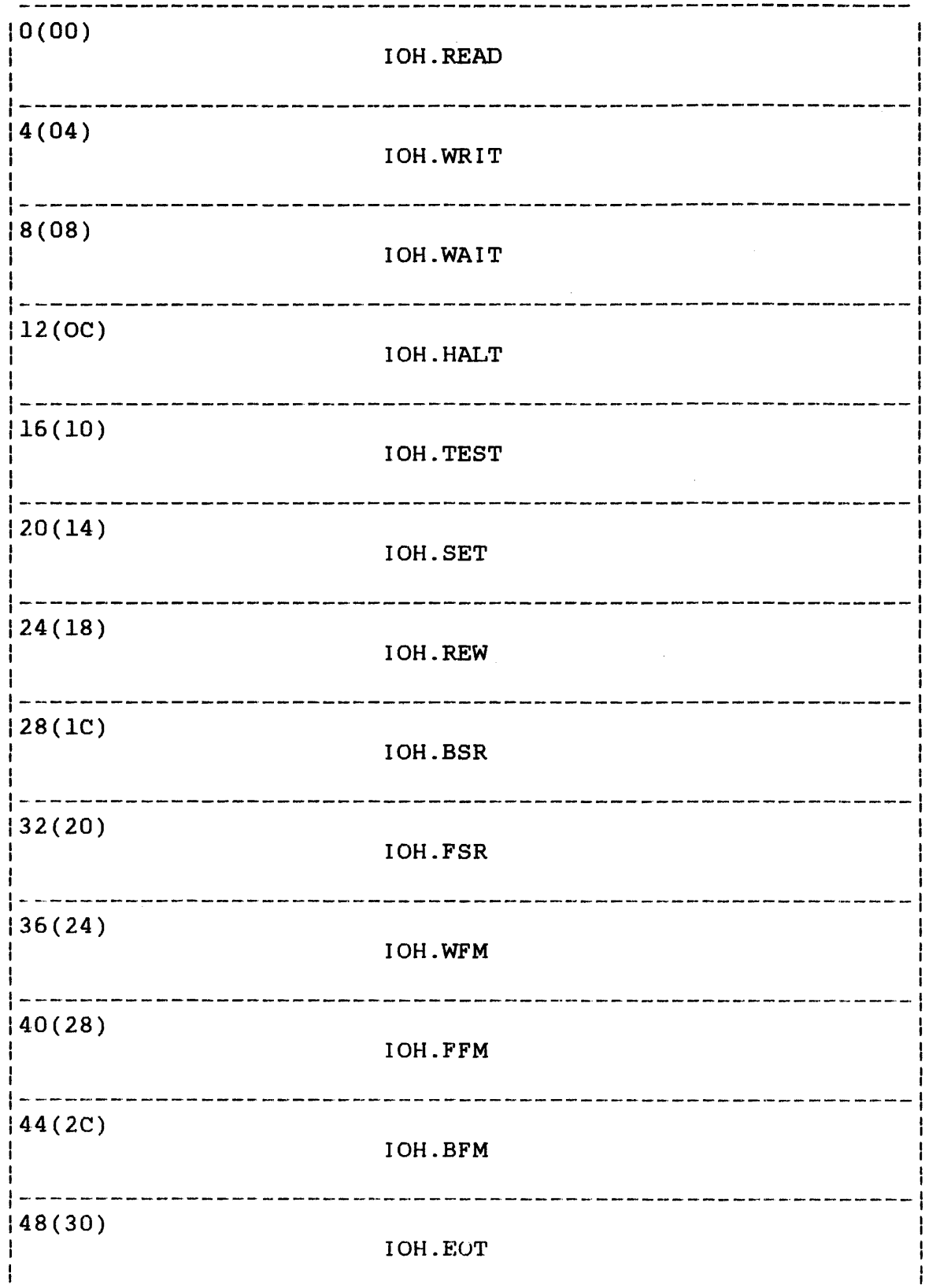


Figure 6-15 IOH Format

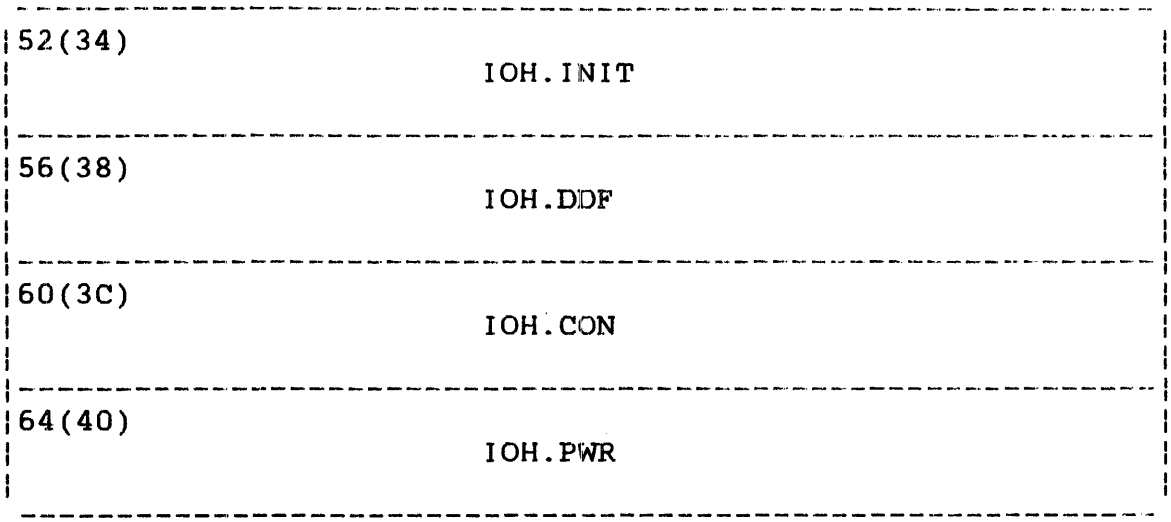


Figure 6-15 IOH Format (Continued)

Fields:

IOH.READ	is the fullword address of the SVC1 read executor.
IOH.WRIT	is the fullword address of the SVC1 write executor.
IOH.WAIT	is the fullword address of the SVC1 wait-only executor.
IOH.HALT	is the fullword address of the SVC1 halt I/O executor.
IOH.TEST	is the fullword address of the SVC1 test I/O completion executor.
IOH.SET	is the fullword address of the SVC1 test and set executor.

IOH.REW — is the fullword address of the SVC1 rewind executor.

IOH.BSR — is the fullword address of the SVC1 backspace record executor.

IOH.FSR — is the fullword address of the SVC1 forward space record executor.

IOH.WFM — is the fullword address of the SVC1 write file mark executor.

IOH.FFM — is the fullword address of the SVC1 forward file mark executor.

IOH.BFM — is the fullword address of the SVC1 backspace file mark executor.

IOH.EOT — is the fullword address of the SVC task termination executor.

IOH.INIT — is the fullword address of the device initialization routine. This routine is entered when the operating system is first started.

IOH.DDF — is the fullword address of the device-dependent function executor.

IOH.CON — is the fullword address of special entry for the operating system console.

IOH.PWR — is the fullword address of the power restore initialization routine. This routine is entered after a power restoration.

6.3.12 File Manager Handler (FMH)

The FMH is a system structure that handles SVC7 functions for data communications. The format of the FMH is shown in Figure 6-16.

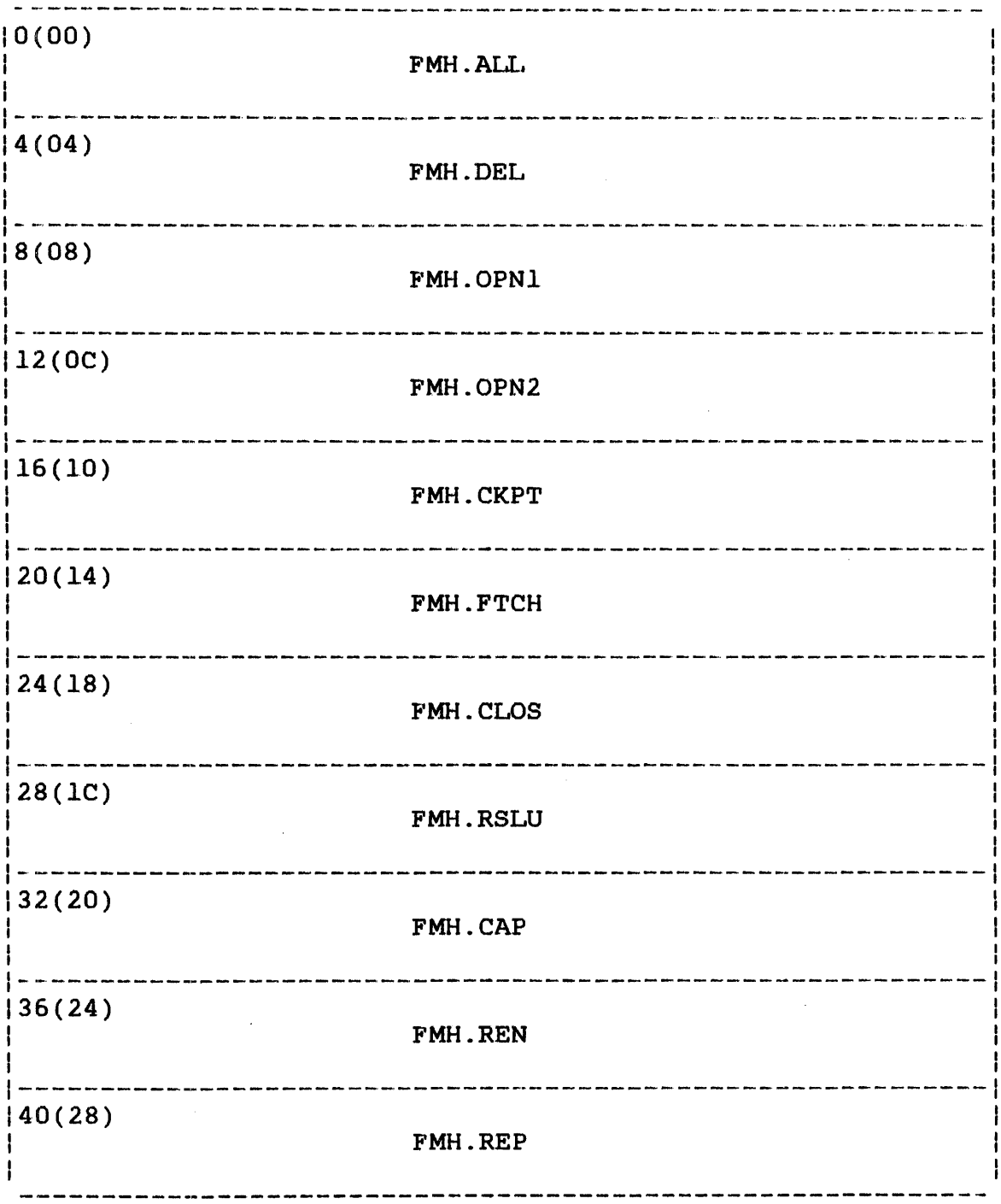


Figure 6-16 FMH Format

Fields:

FMH.ALL	is the fullword address of the SVC7 allocate routine.
FMH.DEL	is the fullword address of the SVC7 delete routine.
FMH.OPN1	is the fullword address of the standard SVC7 assign routine.
FMH.OPN2	is the fullword address of an optional SVC7 assign routine, which can be used by buffered terminal managers for additional processing, after FMH.OPN1 branches to the operating system for common processing.
FMH.CKPT	is the fullword address of the SVC7 checkpoint routine.
FMH.FTCH	is the fullword address of the SVC7 fetch attributes routine.
FMH.CLOS	is the fullword address of the SVC7 close routine.
FMH.RSLU	is the fullword address of the SVC6 lu/TCB exchange lu routine.
FMH.CAP	is the fullword address of the SVC7 change access privileges routine.
FMH.REN	is the fullword address of the SVC7 rename routine.
FMH.REP	is the fullword address of the SVC7 reprotect routine.

6.4 DEVICE CONTROL BLOCK (DCB) POINTERS FOR LINE DRIVER COMMAND INTERPRETATION

Sysgen includes one DCB for each adapter (line) configured in the system. The DCB contains information about that line, its modem, and sometimes, the attached terminals. Line driver command interpretation is controlled by pointers contained in the DCB. These pointers are the:

- Driver initiation routine — DCB.SVCF
- Command table used for decoding commands — DCB.CTA
- Translation tables available for this device — DCB.XLT
- Driver command termination routine — DCB.TERM

• Actual output commands required by the adapter	DCB.DOCR DCB.DOCW DCB.MOCR DCB.DISC
• Programmable adapter information	DCB.AOC
• SVC1 terminal manager entry (if supported)	DCB.INIT
• Adapter device numbers	DCB.RDN DCW.WDN
• Error timeout values	DCB.ITV DCB.OTV
• Next entry into the driver (ESR scheduling)	DCB.ESR
• CCB required for the adapter	DCB.RCCB DCB.WCCB

6.4.1 DCB.TERM Pointer

When a driver schedules its own ESR, it branches to an address pointed to by the driver termination code which usually checks to see if the command is chained. If the command is chained, it continues with command fetching or it terminates the entire SVC15 request by branching to CMTERM, the data communications equivalent of IODONE.

6.4.2 DCB.DOCR, DCB.DOCW, DCB.MOCR and DCB.MOCW Pointers

A single type of adapter, a programmable asynchronous line adapter (PASLA) for example, may require different commands to control it, depending on how it is strapped and how it is to be used (i.e., half-duplex, full-duplex, echoplex or reverse channel). The DCB has space to hold the actual bytes used for the output commands. A single driver can therefore issue the proper output command to several differently strapped adapters.

DCB.DOCR	is the command used to disable an adapter leaving the read mode.
DCB.DOCW	is the command used to disable an adapter leaving the write mode.
DCB.MOCR	is the command used to enable an adapter and place it into read mode, assuming a previously unknown state.
DCB.MOCW	is the command used to enable an adapter and place it into the write mode, assuming a previously unknown state.

6.4.3 DCB.AOC Pointer

This byte is used to pass required information (such as mode, parity, line speed, number of stop bits, etc.) to a programmable adapter. It is generally the CMD2 byte. This allows one driver to communicate with various similar terminals operating at different speeds.

6.4.4 DCB.INIT Pointer

Terminals supported for SVC1 access must have a pointer to the beginning of the terminal manager code for the attached terminal. This is provided in the DCB. The label referenced by DCB.INIT is of the form INITxxxx and is declared an EXTRN in the DCB.

6.4.5 DCB.RDN and DCB.WDN Pointers

Communication adapters can have one or two device numbers, again depending on the strapping. A 2-wire (half-duplex) adapter uses the same device number for both reading and writing, while a 4-wire (full-duplex) adapter uses two different numbers. Drivers can remain transparent to this difference by using the numbers supplied in the DCB when needed.

A simplex device has only one of these numbers (the other is zero, signifying no device).

These numbers are set up at assign time by ITFM, which uses DCB.DN and DCB.XDCD, both initialized at sysgen. The real device number should be specified at sysgen time.

6.4.6 DCB.ITV and DCB.OTV Pointers

The time values used to abort an I/O if not completed within the allotted time period are obtained from the DCB. There are two halfwords, one for read and one for write. Thus, similar devices can have time-outs that vary depending on the intended use of the line and the attached device. These time values are in 1-second increments with a resolution of +0, -1 second.

6.5 EVENT SERVICE ROUTINE (ESR) SCHEDULING

Execution of data communications line driver code can overlap execution of certain buffer management routines that execute in ES(NSU) state, except for actual I/O processing. For I/O processing, these routines execute in the IS state. Therefore, ESR scheduling is required because concurrent events might result from overlapped processing.

Examples of concurrent events are:

- 1-second error timer time-out
- 100ms timer time-out
- System console cancellation of a task
- Closing of an lu in the middle of an I/O operation

The coordination needed for such concurrent events makes data communications ESR scheduling different from that of other drivers.

To coordinate the asynchronous occurrence of ESR requests to a driver, all line drivers use a special routine (ITSRABS) for scheduling ESRs. Standard data communications drivers might require several ESR functions (mainly buffer management) to be performed concurrently with IS processing. Other functions, such as cancel, halt I/O, power fail or time-out, can occur whenever the operating system or the command processor requests them.

Since several of these reqests may be outstanding before any or all of them can be completed, there is a mechanism to handle multiple requests for ESR scheduling. This mechanism consists of a series of bits located in the DCB (DCB.ITB). Each bit indicates a standard function to be performed during ESR processing. The bits are set by the subroutine ITSRABS, which schedules an ESR when it adds the address of the leaf to the system queue if no previous bits were set before this one. Thus, only one entry to the system queue is made for many outstanding ESR requests.

The actual ESR code executed as a result of an ATL instruction to the system queue is a routine called ISSEXEC, which scans the bits from left to right and executes the indicated subroutine corresponding to each bit set. The bit numbers and their correpsonding functions are shown in Table 6-3.

The ISSEXEC routine is always set up as the next ESR by the SVC15 executor at connect time. DCB.ESR is set up with the ISSEXEC routine address. Data communications drivers should never call EVMOD unless the routine used can perform exactly as ISSEXEC performs, especially as far as power fail, time-out or halt I/O is concerned; EVMOD modifies the DCB.ESR field.

TABLE 6-3 DATA COMMUNICATIONS SUBROUTINE REQUEST BITS

BIT	FUNCTION	ROUTINE
0	Get second buffer for read on RAW	ITRAWNX
1	Generate a buffer trap for writes	ISSBTW
2	Set up next chained buffer for write CCB	IT.NXBF
3	Perform end buffer processing for write	IT.ENDBF
4	Generate a command trap	ISSCT
5	Generate a buffer trap for reads	ISSBTR
6	Set up next chained buffer for read CCB	IT.NXBF
7	Perform end buffer processing for read	IT.ENDBF
8	Time-out occurred (abort I/O)	IT.STOP
9	Independent time-out pending expiration	A(CCB.TADR)
12	Halt I/O request	IT..STOP
13	Cancel operation (power fail)	IT..STOP
30	Final ESR processing	A(DCB.TERM) or CMEXIT
31	Schedule initiation phase processing within driver/terminal manager	A(DCB.INIT)

6.6 SUPERVISOR CALL 15 (SVC15) STRUCTURE AND FLOW

The elements involved in the flow of a typical SVC15 request are illustrated in Figure 6-12. Refer to this figure while reading the following description of a data communications driver SVC15 request.

The task sets up an SVC15 parameter block and performs an SVC15 instruction. The SVC15 executor uses the lu assignments for the task to find the appropriate DCB. If this DCB is a data communications DCB and was assigned for SVC15 access, the driver initiation routine address is obtained from the DCB.SVCF, and the driver is entered in ES(NSU) state. Line drivers begin interpretation of each DCW command at this time. The fullword at DCB.CTA contains the address of the command table to be used for command interpretation. The command table contains the address of the routines for each of the eight basic categories of commands supported by this driver.

Each routine contains an address table of the specific routine to perform the exact function indicated by the modifier. Commands that require I/O must enter an ISR. The ISRs, executing in IS state, process all device interrupts. They are responsible for placing the adapter and modem into the proper modes required for I/O. Once the hardware adapter indicates I/O can proceed, all data transfer, character translation and special character detection are performed through the auto driver channel of the processor.

The ISRs can schedule ESRs (e.g., the system support routines, the buffer management subroutines or the driver termination routine) by calling the subroutine ITSRABS. This subroutine, by interfacing with the operating system, schedules ISSEXEC as a subroutine of the calling task.

ISSEXEC is responsible for calling the support subroutines requested by the driver ISRs and for handling certain abort situations imposed by the operating system.

Via the DCB.ITB bit (ITB.ESR), ITSRABS and ISSEXEC also schedule the driver termination routine. This routine is responsible for continuing DCW command interpretation. After all DCW commands are executed, the driver termination routine terminates by branching to CMTERM. CMTERM returns the results (e.g., status) of the SVC15 call to the user parameter block, generates a termination trap, if required, and returns control through the operating system.

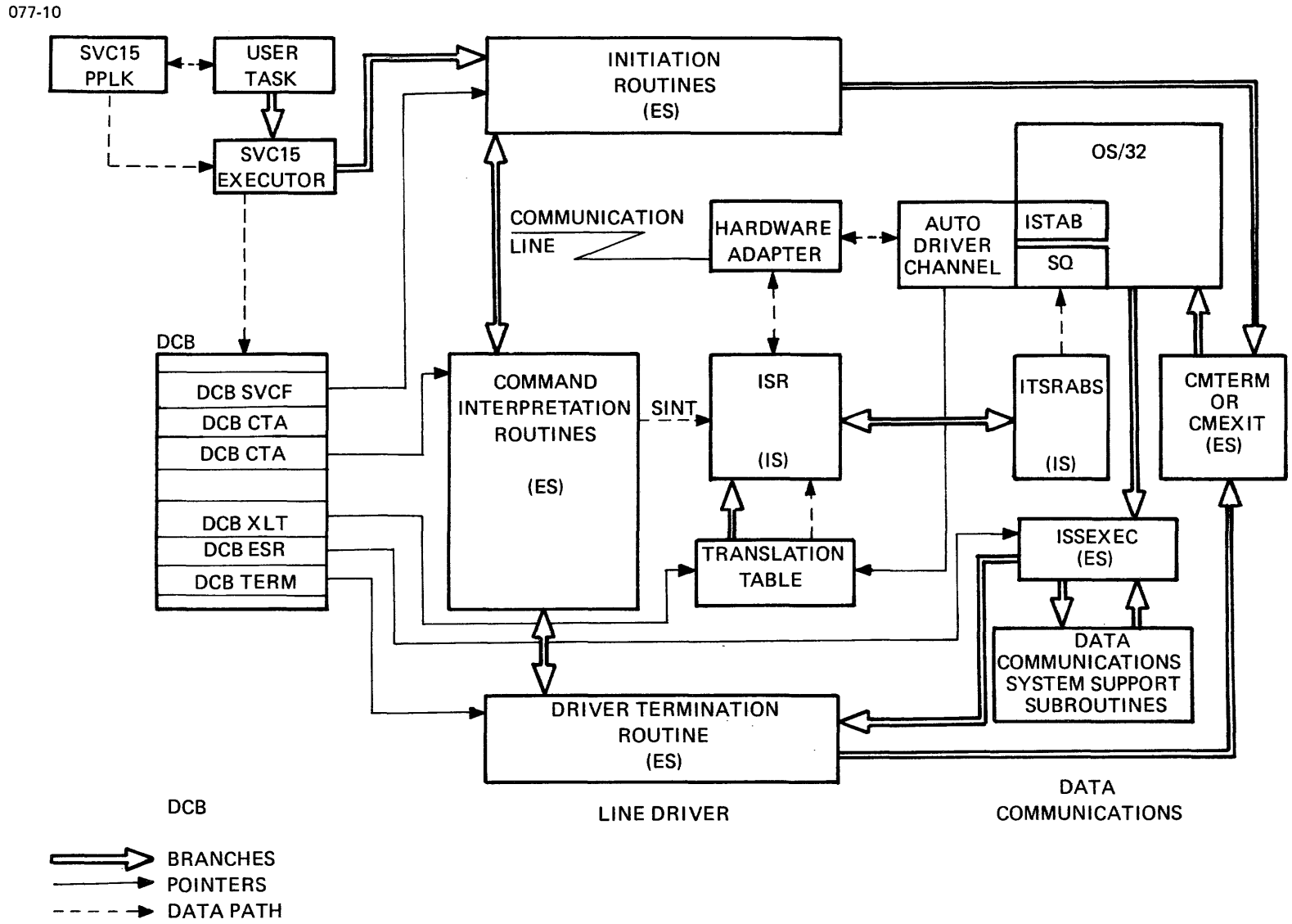


Figure 6-17 SVC15 Line Driver Modules - Data Communications Operating System Interface

6.7 COMMON DATA COMMUNICATIONS SUBROUTINES

This section details the subroutines included in Perkin-Elmer's OS/32 Data Communications Subsystem. Of these subroutines, the first five (SVC15, ITRABS, CMTERM, CMEXIT and ISSEXEC) must be used by all data communications drivers for entering and exiting the drivers.

6.7.1 Supervisor Call 15 (SVC15) Subroutine

This subroutine consists of the SVC15 executor entered on any SVC15 instruction. It is responsible for validating a request and entering the correct driver at the initiation entry.

The SVC15 routine:

1. Sets the user condition code (CC) to zero.
2. Finds the DCB via the lu.
   a. If the DCB is not assigned for SVC15 access, rejects call, CC=2, status=illegal lu.
   b. If the DCB is a null device, returns to task, CC=4, status=0.
3. Connects the DCB by calling EVCON.
   a. If the DCB is already connected, rejects call, CC=1, status unchanged.
   b. Connects and continues.
4. Relocates DCW pointer.

   An out-of-bounds address terminates the call, status=memory fault 1.
5. Stores the following values in the DCB:
   a. Relocated DCW pointer
   b. Relocated data area pointer
   c. Unrelocated parameter block address
   d. Relocated parameter block address
   e. TCB number and TCB address
   f. Function code

6. Sets a busy status in the user parameter block.

7. Sets zero status in the DCB.

8. Zeros out the following:

   a. DCB.NCE (byte) number of commands executed

   b. DCB.AA (halfword) absolute address indicator

   c. DCB.CPTR (fullword) channel program continuation program

   d. DCB.LLR and DCB.LLW (two halfwords) length of last read and write

9. Sets timer (DCB.TOUT) to X'7FFF'.

10. Using the address contained in the first data field, finds the absolute limits of this logical segment and saves these in DCB.BSB and DCB.BSE (if an executive task (e-task), 0 and MTOP are used, respectively).

11. Puts the DCB on time chain.

12. Goes to the driver initiation code by branching to the address located in DCB.SVCF.

6.7.2 ITSRABS Subroutine

This is the special routine necessary for scheduling ESRs from ISRs that must be used by all data communications line drivers.

ITSRABS is called from drivers during IS to schedule an ESR routine. Register 7 is set to the absolute value of the bit number that can be set for the required routines. ITSRABS performs an ATL of the leaf to the system queue only if no other ESR processing bits are set.

```
ENTRY:      BAL E6      IS state
            E4 = CCB address
            E7 = ESR reason bit number absolute (see ISSEXEC)
            Destroys E3, E5, E7
```

6.7.3 CMTERM Subroutine

This is the equivalent of IODONE2 or IODONE. It is called from drivers in either an RS or ES state when the entire SVC15 request is completed, either normally or because of error.

CMTERM is branched to by the last command in an SVC15 request. It is essentially the data communications equivalent of IODONE. Functionally, it:

1. Updates all pertinent fields of the SVC15 parameter block using values from the DCB.

   a. Status

   b. Length of last read and write

   c. Number of commands executed

2. Removes the DCB from time chain.

3. Generates termination trap if requested in function code.

4. Disconnects leaf.

5. Exits from ES state via EVRTE.

ENTRY:	Branch CMTERM RS or ES state UD=DCB address
EXIT:	Does not return

6.7.4 CMEXIT Subroutine

This is the equivalent of TMRSOUT or EVRTE. It is called from drivers in the ES state when they exit control so that other operations can continue. An interrupt or other operation is necessary to get control back to the driver.

THe CMEXIT routine branches to the channel program/terminal manager via A(DCB.CPCR) if a return was requested or via A(DCB.BTRP) if a read buffer trap return was requested. Otherwise, this routine branches to EVRTE.

ENTRY:	Branch CMEXIT in ES state
EXIT:	Does not return

6.7.5 ISSEXEC Subroutine

This special ESR is always set up as the ESR to be scheduled by any add to top of list (ATL) to system queue for all data communications drivers. It is responsible for scanning the DCB.ITB bits and calling the proper routines to handle any bits set, especially the three bits set by the operating system: time-out, power fail and halt I/O. This is the data communications system subroutine executor entered in the ES state whenever an ATL instruction to system queue (SQ) is performed for a device. It is responsible for handling buffer management routines and certain buffer and command trap generation functions. This routine scans the bits in DCB.ITB from left to right and calls the routines corresponding to each set bit until all bits are processed. Then it exits to CMEXIT. The bit numbers of DCB.ITB and their corresponding functions are:

Bit 0 — RAW second buffer

Scheduled by a read ISR if the read is a result of a RAW. Used to get the second buffer of a chained buffer since only one buffer was set up during read initialization. This routine loads UC with the address of the read CCB and calls ITRAWNX.

Bit 1 — Generate buffer trap for write

Scheduled by a write ISR to generate the initial buffer trap when I/O begins. This routine calls IT.TRAP with buffer trap as the reason.

Bit 2 — Next buffer write

Scheduled by write ISRs using chained buffers when each buffer is exhausted. This routine loads UC with the address of the write CCB and calls IT.NXBF.

Bit 3 — End buffer write

Scheduled by write ISRs after terminating write to load register UC with the address of the write CCB and call IT.ENDB.

Bit 4 — Generate command trap

Scheduled by ISRs after completing a write and proceeding to the read ISR in the RAW situation when a command trap is specified in the read command. This routine calls IT.TRAP with command trap as the reason.

Bit 5 Generate buffer trap for read

Scheduled by a read ISR to generate the initial buffer trap when I/O begins. This routine calls IT.TRAP with buffer trap as the reason.

Bit 6 Next buffer read

Scheduled by read ISRs using chained buffers when each buffer is exhausted. This routine loads UC with the address of the read CCB and calls IT.NXBF.

Bit 7 End buffer read

Scheduled by read ISRs after terminating the read. This routine loads UC with the read CCB and calls IT.ENDB.

Bit 8 Time-out

Scheduled by the system clock whenever the error timer (DCB.TOUT) is decremented to zero. This routine calls IT.STOP with time-out status, effectively killing the I/O and the entire SVC15 request.

Bit 9 Independent time-out expiration

Scheduled by the channel terminal manager for polling purposes. Branches to the expiration routine address given in the timer CCB.

Bit 12 Halt I/O

Scheduled by the SVC15 executor whenever a task issues a halt I/O while the task is connected for an I/O to the device. The file manager also schedules the halt I/O routine when closing an lu connected for power up I/O if the device has no other assignments. The halt I/O routine branches to IT.STOP, with a halt I/O status.

Bit 13 Cancel (power fail)

Scheduled by the operating system at power up after a power failure. The operating system also schedules the cancel routine to cancel a task connected to a data communications device. The cancel routine branches to IT.STOP with a power fail status.

Bit 30 Final ESR (termination) processing

Scheduled by line drivers for end of command processing. This routine branches to the driver termination phase via A(DCB.TERM) or to CMEXIT and returns to the terminal manager or to the operating system.

Bit 31 Schedule initiation phase processing

Scheduled by the IOH functional routines (IOHCZBD and IOHMBSC) to invoke terminal manager initiation via A(DCB.INIT) for connected devices.

6.7.6 ITSETREA Subroutine

This is a subroutine called from driver IS to schedule ESR processing. It is similar to ITSRABS except that the reason numbers do not have to be absolute. The subroutine decides whether it is performing read or write and modifies the reason numbers for reads by adding the appropriate constant (4). It is used for common processing for:

- Buffer traps, reason code=01
- Next buffer, reason code=02,
- End buffer requests, reason code=03

```
ENTRY:      BAL E6      IS state
            E4=CCB address
            E7=Reason bit number (as shown above)
            Destroys E3, E5, E7
```

6.7.7 ITXFRISR Subroutine

This is a subroutine that can be called from drivers in IS state after the auto driver channel sends a buffer limit interrupt. This subroutine checks to see if a buffer is available to the CCB as the currently selected buffer.

Auto driver microcode complements the buffer select bit before generating a buffer limit interrupt. If no buffer is available, it means all possible buffers have been exhausted and the routine returns to the caller. If a buffer is available, a next buffer routine is scheduled by calling ITSETREA with E7 specifying next buffer request and the routine exits by an LPSWR E0.

ENTRY: BAL E5 IS state
E4=CCB address

EXIT: Destroys E3, E6, E7

1. Buffer is available; exit by an LPSWR E0.

2. No buffer available.

   a. Used last buffer; return to caller.

   b. RAW next buffer was scheduled but did not execute. Abort I/O with buffer overrun 2 status.

   c. Buffer in CCB but the count is positive; i.e., already used. Abort I/O with buffer overrun 2 status.

   d. Buffer in CCB is flagged as busy.

   e. Buffer in CCB is flagged as done. Abort I/O with buffer overrun 1 status.

6.7.8 ITISSTOP Subroutine

This routine can be called from IS code to terminate I/O because of errors. E7 is loaded wtih the status for the error. ITISSTOP disables both read and write sides of the adapter and clears the interrupt service pointer (ISP) table entries to III. ITISSTOP then ORs the new status to the accumulated status, schedules a halt I/O by calling ITSRABS and exits by an LPSWR E0.

ENTRY: Branch ITISSTOP IS state
E4=CCB address
E7=Error status

EXIT: Does not return

6.7.9 IT..STOP Subroutine

This is a common data communications routine used to abort an I/O call because of error conditions. This subroutine:

1. Disables read and write sides of a device using DCB.DOCR with DCB.RDN and DCB.DOCW with DCB.WDN, respectively.

2. Sets ISPTAB entry to III for read and write device numbers.

3. ORs status halfword contained in register 7 to the latest status in DCB.ISTA. If the encoded portion of DCB status is nonzero, the encoded portion of the status in register 7 is stripped off before it is ORed.

4. Branches to driver ESR obtained from DCB.TERM.

```
ENTRY:    Branch IT..STOP   ES state
          UD(13)=DCB address
          U7=Status (new status)

EXIT:     Either CMEXIT or
          branch via A(DCB.TERM)
```

6.7.10 ITIMLINK Subroutine

This subroutine is used to enter a data communications CCB into the timer chain for the 100ms clock.

At time-out, the clock removes CCB from the timer chain and generates an SINT.

CCB.SUBA and the ISPTAB should be set up prior to any call to ITIMLINK.

```
ENTRY:    BAL U8   ES state
          UC(12)=CCB address

EXIT:     Returns via U8
          Destroys U9, UA, UB, UE
```

6.7.11 ITIMUNLK Subroutine

This subroutine is used to remove a CCB from the timer chain. On return, the condition code (CC) indicates whether or not the CCB was on the chain at the time of the call.

```
CC=G      if CCB was on chain.
CC=0      if CCB was not on chain.

ENTRY:    BAL U8   ES state
          UC=CCB address

EXIT:     Returns to address in register 8
          Destroys U9, UA, UB, UE
```

6.7.12 ITISTOTC Subroutine

This subroutine is used to remove a CCB from a timer chain. It is called from the driver IS state and is similar in action to ITIMUNLK except that it must be called from the ISRs of drivers and it uses different registers. On return:

CC=0	if CCB was not on chain.
CC=G	if CCB was removed from chain.

ENTRY:	BAL ES IS state E4=CCB
EXIT:	Returns via E5 Destroys E6, E7

6.7.13 ITISPOTC Subroutine

This subroutine adds a CCB to the timer chain for the 100ms clock; called from IS state only.

ENTRY:	BAL E5 IS state E4=CCB
EXIT:	Returns via ES Destroys E6, E7

6.7.14 ICMDINT Subroutine

This subroutine fetches and interprets DCW commands by performing the following six functions:

1. Increments the number of commands executed in DCB.NCE by 1.
2. Fetches the next DCW command from the halfword pointed to by DCB.DCW.
3. ANDs the buffer trap and command trap bits of the command with the corresponding bits of the function code and saves in register 3.
4. If the command trap is requested and enabled, generates a command trap by calling IT.TRAP with the command trap as the reason (reason code X'0A').
5. If the command indicates time-out, sets the error timer (DCB.TOUT). Otherwise it stops the timer by setting DCB.TOUT to X'7FFF'. The time value used is obtained from DCB.ITV for READ and PREPARE commands. All other commands use DCB.OTV.

6. Uses the least three bits of the command to index into the command table (located by DCB.CTA), obtains the code address to handle this command, and branches to it. A zero entry in the table indicates that the command is unsupported and results in an illegal command status.

ENTRY: Branch ICMDINT ES state
UD(13)=DCB address

EXIT: Branches to proper command obtained from command table
UD=DCB address
UA=Address of ITGETMOD
U5=Address of ICMDILL (illegal command handler)
U3=Command

Branches to illegal command handler for illegal commands.

Destroys U7, U8, U9, UB, UC, UE

6.7.15 ITGETMOD2 Subroutine

This subroutine is the second entry to ITGETMOD and is used by commands that do not require a CCB (e.g., the NULL command). The routine performs the same functions as ITGETMOD, but it does not store the commmand in the CCB or clear the CCB, since none is provided.

ENTRY: See ITGETMOD

EXIT: See ITGETMOD

6.7.16 ITGETMOD Subroutine

This subroutine is used by driver commands to enter the code for a specific modifier. Each driver command specifies its appropriate modifier table, maximum modifier and CCB. ITGETMOD stores the commands in the CCB, clears the CCB of any buffers and validates that the modifier is less than or equal to the maximum allowable modifier. It is then used to index into the modifier table to fetch the address of the routine responsible for handling this specific request. If the address fetched is negative, a branch on register 5 is performed. This allows commands to perform some common preprocessing before actually branching to the code for a specific modifier.

ENTRY: Branch ITGETMOD ES state
U3=Command
U5=Address for negative modifier return
U6=Number of entries in modifier table
U7=Address of modifier table
UC=CCB address (not required for ITGETMO2)
UD=DCB address

EXIT: If a modifier table entry is positive, branches to that location.

If a modifier table entry is negative, branches to address in U5 entry fetched from the table is in U6.

If a modifier table entry is zero, aborts entire SVC15 request by branching to illegal command handler.

Destroys U8, UA, U6

6.7.17 ITGETDAT Subroutine

This subroutine is called from commands that require a data field. This routine fetches the next data field in order and returns to the caller with the data field in register 7. The pointer to the next data field (DCB.NDA) is incremented by 4, causing successive calls to ITGETDAT fetch sequential data fields. If the data field fetched is negative (data codes of X'80'), ITGETDAT relocates the address contained in the field by calling ADCHK and replaces the data field pointer (DCB.NDA) with the new value. Data fetch then continues from the beginning, effectively causing a transfer out of the normal data field sequence.

ENTRY: BAL UF ES state
UD=DCB address

EXIT: U7=Data field fetched
Destroys U8, U9, UA, UB

6.7.18 ITGETBUF Subroutine

This subroutine is used by all data communications drivers supporting standard buffer management. The routine fetches the data field, calls ITBFREL to relocate the address and sets up the CCB for a data transfer of the proper size using the buffer type specified by the data code. Options contained by U4 are:

- If the get one bit is set (X'4000'), only the first buffer of a possible set of chained buffers is set up in the CCB. Otherwise, two buffers are set up. This option is usually requested in the read of a RAW, where only one buffer must be available. Get one has no effect on direct or indirect buffers.
- If the data code indicates a data area instead of a buffer (data code X'01') as used by READ1 or READ2, the low-order four bits of the option word contain the size of the I/O. Routine ITGETBUF uses ITBFREL to relocate buffers and ADCHK to relocate data areas.

ENTRY:	BAL U2 ES state U4=Options UC=CCB address UD=DCB address
EXIT:	Destroys U5, U6, U7, U8, U9, UA, UB, UE, UF

6.8 SUPERVISOR CALL1 (SVC1) PROCESSING

Processing an SVC1 call for a data communications device differs from processing standard devices in the following ways:

- For data-transfer calls to buffered terminals, terminal manager entry is made in ES/NSU state.
- Upon termination of an SVC1 call (in IODONE), the ISP table entry is not modified for data communications devices.

6.9 ADDITIONAL EXECUTIVE FUNCTIONS

OS/32 executive handling of certain functions is redefined for data communications devices as explained in the following sections.

6.9.1 Cancellation of Input/Output (I/O)

On power restoration and when a task is cancelled, the TIMEOUT routine is called. For data communication devices, time-out calls the HALTITAM routine in the data communications system support module. This routine performs the normal actions of timing out an I/O. However, it also sets a bit in DCB.ITB that can be used by the driver or terminal manager to determine if the situation is a halt I/O (unrecoverable) rather than a recoverable time-out.

6.9.2 Add To Task Queue

The SV9.ATQ routine is modified to accept reason codes 10, 11, 12, 13, 14, 15, 16 and 17 whenever these codes are enabled by bit 23 of the TSW. It is also possible to determine if an unsuccessful return is due to a full, nonexistent or invalid queue; the former sets only the L bit in the CC, the latter two set both C and L bits. To indicate the reason for an error, SV9.ATQ returns with the L bit set in the CC if a queue is full or with both L and C bits set for a nonexistent or invalid queue.

6.9.3 System Initialization

The SYSINIT routine is modified to set up initial values for the data communications timer and DCBs for data communications devices.

6.9.4 Timer Management

An additional timer is maintained for data communications devices. It allows interval timing with a granularity of 100ms and is maintained by a chain of CCBs. Routine ISRLFC is modified to maintain an additional counter, ITM.FREQ, which is initialized to the line frequency divided by five. At every interrupt from the LFC, this counter is decremented by one. When it reaches zero, the counter is reset to one-fifth the line frequency and ISRLFC continues to decrement TM.FREQ by one-fifth the line frequency, rather than one. If the address of the first CCB in the ITAM timer chain (ITAMTIMC) is nonzero, an ESR is scheduled.

Routine TIMESR is modified for the possibility that it may be entered once every 100ms for data communications timer handling. When this happens, it follows the CCB chain and decrements the value of CCB.ITAM. When this value goes to zero, the device is SINTed and ITISTOTC is called to remove the CCB from the chain.

6.10 SUPERVISOR CALL 7 (SVC7) PROCESSING

The standard OS/32 module FMS7 does preliminary processing on all SVC7 function calls. For any data communication-related request, FMS7 branches to individual function routines within the ITFM, the data communication file and memory manager program.

6.10.1 Allocate

The IT.ALLOC routine processes allocate requests for data communication devices. Entry to this routine is in the RS state. To obtain the DCB address, IT.ALLOC calls the DMTLOOK routine. To enter the appropriate allocation routine (i.e., IT.ABSC for allocating a BISYNC LCB, or IT.ADCT for allocating a ZBID DCT), IT.ALLOC uses the main line-type index:

0 = Illegal
1 = BISYNC LCB
2 = ZBID DCT
3 = ZDLC LCB

At entry, each of the allocation routines:

1. Disable the SQS routine.
2. Verifies filename and station-address uniqueness.
3. Obtains and initializes memory space.

On exit, each routine enables the SQS routine.

6.10.2 Delete

The IT.DELET routine processes deletion requests for data communications devices. Entry to this routine is in the RS state. To enter the appropriate deletion routine (i.e., IT.DBSC for deleting a BISYNC LCB, or IT.DDCT for deleting a ZBID DCT), IT.DELET uses the main line-type indexes specified above.

At entry, each of the deletion routines:

1. Disable the SQS routine.
2. Finds a matching LCB or DCT and removes it from the chain.
3. Releases memory space.

On exit, each routine enables the SQS routine.

6.10.3 Assign

The IT.OPEN routine processes assignment requests for data communications devices. Entry to this routine is in the RS state. After checking the extended device code to see if the line is 2-wire or 4-wire, IT.OPEN sets up the device-number entries (for reads and writes) within the DCB.

For an SVC15 assignment, IT.OPEN calls OPEN.SVCF to complete any necessary validity checks, to set up the read count and write count within the DCB and to set up the lu table. For an SVC1 assignment, IT.OPEN calls OPEN.DEV to perform normal validity checking, as with all nonbulk devices, and calls IT.ORJE to perform data communications-related functions.

To enter the appropriate assignment routine (i.e., OPEN.RJE for a BISYNC LCB, or OPD.NRJE for all other devices), IT.OPEN uses the main line-type indexes. If IT.OPEN branches to OPEN.RJE, this allocation routine calls LCBLOOK to:

1. Find the LCB address.
2. Ensure no previous assignment to the found LCB.
3. Set up the read count and write count within the DCB/LCB.
4. Set up the lu table to include the LCB address.

At exit, either assignment routine returns control to the SVC7 mainline coding.

6.10.4 Close

The IT.CLOSE routine processes closing requests for devices assigned SVC15 access. The CLOS.RST routine processes closing requests for nonbuffered devices assigned SVC1 access. For SVC1 buffered devices using main line-type indexes, IT.CBSC closes BISYNC devices, and IT.CDCT closes ZBID devices.

6.10.5 Checkpoint

The IT.CHKPT routine processes checkpointing requests for data communications devices. For nonbuffered devices and for devices assigned SVC15 access, IT.CHKPT waits for ongoing I/O to end before returning control to the operating system. For SVC1 buffered device using main line-type indexes, IT.KBSC checkpoints BISYNC devices and IT.KDCT checkpoint ZBID devices. Both of these routines:

1. Wait for ongoing I/O to end.
2. Call HALTITAM.
3. Flush the various buffer management queues.
4. Exit to the operating system.

6.10.6 Fetch Attributes

The IT.FETCH routine fetches attributes for data communications devices. For nonbuffered devices, IT.FETCH returns only the device mnemonic to the user's parameter block. For buffered devices, IT.FETCH extracts the block size, fd and device code and returns them to the user.

6.10.7 Change Access Privileges

For data communications devices assigned SVC15 access, read-only or write-only privileges cannot be granted. For buffered devices assigned SVC1 access, the address of the CB must be obtained from the LCT.

6.10.8 Rename

The REN.DCB routine changes the DMT entry for the device. Also, this routine changes LCB.NAME and LCB.EXT when it finds a buffered data communications terminal.

6.10.9 Reprotect

As determined by SVC7 parameters passed by the user, the REP.DCB routine changes the read keys and the write keys. Also, if an ITAM device is buffered, REP.DCB must get the DCB address from the LCB.

CHAPTER 7
HOW TO WRITE AND USE A TERMINAL MANAGER

7.1 INTRODUCTION

In data communications, a large number of special-purpose line protocols and data format variations within standard protocols are possible. Consequently, it may be necessary to provide a new terminal manager that satisfies a nonstandard protocol or an interface not currently supported by Perkin-Elmer. The purpose of this chapter is to assist the design analyst and system programmer with the creation of a new terminal manager. It is not oriented toward the modification of any specific existing terminal manager and should be considered a general guideline to be used along with other pertinent Perkin-Elmer publications and protocol specifications. Where possible, examples of functional requirements are provided with reference to existing terminal managers.

7.2 TERMINAL MANAGER MODIFICATION

Terminal manager modification might involve preparation of a newly designed terminal manager to add new features to channel program line control using existing format routines. It could also add new format capabilities using existing channel programs or could remove existing nonrequired features from a standard terminal manager.

Existing terminal manager utility routines and tables designed to support general format and protocol control functions, should be used whenever possible as they can be easily applied to new terminal manager applications. Determine if the terminal manager needs to be modified. For example, if only new line control procedures are required, using a standard supervisor call 15 (SVC15) driver request from an application program may be more appropriate. If only new format procedures are required, adding format subroutines to an application program and obtaining line access via SVC1 image reads and writes might be sufficient.

7.3 BACKGROUND INFORMATION

Terminal manager modification might require some background information depending on how much is to be modified and the program areas that will be affected by the changes.

To remove trailing blank suppression from the existing teletype/video display unit (TTY/VDU) terminal manager, only cursory knowledge of the operating system and data communications is required and the TTY/VDU terminal manager program description might suffice as an information source. To design a totally new buffered terminal manager, extensive background information concerning the operating system and data communications may be required.

- An understanding of the internal structural design, module interface and table structure of the operating system Executive and File Manager is recommended if any interface to the operating system is to be modified or created. While such knowledge of the operating system is highly recommended, existing data communications/operating system interface subroutines should be sufficient for most applications.

- Design criteria, internal structure and module interfaces are required for formatting or channel program modification using common data communications subroutines.

- Design criteria and detailed descriptions of the SVC15 line driver supporting your terminal manager are required for any terminal manager modification involving interface with the line driver or format routine modification to change the control character sequence passed to the line driver.

- General-purpose driver design criteria should be obtained prior to terminal manager modification.

- Detailed specifications concerning the protocol to be supported are required for all terminal manager modifications.

7.4 TERMINAL MANAGER STRUCTURE

While the specific actions to support terminal manager protocol control and data formatting vary for different applications, the high-level functions performed by all terminal managers are similar. Three terminal manager functional flowcharts are provided in Figures 7-1, 7-2 and 7-3. While not indicating every piece of code a terminal manager can contain, these flowcharts provide a guideline to general terminal manager design.

7.4.1 Nonbuffered Terminal Manager

Based on the design of the existing TTY/VDU terminal manager, Figure 7-1 depicts terminal manager requirements for using standard SVC1 task connection control and for performing input/output (I/O) into and out of the task data buffer. This design is similar to a standard OS/32 driver except that it uses SVC15 routines for actual line I/O.

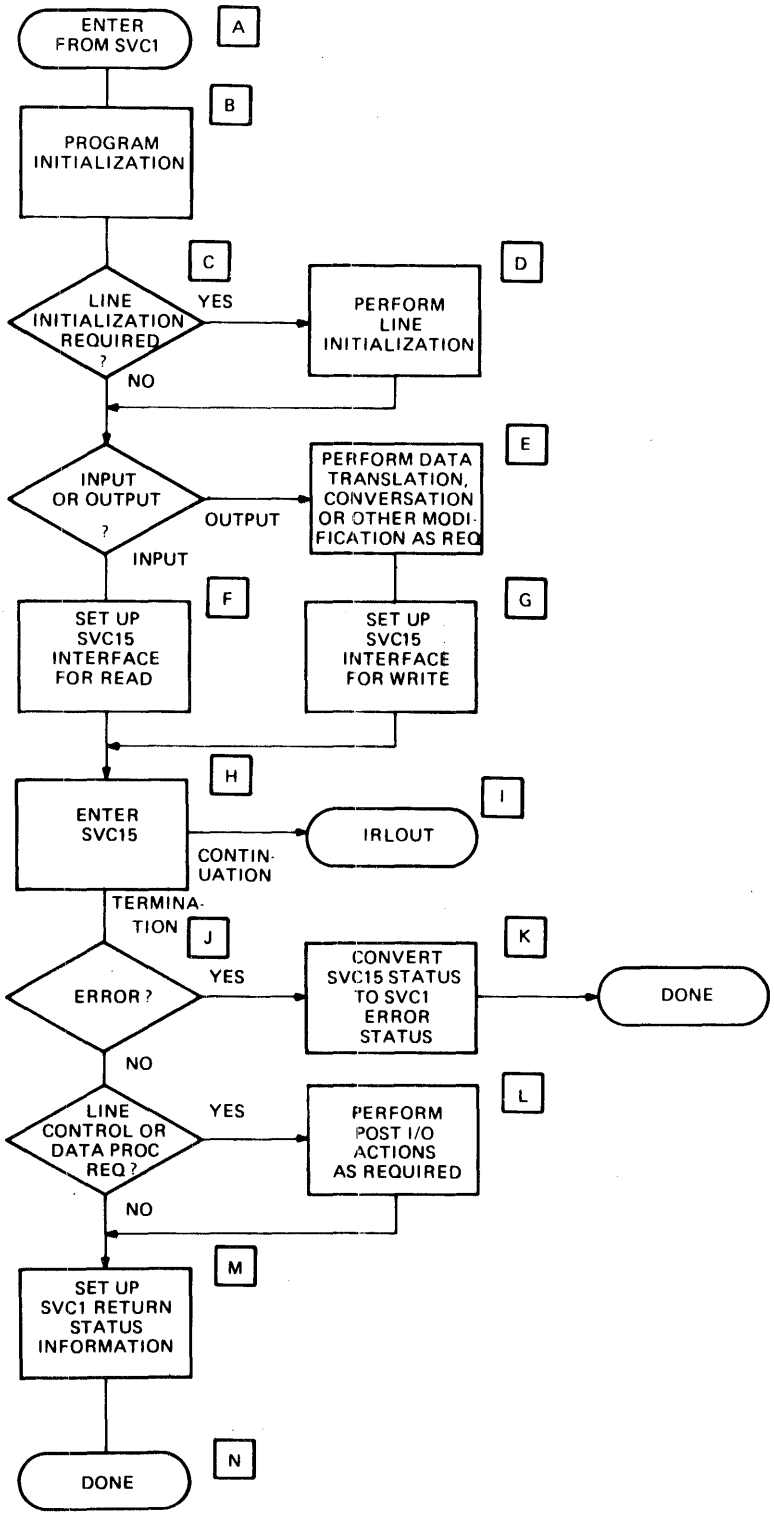


Figure 7-1 Nonbuffered Terminal Manager

7.4.2 Buffered Terminal Manager (Input)

Based on the design of the existing binary synchronous terminal manager, Figure 7-2 depicts the requirements of a terminal manager to control input through single or dual internal buffers, attempt to read ahead of the user task (u-task) and perform its own task connection control.

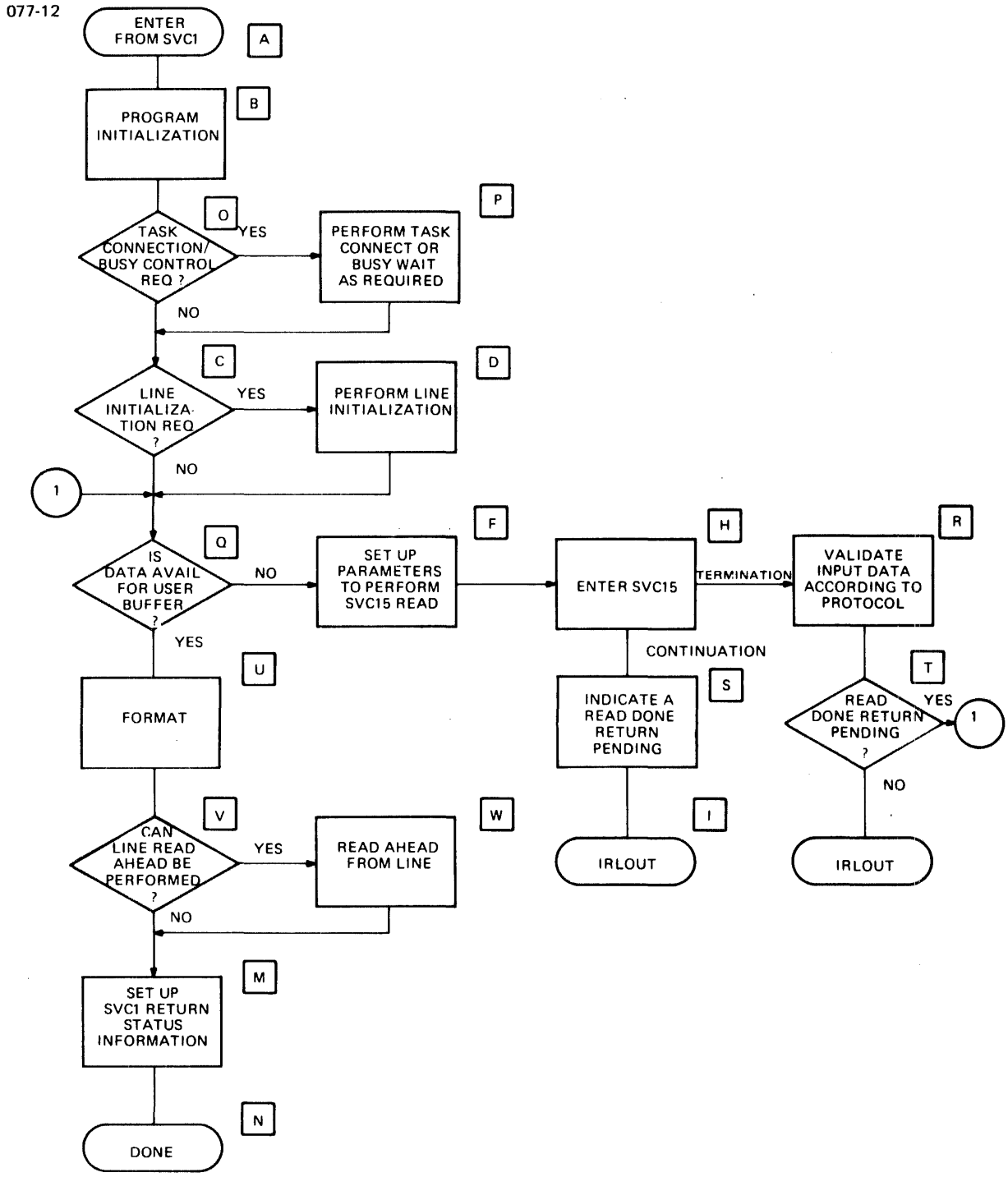


Figure 7-2 Buffered Terminal Manager (Input)

7.4.3 Buffered Terminal Manager (Output)

Based on the design of the existing binary synchronous terminal manager, Figure 7-3 depicts the requirements of a terminal manager that controls output through single or dual internal buffers, is behind the u-task in actual writes due to data buffering and performs its own task connection control.

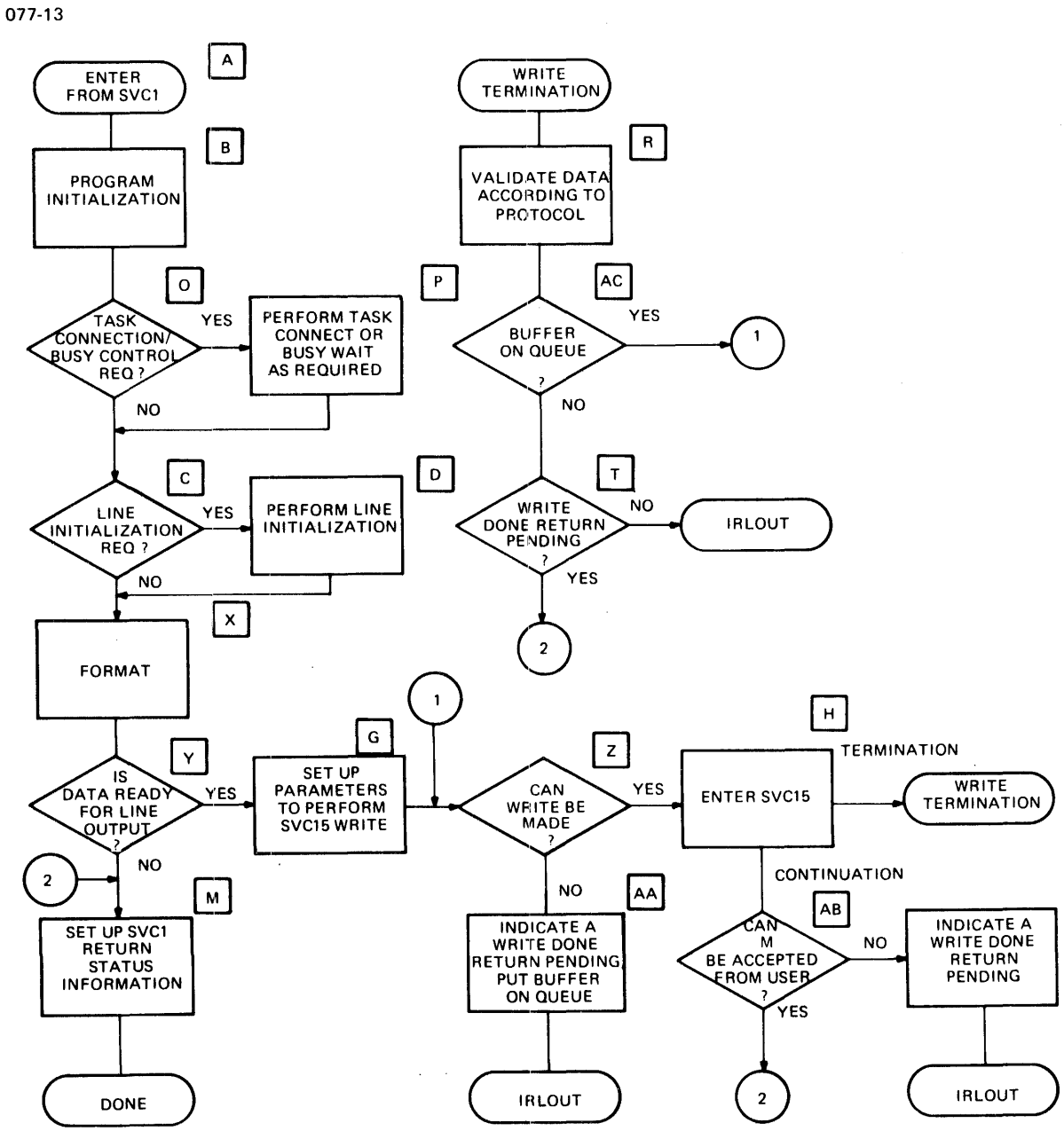


Figure 7-3 Buffered Terminal Manager (Output)

7.5 TERMINAL MANAGER FUNCTIONS

This section describes the individual blocks within the terminal manager flowcharts. Functions performed in more than one flowchart are described only once. These block descriptions should be read as an addendum to Figures 7-1, 7-2 and 7-3.

- BLOCK A

  All terminal managers are initially entered from SVC1 handlers via the I/O handler (IOH) list. For entrance to nonbuffered terminal managers, the device control block (DCB) address is in register U13. For entrance to buffered terminal managers, the line control block (LCB) address is in register U13.

- BLOCK B

  Basic program initialization procedures include interpreting data communications extended options, setting default extended options as required, determining and setting up the register environment, etc. Routine PCINT of the binary synchronous terminal manager can perform these initialization functions.

- BLOCK C

  If a communications line was not readied to accord with a protocol, certain special-purpose actions might be required. Within existing terminal managers, the busy bit (LNS.BSYB) in the line status description word, DCB.LNST, determines if the line was previously readied. The control actions required can be determined from the extended device code, DCB.XDCD. If line initialization is not required for a particular application, exclude this area.

- BLOCK D

  Line initialization is the first of a sequence of functions to ready a communications line for data transfer. Depending on individual requirements, it can involve dial-in/out, polling or terminal selection, line bid transmission/reception or line password or security check. Priority control of master/slave stations, prevention of line bid clashes and similar error control can also be included.

- BLOCK E

  This is a simple format subroutine. For example, within the existing TTY/VDU terminal manager, a carriage return (CR) character terminates an output data buffer and all trailing blanks are truncated. Similar format control routines can be included at this same functional location. A nonbuffered terminal manager should not modify data within a user buffer, as the same buffer could be used simultaneously by the u-task or another driver or terminal manager.

- BLOCK F

  Preparation for an SVC15 read includes setting time-out values and building SVC15 command and data chains. It is convenient to define standard command chains within the terminal manager and build data chains depending on individual circumstances within the DCB.

- BLOCK G

  Preparation for an SVC15 write is similar to a read preparation, described in Block F. Within either of these areas, it might be necessary to cancel any outstanding timers (SVC15 wait requests) required by the individual protocol.

- BLOCK H

  Actual entry into the line driver and final DCB initialization for SVC15 is best achieved by a common subroutine. The terminal manager/line driver interface is currently provided by a branch to the address found within the DCB at DCB.SVCF with the DCB address contained in register UD. Dual return paths are provided to a continuation address found in DCB.CPCR and to a final I/O termination address found in DCB.CPTR. The existing SVC15 interface subroutine within the binary synchronous terminal manager (SVC15GO) saves the command chain address for possible retries, sets an I/O active bit within the line status word, clears SVC15 status words and puts the DCB on the common OS/32 timer chain.

- BLOCK I

  The common exit subroutine IRLOUT is a way to terminate driver action pending the final satisfaction of the user request. The exit is to EVRTE. If the application program performed proceed I/O, it can regain control at this point.

- BLOCK J

  A common subroutine can be used to check for possible errors following all communications line I/Os. This might include a line/hardware error or an invalid data response depending on protocol requirements. It is necessary to differentiate between errors the terminal manager might retry and unrecoverable errors. As this is a common point following all SVC15 line interfaces, it is a possible point at which to include an OS/32 journal entry. The binary synchronous ERRORSET subroutine can be used to perform these basic functions.

- BLOCK K

  When an error return to the user program must be made, convert the SVC15 status or protocol failure to a standard SVC1 error code by using an SVC1 status table indexed by the SVC15 status return.

- BLOCK L

  Within even basic nonbuffered terminal managers, certain protocol-dependent actions might be required following I/O completion. The TTY/VDU terminal manager, for example, determines if a line delete character was received on a read and transmits a line feed (LF)/CR/CR sequence on writes.

- BLOCK M

  Before a return to the user program, certain common status areas must be set for the SVC1 parameter block. These include device-independent status, DCB.STAT, driver-dependent status, DCB.DDPS, if applicable, plus length of last transfer, DCB.LLXF.

- BLOCK N

  The exit subroutine DONE is a way to terminate driver action following completion of a user request. The exit may be to IODONE or to a specialized routine for post-I/O processing and task disconnection, as required. Normally, the application program performing SVC1 wait I/O regains control through this exit point.

- BLOCK O

  Within buffered terminal managers, it might be advisable to perform task connection within the terminal manager instead of having SVC1 perform it. This allows a task to be connected to a line at the beginning of a transmission sequence and remain connected to it throughout the entire transmission. A common application of this procedure allows a task to be connected for a read with the line initialization sequence (see Block D), and later suspend the read before it starts in favor of a write request. This technique requires that the terminal manager perform internal checks to ensure proper handling of proceed I/O requests. A task can perform proceed I/O, regain control through IRLOUT, and issue a second request before completing the first. For proper handling of such requests, the standard terminal manager technique is to set a hold bit, LNS.HLDB, within the line status word if exit is made to IRLOUT prior to request satisfaction. If this bit is set upon terminal manager entry, the second request is put into a wait state pending completion of the first.

- BLOCK P

  The standard OS/32 subroutines EVCON and EVQCON are used for task connect. An OS/32 utility subroutine, WAIT, puts a task into wait I/O pending completion of an earlier request.

- BLOCK Q

  No data can be moved into a user buffer unless a line I/O was previously performed. If data is not ready for the user, initiate a read and a return to this point, following a successful line read. Initiate this code in either reentrant state (RS) or event state (ES) state. Because of this, it is important to ensure against interlace of proceed requests as described in Block O.

- BLOCK R

  This block contains the crux of the protocol-dependent channel program with all ACK/NAK sequences, line error detection and corrections.

- BLOCK S

  This block indicates setting the LNS.HLDB prior to an exit to IRLOUT and before completing a user request. See Block O.

- BLOCK T

  This block is reached only after completing a successful line I/O. It depicts checking LNS.HLDB and returning to data formatting, if required. See Block O.

- BLOCK U

  This block shows all data input deblocking, formatting and moving of data to a user buffer. Should a new protocol subformat be required, such as a 3270 emulation within binary synchronous, only this block requires modification such as removing existing format subroutines and replacing them by 3270 format procedures or exception code to perform 3270, 3780, 2780 or processor-to-processor formatting, as required. An existing subroutine within binary synchronous, DCDINDEX, can provide an exception code table index dependent on format decode.

- BLOCK V

  This block depicts checking for a possible readahead if dual block buffering is used. Before any readahead, the terminal manager checks for the availability of an input buffer, existence of any concurrent line I/O and other conditions that might preclude such an action.

- BLOCK W

  A readahead is performed like any data read. Following successful read completion, the block is put on queue for deblocking and checking LNS.HLDB for possible return to data formatting.

  The readahead technique generally allows an interface to operate at maximum line speed as opposed to a slower combination of line speed and user program return speed.

- BLOCK X

  This block depicts all data output blocking, formatting and moving of data from a user buffer. Since I/O is performed in an internal buffer as opposed to the user buffer, data modification is permitted in accordance with protocol standards. See Blocks E and U.

- BLOCK Y

  After the last user record is moved into an internal buffer, that data buffer is ready for output. This ready condition should be indicated to the channel program by the format subroutine.

- BLOCK Z

  Following write preparation, the actual write may be delayed and the write buffer only put on queue for output. Reasons for a delay could include existing data I/O if dual buffering is used, the expiration of the wait timer or other protocol-dependent conditions. Return to this point is required following removal of the delay. This area must be able to operate in either RS or ES state, similar to the dual-state read sequence described in Block Q.

- BLOCK AA

  If a ready buffer cannot be output, put it on queue for output and establish a hold state until output initiation. See Block O.

- BLOCK AB

  After line data I/O initiation, return to the user is accomplished only if it is possible to accept a subsequent user request; i.e., another buffer is available for deblocking.

- BLOCK AC

  After a successful data write, determine if a subsequent data block is ready for output. If not, return to the appropriate write preparation routine.

7.5.1 Special Terminal Manager Functions

Special terminal manager functions are performed within existing terminal managers. If an existing similar terminal manager is available, study each major function in it before preparing any similar routines for a new terminal manager. Wherever possible, terminal manager subsections and utility subroutines were prepared in a modular format to permit easy modification, replacement, removal or use in a different environment.

7.5.1.1 Format Control

As shown in Figures 7-1, 7-2 and 7-3, data format blocking and deblocking routines are subroutines of the data transfer preparation section. Inserting a new format subroutine is a straightforward process. Within existing format control subroutines, special character scans are performed by moving the involved data through a translate table. Single format subroutines achieve multiple format techniques with control bit registers. For example, internal space suppression is performed only if the perform space suppression bit is set in a control register. The correct control bit register is generally set by a subroutine based on device code DCB.DCOD, data communications extended device code DCB.XDCD and extended options DCB.XITO.

7.5.1.2 Time-out Control

Usually, communications protocols require time-out delays (SVC15 wait) to be initiated between data transfers. Time-out expiration may require transmission of a protocol-dependent data delay signal or request. Common subroutines exist to set timers via the SVC15 WAIT command and to cancel outstanding time-outs prior to data transfer and determine if the time-out has already expired. An invalid return to the program might occur if a data transfer is attempted simultaneously with an earlier SVC15 time-out data transfer request. Within the binary synchronous terminal manager, use the ONTIMER subroutine to set a delay timer and the OFFTIMER subroutine to cancel an existing time-out.

7.5.1.3 Buffer Control

There are subroutines to access or change the status of internal data buffers. Where appropriate, an error return is provided. These routines are:

BUFFER0	Release buffer Error exit if invalid address provided
BUFFER1	Obtain write blocking buffer Error exit if nonavailable
BUFFER2	Obtain active read deblock buffer Error exit if nonavailable
BUFFER3	Obtain free buffer Error exit if nonavailable
BUFFER4	Obtain write buffer on queue for output Error exit if nonavailable
BUFFER5	Is there a free buffer? Error exit if no free buffer
BUFFER6	Is there a buffer with I/O in progress? Error exit if not
BUFFER8	Free all buffers No error exit
BUFFER9	Place buffer on queue No error exit

7.6 SYSTEM GENERATION (SYSGEN) CONVENTIONS

See the System Generation/32 (Sysgen/32) Reference Manual for sysgen considerations. To generate the proper basic data communications code, SGN.ITAM must be set to 1.

7.6.1 Register Conventions

In the binary synchronous terminal manager, strict adherence to register conventions was an aid in system implementation. These conventions are recommended for future terminal manager modifications:

U0	Work (may be destroyed by any subroutine)
U1	DCB address
U2	LCB address
U3	Block descriptor address for format subroutines (LCB.BLK)
U4	Function code (DCB.FC)
U5	Extended option function code (DCB.XITO)
U6	Line status code (DCB.LNST)
U7	Level 0 subroutine entry or return
U8	Level 1 subroutine entry or return
U9	Level 2 subroutine entry or return
U10-U15	Work (may be destroyed by any subroutine)

7.6.2 Device Control Block/Line Control Block (DCB/LCB) References

Future data communications modifications may involve reorganization of a DCB or LCB. For this reason, do not use the following coding techniques:

- Reference to any DCB or LCB element by an absolute value as opposed to its symbolic name.
- Reference to any DCB or LCB element to require certain DCB/LCB elements to be adjacent. This includes load halfword of currently adjacent bytes, load fullword of currently adjacent halfwords and similar multiple load techniques.

7.6.3 EXTRN/ENTRY References

Certain EXTRN/ENTRY references are required to ensure proper OS/32 interface. Although specific applications might require different interface points, the list of the external references currently used by the binary synchronous terminal manager should suffice.

IT.HALT	is an external reference to the halt I/O subroutine.
BEBC.ASC	is an external reference to the binary synchronous line driver ASCII-to-EBCDIC translation table.
ISSEXEC	is an external reference to the system subroutine executor.
WAIT	is an external reference to the executive wait subroutine.
TMREMW	is an external reference to the OS/32 remove task wait subroutine.
JOURNAL	is an external reference to the OS/32 system journal subroutine.
TOCHON	is an external reference to the OS/32 subroutine to put a DCB on a timer chain.
TOCHOFF	is an external reference to the OS/32 subroutine to remove a DCB from a timer chain.
III	is an external reference to the OS/32 subroutine to ignore interrupts.
ISPTAB	is an external reference to the interrupt service pointer table.
IODONE/IODONE2	is an external reference to the OS/32 I/O completion exit points.
EVRTE	is an external reference to the OS/32 exit points.
TWT.RJE	is an entry reference to the subroutine to determine whether or not a terminal manager is busy.
CPT.RJE	is an entry reference to the terminal manager checkpoint subroutine.
CLOSMBSC	is an entry reference to the terminal manager close subroutine.
INITMBSC/FUNCMBSC	is an entry reference to the terminal manager entry points from SVC1.

7.6.4 System Generation (Sysgen)

Following assembly of a new terminal manager and its associated DCBs, standard sysgen procedures can be used to add the new terminal manager and its associated DCBs to the OS/32 system. The new terminal manager and DCBs must be merged into the combined driver library using the OS/32 Library Loader. The only restrictions on the required order of modules within the library is that the DCBs for all devices supported by a particular driver or terminal manager must precede the driver or terminal manager.

7.7 WRITING TERMINAL MANAGERS SUMMARY

The following procedures provide a user-written terminal manager:

- Determine if a new terminal manager is actually required. Where practical, use a standard terminal manager in image mode or applications program SVC15 access.
- Gain thorough familiarity with areas involved. Study existing terminal managers to determine which major sections or subroutines can be used. Consult the individual line driver descriptions to determine how these capabilities can be best used.
- Test and implement the new integrated communications package.

7.8 HOW TO USE DATA COMMUNICATIONS TERMINAL MANAGERS

This section provides a sample program containing four coding examples to illustrate the structure of data communications SVC1 parameter blocks and the execution of SVC1 reads and writes. A CAL STRUC is provided to generate equates for all parameter block references, and each used parameter block is laid out with explanatory comments. The parameter block labelled LINEBLK is used for all functions, and the parameter blocks READBLK and PRINTBLK are used for other SVC1 accesses. The following four coding examples describe these functions.

Example 1:

This example illustrates a loop reading from an input device (e.g., a terminal) and writing each record to a bisynchronous terminal manager. The loop terminates when it recognizes an input sentinel and an end of file (EOF) is written to the line. No extended options are used.

```
EXAMPLE1 EQU   *
*
*              THE FOLLOWING USER EXAMPLE WILL READ
*              CARDS FROM LOGICAL UNIT 1 AND
*              WRITE EACH CARD RECORD TO AN ITAM
*              COMMUNICATIONS LINE ON LOGICAL UNIT 2
*              UNTIL A
*              CARD BEGINNING WITH // IS FOUND.  IT
*              WILL THEN WRITE AN EOF.
*
         SVC   1,READBLK                READ A CARD
         LH    U15,READBLK+SVC1.STA     CHECK FOR ERROR
         BNZ   DONE
         LH    U15,BUFSTART             SEE IF LOOP IS DONE
         CLHI  U15,C'//'
         BE    EOF
         SVC   1,LINEBLK                WRITE CARD TO ITAM COMM LINE
         LH    U15,LINEBLK+SVC1.STA     CHECK FOR ERROR
         BNZ   DONE
         B     EXAMPLE1                 CONTINUE LOOP
EOF      EQU   *
         LHI   U15,X'88'                MODIFY PARAMATER BLOCK
         STB   U15,LINEBLK+SVC1.FC      TO WRITE EOF
         SVC   1,LINEBLK                WRITE EOF TO COMM LINE
DONE     EQU   *
         SVC   3,0                      END OF JOB SVC


         ALIGN ADC
READBLK  EQU   *
         DB    X'48',1,0,0
         DAC   BUFSTART,BUFEND
         DAC   0,0,0

LINEBLK  DB    X'28',2,0,0
         DAC   BUFSTART,BUFEND
         DAC   0,0,0
```

Example 2:

This example illustrates the use of data communications extended options. The data communications parameter block (LINEBLK) is modified to include a write with extended options and the format, transparent and transmission extended options, are set. A single record is then read from an input device and written to the terminal manager. After write completion, the example goes to end of job. The transparent and transmission extended options are not honored on all devices.

```
EXAMPLE EQU     *
*               THE FOLLOWING USER EXAMPLE WILL
*               READ A SINGLE CARD FROM LOGICAL
*               UNIT 1 AND WRITE IT AS A SINGLE
*               RECORD TRANSMISSION, TRANSPARENT
*               TEXT TO AN ITAM COMMUNICATIONS
*               LINE ON LOGICAL UNIT 2.
*
*
          LHI   U15,X'29'                          SET UP WRITE FUNCTION CODE
          STB   U15,LINEBLK+SVC1.FC                USING ITAM EXTENDED OPTIONS
          LI    U15,SV1X.FM+SVC1X.TRM+SV1X.TM      SET UP ITAM EXTENDED
          ST    U15,LINEBLK+SVC1.EXO               OPTIONS
          SVC   1,READBLK                          HEAD A CARD
          LH    U15,READBLK+SVC1.STA               CHECK FOR ERROR
          BNZ   DONE
          SVC   1,LINEBLK                          WRITE TO COMMUNICATIONS LINE
          B     DONE
```

Example 3:

Illustrates a loop reading record from a terminal manager and writing each input record to an output device (e.g., a printer). When an error from the line is received (e.g., an EOF), the loop terminates and the example goes to end of task.

```
EXAMPLE3 EQU    *
*
*               THE FOLLOWING USER EXAMPLE WILL
*               READ SUCCESSIVE CARD RECORDS
*               FROM AN ITAM COMMUNICATIONS
*               LINE ON LOGICAL UNIT 2.
*               EACH RECORD WILL BE PRINTED ON
*               LOGICAL UNIT 1.  UPON RECEIPT
*               OF AN EOF OR OTHER ERROR RETURN
*               FROM ITAM, THE TASK WILL
*               TERMINATE
*
          LHI   U15,X'48'                  SET ITAM PARAMETER BLOCK FUNCTION
          STB   U15,LINEBLK+SVC1.FC        CODE FOR A READ.
EX3LOOP   EQU   *
          SVC   1,LINEBLK                  READ FROM ITAM
          LH    U15,LINEBLK+SVC1.STA       CHECK FOR ERROR
          BNZ   DONE                       DEPART FOR AN ERROR
          SVC   1,PRINTBLK                 WRITE TO THE PRINTER
          LH    U15,PRINTBLK+SVC1.STA      CHECK FOR ERROR
          BNZ   DONE                       DEPART FOR ERROR
          B     EX3LOOP                    CONTINUE LOOP
```

Example 4:

Illustrates the use of the disconnect extended option. A data communications parameter block is built to write a record using extended options. The format and disconnect extended options are set. Upon completion, the example goes to end of job.

```
EXAMPLE4 EQU   *
*              THE FOLLOWING USER EXAMPLE
*              WILL MAKE A SINGLE RECORD
*              WRITE TO THE COMMUNICATIONS
*              LINE ON LOGICAL UNIT 2.  THE
*              USER REQUEST USES ITAM EXTENDED
*              OPTION WITH A REQUEST
*              TO DISCONNECT THE LINE FOLLOWING
*              THE WRITE.  AFTER THE WRITE, THE
*              TASK GOES TO END OF JOB.
*
         LHI   U15,X'29'                PUT WRITE WITH EXTENDED OPTIONS
         STB   U15,LINEBLK+SVC1.FC      FUNCTION INTO PARAMETER BLOCK
         LI    U15,SV1X.FM+SV1X.DM      FORMAT AND DISCONNECT
         ST    U15,LINEBLK+SVC1.EXO     EXTENDED OPTIONS
         LA    U15,PRINTOUT             SET UP PRINT-LINE START AND
         ST    U15,LINEBLK+SVC1.SAD     END ADDRESSES
         LA    U15,PRINTEND
         ST    U15,LINEBLK+SVC1.EAD
         SVC   1,LINEBLK                EXECUTE THE TERMINAL MANAGER SVC
         SVC   3,0                      GO TO END OF JOB
```

CHAPTER 8
HOW TO WRITE AND USE DATA COMMUNICATIONS LINE DRIVERS

8.1 INTRODUCTION

Data communications covers a broad range of transmission types, line protocols, character sets, modems, terminals and terminal idiosyncrasies. Since the basic data communications subsystem cannot support every possible configuration, the purpose of this chapter is to assist the system programmer responsible for modifying or writing a data communications line driver.

8.2 MODIFYING A LINE DRIVER

Before modifying a driver or writing a new driver, determine the functions performed by a typical line driver, where they are performed and which functions require modification with existing drivers as a working base.

In general, data communications line drivers perform some or all of the following steps during a supervisor call 15 (SVC15) request:

1. Common driver initiation processing
2. Command fetch of the basic command category
3. Common command processing
4. Modifier fetch of the specific command
5. Data field fetch
6. Address relocation
7. Data buffer initialization and channel control block (CCB) setup
8. Initialization of adapter and modems

9. Transfer of actual data

    a. Internal/external character set translation

    b. Detection of special characters

    c. Buffer limit processing

    d. Error status processing

10. Ending sequence processing and adapter disabling

11. Event service routine (ESR) processing

    a. Restart at Step 2 if command is chained

    b. Terminate the call if in error or completed

Steps 1 through 7 are nearly device-independent. They are basically the same for all drivers observing the standard data communications command and buffer format. Steps 8, 9 and 10 are concerned with the type of modem and characteristics of the attached terminal. Steps 8, 9c, 9d and 10 are performed by the interrupt service routines (ISRs) of the command. The auto driver channel, with the assistance of the translation table whose address is in CCB.XLT, performs Steps 9a and 9b.

For example, the existing asynchronous line driver supports a 103-type modem. If support for a 202-type modem using reverse channel is desired, new ISRs are required to handle the different status interrupts caused by the reverse channel. Different output commands are required to enable and disable the adapter. These should be assembled in the new device control block (DCB).

If the attached terminal communicates in ASCII, only the new ISRs that handle the interrupts are required along with the new DCB containing the desired output command bytes. Figure 8-1 shows the structure of a typical data communications line driver.

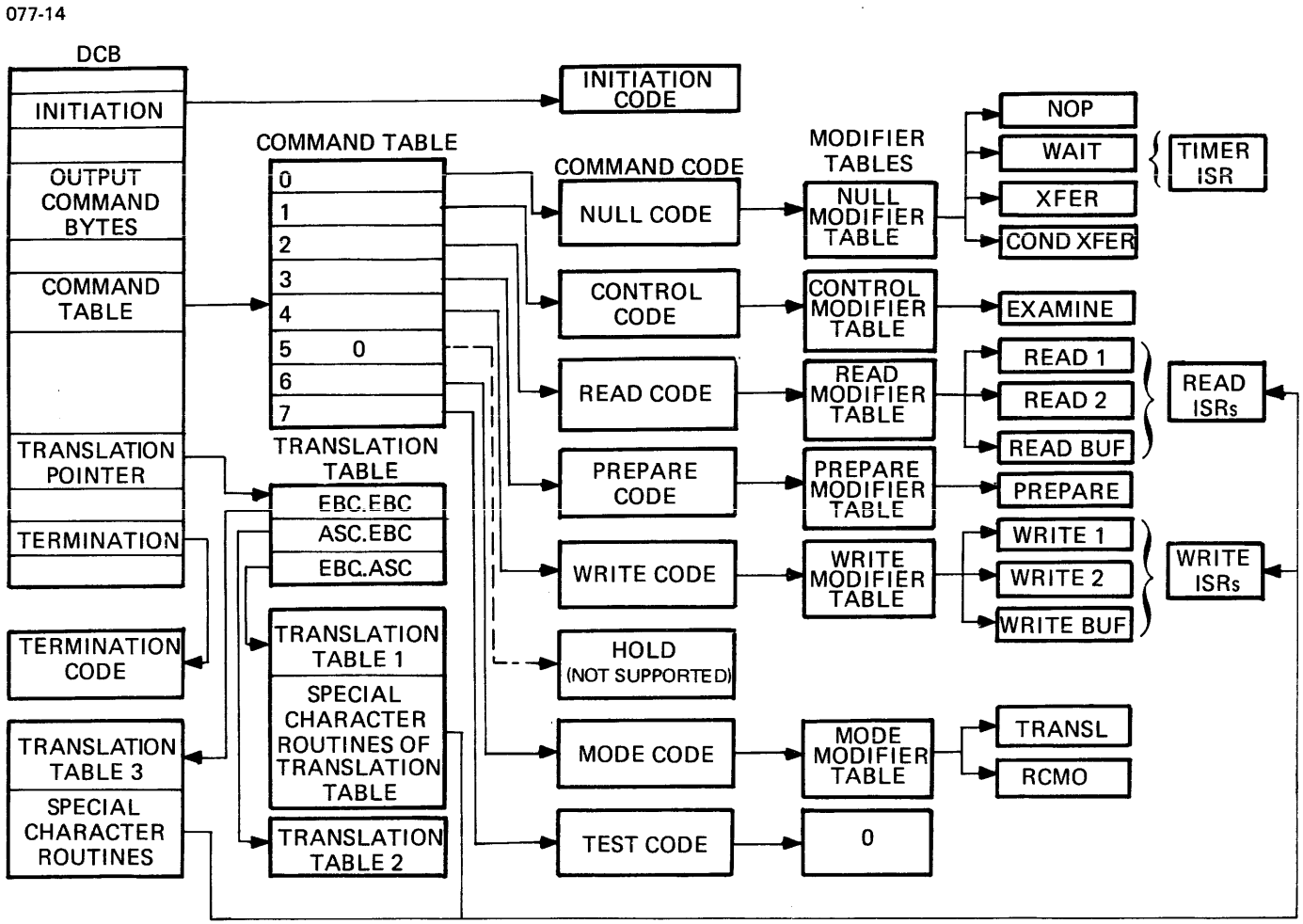


Figure 8-1 SVC15 Driver Structure

As another example, assume that the attached terminal can interface to a 103-type modem but communicates in EBCDIC (or any other character set). The addition of two new translation tables, ASCII-to-EBCDIC for output and EBCDIC-to-ASCII for input, would be required.

Most specialized requirements would need:

- Different translation tables
- Special character handling subroutines
- ISRs to handle interrupts in a device-dependent (modem) manner

The standard command format and buffer management should be maintained. A similar asynchronous or binary synchronous line driver should be studied as an example. Much of the basic data communications subsystem is table-driven, so modifications or additions require a table change and the additional code necessary to perform the new feature.

8.3 LINE DRIVER USE OF THE DEVICE CONTROL BLOCK (DCB)

A separate DCB is maintained for each device (adapter) in the system. It is the DCB that controls the flow of any SVC15 request by allowing each device to specify its particular requirements. The DCB maintains control of the SVC15 request through these fields:

DCB.INIT	is the pointer to the initiation code for all SVC15 requests to this device.
DCB.CTA	is the address of the command table that contains pointers to the code for each of the commands supported by this device. Some commands may be common with other drivers; other commands may be specially designed.
DCB.MOCR DCB.MOCW DCB.DOCR DCB.DOCW DCB.DISK	are the bytes used with output command instructions to control the adapter for a particular attached modem and terminal.
DCB.AOC	is the byte used to load programmable adapters with the information required to control the communications line attached (line speed, character size and parity information).
DCB.XLT	is the pointer to the table of valid driver translation tables.

DCB.TERM — is the pointer to the driver termination code.

DCB.RDN
DCB.WDN — are the halfwords indicating the device numbers to use when reading and writing to the connected adapter. Depending on the adapter strapping, these numbers may be the same or different. These numbers are initialized by the file manager using the 2-wire/4-wire indication in DCB.XDCD to determine device strapping.

DCB.XDCD — is the extended device code field initialized at sysgen that supplies information about the intended use of the communications line.

DCB.ESR — is the address of the ESR to be scheduled during system queue service (SQS).

8.4 LINE DRIVER STRUCTURE

The actual code to handle a specific request (command and modifier) is depicted by the boxes on the right side of Figure 8-1. The routines are entered by indexing through a series of tables beginning at the DCB (left side of Figure 8-1). New commands or features can be added by specifying a new table that points to the old command routines that remain unchanged and to the new routines for the added features.

Much of the code needed by line drivers handles command and modifier fetching and buffer management. This is usually consistent throughout all line drivers supporting the standard data communications format and has been implemented in subroutines maintained in the data communications module. The user-written line drivers should use these supplied routines. If the standard data communications command format and buffer management is followed, a specialized driver would require ISR code to handle the adapter and translation tables to handle the character set involved.

8.4.1 Driver Initiation Routine

Drivers are entered in the ES(NSU) state and command interpretation begins immediately. Commands performing input/output (I/O) require interrupt service (IS) (execution is in the IS state), eventually ending in event state (ES). Any additional commands in the command chain are interrupted and executed in the ES state. Normally, the drivers exit to the supplied routines CMEXIT and CMTERM.

Routine CMEXIT is like TMRSOUT or EVRTE and is used whenever the driver desires to exit and wait for a condition to reactivate it. Routine CMTERM is like IODONE or IODONE2 and is used to end an SVC15 request and, if required, to generate a termination trap to the calling task.

SVC15 instruction execution by a task results in entry to the line driver in the ES(NSU) state at the initiation address given by the DCB.SVCF, with UD13 containing the address of the DCB. Most drivers at this time begin driver command word (DCW) command execution by branching to the supplied routine ICMDINT. However, the driver might initialize programmable adapters at this time or perform some other introductory operation. Once ICMDINT is entered, the driver is controlled by the command table and the DCB.

8.4.2 Translation Tables

Translation tables can exist anywhere in memory. However, all special character subroutines referenced by the tables must be in pure code located below 64K.

To make changes in translation as easy as possible, drivers make no absolute reference to any character. A driver can have a code to handle a terminating character, but the translation table and the translation table routines decide what constitutes such a character and branch to the code that handles it. A change of translation table and possibly some output commands supplied in the DCB, should orient the supplied driver toward a new device while maintaining compatibility with the devices for which the driver was originally designed.

All basic data communications special character routines begin with a Load Halfword Immediate (LHI) instruction to load a register with the translated character. This instruction must not be squeezed by CAL/32 to allow line drivers to perform a software translation of all characters, including special characters, since the translated value of any character is always the second halfword of each routine.

8.5 DATA COMMUNICATIONS LINE DRIVER EXAMPLE

This section is to guide the reader through the code of a typical line driver. To write a new line driver that handles reads and writes using the existing buffer management and command format but requires a new protocol and different handling of the modem interface, the standard routines supplied for NULL and CONTROL should be adequate. No support is required for the MODE or PREPARE command.

8.5.1 Command Table

First, a command table referencing the necessary commands is required. This table usually is a part of the driver and must be aligned on a fullword boundary and have a label declared as an ENTRY in the driver. This label is referenced as an EXTRN by the DCB. The command table can be coded as follows:

```
            ENTRY   UWCOMTAB
            EXTRN   ITAMNULL,ITAMCTRL
            .
            .
            .
            ALIGN   ADC                 ALIGN TABLE ON FULLWORD
UWCOMTAB    EQU     *
            DAC     ITAMNULL            USE STANDARD NULL
            DAC     ITAMCTRL            USE STANDARD CONTROL
            DAC     UWREAD              USER WRITTEN READ
            DAC     0                   NOT SUPPORTED (PREP)
            DAC     UWRITE              USER WRITTEN WRITE
            DAC     0                   NOT SUPPORTED (HOLD)
            DAC     0                   NOT SUPPORTED (MODE)
            DAC     0                   NOT SUPPORTED (TEST)
```

The initiation code for the driver must begin with a label of the form INITxxxx just as for general-purpose drivers. This is declared as an ENTRY in the driver and an EXTRN in the DCB. The reference to the line driver in the DCB is in DCB.SVCF instead of DCB.INIT, since DCB.INIT is reserved for the address of the terminal manager, if supported.

8.5.2 Command Fetch

Routine ICMDINT locates the first command in the DCW string, uses the least significant three bits to index into the command table and branches to the code to perform that command. If the command specifies command traps or time-out, the trap is generated or the timer started before entering the command. If the command is a NULL or CONTROL, the routine ITAMNULL or ITAMCTRL is entered, indicated by the command table. If the command is a WRITE, the user-written routine UWWRITE is entered. The command halfword is in U3 and the DCB in UD (the user-written routine in the center of Figure 8-1) can perform any desired function or initialize programmable adapters, if necessary. To enter the code to perform the exact function indicated by the modifier, use a data communications routine and a user-supplied table. The basic command (WRITE, in this case) can access the code for each modifier by the following steps:

1. Loading UC with the address of the write channel control block (CCB)
2. Loading U7 with the address of the write modifier table
3. Loading U6 with the maximum allowable modifier
4. Branching to the ITGETMOD routine

The command halfword is still contained in U3 and the DCB address is always maintained in UD.

8.5.3 Modifier Fetch

The CCB is cleared of all buffers by ITGETMOD, which then places a copy of the command in the CCB after clearing the unused four bits (bits 4 through 7) and branches to the code for the function specified in the modifier table.

NOTE

All loads of CCB addresses must use the LHI instruction since the CCB address is stored in a halfword that might be located above 32K.

8.5.4 Command/Modifier Routines

Each particular command (right side of Figure 8-1) must perform all the processing required to complete its defined function. If data fields are required, they can be fetched by calling ITGETDAT, which returns with the next data field in U7. If bit 0 of the data field is set (data code X'80'), ITGETDAT performs a data field transfer by relocating the address contained in the data field and using this to continue the data field fetch.

Data fields contain the addresses in user program space of data required by the driver. These addresses must be relocated to absolute values to be useful to drivers. Data buffer addresses can be relocated by calling ITBFREL. All other addresses must be relocated by calling ADCHK.

The routine ITBFREL assumes all buffers are in the same logical segment as the address contained in the first data field of the SVC15 parameter block.

The command can now load the data from this address and perform the required operation. Mode commands, for example, merely store the data in selected places in the DCB. READ and WRITE commands must transfer data into or out of this location. They use the address and the data code to set up the CCB for a data transfer using the buffer type specified by the data code through the ITGETBUF routine.

A call to ITGETBUF works to fetch the data field, relocate the address and set up the CCB for a transfer of the proper size data. It uses the buffer type specified by the data code. The least significant four bits of the data code are also stored in bits 4 through 7 of the command kept in the CCB. If the command is a short write such as WRITE2 (maximum of 15), this number is placed (ORed) into the low-order four bits of the options register, U4, before calling ITGETBUF.

If read after write lookahead is supported, a check is performed to see if the present command (WRITE) is chained. If so, RAWCHKR (referenced as an EXTRN) is called. This routine checks to see if the next command is a READ. If so, RAWCHKR sets a read after write (RAW) pending flag in the DCB, sets the get one buffer flag in the options register, and fetches and enters the READ code. The READ command performs as if it were fetched normally and can initialize programmable adapters or any other introductory read operation. It performs the following steps:

1. Loads UC with the address of the read CCB

2. Loads U7 with the address of the modifier table

3. Loads U6 with the maximum valid modifier

4. Branches to ITGETMOD

The ITGETMOD routine performs the same operations conducted in fetching the WRITE and enters the code for the indicated type of READ. This code also sets up the read CCB for a READ using the appropriate buffer type by calling ITGETBUF. The READ is now ready to begin. However, if a RAW pending flag is found, it returns to the WRITE code.

8.5.5 Entering Interrupt Service Routines (ISRs)

Returning from a call to RAWCHKR, the WRITE code sets up the channel command word (CCW) of the CCB (CCB.CCW) to ensure that the execute bit is off and that the other bits are appropriate for the write. Most likely, the write and the translate bits are also set. The subroutine pointer in the CCB (CCB.SUBA) is loaded with the address of a pure routine, an ISR in the driver. The address of the CCB+1 is stored in the ISPTAB entry for the write device number, and a simulate interrupt (SINT) instruction is performed using a write device number. The routine now exits by branching to CMEXIT.

The SINT instruction simulates an interrupt and therefore enters the ISR. User-written drivers can perform whatever is necessary here; however, most drivers try to place the adapter and the modem into write mode, assuming that the adapter is in an unknown state. This is done by issuing an output command using the byte in the DCB reserved for placing the adapter in write mode with interrupts enabled (DCB.MOCW). The address of CCB.SUBA can now be changed to point to the next ISR and the routine exits by a Load Program Status Word Register (LPSWR) instruction. Each interrupt causes the second ISR to be activated. The following registers are loaded by firmware:

E0	Program status word (PSW) status before the interrupt
E1	PSW location before the interrupt
E2	Device number causing the interrupt
E3	Status of device causing the interrupt
E4	Address of the CCB

This second ISR generally ignores all interrupts until the adapter is ready for data transfers (indicated by zero status). ISRs exit by the LPSWR using register E0. They can use registers E2 through E7 without restoring them; E8 through E15 can be used only if saved and stored.

Once the adapter is ready for the transfer (status equal zero), the CCB.SUBA is loaded with the address of an ISR to handle the interrupts from the auto driver channel. The execute bit is turned on in the CCW. The first character is obtained from the buffer by a call to ITFC, translated by the TLATE instruction and written to the adapter. The CCB pointers are adjusted by the Simulate Channel Program (SCP) instruction, and the ISR exits by loading the PSW from register E0.

All future interrupts from the adapter are handled by the auto driver channel (microcode) and assistance from the driver is required only for one of the following reasons:

- Error status interrupts
- Buffer limit interrupts
- Special character processing

An error status interrupt is usually an indication to abort I/O by loading the appropriate status in register E7 and branching to ITISSTOP. This routine stores the status, disables the device (both read and write), clears the ISPTAB entries and schedules the halt I/O routine, terminating the entire SVC15 request.

A buffer limit interrupt means the CCB has just used the last byte from the buffer and complemented the buffer select bit in the CCW of the CCB. When using chained buffers, the driver must determine if another buffer is available. If another buffer is available, the driver keeps the I/O going with the second buffer while a third buffer is readied by scheduling the next buffer write routine. All these operations are done by calling ITXFRISR to schedule the next buffer routine, if there is a need for it, or to return to the caller and indicate that no more buffers are available. Each individual driver can treat this situation as appropriate. The binary synchronous driver treats it as an error and indicates that a proper terminating sequence was not encountered. The asynchronous driver considers it a normal completion. The ITXFRISR routine checks for certain error conditions, and if any exist, aborts the entire SVC15 request with the appropriate status. These errors are concerned with chained buffers and the condition of the busy and done bits or with the fact that a next buffer write routine was scheduled, but had not yet executed.

8.5.6 Special Character Routines

Special character routines are entered as a result of the translation table and are responsible for the following procedures:

- Translating the character by an LHI instruction, since each routine is for a specific character.
- Performing the required function of the character (change modes, terminate the I/O or backspace).
- Writing the translated character to the adapter or storing it in the buffer, as appropriate.
- Performing cyclic redundancy checks (CRC) or longitudinal redundancy checks (LRC), if necessary.
- Incrementing the pointers of the CCB so that character count is correct.
- Handling any buffer limit situations that occur during the previous step.

8.5.7 Read After Write (RAW) Turnaround

At some point, the ending character, ending sequence or buffer limit condition can terminate I/O. The adapter is disabled by issuing an output command using the command byte in DCB.DOCW and is usually set up to disable the adapter while keeping it in write mode. The end buffer write routine is scheduled by calling ITSRABS. This routine takes care of the last buffer and adjusts the length of the last write indicator in the DCB. If the RAW pending flag is set, the driver ISR calls the routine ITWR.RD to perform the following steps:

- Increment the current DCW pointer
- Set transfer not begun in status halfword
- Set up the ISPTAB entry for the read device number
- Load E2 with the read device number
- Load E3 with the read status
- Load E4 with the read CCB
- Start read error timer if command requests time-out
- Schedule RAW second buffer if chained buffers are used
- Schedule command trap read if command requests it
- Increment number of commands executed
- Adjust current DCW pointer
- Return to caller, who branches to the first read ISR

The read ISR now performs like the write ISR, placing the adapter in read mode by using DCB.MOCR and waiting for an interrupt with the proper status (usually zero). If buffer traps are requested in the command, the trap is scheduled by calling ITSRABS with the proper reason number. CCB.SUBA is loaded with the address of the read ISR to handle the interrupts from the auto driver, the execute bit is set and the I/O proceeds as in the write. Buffer limit, error status and special characters are handled just as in the write ISR. The read ISR, using DCB.DOCR, eventually terminates by disabling the adapter, schedules the end buffer read routine and the driver-termination routine by two calls to ITSRABS, and exits by the LPSWR instruction.

8.5.8 Driver-Termination Phase

The driver-termination phase routine, whose address is in DCB.TERM, checks the status halfword in DCB.ISTA and terminates the SVC15 request if any errors exist by branching to CMTERM. When no errors exist, the driver ESR checks to see if the current command is chained and, if so, fetches and executes the next command; this is accomplished by the routine ITNXTCMD. This routine checks the current DCW command and, if it is chained, increments the DCW pointer and branches to ICMDINT, which continues with command interpretation. When the command is not chained, ITNXTCMD returns to the caller. The entire SVC15 request can then be terminated by branching to CMTERM.

8.6 USING DATA COMMUNICATIONS LINE DRIVERS

Except for interrupt handling routines whose code is executed in the IS state, data communications line drivers execute in the ES(NSU) state with SQS turned off.

Drivers fetch and execute each command of a DCW chain. When a command goes to an error-free completion and if the chain command bit of the command is set, the next command in line is fetched and executed. This sequence of fetch and execute is repeated until the entire DCW chain is successfully interpreted or until an error condition in any command terminates the SVC request.

Commands can be arranged in any order; however, each command is interpreted one at a time and at user priority. A command chain consisting of three consecutive writes is interpreted as three complete and separate write operations that are fetched, set up, executed and terminated one at a time. On a high-speed CRT, delays may appear between the writes.

If the preceding sequence of three writes were issued to the binary synchronous driver, each write would begin by sending a series of leading synchronous characters. In addition, each write would end with the proper binary synchronous termination sequence. If this sequence is not supplied, an error results. An analogous situation exists for chained reads. Data communications is generally interactive and involves a write request followed by a read request, such as:

- Write prompt character(s) and read data
- Write a buffer and read acknowledge (ACK) or negative acknowledge (NAK)
- Write ACK or NAK and read buffer
- Write poll or select and read response

Data communications allows a write command chained to a read command to be handled as one continuous command, both of which are fetched and set up before any I/O begins. The write is then performed. Upon completion, the line is immediately turned around and the read is performed. Once I/O for the write actually begins, the entire sequence (write and read) is performed regardless of task priority and in a manner totally transparent to the user.

In addition to the RAW lookahead, there is a read after prepare lookahead that allows a task to chain a read to a prepare. Thus, a task can scan a communications line for a special (prepare) character and immediately enter the read command to read any following data.

8.6.1 Buffer Management

Buffer management currently supports four buffer types:

- Direct
- Indirect
- Chained
- Queued

Direct and indirect buffers each consist of a single buffer that must be sufficient to complete a single write or read command. The term complete means that the buffer contains one or more terminating characters (if required for writes) or enough buffer space to hold one or more terminating characters (if required for reads). Chained and queued buffers consist of one or more linked buffers. It is through the use of chained and queued buffers that most flexibility is achieved. The entire series of buffers must be sufficient to complete a command. The use of buffers is illustrated as follows:

- Figure 8-2 is a sample SVC15 parameter block to write data from a direct buffer.
- Figure 8-3 is a sample SVC15 parameter block to read data (an SVC15 parameter block reading) into an indirect buffer.
- Figure 8-4 is a sample SVC15 parameter block to read data into a series of chained buffers.
- Figure 8-5 is a sample SVC15 parameter block to read data into a series of queued buffers.

I/O has a double-buffering capability. However, when a single buffer (direct, indirect or single chained) is specified for a read or write, only one buffer can be used. If I/O reaches the end of the buffer, the line driver receives a buffer limit interrupt. Buffer limit interpretation is driver-dependent. For example, the asynchronous driver terminates error-free when buffers are exceeded, while the binary synchronous driver returns a buffer limit status if a proper terminating sequence did not occur before buffer limits were exceeded.

077-15

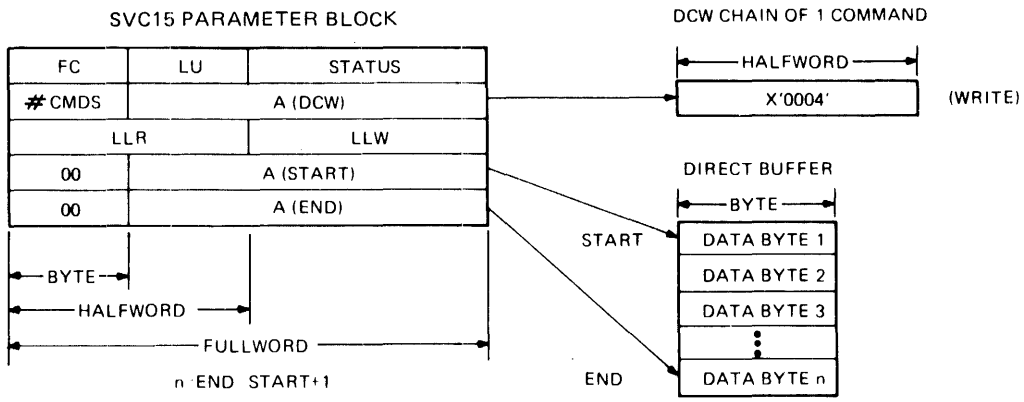


Figure 8-2 SVC15 Using Direct Buffers

077-16

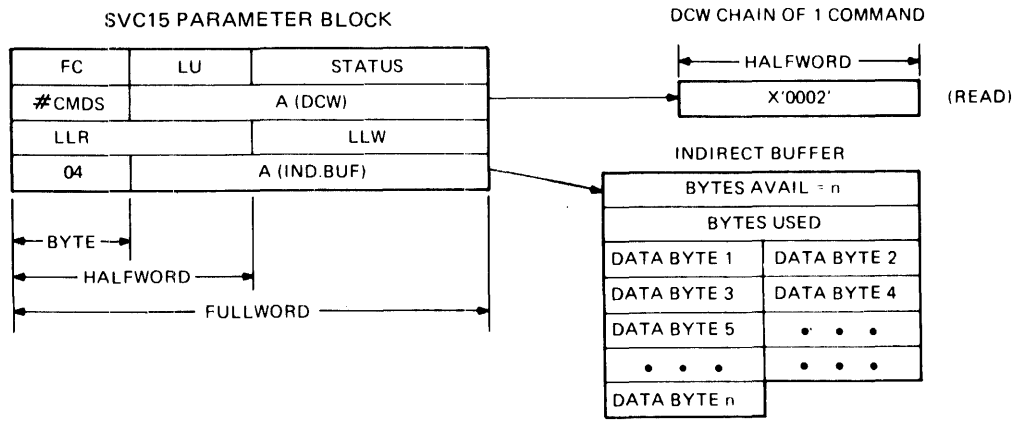


Figure 8-3 SVC15 Using Indirect Buffers

077-17

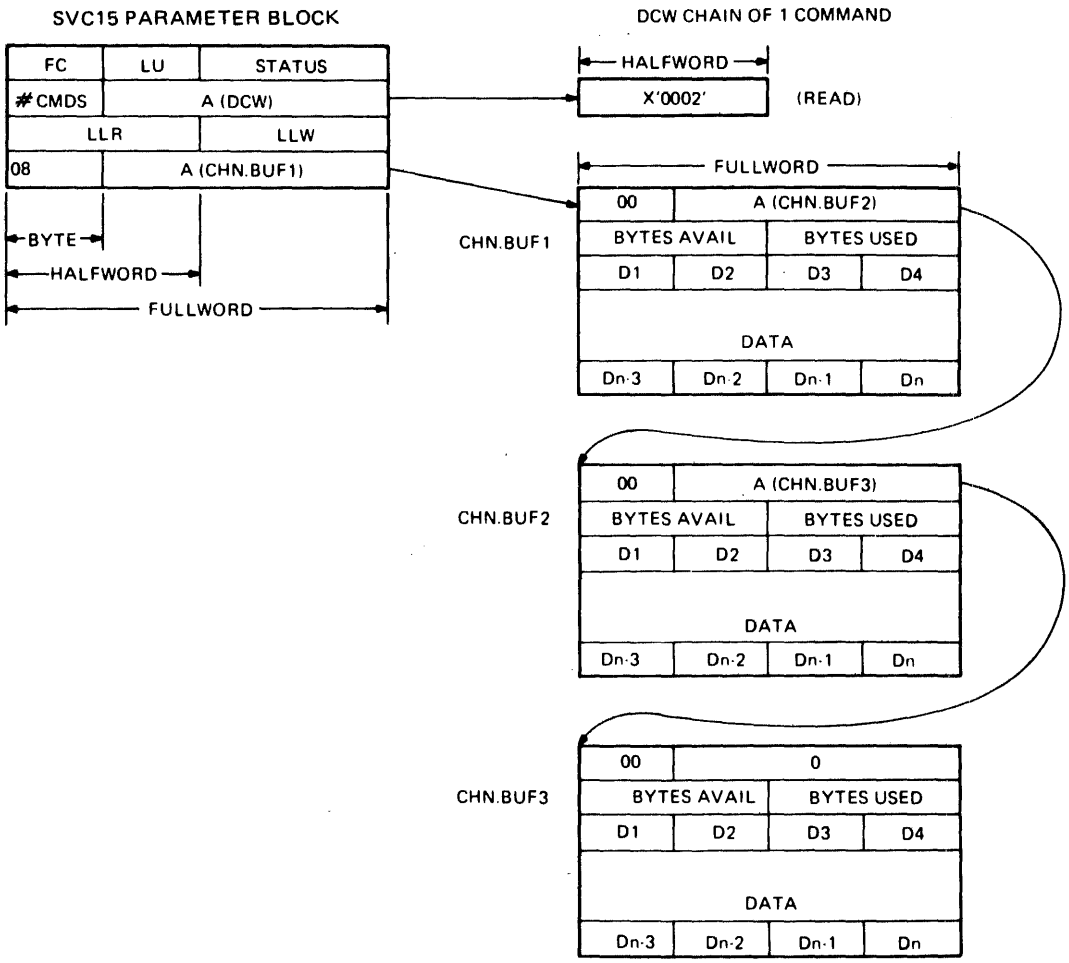


Figure 8-4 SVC15 Using Chained Buffers

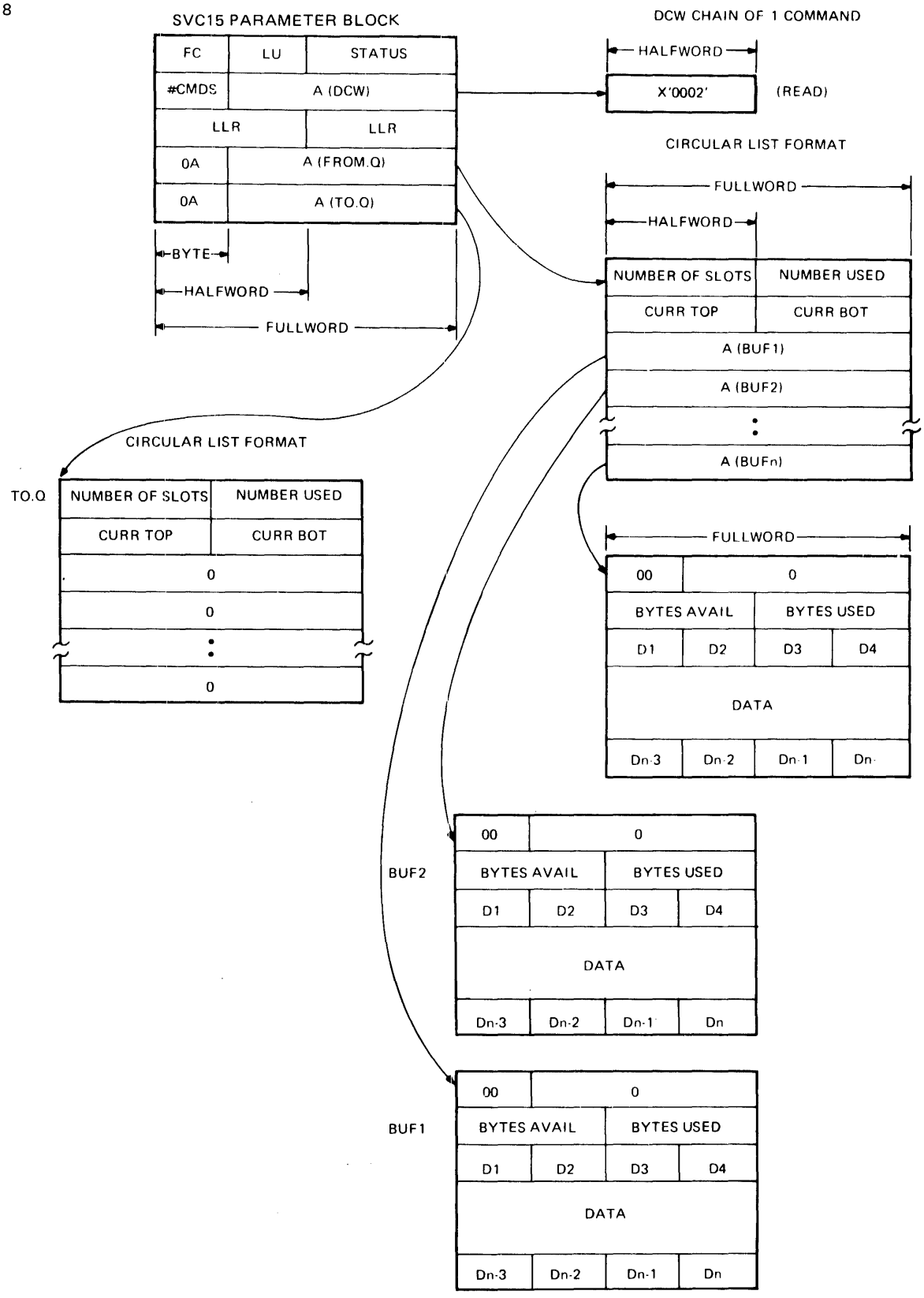


Figure 8-5 SVC15 Using Queued Buffers

8.6.1.1 Chained Buffers

When chained buffers are specified, the driver attempts to set up two I/O buffers. This means that for the task to use more than a single chained buffer for I/O, it must supply at least two linked buffers when the SVC15 request is issued. These two buffers are set up; the first buffer is flagged as busy (bit 0 of link word set) a buffer trap is generated, if specified and I/O starts. When buffer 1 is exhausted, the driver receives a buffer limit interrupt and finds buffer 2 is available (busy and done bits reset) and uses this buffer to continue I/O. Meanwhile, a routine is scheduled to flag buffer 1 as done, flag buffer 2 as busy, and attempts to set up the first I/O buffer using the current link word of buffer 2 as a pointer. If this link is zero, the current buffer is the last buffer of the chain and the I/O must terminate within it. If the link word is nonzero, it indicates the location of buffer 3. At this time, buffer 3 does not necessarily have to be available (busy and done bits reset) as long as its address is specified. A buffer trap is generated to the calling task. However, buffer 3 must be available before the next buffer limit interrupt. If it is not, the buffer overruns and I/O aborts; the status reflects this overrun.

In RAW, the driver looks ahead and sets up only one read buffer. Therefore, if chained buffers are used for the read, only one buffer is set up in advance. When the driver terminates the write, one buffer is ready for the read. If the read uses chained buffers, a subroutine to get the next buffer schedules immediately after performing the write to read turnaround. The buffer trap for the read is performed after read I/O begins. Getting only one read buffer instead of two is useful when using chained buffers for both write and read. However, the buffer trap for the next to last write buffer must be identified by the u-task because the task must specify the second buffer in the link word of the first read buffer before the driver completes the write.

When interpreting commands, the driver uses two pointers that are updated during command fetching and execution:

- The current DCW pointer that points to the current DCW being executed. If the command is chained, it is updated after each command goes to completion.
- The next data field pointer pointing to the data field to be used on the next data field fetch. It is updated after each data field is fetched.

This example illustrates how these pointers are used. It also illustrates the transfer in Data Facility and the TRANSFER command.

NUMBER	DATA FIELD	DCW COMMAND CHAIN	COMMAND NUMBER
1	00 BUF1.STRT	cc WRITE BUFFER	1
2	00 BUF1.END	cc READ BUFFER	2
3	04 BUF2	cc XFER	3
4	01 A(command 1)		
5	80 A(data 1)		

Execution of the command chain and data chain results in continuous I/O consisting of the following:

- A WRITE from buffer 1 (direct text)
- A READ into buffer 2 (indirect text)
- A branch back to a write from buffer 1 (direct text), thus repeating the write/read sequence indefinitely or until an error occurs.

At starting time, the:

- current DCW points to the WRITE (command 1), and
- the next data field points to BUF1STRT (data field 1).

The first command is fetched and the write routine is entered, which sets up for the write by fetching a data field. Since the data field indicates a direct buffer, a second data field is fetched. Now the pointers show the following:

- The current DCW pointing to WRITE (command 1)
- The next data field pointing to BUF2 (data field 3)

Before actually writing the data, the driver looks at the next command and finds it to be a read. Therefore, the read code is entered. The read routine sets up for the read by fetching a data field. Since the data field indicates an indirect text buffer, the driver is satisfied. After setting up for the read, the read routine finds that it was entered from a write and returns to write without performing any I/O. The pointers now show the following:

- The current DCW pointing to WRITE (command 1)
- The next data field pointing to data field 4

Write I/O is performed and, on error-free completion, the write termination routine finds a RAW situation, bumps the current DCW pointer and enters the READ code. The pointers now show:

- The current DCW pointing to READ (command 2)
- The next data field pointing to data field 4

After the READ goes to completion, a routine is scheduled to check the command to see if it is chained. Since it is, the current DCW pointer changes to fetch the next command. The pointers now show:

- The current DCW pointing to XFER (command 3)
- The next data field pointing to data field 4

The XFER routine fetches data field 4 (the address of command 1) and changes the DCW pointer to the value contained in that data field, affecting a branch. The next command is fetched again. However, the pointers show:

- The current DCW pointing to WRITE (command 1)
- The next data field pointing to data field 5

The command fetch brings the WRITE routine into action again. It fetches data fields and finds the X'80' data code, indicating that the data field contains the address of another data field. The next data field pointer is updated to the contained value. The pointers now show:

- The current DCW pointing to WRITE (command 1)
- The next data field pointing to BUF1.STRT (data field 1)

The WRITE code again attempts to fetch a data field to use for buffers.

8.6.1.2 Line Driver Data Communications Device Interface

The line drivers provide a simple interface to standard data communications devices and allow enough flexibility for effective control of the particular communications network.

A simple communications network can be controlled by the use of only three commands: READ BUFF, WRITE BUFF and WAIT. Assuming that move commands are not needed to change default parameters, the user merely reads or writes information regardless of communication variables like line speed, character size, line parity, type of line (2-wire, 4-wire, leased), type of adapter, programmable asynchronous single line adapter (PASLA), quad-synchronous adapter (QSA) or type of modem.

While the line drivers do handle the specific operating requirements of the actual communications line interface (computer, adapter and modem), they are unaware of any line discipline or protocol. The user provides the correct handshaking procedure or dialogue with the other end of the line, if any is required, by chaining appropriate WRITE and READ commands.

While complex DCW chains can be constructed, commands are fetched and executed in an interruptible state at the priority of the calling task. Ensure that the task's priority is commensurate with the requirements of the line, especially when using the more sophisticated chained buffers.

Although commands are fetched and interpreted in an interruptible state, special provision is made for the unique sequence of write chained to read. In this situation, after the write is set up, but before actually starting I/O, the driver checks to see if the write is chained to a read. If it is, the read is also set up. This way the driver can turn a line around very quickly and be ready to receive the data regardless of task priority.

Intelligent terminals usually operate with a protocol that requires each end to ACK or NAK receipt of a message. Therefore, to communicate information to such a device, issue a write (containing your message) chained to a read (to accept the ACK or NAK). The driver relocates and sets up the write and read buffers and the write is performed. When write completes, the line is turned around and read is performed. When read completes, the driver has finished the SVC15 request. The status is stored in the user's SVC15 parameter block, and if canceled, a termination trap is generated. Now look at the read buffer and determine whether the message should be repeated (received an NAK) or the next message should be written (received an ACK).

If the user is receiving a data message from the device, the message must be acknowledged to indicate error-free (ACK) or invalid (NAK) reception by the write chained to read sequence.

First issue an SVC15 read to get the message. After SVC15 completes, check the status (and optionally the data) and reply with an SVC15 write chained to read. The write specifies ACK or NAK (reply to the last message), and the chained read readies the driver to receive a new message (if ACK) or a repeat of the message (if NAK).

Once write to read turnaround is done, the driver has one buffer transfer time to set up additional required buffers. If, due to heavy processor loading, these operations are not performed in time, I/O is terminated on detection of the buffer overrun situation and the status is set to reflect this (status=buffer overrun). Take appropriate recovery action (retry I/O).

Assume that the task is an interactive interpreter that, once started, requests input from a remote terminal. Normally the computer writes out a statement number (sequence number) and then initiates a read to the terminal in several ways, using the SVC15 request.

Method 1:

The sequence number is maintained in one buffer.

Buffer 1 is a direct buffer of eight characters, containing two carriage return (CR) characters, one line feed (LF) character, a 4-digit ASCII sequence number and a space. Writing this buffer causes the four sequence numbers to appear on the terminal at the left margin of a new line.

Buffer 2, an indirect buffer of 60 characters, receives the input. By using an indirect buffer, the number of characters actually entered can be determined without scanning the input for special characters. The first supplied halfword indicates to the driver how many bytes are available for data character storage. The actual number of characters read by the driver is stored in the second halfword of the buffer.

A user could issue a write from buffer 1 and then a read into buffer 2. Adequate response time for the read can be assured and system overhead can be saved by issuing only one SVC15 for both the write and read. The SVC15 parameter block and associated data are shown in Figure 8-7.

The function code specifies termination trap. The DCW chain contains the following:

- A WRITE command specifying a chain to the next command when this write is error free.
- A READ command. The read does not have the time-out bit set and is able to wait indefinitely for input. The read is not chained. When this command completes, SVC15 terminates by storing certain values in the parameter block and by generating a termination trap (requested in function code) to the user program.

If the response at the terminal was A=B CR, the parameter block and associated fields might be as shown in Figure 8-7.

Method 2:

Since the interpreter is just one of many programs running in a multitasking environment, unacceptable delays might occur between responses from the task because of higher priority tasks.

To inform a remote terminal of the status of the input attempt, it might be desirable to establish the dialogue as:

- The computer starts a new line and types out a sequence number when it wants input.
- The user types in a line of data and signals the end of input with the ASCII ETX, which leaves the TTY carriage at the end of the input line.
- The computer, on recognizing the end character (ETX), writes out a CR sequence to signal that the computer is still there and running, and that input was received. Any delay between the CR and the next sequence number is due to computation time within the computer.

077-19

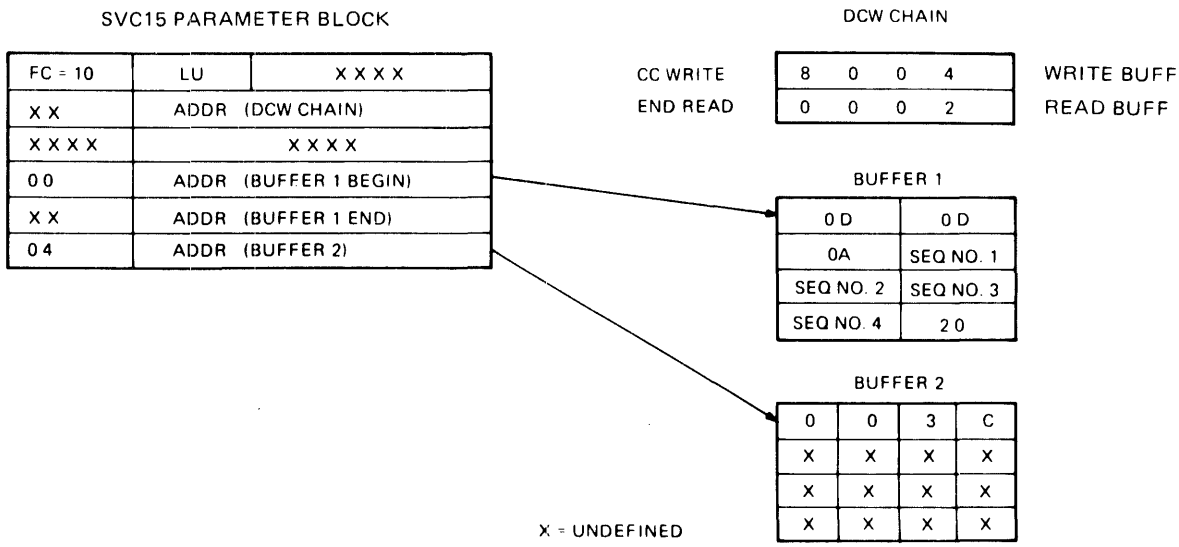


Figure 8-6 Example of an SVC15 Parameter Block and Associated Data

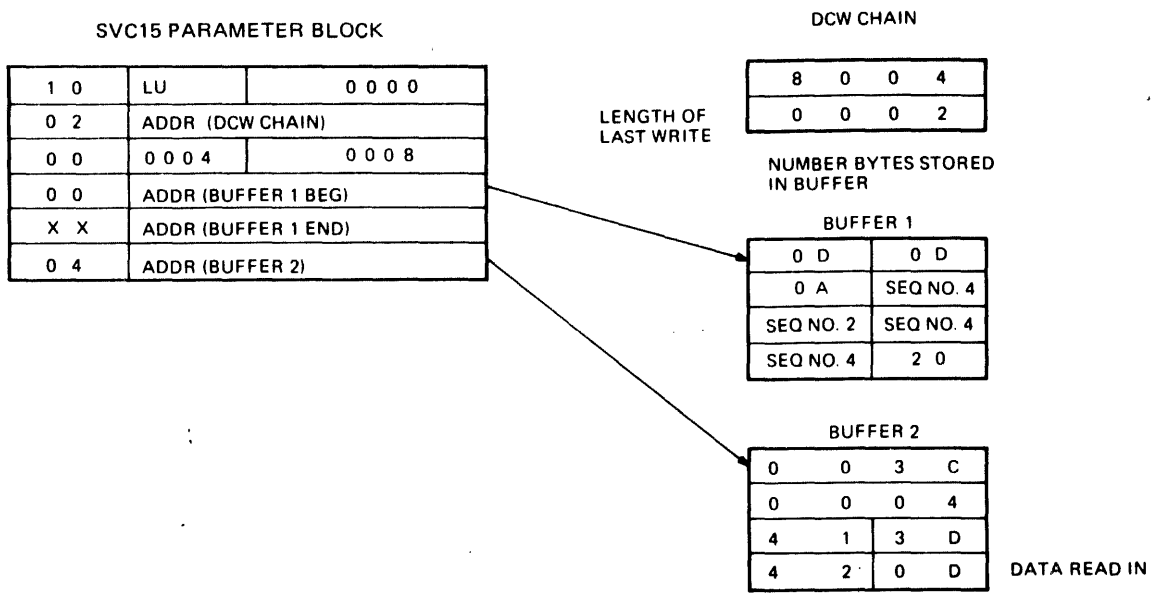


Figure 8-7 Parameter Block and Associated Fields After SVC15 Termination

If the interpreter needs to store the input line image, along with its sequence number on a disk or tape, the program with sequence numbers can be listed when requested. For example:

- Assume buffer 1 is a direct buffer of 72 characters and buffers 2 and 3 are direct buffers that are subsets of buffer 1
- Buffer 2 begins where buffer 1 begins
- Buffer 2 ends where buffer 1 began +7; i.e., buffer 2 is just the first eight characters of buffer 1
- Buffer 3 begins where buffer 1 began +8
- Buffer 3 ends where buffer 1 ends.
- Assume, as in the previous example, that buffer 2 contains a CR, LF and SEQ number space.

By performing a write from buffer 2 and a read into buffer 3, buffer 1 then contains the sequence numbers and the input. These can then be written to a disk or magnetic tape by simply starting past the CR-LF sequence. See Figure 8-8.

Although considerably more complex, the program performs no additional work other than initially setting up the block. The commands (DCWs) specify time-out, which is normally an error time-out used to abort the call should the command not go to completion (possibly due to hardware problems) within the allotted time. Store a null in the first position of a read buffer to determine if the read timed-out before or during data input.

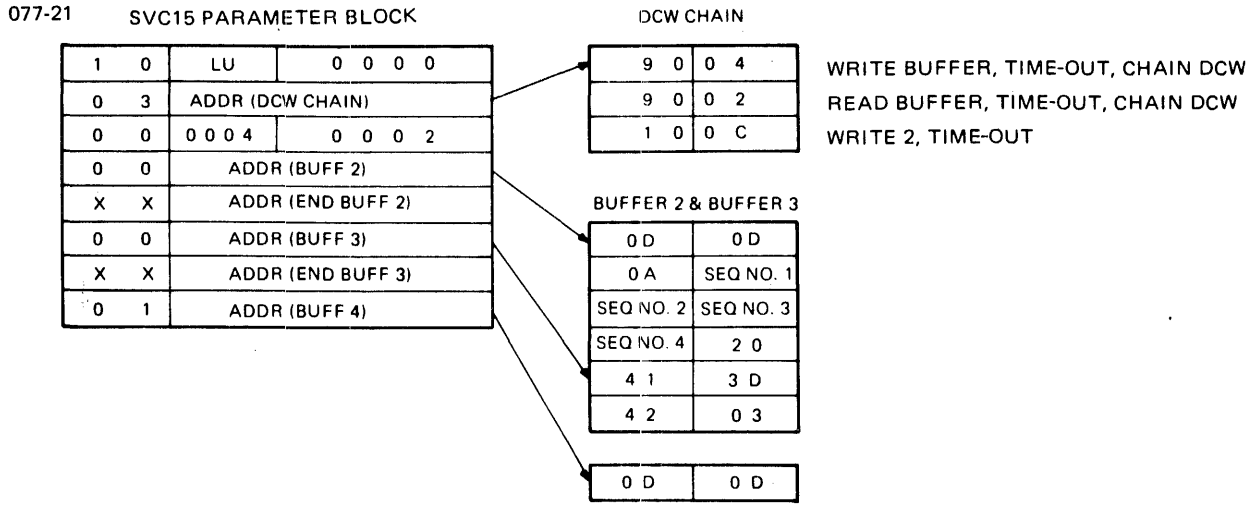


Figure 8-8 SVC15 Parameter Block After Termination

CHAPTER 9
GENERATING AN OPERATING SYSTEM WITH DATA COMMUNICATIONS DEVICES

9.1 INTRODUCTION

An operating system configured with data communications devices is generated using the System Generation/32 (Sysgen/32) program.

Sysgen/32 enables you to create and tailor an operating system to accommodate particular system requirements. Hardware and software features are selected and defined through sysgen configuration statements. These statements form a sysgen configuration input file. Driver and system modules provided in the OS/32 package are selected by Sysgen/32 based on the requirements indicated in these sysgen statements.

You can create a new configuration input file or modify an existing one through Sysgen/32 commands. Once a configuration input file is created, it is processed by the Sysgen/32 program to produce macro calls. These macros are subsequently expanded, assembled and linked to yield the operating system. See the System Generation/32 (Sysgen/32) Reference Manual for a description of sysgen commands.

9.2 DATA COMMUNICATIONS CONFIGURATION STATEMENT

The sysgen ITAM configuration statement is used to configure data communications support in the operating system. Communications support consists of system modules, drivers and device control blocks (DCBs) and channel control blocks (CCBs), etc. The drivers are stored in either the commmunications driver library or extended communications driver library. The system modules are stored in the system communications library.

Every communications device to be configured in the operating system must be defined by a device descriptor statement. The sysgen DEVICES...END statements are used to delimit the device descriptor statements. Each communications device descriptor statement requires three parameters; the device name, the device address and the device code. The three required parameters must be entered in the order described. There are optional parameters describing other device details that can be entered in any order.

Example:

In the following example, an asynchronous communications line is to be configured in the operating system. The first required parameter is the device name. The second parameter is the required device address, in this example, 40. The third required parameter, XD=X0830, is the device code. Note that communications device code specifications are preceded by XD specifying additional device configuration information. This is called the extended device code.

```
LINE:,40,144,XD=X0830,REA=XE1C9,WRI=XE809,PAD=2
```

9.3 SYSTEM LIBRARIES

Sysgen for OS/32 configured with the basic data communications subsystem is performed in the same manner as any other configuration of OS/32. However, the OS/32 Library Loader must first be used to merge the OS/32 System Object Module Library with the Data Communications System Object Module Library and the OS/32 General-Purpose Driver Library with the Data Communications Driver/Terminal Manager Library. Sysgen/32 is used to process the resulting combined driver library. The OS/32 Library Loader is then used to generate the final load module as described in the System Generation/32 (Sysgen/32) Reference Manual.

9.3.1 The Driver Library

The recommended sysgen procedure requires that the driver library processed by Sysgen/32, and the system module library processed by the OS/32 library loader, each reside on a single disk file, magnetic tape or cassette. These libraries must, for an OS/32 system with basic data communications, include object modules from the OS/32 software package and the basic data communications software package. The procedure for creating the combined libraries involves the use of the OS/32 Library Loader facilities for library manipulation. The reader should be familiar with the material contained in the OS/32 Library Loader Reference Manual.

The only restriction on the order of modules in the driver library is that the DCB for each device code precedes the driver for that device. Since all Perkin-Elmer supplied libraries are in this order, the Basic Data Communications Driver/Terminal Manager Library can simply be appended to the existing general purpose driver library by a single DUPE operation.

The combined system library situation is a bit more complex; the required order is that all EXEC object modules precede the data communications module and UBOT be last. Thus, it is necessary to create the combined system library on a third file or tape by duplicating the OS/32 System Module Library to the combined system library, duplicating the Data Communications System Module Library, and finally, copying UBOT from the OS/32 System Module

Library. These procedures are described in detail in the Data Communications Packaging Information Document.

9.3.2 Including User-Written Drivers

To include a nonstandard device in the operating system, it must be defined in the sysgen device statements. The library containing the user-written driver (USERDLIB.LIB) for the device must be specified during the link phase of the Sysgen/32 process.

Use reserved device codes 240-254 to configure a user-written driver in the system.

Each device configured in the system gets an appropriate DCBxxx macro call written to the .MAC output file where xxx is the device code (e.g., DCB39, DCB147, DCB245, etc.). The DCBxxx macro creates the device DCB and external references to the device driver (in DRIVER.LIB or USERDLIB.LIB). The user must create the DCBxxx macro definition and put it in the user USERDLIB.MLB file.

9.3.2.1 Creating the DCBxxx Macro

Creating the DCBxxx macro entails these six intermediate steps:

1. Use MLU32 to get the DCBFORM macro from the SYSGEN32.MLB file to use as the pattern.

2. Make the appropriate changes noted in DCBFORM to create the DCB macro.

3. Save the file as DCBxxx.MAC.

4. Use the MLU32 (Macro Library) Utility to add the DCBxxx macro definition to your USERDLIB.MLB file. This library will be searched by MACRO32 before the SYSGEN32.MLB in the normal sysgen process. Use care when creating definitions of macros with names identical to macro names in other libraries.

5. Use Copy/32 or the LIBLDR Utility to add the driver code to your USERDLIB.LIB file. The USERDLIB.LIB file will be edited by OS/32 Link before the standard DRIVER.LIB file. Therefore, modified Perkin-Elmer drivers that use standard Perkin-Elmer device codes can also be placed in USERDLIB.LIB, thereby preempting the standard Perkin-Elmer driver.

6. Perform a sysgen using the standard SYSGEN.CSS. The USERDLIB.MLB file will be assigned and the DCBxxx definition will be used.

See the DCBFORM macro in the SYSGEN32.MLB file.

APPENDIX A
LINE DRIVER COMMAND SUMMARY

MODE	COMMAND	MODIFIER/ COMMAND BYTE HEX	VALID COMMAND BITS	NUMBER DATA FIELDS	DATA FIELD SPECIFIES
NULL					
	NOP	XX00	|CC|CT| X| X|XXXX|00000|000|	1	Any valid address
	WAIT	XX08	|CC|CT| X| 0|XXXX|00001|000|	1	Halfword
	XFER	XX10	|CC|CT| X| X|XXXX|00010|000|	1	Halfword
	CXFER	XX18	|CC|CT| X| X|XXXX|00011|000|	2	2 halfwords valid address
CONTROL					
	EXAMINE	XX01	|CC|CT| X|TO|XXXX|00000|001|	1	Byte
	RING WAIT	XX09	|CC|CT| X|TO|XXXX|00001|001|		None
	ANSWER	XX11	|CC|CT| X|TO|XXXX|00010|001|		None
	DISCONNECT	XX19	|CC|CT| X|TO|XXXX|00011|001|		None
READ					
	READ BUFFER	XX02	|CC|CT|BT|TO|XXXX|00000|010|	1.2	Buffer
	READ1	XX0A	|CC|CT BT|TO|XXXX|00001|010|	1	Byte
	READ2	XX12	|CC|CT|BT|TO|XXXX|00010|010|	2	Byte
PREPARE					
	PREP	XX03	|CC|CT| X|TO|XXXX|00000|011|	1	Byte

MODE	COMMAND	MODIFIER/ COMMAND BYTE HEX	VALID COMMAND BITS	NUMBER DATA FIELDS	DATA FIELD SPECIFIES
WRITE					
	WRITE BUFFER	XX04	|CC|CT|BT|TO|XXXX|00000|100|	1.2	Buffer
	WRITE1	XX0C	|CC|CT|BT|TO|XXXX|00001|100|	1	Byte
	WRITE2	XX14	|CC|CT|BT|TO|XXXX|00010|100|	2	Byte
HOLD					
	BREAK	XX05	|CC|CT| X|TO|XXXX|00000|101|	1	Halfword
	TOUT	XX06	|CC| X| X| X|XXXX|00000|110|	1	Fullword
	CMD2	XX0E	|CC| X| X| X|XXXX|00001|110|	1	Byte
	RCMD	XX16	|CC| X| X| X|XXXX|00010|110|	1	Byte
	WCMD	XX1E	|CC| X| X| X|XXXX|00011|110|	1	Byte
	RDIS	XX26	|CC| X| X| X|XXXX|00100|110|	1	Byte
	WDIS	XX2E	|CC| X| X| X|XXXX|00101|110|	1	Byte
	DISK	XX36	|CC| X| X| X|XXXX|00110|110|	1	Byte
	SYCT	XX3E	|CC| X| X| X|XXXX|00111|110|	1	Byte
	TRNSL	XX46	|CC| X| X| X|XXXX|01000|110|	1	Byte
	SPEC CHAR	XX4E	|CC| X| X| X|XXXX|01001|110|	1	Fullword
	TRECS	XX56	|CC|CT| X| X|XXXX|01010|110|	1	Halfword

Default values are assembled in the DCB.

APPENDIX B
INTERFACE SIGNAL DEFINITIONS

The following signals, defined by EIA Standard RS-232C, are used by the 103 and 201 series modems and are supported by the basic data communications subsystem.

SIGNAL (RS-232C DESIGNATION)	PIN	COMMENTS
Transmit data (BA)	2	Serial data sent from adapter to modem.
Received data (BB)	3	Serial data received by adapter from modem.
Request to send (CA)	4	Set by adapter when user program wishes to transmit.
Clear to send (CB)	5	Set by modem when transmission can commence.
Data set ready (CC)	6	Set by modem when it is powered on and ready to transfer data in response to data terminal ready (CD).
Carrier detect (CF)	8	Set by modem when signal present.
Data terminal ready (CD)	20	Set by adapter to enable modem to answer an incoming call on a switched line. Reset by adapter to disconnect call.
Ring indicator (CE)	22	Set by modem when telephone rings.

INDEX

A

Access, device-dependent 2-10, 5-1
  access, line driver 5-1
Access, device-independent 2-5, 4-1
  LCB 4-2
  nonbuffered access 4-1
  operation 4-2
Access, line driver 5-1
  sequence of operations 5-2
Access, terminal manager 4-1
  buffered access 4-2
Adapters
  asynchronous 2-2
  bisynchronous adapters 2-3
  EDLC
  parallel 2-2
  serial 2-2
  ZBID adapters 2-4
Add to task queue 6-71
Allocate
  function 3-6
  routine 6-72
ALLOCATE command 3-11
American standard code for information interchange. See ASCII.
ANSWER command 5-26
ASCII 1-8
Assign
  function 3-6
  routine 6-73
ASSIGN command 3-13
Asynchronous adapters 2-2
  MPC 2-3
  MUX, 2-line 2-3
  MUX, 8-line 2-3
  PASLA 2-3
Asynchronous mode 1-6

B

Baudot code 1-8
Bisynchronous adapter
  QSA 2-4
  SELCH 2-4
  SSA 2-4
  synchronous 201 data set 2-3
Buffer control 7-13
Buffer types SVC15 5-14
  chained buffers 5-16
  chained/queued buffer link word flag byte 5-17
  direct buffers 5-15
  indirect buffers 5-15
  queued buffers 5-19
Buffered terminal manager
  input 7-5
  output 7-6

C

CA/CD 1-3
Cancellation I/O 6-71
CCB
  device-dependent portion 6-27
  device-independent portion 6-26
Change access privileges 6-74
Change access privileges function 3-7
Channel control block. See CCB.
Checkpoint
  function 3-9
  routine 6-74
Close
  function 3-8
  routine 6-73
CLOSE command 3-15
CMEXIT subroutine 6-61
CMTERM subroutine 6-60
Collision avoidance/collision detection
Command
  fetch 8-7
  modifier routines 8-8
  number field SVC15 5-12
  table 8-6
Command functions
  allocate 3-6
  assign 3-6
  change access privileges 3-7
  checkpoint 3-9
  close 3-8
  delete 3-8
  fetch attributes 3-9
  rename 3-10
  reprotect 3-10
  VFC 3-9
Communications methods 1-2
  CA/CD 1-3
  polling 1-3
  selection 1-3
Conditional transfer command. See CXFER command
Control block formats 6-2
  DCB data communications related portion 6-7
  device-dependent portion 6-16
  device-independent portion 6-3
Control-type commands 5-25
  ANSWER 5-26
  DISCONNECT 5-26
  EXAMINE 5-25
  READ BUFFER 5-26
  read-type commands 5-26
  READ1 5-26

Control-type commands (Continued)
READ2 5-27
RING WAIT 5-25
CSMA/DC 2-2
CXFER command 5-24

D

DASY driver 2-5
DAT for asynchronous multidrop communications 6-38
Data block descriptor, LCB 6-25
Data codes
ASCII 1-8
Baudot 1-8
EBCDIC 1-8
Data communications 1-1
analog 1-2
distributed processing networks 1-3
networks 1-2
subsystem 2-1
Data communications subroutines
ITSRABS 6-60
Data fields SVC15 5-13
Data set link control transmission modes. See SDLC transmission modes.
Data transmission modes. See modes, data transmission
DCB data communications 6-2
DCB data communications related portion 6-7
DCB line driver use 8-4
DCB pointers 6-52
DCB.AOC 6-54
DCB.DOCR 6-53
DCB.DOCW 6-53
DCB.INIT 6-54
DCB.ITV 6-54
DCB.MOCR 6-53
DCB.MOCW 6-53
DCB.OTV 6-54
DCB.RDN 6-54
DCB.TERM 6-53
DCB.WDN 6-54
DCB/LCB references 7-14
DCSY driver 2-5
DCT for asynchronous multidrop communications 6-36
DCT for ZDLC communications 6-30
DCW 5-21
DCW pointer SVC15 5-12
DDT for 3270 Emulator 6-44
DDT for ZDLC communications 6-34
Delete
function 3-8
routine 6-72
DELETE command 3-16
DETH driver 2-5
Device control block. See DCB data communications. 6-2
Device definition table. See DDT.
Device-dependent portion
CCB 6-27
DCB 6-16
Device-dependent portion LCB 6-22
Device-independent portion
CCB 6-26
DCB 6-3
LCB 6-16
Device mnemonic field 3-5
DISCONNECT command 5-26
Distributed processing networks 1-3
Driver command work. See DCW.
Driver initiation routine 8-5
Driver library 9-2
DCBxxx macro creation 9-3
user-written drivers 9-3
Driver-termination phase 8-13
Drop access table. See DAT.
Drop control table. See DCT.
Drop definition table. See DDT. 6-34
DZBD driver 2-5

E

EBCDIC 1-8
EDLC
Error handling
CRC 2-9
LRC 2-9
VRC 2-9
Error status field 3-3
ESR 6-54
Ethernet data link controller. See EDLC.
Ethernet network 2-2
Event service routine. See ESR.
EXAMINE command 5-25
Extended binary coded decimal interchange code. See EBCDIC.
Extension field 3-5
EXTRN/ENTRY references 7-14
BEBC.ASC 7-15
CLOSMBSC 7-15
CPT.RJE 7-15
III 7-15
INITMBSC/FUNCMBSC 7-15
IODONE/IODONE2 7-15
ISPTAB 7-15
ISSEXEC 7-15
IT.HALT 7-15
JOURNAL 7-15
TMREMW 7-15
TOCHOFF 7-15
TOCHON 7-15
TWT.RJE 7-15
WAIT 7-15
7-15

F,G

Fetch attributes function 6-74 3-9
File manager handler. See FMH.
File size field 3-6
Filename field 3-5
FMH 6-50
Format control 7-12
Full-duplex mode 1-5
Function code field 3-3
Functions, terminal manager
BLOCK A 7-7
BLOCK AA 7-11
BLOCK AB 7-12
BLOCK AC 7-12
BLOCK B 7-7
BLOCK C 7-7
BLOCK D 7-7
BLOCK E 7-7
BLOCK F 7-8
BLOCK G 7-8
BLOCK H 7-8
BLOCK I 7-8
BLOCK J 7-8
BLOCK K 7-9
BLOCK L 7-9
BLOCK M 7-9
BLOCK N 7-9
BLOCK O 7-9
BLOCK P 7-10
BLOCK Q 7-10
BLOCK R 7-10
BLOCK S 7-10
BLOCK T 7-10
BLOCK U 7-10
BLOCK V 7-11
BLOCK W 7-11
BLOCK X 7-11
BLOCK Y 7-11
BLOCK Z 7-11
buffer control 7-13
format control 7-12
special functions 7-12
time-out control 7-12

H

Half-duplex terminal 1-5
Hard-wired modems 1-9
Hold space (line break) command 5-28
Hold-type commands
hold space (line break) 5-28

I,J,K

ICMDINT subroutine 6-67
Input/output block. See IOB.
Input/output handler. See IOH.
Interrupt service routine. See ISR.
IOB for asynchronous multidrop communications 6-40
IOH 6-47
ISR 8-9
ISSEXEC subroutine 6-62
IT..STOP subroutine 6-64
ITGETBUF subroutine 6-70
ITGETDAT subroutine 6-69
ITGETMOD subroutine 6-68
ITGETMOD2 subroutine 6-68
ITIMLINK subroutine 6-66
ITIMUNLK subroutine 6-66
ITISPOTC subroutine 6-67
ITISSTOP subroutine 6-64
ITISTOTC subroutine 6-67
ITSABS subroutine 6-60
ITSETREA subroutine 6-64
ITXFRISR subroutine 6-64

L

LAN 2-2
LCB 4-2
6-16
data block descriptor portion 6-25
device-dependent portion 6-22
device-independent portion 6-16
Leased lines 1-10
Length of last read field SVC15 5-12
Length of last write field SVC15 5-12
Line control block. See LCB.
Line driver
data communications 6-1
example 8-6
Line driver command types SVC15 5-23
control-type 5-25
CXFER 5-24
hold-type 5-28
mode-type 5-29
NOP 5-24
null-type 5-24
prepare-type 5-27
test-type 5-30
WAIT command 5-24
write-type 5-27
XFER 5-24
Line drivers
buffer management 8-14
chained buffers 8-18
command fetch 8-7
command table 8-6
command/modifier routines 8-8
DASY 2-5
DCB line driver use 8-4
DCSY 2-5
DETH 2-5
driver-termination phase 8-13

Line drivers (Continued)
  DZBD 2-5
  ISR 8-9
  line driver device interface 8-20
  modifier fetch 8-8
  modifying 8-1
  RAW turnaround 8-12
  special character routines 8-11
  structure 8-5
  use 8-13
Lines
  multidrop 1-2
  point-to-point 1-2
Local area network. See LAN.
Logical record length field 3-5
Logical unit. See lu.
lu field 3-5

M

MASY nonbuffered terminal manager 2-8
MBSC buffered terminal manager 2-8
MMSM nonbuffered terminal manager 2-8
MODE CMD2 (adapter) command 5-29
MODE DISC command 5-30
MODE RCMD and MODE WCMD commands 5-30
MODE RDIS and MODE WDIS commands 5-30
MODE SPCHAR command 5-30
MODE SYNCNT command 5-30
MODE TOUT (time-out interval) command 5-29
MODE TRANSL command 5-30
Mode-type commands
  MODE CMD2 (adapter) 5-29
  MODE DISC 5-30
  MODE RCMD and MODE WCMD 5-30
  MODE RDIS and MODE WDIS 5-30
  MODE SPCHAR 5-30
  MODE SYNCNT 5-30
  MODE TOUT (time-out interval) 5-29
  MODE TRANSL 5-30
Modems
  hard-wired modems 1-9
  voice grade 1-9
  wideband 1-9
Modes, data transmission
  asynchronous mode 1-6
  synchronous mode 1-7
  ZBID 2-4
Modes, terminal communications 1-4
  full-duplex mode 1-5
  half-duplex terminal 1-5
  simplex terminal uses 1-5
Modifier fetch 8-8
MPC 2-3
Multidrop line 1-2
Multiperipheral controller. See MPC.
Multiplexors See MUXs.
MUXs 1-11

N

Networks
  data communications 1-2
  Ethernet 2-2
  LAN 2-2
  PENnet 2-2
No operation command. See NOP command.
Nonbuffered terminal manager 7-2
NOP command 5-24
Null-type commands 5-24

O

Operation, sequence of 4-2
Operator commands 3-10
  ALLOCATE 3-11
  ASSIGN 3-13
  CLOSE 3-15
  DELETE 3-16
  XALLOCATE 3-17
  XDELETE 3-19

P

Parameter block 3-1
  4-4
  command functions 3-6
  device mnemonic field 3-5
  error status field 3-3
  extension field 3-5
  file size field 3-6
  filename field 3-5
  function code field 3-3
  logical record length field 3-5
  lu field 3-5
  read and write key fields 3-5
Parameter block SVC15. See SVC15 parameter bloc
Parity 1-8
PENnet network 2-2
Point-to-point line 1-2
Polling process 1-3
PPSM nonbuffered terminal manager 2-8
PREPARE command 5-27
Prepare-type commands 5-27
  PREPARE 5-27
  PREPARE3 5-27
PREPARE3 command 5-27
Processing, SVC1 6-70
Processing, SVC7
  allocate 6-72
  assign 6-73
  change access privileges 6-74

Processing, SVC7 (Continued)
  checkpoint 6-74
  close 6-73
  delete 6-72
  fetch attributes 6-74
  rename 6-74
  reprotect 6-74
Protocols 2-8
  communications 2-9
  data link control 2-9
  error handling 2-9

Q

QSA 2-4
Quad synchronous adapter. See QSA.

R

RAW 5-18
  8-12
  turnaround 8-12
Read after write. See RAW.
Read and write key fields 3-5
READ BUFFER command 5-26
Read-type commands 5-26
READ1 command 5-26
READ2 command 5-27
Register conventions 7-13
Rename
  function 3-10
  routine 6-74
Reprotect
  function 3-10
  routine 6-74
RING WAIT command 5-25

S

SDLC transmission modes 2-4
SDT for 3270 emulators 6-43
SELCH 2-4
Selection mechanism 1-3
Selector channel. See SELCH.
Simplex terminal uses 1-5
Single-synchronous adapter. See SSA.
Special character routines 8-11
Speeds, terminal 1-6
SSA 2-4
Station description table. See SDT.
Structure
  buffered terminal manager (input) 7-5
  buffered terminal manager (output) 7-6
  data communications 6-1
  line driver initiation routine 8-5
  line driver translation tables 8-6
Subroutine, data communications
  ITISPOTC 6-67
Subroutines
  CMEXIT 6-61
  CMTERM 6-60
  data communications 6-59
  ICMDINT 6-67
  ISSEXEC 6-62
  IT..STOP 6-64
  ITGETBUF 6-70
  ITGETDAT 6-69
  ITGETMOD 6-68
  ITGETMOD2 6-68
  ITIMLINK 6-66
  ITIMUNLK 6-66
  ITISSTOP 6-64
  ITISTOTC 6-67
  ITSETREA 6-64
  ITSRABS 6-60
  ITXFRISR 6-64
  SVC15 6-59
Subsystem, data communications 2-1
Supervisor call 1. See SVC1.
Supervisor call 15. See SVC15.
Supervisor call 7. See SVC7.
SVC1
  data transfer function code 4-7
  format and coding 4-5
  parameter block 4-4
SVC1 extended options field 4-9
  connect 4-10
  disconnect 4-10
  echoplex 4-10
  format/image bit 4-10
SVC1 parameter block 4-4
  buffer end address 4-6
  buffer start address 4-6
  device-dependent status 4-6
  device-independent status 4-6
  extended options 4-6
  extended options field 4-9
  function code 4-5
  length of data transfer 4-6
  lu 4-6
  random address 4-6
SVC15 5-3
SVC15 function code field 5-6
SVC15 lu field 5-7
SVC15 parameter block 5-5
  command number field 5-12
  data fields 5-13
  DCW pointe 5-12
  length of last read field 5-12
  length of last write field 5-12
  SVC15 function code field 5-6
  SVC15 lu field 5-7
  SVC15 status information field 5-8
SVC15 status information field 5-8
SVC15 structure and flow 6-56

SVC15 trap handling
  buffer transfer trap 5-4
  command execution trap 5-4
  halt I/O termination trap 5-4
  termination trap 5-4
SVC7
  parameter block 3-1
Switched lines 1-10
Synchronous mode 1-7
Sysgen, data communications 9-1
  driver library 9-2
  system libraries 9-2
Sysgen conventions
  DCB/LCB references 7-14
  EXTRN/ENTRY references 7-14
  register 7-13
System
  data communications terminals 1-3
  initialization 6-71
  libraries 9-2
System, data communications 1-3
System generation. See Sysgen.

T,U

Terminal communications modes. See modes, terminal communications.
Terminal manager modification 7-1
  nonbuffered terminal manager 7-2
  structure 7-2
Terminal managers 2-6
  MASY 2-8
  MBSC 2-8
  MMSM 2-8
  PPSM 2-8
Terminal speeds. See speeds, terminal
Terminals 1-4
Test-type commands 5-30
Time-out control 7-12
Timer management 6-71
Transfer in command. See XFER command.
Translation tables 8-6
Transmission lines
  leased 1-10
  switched 1-10
Trap handling SVC15. See SVC15 trap handling.

V

Vertical forms control. See VFC function.
VFC function 3-9
Voice grade 1-9

W

WAIT command 5-24
Wideband modems 1-9
WRITE BUFFER command 5-28
Write-type commands 5-27
  WRITE BUFFER 5-28
  WRITE1 5-28
  WRITE2 command 5-28
WRITE1 command 5-28
WRITE2 command 5-28

X,Y

XALLOCATE command 3-17
XDELETE command 3-19
XFER command 5-24

Z

ZBID adapters 2-4
Zero-bit insertion/deletion adapters. See ZBID.

PERKIN-ELMER

PUBLICATION COMMENT FORM

We try to make our publications easy to understand and free of errors. Our users are an integral source of information for improving future revisions. Please use this postage paid form to send us comments, corrections, suggestions, *etc.*

1. Publication number _____

2. Title of publication _____

3. Describe, providing page numbers, any technical errors you found. Attach additional sheet if neccessary.

   _____

   _____

   _____

4. Was the publication easy to understand? If no, why not?

   _____

5. Were illustrations adequate? _____

   _____

   _____

6. What additions or deletions would you suggest? _____

   _____

   _____

7. Other comments: _____

   _____

   _____

From _____ Date _____

Position/Title _____

Company _____

Address _____

_____

_____

'APLE STAPLE

OLD FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 22 OCEANPORT, N.J.

POSTAGE WILL BE PAID BY ADDRESSEE

PERKIN-ELMER

Data Systems Group
106 Apple Street
Tinton Falls, NJ 07724

ATTN:
TECHNICAL SYSTEMS PUBLICATIONS DEPT.

OLD FOLD

TAPLE STAPLE