Part No. 070-4664-00
Product Group 16

4110 SERIES HOST PROGRAMMERS

Please Check for CHANGE INFORMATION at the Rear of This Manual

First Printing AUG 1983
Revised JAN 1984

Tektronix®
COMMITTED TO EXCELLENCE

WARNING

This equipment generates, uses, and can radiate radio frequency energy and if not installed and used in accordance with the instruction manual, may cause interference to radio communications. It has been tested to comply with the limits for Class A computing devices pursuant to Subpart J or Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference when operated in a commercial environment. Operation of this equipment in a residential area is likely to cause interference in which case the users at thier own expense will be required to take whatever measures may be required to correct the interference.

This instrument, in whole or in part, may be protected by one or more U.S. or foreign patents or patent applications. Information provided upon request by Tektronix, Inc., P.O. Box 500, Beaverton, Oregon 97077.

MANUAL REVISION STATUS

PRODUCT: 4110 Series Computer Display Terminals

This manual supports the following versions of this product: Serial Numbers B010100 and up.

REV DATE	DESCRIPTION
AUG 1983	Original Issue. This manual replaces the 4112 Host Programmers Manual (061-2565-01), the 4113 Host Programmers Manual (061-2616-01), and the 4114 Host Programmers Manual (061-2564-01).
NOV 1983	Revised: pages 4-7, 4-11, 10-4, 10-6.
JAN 1984	Revised: page 10-4.

CONTENTS

Section 1	**ABOUT THIS MANUAL**	**Page**
	Who Should Read This Manual and Why	1-1
	What This Manual Is	1-1
	What This Manual Is Not	1-1
	What's in This Manual	1-1
	Instruments This Manual Covers	1-1
	Topics this Manual Covers	1-1
	4110 Series Manuals and What's in Them	1-2
	Common Manuals	1-2
	Terminal Specific Manuals	1-3
	How to Put It All Together	1-3
	Where to Find Information	1-3
	Command Syntax Conventions	1-3
	Pseudocode Syntax	1-3
	Syntax of Examples	1-3
	Literal Portions of Commands	1-3
	How to Learn the Terminal	1-4
	Using Setup Mode	1-4
	Using Vector and Marker Modes From the Keyboard	1-4
Section 2	**AN OVERVIEW OF 4110 SERIES TERMINALS**	
	Preview	2-1
	4110 Series Terminal Architecture	2-1
	Communications Systems	2-1
	Display Systems	2-1
	DVST Display	2-3
	Raster Displays	2-3
	The 4110 Series Terminals	2-3
Section 3	**COMMUNICATIONS**	
	Introduction	3-1
	Standard Mode Communications	3-1
	Preview	3-1
	Concepts And Definitions	3-3
	Setting Communications Parameters	3-3
	Bypass Mode	3-4
	Prompt Mode	3-4
	Handshaking	3-5
	Full- and Half-Duplex	3-5
	Block Mode Communications	3-6
	Preview	3-6
	Concepts and Definitions	3-6
	Overview of Block Mode Communications	3-6
	Using Block Mode in Host Programs	3-6
	Block Mode Parameters	3-7
	Notes on Block Mode Parameters	3-7
	Arming for Block Mode	3-7
	Block Format	3-7
	The Block Header	3-7
	Block Data	3-7
	The Block-End and Block-Continue Characters	3-8
	The Block-Master and Non-Transmittable Characters	3-8
	The Block-Control Characters	3-8
	Block Mode Algorithms	3-13

Section 4 **USING 4110 SERIES TERMINALS** **Page**

Introduction 4-1
Terminal Commands, Reports, and Parameters 4-1
Preview 4-1
Terminal Commands 4-1
Initializing the Terminal and Sending Commands 4-2
Command Parameters 4-3
int Parameters 4-3
Packing an *int* Parameter 4-3
xy Parameters 4-3
Packing *xy* Parameters 4-3
char Parameters 4-3
Complex Parameters 4-4
Packing Complex Parameters 4-4
Terminal Reports 4-4
Parsing an *int* or *intc* Report 4-5
Parsing an *xy* Report 4-5
Parsing Array Reports 4-5
Troubleshooting Hints 4-5
Macros and Macro Expansion 4-6
Preview 4-6
Concepts and Definitions 4-6
Host Macros 4-6
Byte Macros 4-6
Key Macros 4-6
Making the Terminal Execute a Key Macro 4-6
Expanding Macros 4-6
Terminal Parsers 4-7
Preview 4-7
Overview 4-7
Snoopy Mode 4-7
The Tek Parser 4-7
The Implicit Command Modes 4-10
Alpha Mode 4-10
Vector Mode 4-10
Marker Mode 4-10
Explicit Command States 4-10
LCE-T State 4-11
20C State 4-11
Parameter Parsing States 4-11
int Parsing States 4-11
xy Parsing States 4-11
Modes That Affect Parsing 4-12
Ignore Deletes Mode 4-12
Entering Ignore Deletes Mode 4-12
Leaving Ignore Deletes Mode 4-12
Effects of Ignore Deletes Mode 4-12
4115 Coordinate Modes 4-12
Changing Coordinate Modes 4-12
12-Bit Coordinate Mode 4-12
32-Bit Coordinate Mode 4-12
The ANSI Parser 4-12
ANSI Alpha Mode 4-12
LCE-A State 4-12
CSI State 4-12

Section 5 **THE OPERATOR INTERFACE** **Page**

Introduction 5-1
Preview 5-1
The Dialog Area 5-1
Preview 5-1
Concepts and Definitions 5-1
Controlling the Dialog Area 5-1
Dialog Area Parameters 5-2
ANSI Mode 5-2
Introduction 5-2
Preview 5-2
Concepts and Definitions 5-3
Screen Editors 5-3
Cursor Positioning 5-3
Tabulation Commands 5-3
Editing Commands 5-4
Display Control Commands 5-4
Terminal Control Commands 5-4
Communications 5-4
ANSI Sub-Modes 5-4
The Keyboard 5-5
The Display 5-5
Display Control 5-5
Controlling Terminal Responses 5-5
CRLF 5-5
LFCR 5-5
SET-MARGINS 5-5
SET-PAGE-FULL-ACTION 5-6
SET-ECHO 5-6
SET-ERROR-THRESHOLD 5-6

Section 6 **GRAPHICS PRIMITIVES**

Introduction 6-1
Graphics Primitives and Primitive Commands 6-1
Concepts and Definitions 6-1
Explicit and Implicit Commands 6-1
Vectors 6-2
Preview 6-2
Concepts and Definitions 6-2
Line Attributes 6-2
Explicit MOVE and DRAW Commands 6-3
Implicit MOVE and DRAW Commands 6-3
Leaving Vector Mode 6-3
Hint 6-3
Markers 6-3
Preview 6-3
Concepts and Definitions 6-3
Markers 6-3
Marker Types 6-3
The Explicit DRAW-MARKER Command 6-3
The Implicit DRAW-MARKER Command 6-4
Uses of Markers 6-4
Hints 6-4

Section 6 (cont) **Page**

Text in the Graphics Area 6-4
Preview 6-4
Concepts and Definitions 6-4
Alphatext 6-4
Alphatext Attributes 6-4
Graphtext 6-5
Graphtext Precision 6-5
String Precision Graphtext 6-5
Stroke Precision Graphtext 6-5
Defining a Graphtext Font 6-5
Initialization 6-5
Graphtext Character Definition 6-6
Hints 6-6
Eliminating Character Definition Display 6-6
Saving a Graphtext Font on Disk 6-6
Panels 6-6
Preview 6-6
Concepts and Definitions 6-6
Panels 6-6
Panel Boundary 6-7
Rectangle Boundaries 6-8
Panel Attributes 6-8

Section 7 **SEGMENTS**

Introduction 7-1
An Introduction To Segments 7-1
Preview 7-1
Concepts and Definitions 7-1
What Is a Segment? 7-1
Retained and Non Retained Segments 7-1
Segment Numbering 7-2
Segment Attributes 7-2
The Pivot Point and the Segment Origin 7-2
Dynamic Segment Attributes 7-3
Use of Segments 7-4
Building Segments 7-5
Preview 7-5
The Segment Definition 7-5
Setting Attributes for Future Segments 7-5
Opening the Segment Definition 7-6
Closing the Segment Definition 7-7
Appearance of the Display When Defining a Segment 7-7
Commands That Are Not Part of a Segment Definition 7-7
Segment Classes and Matching Classes 7-7
Preview 7-7
Concepts and Definitions 7-7
Segment Class Field 7-7
Current Matching Class 7-8
An Example Using Segment Classes 7-9
The Scenario 7-9
Defining the Segment Class Subfields 7-9
Setting the Segment Class Field 7-9
Using the Current Matching Class 7-10

Section 8 **Raster Display Graphics** **Page**

Introduction 8-1
The Raster Display 8-1
Introduction 8-1
Concepts and Definitions 8-1
Raster Memory Buffer 8-3
Color Indices 8-4
Surfaces 8-6
Preview 8-6
Concepts and Definitions 8-6
Writing on a Surface 8-6
Defining a Surface 8-6
Displaying Surfaces 8-6
Surface Priority 8-8
Using Surfaces 8-8
Surfaces and the Dialog Area 8-9
Number of Surfaces 8-9
The Super Surface 8-9
Color 8-10
Preview 8-10
Introduction 8-10
Concepts and Definitions 8-10
Color Indices 8-10
Color Map 8-10
Selecting Color Coordinate Systems 8-10
Background Colors 8-16
Background Indices 8-16
Hint 8-16
Pixels 8-16
Preview 8-16
Concepts and Definitions 8-16
Pixel Operations 8-16
Pixel Viewport and Pixel Beam Position 8-16
The RASTER-WRITE Command 8-16
The RUNLENGTH-WRITE Command 8-17
ALU Modes 8-18
The RECTANGLE-FILL Command 8-18
The PIXEL-COPY Command 8-18
User-Defined Fill Patterns 8-18
The BEGIN-FILL-PATTERN Command 8-18
The END-FILL-PATTERN Command 8-18
An Example of a User-Defined Fill Pattern 8-19
Views 8-19
Preview 8-19
Concepts and Definitions 8-20
Space 8-20
Views 8-20
Window 8-21
Viewport 8-22
Creating and Selecting Views 8-23
Using Views 8-23
The Terminal Viewing Keys 8-23
The Overview and Home Position in 4115 Terminals 8-23
Fixup Level 8-23
Hint 8-23
View Display Cluster 8-23

Section 9	GRAPHICS INPUT	Page
	Introduction	9-1
	Preview	9-1
	Concepts and Definitions	9-1
	GIN Functions	9-2
	Locate Function	9-2
	Pick Function	9-2
	Stroke Function	9-2
	GIN Space	9-2
	GIN Windows and Areas	9-4
	GIN Function Reports	9-5
	Parsing GIN Function Reports	9-5
	GIN Commands	9-5
	Enabling and Disabling GIN	9-5
	Setting GIN Parameters	9-6
	Hints and Examples	9-6
	Picking and Dragging	9-6
Section 10	THE TERMINAL FILE SYSTEM	
	Introduction	10-1
	The Terminal File System	10-1
	Preview	10-1
	Concepts and Definitions	10-1
	Devices and Device Names	10-2
	Types of Devices	10-2
	File Names	10-2
	General File Transfer Operations	10-2
	Devices and Commands in the Standard Terminal	10-2
	Standard Terminal Devices	10-2
	Standard Terminal Commands	10-2
	Hints	10-3
	Local Disk Storage	10-4
	Preview	10-4
	Disk Devices and Commands	10-4
	Devices and Device Names	10-4
	Disk Commands	10-4
	Three-Port Peripheral Interface (3PPI)	10-5
	Preview	10-5
	Devices and Commands	10-5
	3PPI Commands	10-5
	The Color Hard Copier	10-6
	Preview	10-6
	Commands and Devices	10-6
	SC:	10-6
	HC:	10-6
	Hints	10-6
	The DMA Interface	10-6
	Preview	10-6
	Commands and Devices	10-6
	Using DMA From the Host Program	10-6
Section 11	SOFTWARE COMPATIBILITY	
	Using 4010 Programs With 4110 Series Terminals	11-1

Appendix A **ASCII CHART**

Appendix B ***INT* PARAMETERS**

Appendix C **CODE EXAMPLES**

Appendix D **COLOR COORDINATE SYSTEMS** **Page**
Color D-1
The HLS Color Cone D-2
RGB and CMY — the Color Cube D-8

Appendix E **CHARACTER INTERPRETATION BY TEK AND ANSI PARSERS**

INDEX

TABLES

Tables	Description	Page
2-1	4110 Series Terminals Display Type and Address Space	2-3
3-1	Meaning of Bits 1 & 2 in Control Byte 1	3-9
5-1	Function Key Codes	5-5
8-1	Pixel Space for Various Terminals	8-20
B-1	Representing Numbers as *INT* Parameters	B-2
E-1	Character Interpretation by TEK and ANSI Parsers	E-1

ILLUSTRATIONS

Figures	Description	Page
2-1	4110 Series Terminal Subsystems	2-2
3-1	4110 Series Communications System	3-2
3-2	Error-Free Block Transmission	3-10
3-3	Occasional Errors in Block Mode	3-11
3-4	Multiple Errors in Block Mode	3-12
4-1	Implicit Command Modes and Explicit Command States	4-8
4-2	Explicit Command States	4-9
6-1	Line Styles	6-2
6-2	Inside of Panels Filled With a Pattern	6-7
6-3	Rectangles Drawn Inside and Outside a Panel Definition	6-8
7-1	Interchanging the Display Priority of Two Segments	7-4
7-2	The First 15 Bits of the Segment Class Field	7-9
7-3	The Segment Class Field for Segment 43	7-9
8-1	Magnified View of Pixels in a Line	8-1
8-2	An Electron Gun Generating a Raster	8-2
8-3	Screen Pixels and Raster Memory	8-3
8-4	Block Diagram of a Monochrome Raster System	8-4
8-5	Block Diagram of a Color Raster System	8-5
8-6	Graphics Drawn on Two Surfaces	8-7
8-7	Surface Priorities	8-8
8-8	The Super Surface	8-9
8-9	The Effect of Changing the Color Map	8-11
8-10	The HLS System Color Cone	8-12
8-11	The RGB Color Cube	8-13
8-12	The CMY Color Cube	8-14
8-13	Interaction of Colors on Different Surfaces	8-15
8-14	A User-Defined Fill Pattern	8-19
8-15	Windows in Terminal Space	8-21
8-16	Two Viewports Mapped to Two Windows	8-22
9-1	GIN Devices Mapped into GIN Space	9-3
9-2	GIN Areas Mapped on GIN Windows	9-4
D-1	Colors Arranged in a Wheel	D-2
D-2	A Color Wheel With Saturation	D-3
D-3	Adding a Gray Scale to the Color Wheel	D-4
D-4	The HLS Color Cone	D-5
D-5	The RGB and CMY Color Coordinates	D-9
D-6	The Combined RGB and CMY Color Cube	D-8
D-7	Gray Scale in a Cutaway Color Cube	D-10

Section 1

ABOUT THIS MANUAL

Welcome aboard. You are about to explore a unique new world of graphics. Opening this manual was the first step. The remainder of this section gives you an overview of what is to come.

WHO SHOULD READ THIS MANUAL AND WHY

If you are a working programmer writing programs for TEKTRONIX 4110B Series Computer Display Terminals (henceforth referred to as the 4110 Series), this manual is designed for you. If you are not a programmer, but hope to learn something about the 4110 Series and general graphics concepts, this manual should help; however, you need some knowledge of programming theory and practice to benefit from the examples in this manual.

WHAT THIS MANUAL IS

This manual, the *4110 Series Host Programmer's Manual*, contains information on graphics concepts and the commands to implement them on 4110 Series terminals. This manual contains informal discussions on various commands and how they interrelate. In addition this manual contains examples which illustrate how to put terminal commands together to accomplish various tasks.

WHAT THIS MANUAL IS NOT

This manual is not a text on programming theory or practice. Examples given in this manual are usually the simplest possible sequences that will accomplish a given task. Be cautious in attempting to use these examples in working programs, as the code is intended to convey concepts, not to work in an actual implementation. Use the examples as a guide in writing your own programs.

This manual is not complete by itself; it does not cover the detailed syntax of commands and their parameters. Information on syntax and parameters is covered in the *4110 Series Command Reference Manual*. You must have a copy of the *4110 Series Command Reference Manual* to use this manual.

WHAT'S IN THIS MANUAL

INSTRUMENTS THIS MANUAL COVERS

This manual supports the following Tektronix terminals:

- 4112B
- 4113B
- 4114B
- 4115B
- 4116B

Although many of the discussions in this manual apply to Tektronix terminals designed before the 4110B Series, this manual does not attempt to cover earlier terminals. You can use this manual with earlier TEKTRONIX 4110 Series terminals, however, some commands may operate differently.

TOPICS THIS MANUAL COVERS

Section 2, *An Overview of 4110 Series Terminals*, covers 4110 Series terminal architecture, a general discussion of communications and display subsystems, and a comparison of 4110 Series terminal features.

Section 3, *Communications*, covers standard and Block mode communications. It discusses how to set communications parameters and contains a set of algorithms illustrating how to use Block mode.

Section 4, *Using 4110 Series Terminals*, discusses terminal commands and reports and command parameters. This section includes a set of algorithms that show how to issue commands, parse terminal reports, and pack and issue parameters from your host program.

Section 4 also contains discussions on macros: how to define them and how to use them.

The most complex discussion in Section 4 is the discussion of the TEK and ANSI command parsers and how they work. This discussion will be of greatest value while trying to debug a program. Going along with the discussion of TEK and ANSI parsers in Section 4, Appendix E contains a table that details how the TEK and ANSI parsers interpret characters from the host.

Section 5, *The Operator Interface*, covers how to control the terminal features with which the operator interacts. This section discusses controlling the dialog area and the display as well as programming and controlling the keyboard. It also covers the ANSI mode commands available on raster display terminals.

Section 6, *Graphics Primitives*, covers the simplest graphics operations the terminal can perform. It covers vectors and Vector mode, markers and Marker mode, text in the Graphics Area, and panels.

Section 7, *Segments*, discusses what segments are and how to define and use them. It also discusses grouping segments into segment classes and using matching classes.

Section 8, *Raster Display Graphics*, discusses those features available only on raster display terminals. It includes discussions on raster display hardware, surfaces, color and gray scales, pixel operations, terminal spaces, and views and view clusters.

Section 9, *Graphics Input*, discusses how to use graphics input devices from your host program. It discusses the types of devices, the functions available, and the reports returned from the terminal.

Section 10, *The Terminal File System*, discusses the terminal's internal file system — how it operates and how you use it. This section covers using local disk storage, the three-port peripheral interface, the color hard-copy interface, and the DMA interface.

Section 11, *Software Compatibility*, describes software packages designed to use 4110 Series terminals to full advantage.

This section also discusses how to emulate 4010 Series terminals with a 4110 Series terminals. As 4110 Series terminals have retained older terminal features, you can (with some modification) use existing software that was written for 4110 Series terminals.

The appendices to this manual contain material for quick reference. They are:

- Appendix A contains an ASCII chart.
- Appendix B contains a list of *int* values (encoded integers) and a method for converting between numbers and *int*s.
- Appendix C contains code implementations of some pseudocode algorithms.
- Appendix D contains a discussion of color and color systems.
- Appendix E contains a chart showing how the TEK and ANSI parsers interpret characters in various modes and states.

4110 SERIES MANUALS AND WHAT'S IN THEM

Each 4110 Series terminal has a number of manuals, some specific to the particular terminal and some common to the entire series.

Common Manuals

In addition to this manual, two others are common to the entire 4110 Series; you should obtain both before attempting to write programs for a 4110 Series terminal:

- The *4110 Series Command Reference Manual* is the primary reference for command syntax. Whenever you are in doubt as to the exact syntax of a command or its parameters, refer to the *4110 Series Command Reference Manual*.
- The *4110 Series Reference Guide* is a pocket size summary of 4110 Series commands and parameters.

Terminal Specific Manuals

Each terminal has a series of manuals that are specific to it alone. These are:

- An Introduction Brochure.
- An Operators Manual.
- A Service Manual.

You should have the Introduction Brochure and the Operator's Manual for each terminal that your program will drive. The Introduction Brochure contains exercises that quickly familiarize you with the terminal. The Operators Manual contains operating information not covered in this manual.

The service manual for each terminal details the hardware of that terminal. Unless you are an engineer or technician, the service manual is of limited value.

HOW TO PUT IT ALL TOGETHER

WHERE TO FIND INFORMATION

Refer to this manual when you need information about graphics concepts, command sequences, how the terminal interprets information, or how to sequence commands to perform a task.

For information about the specific syntax of a command or the parameters for a command, refer to the *4110 Series Command Reference Manual.*

For operating information, such as how to use Setup mode or reset the terminal, refer to the Operators manual for the specific terminal.

COMMAND SYNTAX CONVENTIONS

Command syntax in this manual follows the conventions used in the *4110 Series Command Reference Manual*; see that manual for details.

In all 4110 Series manuals, nonprinting characters are represented in their 2-character mnemonic form; for example, the character escape is represented by ^{E}c. This manual refers to commands by name rather than by their escape sequence.

PSEUDOCODE SYNTAX

This manual presents algorithms as pseudocode rather than as an actual computer language. The body of a loop is marked by indentation. When an indent ends it marks the end of the loop.

Code examples in Appendix C approximate the pseudocode algorithms, although they do not follow them exactly.

SYNTAX OF EXAMPLES

Literal Portions of Commands

Whenever an escape sequence is one that the terminal will accept (either from the host or in Setup mode) the command sequence is shown in bold type. For example, the escape sequence for the command END-SEGMENT, which takes no parameters, is:

^{E}C SC

The command is three characters long. It is the character ^{E}c followed by an uppercase S followed by an uppercase C. Any necessary spaces in the command are represented as ^{S}P.

HOW TO LEARN THE TERMINAL

As a host programmer, you need to learn the terminal's capabilities an limitations. To use these terminals to full advantage, you need to be familiar with their operation.

Although you can experiment with a terminal by writing simple host programs, some people prefer to experiment from the terminal keyboard in Setup and Local mode. If you prefer this method, start by reading the remainder of this section, then work the exercises in the Introduction Brochure.

USING SETUP MODE

Remember that Setup mode syntax differs from the host syntax in several ways. When you work with the terminal in Setup mode, the escape sequence form of Setup mode commands is closer to the host syntax you will use from your host program.

In Setup mode the terminal automatically performs data packing on command parameters. You can enter parameters directly from the keyboard.

The escape sequence form of a Setup mode command is the ^{E}c character immediately followed by the same one- or two-letter sequence as that command given from the host. If that command requires parameters, you *must* follow it with a ^{S}P.

Parameters for some Setup mode commands must be entered as a name. When a command requires this type of a parameter, enter the string as shown in the *4110 Series Command Reference Manual.*

Numeric parameters require delimiters if more than one must be entered. They may be separated either by a ^{S}P or a, (comma). For example, to draw a line from point (100,100) to point (500,600) in Setup mode, use the sequence:

^{E}cLF^{S}P100,100	MOVE to point (100,100)
^{E}cLG^{S}P500,600	DRAW to point (500,600)

When you send an array parameter from the host, the first item of the array is always a count. In Setup mode this count is done for you, so do *not* give a count before the array data. If a command takes more than one array, enclose each array in the characters < >. Type the < character before the beginning of the array and the > character after the end of the array.

In Setup mode, real numbers consist of two numbers. The first is the mantissa, or multiplicand portion, and the second is the power of two, which multiplies the mantissa. For example, to send the value 1.5 as a real number, send the sequence **3,−1** which means 3×2^{-1}.

You must enclose Setup mode strings in delimiters. In most cases, a backslash (\) at the beginning and end of the string is a suitable delimiter.

USING VECTOR AND MARKER MODES FROM THE KEYBOARD

You cannot enter Vector or Marker mode from Setup mode. To experiment with these modes from the keyboard, put the terminal in Local mode.

In Local mode, the terminal responds to characters from the keyboard as though they had been sent from the host; you must encode parameters before sending them. The sequence of characters must be exactly that which the host will send. To send *xy* parameters for a Vector or Marker mode command, you must encode the values into the *HiY, Extra, LoY, HiX, LoX* format.

For example, to enter Vector mode and draw a triangle, put the terminal into Local mode, type the character ^{G}s, encode the vertices of the triangle as *xy* coordinates, and type them. You will not see any feedback on the screen, so enter the characters with caution.

You are ready to begin the introductory exercises if you have not already done so. You'll find that 4110 Series terminals can do much more than this manual covers. Experiment — try the commands and how they work. You'll find that a 4100 Series terminal is a capable and versatile partner.

Section 2

AN OVERVIEW OF 4110 SERIES TERMINALS

PREVIEW

This section covers the similarities and differences between 4110 Series terminals. You will find the following discussions:

- 4110 Series terminal architecture
- Communications subsystems
- Display subsystems
- A comparison of 4110 Series terminal features

4110 SERIES TERMINAL ARCHITECTURE

Each member of the 4110 Series terminals is a composite of many subsystems. These include a communications system, command parsers, and a display system. Differences between the individual terminals are due to differences in these subsystems.

Figure 2-1 shows how the subsystems of a 4110 Series terminal interrelate. The arrows show how commands and data flow between the subsystems of the terminal. Each subsystem shows the sections where it is discussed.

COMMUNICATIONS SYSTEMS

All 4110 Series terminals handle communications in much the same way. The terminal contains a full-duplex, RS-232C port for communicating with the host system. A terminal file system,driven by a microprocessor, controls the flow of information within the terminal and to and from the host port. Information and commands from the host computer are placed in an input queue, then executed or displayed.

The terminal accepts information and commands from an operator via a keyboard that contains several programmable function keys, several command keys, a set of thumbwheels, and a full ASCII keyboard (most keys can be reprogrammed). The terminal sends information to the operator by passing graphic and alphanumeric information to the display system for the operator to view.

An optional interface adds three additional RS-232C ports (under control of the terminal file system) so that the terminal can control local peripherals such as printers or plotters. Another optional interface allows direct memory access between the terminal's memory and a host computer for extremely high speed data transfer. Interfaces for hard copy units allow the operator to save screen images.

Optional disk interfaces allow the terminal to store files on hard disk or floppy disks. The terminal file system includes utility commands that allow either the terminal operator or the host to create, delete, copy, or rename local disk files.

DISPLAY SYSTEMS

Each 4110 Series terminal contains a display system that takes primitive graphics commands and converts them to a display on the terminal screen. 4110 Series terminals include two types of display system: DVST and raster display. Each 4110 Series terminal responds, as nearly as possible, to the same command with the same display. Thus, a command that causes a DVST terminal to draw a diagonal line from the lower left corner to the upper right corner of the screen, causes a raster display terminal to draw a similar line.

In addition, the display system for each terminal can accept and transform commands that use the terminal specific features. Section 8 *Raster Display Graphics* discusses the features available on raster display terminals.

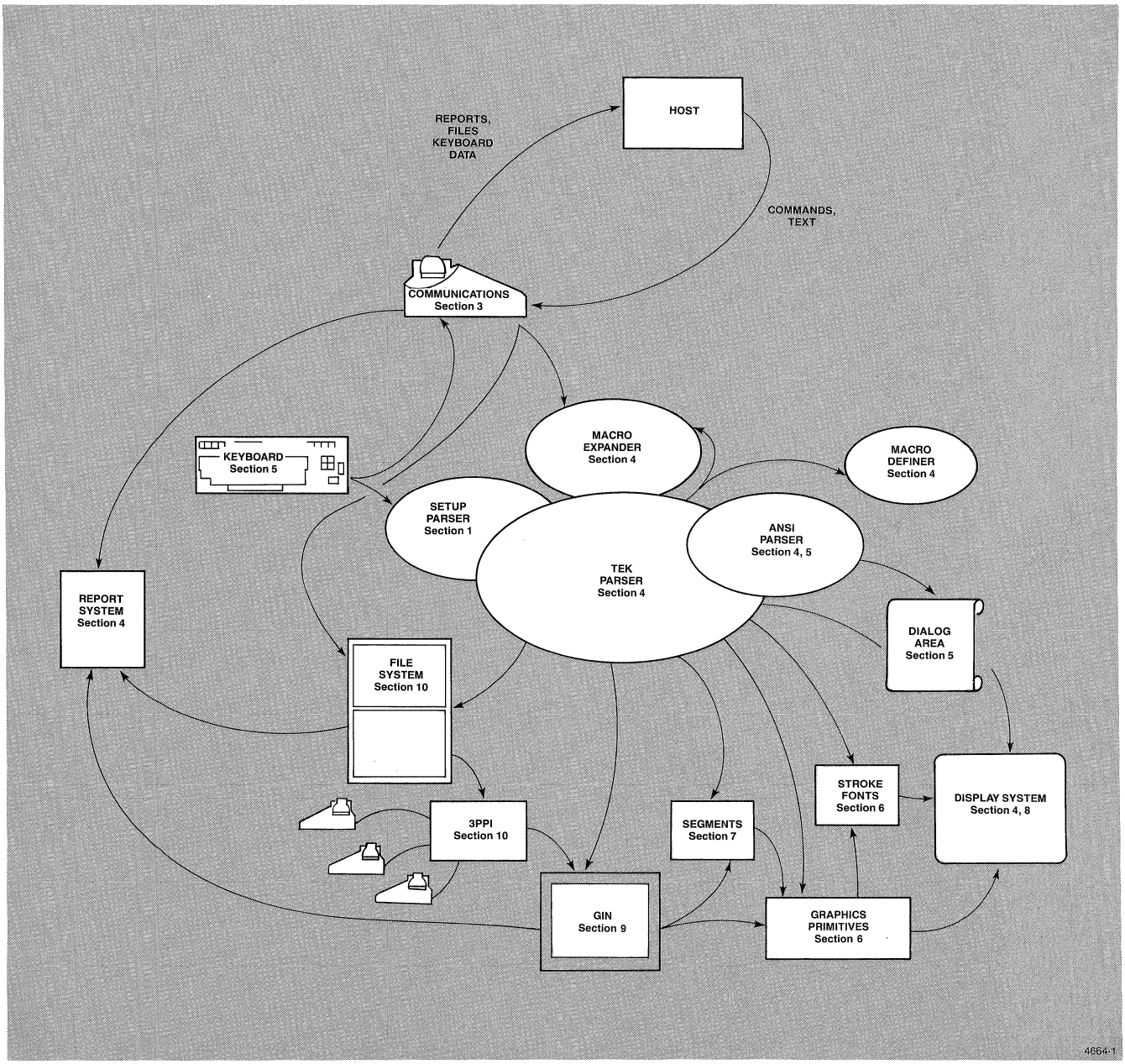


Figure 2-1. 4110 Series Terminal Subsystems.

All terminal display systems support the use of *segments*, graphics constructs that the terminal stores for later use. By using segments, raster terminals can pan and zoom. Both raster and DVST terminals can redraw screens without repeating the commands used to originally draw the display. Section 7 discusses segments in greater detail.

DVST Display

A DVST display is a very high resolution display. Information that is drawn on the screen may be either stored (written until the screen is erased) or refreshed (dynamically drawn and maintained).

You can write any amount of stored information on the screen. Stored information cannot be changed without erasing and redrawing the entire screen. The only limit to the amount of stored information on the screen is the viewer's ability to visually distinguish individual lines.

Refreshed information is displayed on the screen, but must be rapidly redrawn by the terminal to remain visible. Refreshed graphics are easily changed, but take terminal processor time to maintain. The limit for refresh graphics is not absolute, but generally is reached when the terminal cannot redraw the information fast enough to avoid a perceptible flicker on the display.

The display system for a DVST display controls how the terminal draws graphics on the screen. It controls whether the image will be stored or refreshed, and maintains refreshed displays.

Raster Displays

A raster display is much like a television picture. The terminal synchronizes a counter (not accessible to the user) that addresses raster memory locations with a spot of light on the display screen. This spot sweeps horizontally across the screen as the memory counter increments. When the spot reaches the edge of the screen it returns to the other side one line down. When it reaches the bottom of the screen, it returns to the top and repeats the process.

Each location on the display screen is paired with a particular memory location. The spot of light is controlled by the contents of the memory location paired with it. Thus, a picture on the screen is a visual copy of a set of memory.

The display system for a raster display terminal computes and stores the necessary information in the raster memory so that the display hardware will show the desired graphics.

You can access raster memory through *pixel operations*. Section 8, *Raster Display Graphics* examines the raster display and its features in more detail.

THE 4110 SERIES TERMINALS

The following table summarizes the display type and address space of the 4110 Series terminals.

Table 2-1
4110 SERIES TERMINALS DISPLAY TYPE AND ADDRESS SPACE

Terminal	Display	Address Space
4112	13" 640 x 480 Monochrome	4096 x 4096
4113	19" 640 x 480 Color	4096 x 4096
4114	19" 4096 x 4096 DVST	4096 x 4096
4115	19" 1280 x 1024 Color	2^{32} x 2^{32}
4116	25" 4096 x 4096 DVST	4096 x 4096

$2^{32} = 4,294,967,296$

Section 3

COMMUNICATIONS

INTRODUCTION

4110 Series terminals provide an extensive set of communications features that allow them to interface with a wide variety of host computers. Nearly every communications parameter for 4110 Series terminals is programmable. Usually, only a limited number of combinations of settings will work in a particular application. Once you configure the terminal to communicate with the host, leave the parameters set; 4110 Series terminals retain their communications parameters in nonvolatile memory.

CAUTION

The host computer has control of the terminal communications parameters. It can inadvertently reprogram the terminal so that it can no longer communicate, and thus lose control. If this happens, the operator must reprogram the terminal in SETUP mode.

4110 Series terminals communicate in two major modes, Standard mode and Block mode. This section first covers Standard Mode Communications and then covers Block Mode Communications. Figure 3-1 illustrates the 4110 Series communications system.

If you need to use current-loop communications, you must have Option 02. If you need to use one of the half-duplex communications protocols or Block mode, you must have Option 01.

STANDARD MODE COMMUNICATIONS

PREVIEW

- 4110 Series terminals support RS-232C serial communications protocol through programmable communications parameters and various options.
- The standard communications interface is a full-duplex RS-232C port.
- Option 01 supports half-duplex and Block mode communications.
- Option 02 supports current-loop communications.
- Terminal commands set the following communications parameters:
  - Baud rate
  - Parity
  - Stop bits
  - Echo (local or remote)
  - Transmit rate limit
  - Transmit delay
  - EOL string
  - EOF string
  - EOM characters
  - Break time
  - Input queue size
  - Prompt string

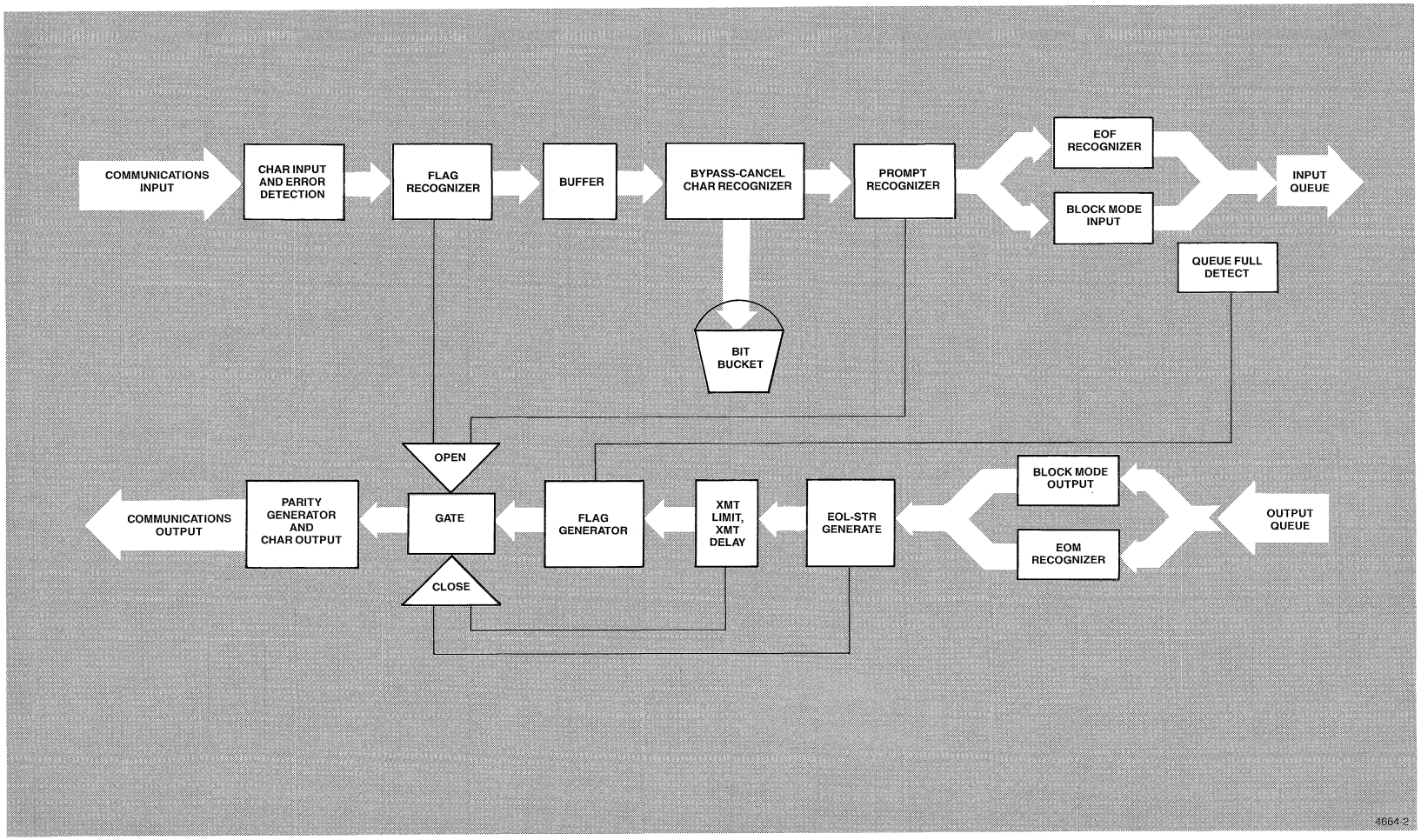


Figure 3-1. 4110 Series Communications System.

- Bypass mode causes the terminal to ignore characters echoed by the host.
- In Prompt mode, the terminal waits for a host prompt before transmitting data.
- You can choose from several types of flagging or handshaking:
  - Character flagging ($^{D}1/^{D}3$)
  - RS-232C signal flagging (DTR/CTS)
  - Message flagging (wait for report from status request)

CONCEPTS AND DEFINITIONS

The standard communications interface on 4110 Series terminals is an RS-232C full-duplex port. 4110 Series terminals communicate using the ASCII 7-bit or 8-bit character set.

Setting Communications Parameters

The communications parameters are retained in the terminal's non-volatile memory. You (or the operator) will normally set the necessary parameters once, and then leave them.

Critical Communications Parameters. Several communications parameters cause the terminal to reset portions of the communications system. These commands must be parsed, passed onto the command processor, and then implemented. Until the communications system has made the changes, incoming characters in the new format cannot be recognized. If you must change a critical communications setting from the host program, follow each change with a delay (1/4 second is usually sufficient). You can delay by simply not sending any characters from the host for 1/4 second or, if that is not possible, send no-op characters such as ^{N}L or ^{S}N for 1/4 second.

The critical settings are:

- Prompt mode
- Prompt string
- Parity
- Stop bits
- EOF string
- Baud rate

Setting Baud Rates. You control the terminals sending and receiving baud rates with the SET-BAUD-RATES command. To issue this command from the host, you must be in communication with the terminal. Unless you can change the host's baud rate to correspond, you will lose communication with the terminal after changing the terminal baud rate.

In addition, should you use this command, do not send the terminal any characters immediately following this command. The terminal can scramble characters sent to it at the wrong baud rate.

4110 Series terminals can operate at baud rates from 50 to 38.4 kilobaud. With high baud rates you may have to use flagging or other handshake methods to prevent the terminal input queue from overflowing.

Setting the Transmit Rate Limit. You can control the average transmitting speed of the terminal with the SET-TRANSMIT-RATE-LIMIT command. If your host is unable to accept characters back-to-back at high baud rates, use this command to reduce the effective baud rate. The terminal will send the individual characters at the high baud rate, then pause between characters long enough to reduce the average baud rate to the value you specify.

Parity. You can use the SET-PARITY command to configure the terminal for your environment.

Number of Stop-Bits. The SET-STOP-BITS command allows you to send either one or two stop-bits, as needed by the host. (The terminal can receive characters with either one or two stop-bits regardless of this setting.)

EOL String. The SET-EOL-STRING command sets the string that the terminal sends to the host at the end of each line of a report or Block mode block. This is usually ^{C}R.

EOF String. The SET-EOF-STRING command sets the string used to define the end of a file sent from the terminal to the host and from the host to the terminal. You should set the EOF string to empty except when actually transferring files.

Break Signal Duration. The SET-BREAK-TIME command sets the duration of a BREAK signal sent to the host by the terminal. 200 milliseconds is standard for most systems.

Input Queue Size. The SET-QUEUE-SIZE command sets the size of the terminal's input queue from 1 to 65535 characters. (The terminal actually allocates memory in 54 byte chunks.) In addition to the programmable input queue, the terminal has a small buffer for holding characters until they can be put in the input queue. When both this buffer and the input queue are filled, the terminal ignores further incoming characters.

With a large input queue, the terminal is less likely to need handshaking. However, the memory allocated to the input queue is not available for other uses. The default input queue size of 300 characters is adequate for most applications.

Controlling Report Length. The SET-REPORT-MAX-LINE-LENGTH command sets the number of characters the report system will send before sending an EOL string. This is useful if the host system can accept only a limited length line. For example, if your program uses a FORTRAN read with an 80 character array to input reports, set the report-max-line-length to 80.

Bypass Mode

Sometimes you want the terminal to ignore characters coming from the host. For example, when the terminal sends a report to the host, if the host echoes these characters, the terminal could print them as alphatext or interpret them as *xy* parameters, depending upon terminal mode.

When the terminal is in Bypass mode, it ignores all characters until it encounters the *bypass-cancel* character. When the terminal receives the bypass-cancel character, it exits from Bypass mode and discards the character.

If the bypass-cancel character is not $^{N}_{L}$, the terminal automatically enters Bypass mode when it sends a report. Since the terminal is in Bypass mode, even though the host echoes the report to the terminal, the terminal does not display the report. If the bypass-cancel character is $^{N}_{L}$, the terminal cannot enter Bypass mode.

You can also explicitly enter Bypass mode with the ENTER-BYPASS-MODE command. (You might want to enter Bypass mode to suppress echoing a password being typed on a terminal.) Due to command processing overhead, the terminal enters Bypass mode a few milliseconds after receiving this command.

Choosing the Bypass-Cancel Character. Use the SET-BYPASS-CANCEL-CHARACTER command to determine which character will bring the terminal out of Bypass mode. Set the bypass-cancel character to be the last character the host echoes when the terminal sends it a report. For example, if the host echoes $^{C}_{R}{}^{L}_{F}$ when it receives a $^{C}_{R}$, set the bypass-cancel character to $^{L}_{F}$. However, if the host echoes only $^{C}_{R}$ set the bypass cancel character to $^{C}_{R}$. If the host does not echo at all, set the bypass-cancel character to $^{N}_{L}$.

Prompt Mode

Some hosts will not accept input from the terminal until they have sent a prompt. Other hosts may work better if your program simulates prompting. Prompt mode operates by opening or closing the terminal's communications output gate. When the output gate is open, the terminal transmits to the host; when the output gate is closed it cannot.

The following actions close the terminal's communications output gate:

- The terminal receives the PROMPT-MODE command with a parameter of 2 (turn Prompt mode on immediately).
- While in Prompt mode and not in Block mode, the terminal sends an EOM character.
- The terminal sends an EOL string — either after a report or, while in Block mode, after a block line.

Any of the following actions open the terminal communications transmit gate:

- The terminal receives the PROMPT-MODE command with a parameter of 0 (turn Prompt mode off).
- The host sends the prompt string, and sends no more characters for a period of time called the *transmit delay*. The characters in Prompt mode must not be separated by more than the transmit delay time.

Controlling Prompt Mode from the Host. The PROMPT-MODE command puts the terminal into or out of Prompt mode. A parameter of 0 disables Prompt mode and opens the output gate, 1 enables Prompt mode but leaves the output gate open, and 2 enables Prompt mode and closes the output gate immediately.

Setting the Prompt String. The SET-PROMPT-STRING command sets the prompt string. The terminal will recognize that string as the prompt from the host when the following three conditions are met:

1. The terminal is in Prompt mode.
2. The output gate is closed.
3. The characters in the prompt string are not separated by more than the transmit delay time.
4. No characters follow the prompt string for the transmit delay time.

If you set the prompt string as a zero length (empty) string, it will always be recognized immediately, effectively disabling Prompt mode. This technique is not recommended.

HANDSHAKING

To prevent the host from sending too much data to the terminal and overflowing the input queue, you should use some form of handshaking. Block mode, discussed elsewhere in this section, automatically provides handshaking. Automatic flagging and frequent reports also provide handshaking. The method you should use depends on your environment.

Character Flagging. You can use the $^{D}1/^{D}3$ protocol to flag messages. With $^{D}1/^{D}3$ flagging, the receiving device sends the $^{D}3$ character when it wants the transmitting device to pause. You can enable $^{D}1/^{D}3$ flagging from the terminal, from the host, or bidirectionally. $^{D}1/^{D}3$ flagging is practical only if your host automatically uses this system.

DTR/CTS Flagging. DTR/CTS flagging is hardware-dependent flagging that uses the DTR (data terminal ready), CTS (clear to send), and RTS (ready to send) and signal lines in the RS-232C interface. This flagging mode is practical only when the host is directly connected to the terminal. When the terminal is connected to the host via a modem, the modem controls the control signal lines and you must use $^{D}1/^{D}3$ or message flagging.

Enabling Flagging. Use the SET-FLAGGING-MODE command to select $^{D}1/^{D}3$ or DTR/CTS flagging. The parameters you give with the command select the type and direction of flagging.

Frequent Reports. Some terminal operations take a relatively long time compared to the commands necessary to cause them. For example, if you command the terminal to load a file from its disk, the terminal will be unable to respond to host commands until the transfer is complete.

When you suspect that the terminal will be tied up for a while by a command of this type, follow the command with a report command such as REPORT-TERMINAL-STATUS or REPORT-4010-STATUS. The command will remain in the terminal's input queue until the operation is complete, then the terminal will send the report. When the host receives the report, you can once again send data.

You can also use this technique to prevent overrunning the terminal during normal communications by keeping track of the number of characters the host sends the terminal. After the host sends enough characters to nearly fill the input buffer, command a report and wait for the reply before sending more characters.

Full- and Half-Duplex

Full-Duplex means that communication is bidirectional. The host and terminal can simultaneously transmit to each other. *Half-Duplex* means that only one or the other can transmit at one time, so the host and terminal must take turns transmitting.

On a terminal without options, you can use only full-duplex. If you have Option 01 installed, you can select one of three half-duplex modes with the SET-DUPLEX-MODE command. The three half-duplex modes are described in the *4110 Series Command Reference Manual*. All other communications and terminal feature operate the same in half-duplex as in full duplex. Your host program should never set the duplex mode, since this could disconnect the terminal from its modem.

BLOCK MODE COMMUNICATIONS

PREVIEW

- You must have Option 01 to use Block mode.
- Block mode is a formal communication protocol located between the standard communications system and the terminal's input and output queues.
- Block mode includes error detection and automatic retransmission of messages on receiving an error.
- Block mode allows the terminal to communicate with a host system that cannot support the full ASCII character set.
- Block mode features automatic handshaking of messages to prevent input buffer overrun.

CONCEPTS AND DEFINITIONS

Overview of Block Mode Communications

Block mode is a nearly symmetrical system of communication between the host and a terminal. The host packs commands, text, and control information into a block, then sends it to the terminal. For each block the host sends the terminal, the terminal sends a similar block to the host, packed with keyboard data and reports if requested.

A count (odd or even) and checksum are packed within each block. The host and terminal use the checksum and odd/even count to check the block contents for errors and signal correct or incorrect transmission.

The host program examines the return block from the terminal to see if the block it sent was received correctly. If the terminal block has the same odd/even count as the block the host sent, and the checksum is correct, the host program assumes that its block was received correctly and that it can send the next block.

If the terminal block does not have the same odd/even count as the block the host sent, or if the checksum does not match, it shows that either the terminal did not receive the host block correctly, or that the host did not receive the terminal block correctly. When the host receives a non-matching block it transmits its block again.

The host controls whether or not the terminal transmits data in the return block. Even if the terminal has date to send to the host (such as a report or keyboard data), the terminal will send data only when the host requests it.

Block data is packed by a method that enables any host to receive and transmit the full 7-bit or 8-bit ASCII character set in blocks, even if the host is normally limited to a subset of the ASCII set.

Using Block Mode in Host Programs

To use Block mode, your host program must perform these functions:

- Set the terminal's Block mode parameters
- Arm the terminal for Block mode
- Handle block transactions

For each block transaction, your program must:

- Send the host block
- Input the terminal block
- Determine whether the terminal block was an acknowledge; if not, repeat the transaction.

In order to send the host block, your program must:

- Compute the host block control bytes
- Pack the block data
- Substitute the non-transmittable characters
- Send the block lines

In order to input the terminal block, the host must:

- Input the block lines
- Translate the master character pairs
- Unpack the block data
- Compute the checksum

Your program should also keep count of retransmissions of bad blocks. If your program made an error in computing a checksum, for example, you would be in an infinite loop. The host would transmit the block, the terminal would not acknowledge it, and the host would transmit the same block again.

BLOCK MODE PARAMETERS

You can use Block mode with full- or half-duplex communications. You can also use Block mode with or without Prompt mode.

You must set the following Block mode parameters before you arm the terminal for Block mode. The terminal stores these parameters in non-volatile RAM.

The following commands set the terminal's Block mode parameters:

- SET-BLOCK-CONTINUE-CHARS
- SET-BLOCK-END-CHARS
- SET-BLOCK-NON-XMT-CHARS
- SET-BLOCK-MASTER-CHARS
- SET-BLOCK-HEADERS
- SET-BLOCK-LENGTH
- SET-BLOCK-PACKING
- SET-BLOCK-TIMEOUT

Notes on Block Mode Parameters

Block Headers. Transmit and receive block headers may be up to 10 characters long.

Block Length. The maximum block length for unpacked data (including the four block control bytes) is set by the SET-BLOCK-LENGTH command. You should set block lengths to four bytes less than your input and output buffer sizes; you must allow for the for block control bytes.

Block Packing. The SET-BLOCK-PACKING command determines how characters will be packed before being sent as packed data. The default 7,6,7,7 works well for most uses. If you need to transfer files with 8-bit bytes, such as to or from a pseudo device or color hard copy device, you can use 8,6,8,7 packing.

Block Timeout. Block timeout is the number of seconds the terminal waits for an ACK block from the host. If this time limit is exceeded, the terminal retransmits its last block. This will help the block exchange to automatically restart if some critical character is lost, such as a block-end character or part of a block header. The block timeout should be longer than the maximum expected host response time.

ARMING FOR BLOCK MODE

To use Block mode from your host, issue the ARM-FOR-BLOCK-MODE command and wait for the command to take effect (REPORT-TERMINAL-SETTINGS for ARM-FOR-BLOCK-MODE is best).

When the terminal is armed for Block mode, it examines incoming data for the special sequence of characters that identifies the beginning of a block. The terminal actually enters Block mode when it recognizes the first block from the host.

BLOCK FORMAT

Each block consists of one or more lines. Each line begins with an identifier, or *block header*, that signals that the following data is a block, continues with the packed block data, and ends with either a block-continue character to indicate that the block is continued on another line, or a block-end character to signal that the block is complete.

The Block Header

The block header is a special sequence of characters that identifies the following character sequence as a block. The terminal, as shipped from the factory, has the block headers preset: the block header for blocks sent by the terminal to the host is the string "HEADTX", the block header for blocks sent from the host to the terminal is the string "HEADRX".

You can alter these strings with the SET-BLOCK-HEADERS command.

Block Data

The packed data contained in a block is in two parts: the first part is the actual data for the block, the second part is four block-control bytes appended to the data. When block data is contained on more than one line, the embedded block header characters and block-continue characters are simply ignored when unpacking data.

Block Control Bytes. The last four bytes of the packed data are block control bytes, appended to the block by the sender. These four bytes are used for error detection and control. The contents and use of the block control bytes are discussed later in this section.

The Block-End and Block-Continue Characters

Each line of a block is terminated by either a *block-continue character*, a character that informs the receiver that the line will be followed by more lines, or a *block-end character*, a character that informs the receiver of the block that the block has ended. You can alter these characters with the commands: SET-BLOCK-CONTINUE-CHARS and SET-BLOCK-END-CHARS.

The Block-Master and Non-Transmittable Characters

Any character that has another meaning cannot appear in the packed block data. These include the block-continue, block-end, and block-master characters as well as any other characters absorbed by the communications system, such as $^{D}1$ and $^{D}3$ when character flagging is in use. Any character that your host cannot input as a normal character, such as ^{C}R, ^{E}C, or ^{N}L must also be excluded from the packed block data.

To exclude characters, include them in a SET-BLOCK-NON-XMT-CHARS command. When the excluded characters appear in your packed block data, replace them with a pair of characters — the block-master character and substitute character.

The substitute character to use for any non-transmittable character depends upon the position of the non-transmittable character in the parameter of the SET-BLOCK-NON-XMT-CHARS command; the first non-transmittable character substitute is A, the second B, and so forth.

For example: if the block-master character were "#", the block-end char were "$", and the block-continue character were "&"; you would set these three characters as non-transmittable characters by including them, in order, as parameters to the SET-BLOCK-NON-XMT-CHARS command. Pack the data and check it for the forbidden characters. If you find the character "#" in the packed data, replace it with "#A", replace "$" with "#B", and "&" with "#C."

You can reduce the number of non-transmittable characters by packing the block data into 6-bit format, which uses only generally transmittable characters: ^{S}P through "_".V

The Block-Control Characters

Your program must compute and include four block-control characters at the end of the unpacked data in each block. After your program receives a block from the terminal, it must interpret the final four characters after unpacking the data.

If the unpacked byte size (as set by the SET-BLOCK-PACKING command) is seven, then each block-control character is seven bits long. If the unpacked byte size is eight, then each block-control character is eight bits long, with the most significant bit set to 0.

Block-Control Character 1. The bits of the first block-control character (with Bit 1 as least significant) have the following meaning:

- Bits 1 and 2 — Block count and end protocol
- Bits 3, 4, 5 — Reserved (always 0)
- Bit 6 — End of File
- Bit 7 — End of Message
- Bit 8 — Not present (7 bits) or 0 (8 bits)

Bit 1 is the block count bit if bit 2 is 0.

If bit 2 is 1, the terminal will exit block mode; the terminal will transmit an acknowledge block if bit 1 is zero, but not if bit 1 is one.

The four possible meanings for Bits 1 and 2 are summarized in Table 3-1. The terminal expects the first block it receives after arming for Block mode to have both bit 1 and bit 2 set to zero.

Bit 6, the EOF bit, is set to 1 at the end of a file transfer. This bit is analogous to the EOF string used when the terminal is not in Block mode. This bit can be set when there is data in the block. This data will considered to be the last data in the file.

Bit 7, the EOM bit has different meaning in host blocks and terminal blocks. You should set the EOM bit to 0, to indicate that the terminal should send an ACK block immediately, whether or not it has data to pack into the block. (This ACK block will consist of only the four block-control bytes) When you set this bit to 1, the terminal will not send a block until it has a block full of characters or encounters an EOM character or EOM indicator. You must set this bit to 1 to input reports, files, or keyboard data.

Table 3-1

MEANING OF BITS 1 & 2 IN CONTROL BYTE 1

Bit 2	Bit 1	Meaning
0	0	This is an even block and the terminal will remain in Block mode
0	1	This is an odd block and the terminal will remain in Block mode
1	0	From host to terminal — Exit from Block mode. From terminal to host — Acknowledge command to leave Block mode.
1	1	From host to terminal — Exit Block mode immediately, do not send ACK to host. Terminal to host — Not allowed

In blocks sent from the terminal to the host, bit 7 is set to 1 when the terminal encounters an EOM character or an EOM indicator in the unpacked data. This bit is 0 when the maximum block length is reached or another block follows containing more of the message. In general, you program can ignore this bit in the block sent from the terminal.

Block-Control Character 2. All the bits of block-control character 2 are unused. The terminal sends a $^{N}_{L}$ for this character, and your program can do the same.

Block-Control Characters 3 & 4. These two characters are the checksum, which is computed by the following algorithm:

1. **Initialize (max-byte), (c-char3), and (c-char4) to have each bit a one. (7-bit characters to a value of 127, 8-bit characters to a value of 255)**
2. **For each character, (char), up to block-control character 2:**

   (c-char3) = (c-char3) + (char)
   If (c-char3) > (max-byte)
   (c-char3) = (c-char3) − (max-byte)
   (c-char4) = (c-char4) + (c-char3)
   If (c-char4) > (max-byte)
   (c-char4) = (c-char4) − (max-byte)
3. **(c-char3) = (max-byte) − (c-char3) − (c-char4)**
   If c-char3 <= 0
   (c-char3) = (c-char3) + (max-byte)

Figure 3-2 illustrates how the host and terminal communicate in Block mode with no errors. The host transmits three blocks, the terminal acknowledges each and communications are unhindered.

Figure 3-3 shows the effect of an occasional error in block mode transmission. In this case the host simply retransmits the block that the terminal failed to acknowledge and receives an acknowledgement on the second try.

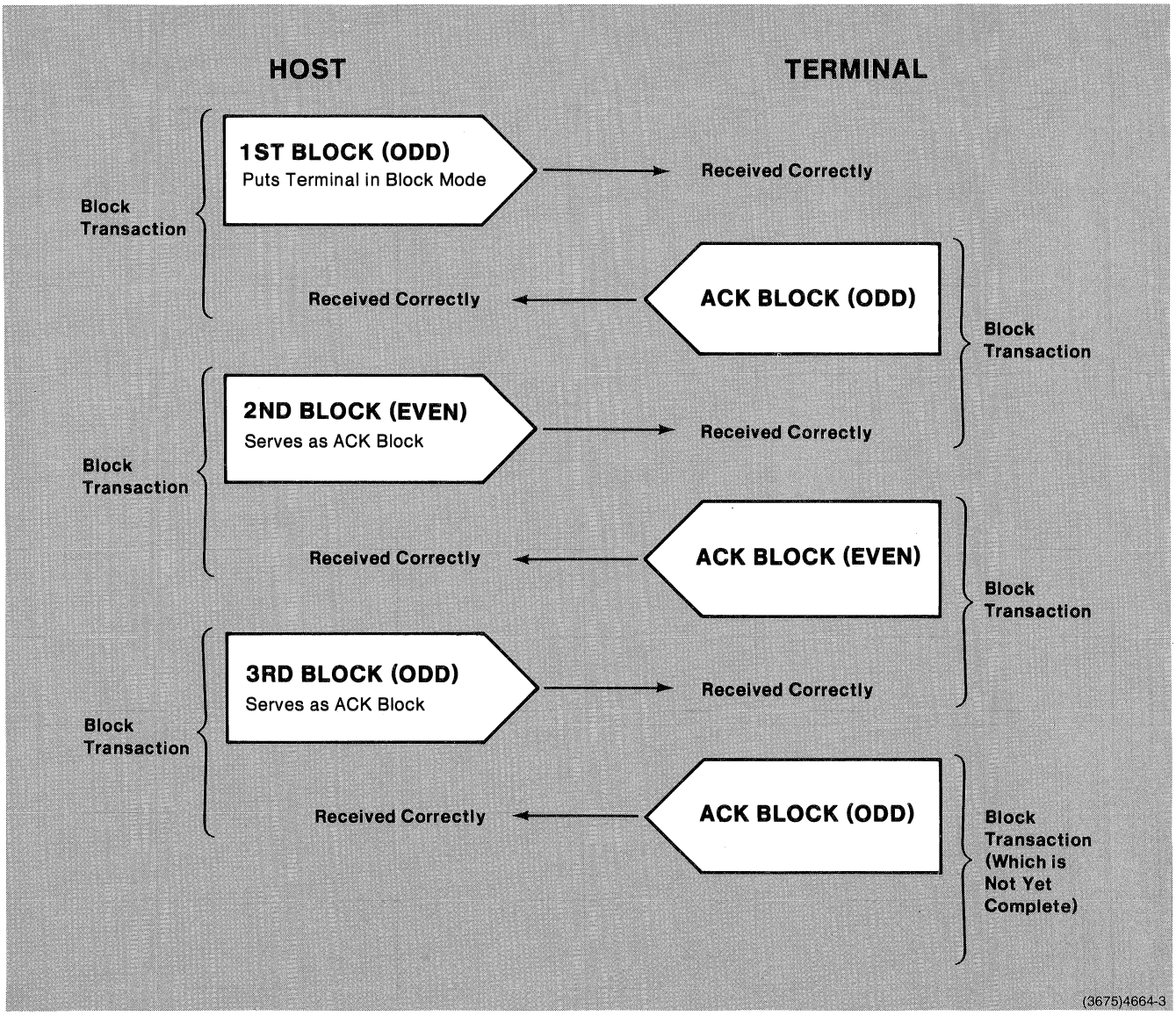


Figure 3-2. Error-Free Block Transmission.

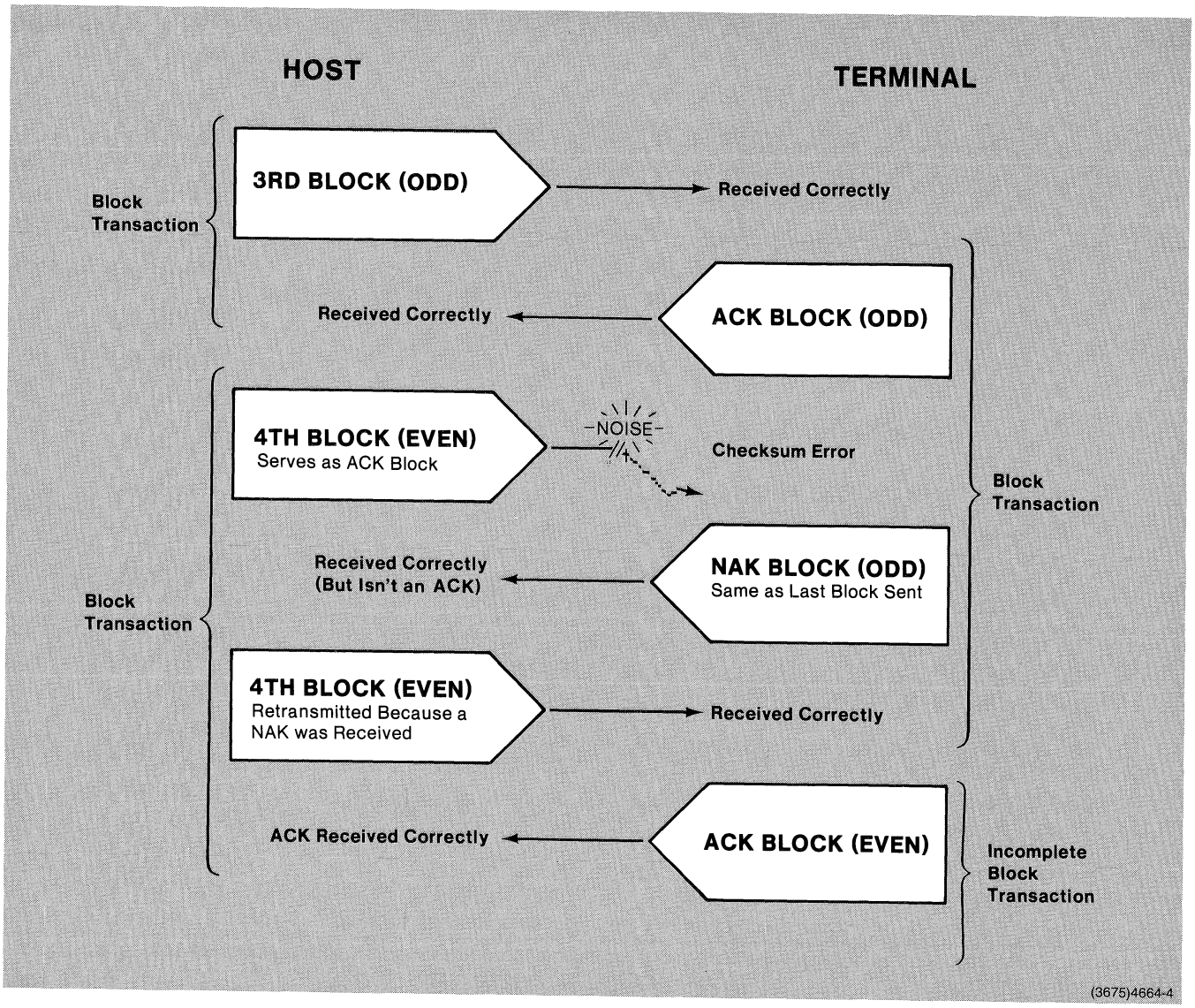


Figure 3-3. Occasional Errors in Block Mode.

Figure 3-4 shows the effect of multiple errors in Block mode. The terminal and host simply exchange blocks until they achieve an error-free exchange.

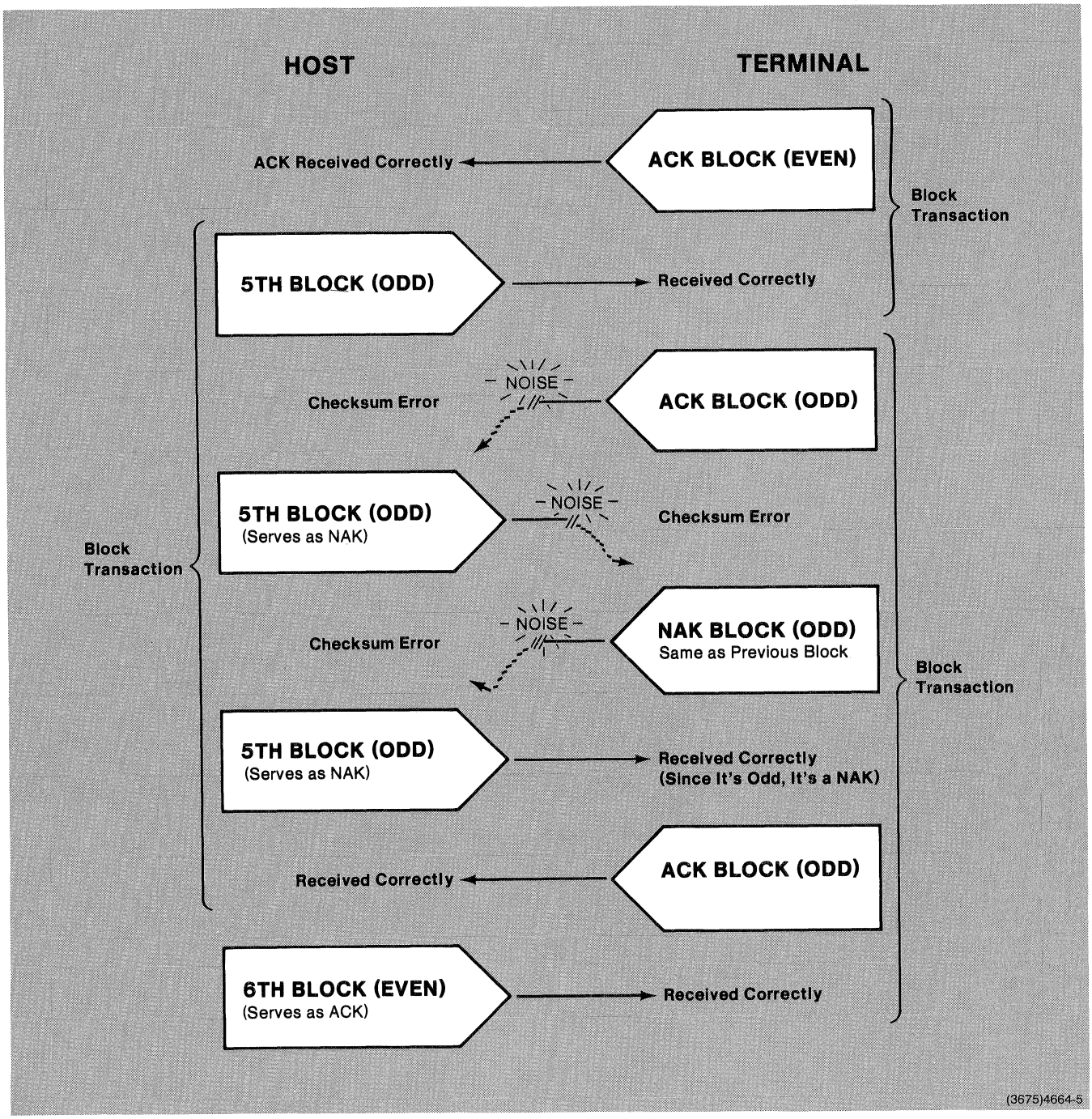


Figure 3-4. Multiple Errors in Block Mode.

Block Mode Algorithms

The following algorithms illustrate how to communicate with the terminal in Block mode.

```
Procedure Start-block-protocol:
   global-reference: (armed-for-block-mode),
   (block-count-bits),
      (host-eom-bit),(host-eof-bit),(host-unpacked-bits),
      (host-packed-bits),(term-unpacked-bits),
      (term-packed-bits),
      (host-master-char),(host-continue-char),(host-end-char),
      (term-master-char),(term-continue-char),(term-end-char),
      (host-number-of-non-xmts),(host-non-xmts),
      (term-non-xmts),
      (chars-in-input-buffer),(chars-in-output-buffer)
   send-2-op-code-command: (O),(B)
   send-packed-integer: (1)
   send-2-op-code-command: (I),(Q)
   send-character: (O)
   send-character: (B)
   input-character: (char)
   input-character: (char)
   input-integer: (armed-for-block-mode)
   if (armed-for-block-mode) = 1
      (block-count-bits)=1
      (host-eom-bit)=0
      (host-eof-bit)=0
      (host-unpacked-bits)=7
      (host-packed-bits)=6
      (term-unpacked-bits)=7
      (term-packed-bits)=6
      (host-master-char)=(#)
      (host-continue-char)=(&)
      (host-end-char)=($)
      (term-master-char)=(#)
      (term-continue-char)=(&)
      (term-end-char)=($)
      (host-number-of-non-xmts)=3
      (host-non-xmts)=<(#),(&),($)>
      (term-non-xmts)=<(#),(&),($)>
      (chars-in-input-buffer)=0
      (chars-in-output-buffer)=0

Procedure End-block-protocol:
   global-reference: (armed-for-block-mode),
   (block-count-bits)
   if (armed-for-block-mode) = 1
      (block-count-bits)=2
      block-transaction:
      (armed-for-block-mode)=0
      send-2-op-code-command: (O), )B)

Procedure Send-character: (char)
   global-reference: (chars-in-output-buffer),(output-buffer),
      (armed-for-block-mode)
   if (armed-for-block-mode) = 1
      if (char) < 0 or (chars-in-output-buffer) = 256
       block-transaction:
   else
      increment (chars-in-output-buffer)
      (output-buffer(chars-in-output-buffer))=(char)
   else
      send-char-to-TTY: (char)

Procedure Input-character: (char)
   global-reference: (chars-in-input-buffer),(input-buffer),
   (host-eom-bit),
      (armed-for-block-mode)
   if (armed-for-block-mode) = 1
      if (chars-in-input-buffer) = 0
        (host-eom-bit)=1
        block-transaction:
        (host-eom-bit)=0
        (front-of-input-buffer)=0
      if (chars-in-input-buffer) > 0
        increment (front-of-input-buffer)
        (char)=(input-buffer(front-of-input-buffer))
        decrement (chars-in-input-buffer)
      else
         (char)=(SP)
   else
      input-char-from-TTY: (char)

Procedure Block-transaction:
   global-reference: (armed-for-block-mode),
   (block-count-bits),
      (chars-in-output-buffer),(output-buffer),
      (chars-in-input-buffer),
      (input-buffer)
      if (armed-for-block-mode) = 1
         (repeat-counter)=0
         (nack)=0
         until (repeat-counter) = 4 or (nack) = 0
            block-send: (chars-in-output-buffer),
            (output-buffer)
            increment (repeat-counter)
            if (block-count-bits) < 3
               block-input: (chars-in-input-buffer),
               (input-buffer),(nack)
            if (nack) = 0
               (block-count-bits)=1-(block-count-bits)
            (chars-in-output-buffer)=0
```

```
Procedure Block-send: (chars-in-output-buffer),
(output-buffer)
   global-reference: (line-length),(max-line-length),(register),
      (bits-in-register),(offset),(max-byte),
      (host-unpacked-bits),
      (host-packed-bits),(host-end-char),(check1),(check2)
   (register)=0
   (bits-in-register)=0
   (offset)=0
   if (host-packed-bits) = 6
      (offset)=32
   (max-byte)=2**(host-unpacked-bits)-1
   (check1)=(max-byte)
   (check2)=(max-byte)
   (line-length)=0
   increment (chars-in-output-buffer)
   (output-buffer(char-in-output-buffer))=
   (block-count-bits)+
      32*(block-host-eof-bit)+64*(block-host-eom-bit)
   increment (chars-in-output-buffer)
   (output-buffer(char-in-output-buffer))=0
   for (counter) = 1 to (chars-in-output-buffer)
      pack-and-send-block-char: (output-buffer(counter))
      checksum: (out-buffer(counter))
   (char)=(max-byte)-(check1)-(check2)
   if (char) <= 0
      increment (char) by (max-byte)
   (char2)=(check2)
   pack-and-send-block-char: (char)
   pack-and-send-block-char: (char2)
   if (bits-in-register) > 0
      shift (register) left (host-packed-bits)-(bits-in-register)
      increment (register) by (offset)
      send-char-to-TTY: (register)
   send-char-to-TTY: (host-end-char)

Procedure Pack-and-send-block-char: (char)
   global-reference: (register),(bits-in-register),(offset),
   (max-byte),
      (host-unpacked-bits),(host-packed-bits),
      (host-number-of-non-xmts),
      (host-non-xmts)
   shift (register) left (host-unpacked-bits)
   increment (register) by (char)
   increment (bits-in-register) by (host-unpacked-bits)
   while (bits-in-register) => (host-packed-bits)
      (out-char)=(register) modulo 2**(host-packed-bits)
      shift (register) right (host-packed-bits)
      decrement (bits-in-register) by (host-packed-bits)
      increment (out-char) by (offset)
      (sub-pointer)=(host-number-of-non-xmts)
      until (sub-pointer) = 0 or (out-char) =
      (host-non-xmts(sub-pointer))
         decrement (sub-pointer)
      if (sub-pointer) = 0
         send-block-char: (out-char)
      else
         send-block-char: (host-master-char)
         send-block-char: (sub-pointer)+64

Procedure Send-block-char: (char)
   global-reference: (host-header-length),(host-header),
   (line-length),
      (host-continue-char),(max-line-length)
   if (line-length) = (max-line-length)-1
      send-char-to-TTY: (host-continue-char)
      (line-length)=0
   if (line-length) = 0
      for (counter) = 1 to (host-header-length)
         send-char-to-TTY: (host-header(counter))
      (line-length)=(host-header-length)
   send-char-to-TTY: (char)
   increment (line-length)

Procedure Block-input: (chars-in-input-buffer),
(input-buffer),(nack)
   global-reference: (register),(bits-in-register),(offset),
   (max-byte),
      (term-unpacked-bits),
      (term-packed-bits),(check1),(check2),
      (end-of-data),(term-eof-bit),(term-eom-bit),
      (in-block),(end-of-block)
   (register)=0
   (bits-in-register)=0
   (offset)=0
   if (term-packed-bits) = 6
      (offset)=32
   (max-byte)=2**(term-unpacked-bits)-1
   (check1)=(max-byte)
   (check2)=(max-byte)
   (in-block)=0
   (end-of-block)=0
   (end-of-data)=0
   until (end-of-data) = 1
      input-and-unpack-block-char: (char)
      if (end-of-data) = 0
         checksum: (char)
         increment (chars-in-input-buffer)
         (input-buffer(chars-in-input-buffer))=(char)
   (control-bit-char)=(input-buffer((chars-in-input-buffer)-3))
   decrement (chars-in-input-buffer) by 4
   (term-count-bits)=(control-bit-char) modulo 4
   (term-eof-bit)=(control-bit-char)/32 modulo 2
   (term-eom-bit)=(control-bit-char)/64
   if (check1) = (max-byte) and (check2) = (maxbyte)
   and (term-count-bits) = (block-count-bits)
      (nack)=0
   else
      (nack)=1
```

```
Procedure Input-and-unpack-block-char: (char)
   global-reference: (register),(bits-in-register),(offset),
   (max-byte),
      (term-unpacked-bits),(term-packed-bits),
      (term-non-xmts),(end-of-data),
      (end-of-block)
   if (end-of-block) = 0
      until (bits-in-register) => (term-unpacked-bits)
         input-block-char: (in-char)
         if (end-of-block) = 0
            if (in-char) = (term-master-char)
               input-block-char: (in-char)
               (in-char)=(term-non-xmts((in-char)-64))
            decrement (in-char) by (offset)
            shift (register) left (term-packed-bits)
            increment (bits-in-register) by (term-packed-bits)
            increment (register) by (in-char)
         else
            if (bits-in-register) = 0
               (end-of-data)=1
            shift (register) left (term-unpacked-bits)-
            (bits-in-register)
            (bits-in-register)=(term-unpacked-bits)
      (char)=(register) modulo 2**(term-unpacked-bits)
      shift (register) right (term-unpacked-bits)
      decrement (bits-in-register) by (term-unpacked-bits)
   else
      (end-of-data)=1
```

```
Procedure Input-block-char: (char)
   global-reference: (term-header-length),
   (term-header),(term-continue-char),
      (term-end-char),(in-block),(end-of-block)
   if (in-block) = 1
      input-char-from-TTY: (char)
      if (char) = (term-end-char)
         (end-of-block)=1
      elseif (char) = (term-continue-char):
         (in-block)=0
      if (in-block) = 0
         (header-pointer)=0
         until (header-pointer) = (term-header-length)
            increment (header-pointer)
            input-char-from-TTY: (char)
            if (char) <> (term-header(header-pointer))
               (header-pointer)=0
         (in-block)=1
         input-char-from-TTY: (char)

Procedure Checksum: (char)
   global-reference: (max-byte),(check1),(check2)
   increment (check1) by (char)
   if (check1) > (max-byte)
      decrement (check1) by (maxbyte)
   increment (check2) by (check1)
   if (check2) > (max-byte)
      decrement (check2) by (max-byte)
```

Section 4

USING 4110 SERIES TERMINALS

INTRODUCTION

To drive a 4110 Series terminal from a host, your host program must send commands to the terminal and parse reports that the terminal sends to the host. You must pack command parameters and parse the parameters that the terminal returns. This section contains discussions on:

- Terminal commands, reports, and parameters
- Macros and macro expansion
- The TEK and ANSI command parsers

TERMINAL COMMANDS, REPORTS, AND PARAMETERS

PREVIEW

- Terminal commands
- Initializing the terminal and general procedures for sending commands
- Command parameters
- Packing and sending simple parameters
- Packing and sending complex parameters
- Terminal reports
- Parsing simple reports
- Parsing complex reports

TERMINAL COMMANDS

4110 Series terminals respond to commands from the host or, if the terminal is in Setup mode or Local mode, to the terminal keyboard. Commands presented in this discussion are commands from the host to the terminal.

A 4110 Series command is one of the following:

- A single-character command — such as a printable character in Alpha mode (implied PRINT-THIS-CHARACTER command), the Alpha-cursor positioning control characters ($^{C}_{R}$, $^{L}_{F}$, $^{B}_{S}$, $^{V}_{T}$, and $^{H}_{T}$), special-purpose control characters ($^{B}_{L}$, $^{S}_{I}$, and $^{S}_{O}$), and the mode-controlling characters ($^{U}_{S}$, $^{G}_{S}$, and $^{F}_{S}$).
- A two-character sequence beginning with the character $^{E}_{C}$ followed by a single character. These two-character escape sequences are also called *one op-code* commands. The one-op-code commands are:

  $^{E}_{C}{}^{S}_{B}$ = ENABLE-4010-GIN
  $^{E}_{C}{}^{C}_{N}$ = ENTER-BYPASS-MODE
  $^{E}_{C}{}^{F}_{F}$ = PAGE
  $^{E}_{C}{}^{E}_{Q}$ = REPORT-4010-STATUS
  $^{E}_{C}{}^{S}_{I}$ or $^{E}_{C}{}^{S}_{O}$ = SET-ALPHATEXT-FONT
  $^{E}_{C}$*char* = SET-4014-LINE-STYLE
  $^{E}_{C}{}^{E}_{B}$ = 4010-HARD-COPY

- A three-character sequence beginning with the character $^{E}_{C}$ followed by parameters if the command takes them. These three-character escape sequences are also called *two op-code* commands. Most 4110 Series commands are these three-character escape sequences. See the *4110 Series Command Reference Manual* for an alphabetic listing of all 4110 Series commands.

Initializing the Terminal and Sending Commands

The following algorithms illustrate how to initialize the terminal and send two-op-code commands.

```
Procedure Initialize-system: (terminal-model)
    global-reference: (implicit-mode), (abs-xy),
    (terminal), (last-HiY), (last-exLoY), (last-LoY), (last-HiX),
    (home-x), (home-y)
    (implicit-mode)=-1
    send-enter-alpha-mode-command:
    send-set-coordinate-mode-command: 0, 3
    send-arm-for-block-mode-command: 0
    (terminal)=(terminal-model)
    (last-HiY)=0
    (last-exLoY)=0
    (last-LoY)=0
    (last-HiX)=0
    (home-x)=0
    (home-y)=3071
    if (terminal)=4115
        (home-y)=3190
    send-page-command

Procedure Send-enter-alpha-mode-command:
    global-reference: (implicit-mode)
    if (implicit-mode) <> 1
        send-character: (US)
        (implicit-mode)=1

Procedure Send-enter-vector-mode-command:
    global-reference: (implicit-mode), (move-draw-flag)
    if (implicit-mode) = 0
        send-character: (US)
    send-character: (GS)
    (implicit-mode)=2
    (move-draw-flag)=0

Procedure Send-enter-marker-mode-command:
    global-reference: (implicit-mode)
    if (implicit-mode) <> 0
        send-character: (FS)
        (implicit-mode)=0

Procedure Send-set-coordinate-mode-command: (c-mode),
(r-length)
    global-reference: (coord-mode),(report-length)
    send-2-op-code-command: (U),(X)
    send-packed-integer: (c-mode)
    send-packed-integer: (r-length)
    (coord-mode)=(c-mode)
    (report-length)=(r-length)

Procedure Send-2-op-code-command: (char1),(char2)
    Send-character: (ESC)
    Send-character: (char1)
    Send-character: (char2)

Procedure Send-page-command:
    global-reference: (implicit-mode),(d-a-enable),
    (beam-x),(beam-y),
        (home-x),(home-y),(4010-GIN-on)
    send-character: (ESC)
    send-character: (FF)
    if (d-a-enable) = 0
        (implicit-mode) = 1
        (beam-x)=(home-x)
        (beam-y)=(home-y)
        (4010-GIN-on)=0

Procedure Send-draw-command:
    global-reference: (implicit-mode), (move-draw-flag)
    if (implicit-mode) <> 2
        send-enter-vector-mode-command:
    if (move-draw-flag) = 0
        send-character: <BL>
    send-xy: (x), (y)

Procedure Send-move-command: (x), (y)
    send-enter-vector-mode-command:
    send-xy: (x), (y)

Procedure Send-marker-command: (x), (y)
    send-enter-marker-mode-command:
    send-xy: (x), (y)
```

COMMAND PARAMETERS

4110 Series command parameters are of three basic types: *int*, *xy* or *char*. Other parameter types are made up of combinations of these basic types.

int Parameters

int parameters are a sequence of ASCII characters from ^{S}P through ^{D}L that represent an integer value. The packing scheme used places the sign of the integer in the final character (called the *LoI* character) and the values whose absolute value exceeds 15 in other characters (called *HiI* characters). Appendix B gives examples of *int* parameters and a method of converting between integers and *int* parameters. The following general-purpose algorithm shows how to pack *int* parameters.

Packing an *int* Parameter

```
Procedure Send-packed-integer: (int)
   initialize (stack)
   (abs-int)=absolute value of (int)
   (loi)=(abs-int) modulo 16 + 48
   if (int) < 0
      decrement (loi) by 16
   push (loi) onto (stack)
   shift (abs-int) right 4 bits
   while (abs-int) > 0
      push (abs-int) modulo 64 + 64 onto (stack)
      shift (abs-int) right 6 bits
   until (stack) is empty
      pop (stack) into (ADE-char)
      send-character: (ADE-char)
```

xy Parameters

xy parameters are a sequence of ASCII characters from ^{S}P through ^{D}L that represent a coordinate in 12-bit terminal space. The packing scheme is the same as that used on TEKTRONIX 4010 Series terminals. The 4115 can use a pair of integers to represent *xy* parameters in its larger address space. The following algorithms illustrate how to pack *xy*s including 32-bit *xy*s for the 4115.

Hint. If you would like to reduce communications overhead and don't mind slightly reduced resolution, you can omit the Extra-LoY character in the 12-bit *xy* parameter. This gives 10-bit instead of 12-bit resolution and can reduce the number of characters transmitted per *xy* by up to 33%. Tektronix 4006, 4010 and 4012 terminals use the 10 bit format.

Packing *xy* Parameters

```
Procedure Send-xy: (x),(y)
   global-reference: (coord-mode),(last-x),(last-y),(abs-xy)
   if (coord-mode) = 0
      send-12-bit-xy: (x),(y)
   else
      if (abs-xy) = 1
         (x-sent)=(x)
         (y-sent)=(y)
      else
         (x-sent)=(x)-(last-x)
         (y-sent)=(y)-(last-y)
      send-packed-integer: (x-sent)
      send-packed-integer: (y-sent)
   (last-x)=(x)
   (last-y)=(y)

Procedure Send-12-bit-xy: (x),(y)
   global-reference: (last-hiy),(last-exloy),(last-loy),(last-hix)
   global-reference: (margin-control-bit)
   (new-hiy)=(y)/128+32
   (new-exloy)=((y) modulo 4)*4+((x) modulo 4)+
   (margin-control-bit)*16+96
   (new-loy)=((y)/128 modulo 32)+96
   (new-hix)=(x)/128+32
   (new-lox)=((x)/128 modulo 32)+64
   if (new-hiy) <> (last-hiy)
      send-character: (new-hiy)
      (last-hiy)=(new-hiy)
   if (new-exloy) <> (last-exloy)
      send-character: (new-exloy)
      (last-exloy)=(new-exloy)
   if (new-loy) <> (last-loy)
   or (new-exloy) <> (last-exloy)
   or (new-hix) <> (last-hix)
      send-character: (new-loy)
      (last-loy)=(new-loy)
   if (new-hix) <> (last-hix)
      send-character: (new-hix)
      (last-hix)=(new-hix)
   send-character: (new-lox)
```

char Parameters

A *char* parameter is a single ASCII character from ^{S}P through ~ . Char parameters have ASCII decimal equivalents from 32 through 126.

Complex Parameters

Other parameters are made up from the simple parameter types. These are:

- *array* parameters. An *array* is a series of one of the basic parameter types preceded by its count in an *int*. 4110 Series *array*s are:

  int-array —a series of *int*s preceded by a count *int*.

  xy-array —a series of *xy*s preceded by a count *int*

  char-array —a series of *char*s preceded by a count *int or string*

- *real* parameters. A *real* parameter is a pair of *int* parameters: a mantissa and an exponent where the exponent is a power of two.

- 32-bit *xy* parameters. A 32-bit *xy* parameter is a pair of *int*s, one x-value and one y-value.

The following general-purpose algorithms show how to pack *real*, *int-array*, *char-array*, and *xy-array* parameters.

Packing Complex Parameters

```
Procedure Send-packed-real: (real)
   (exponent)=0
   (abs-real)=absolute value of (real)
   (mantissa)=integer part of (abs-real)
   while (mantissa) < 2**15
   and absolute value of ((abs-real)-(mantissa)) > 2**-15
      (abs-real)=(abs-real)*2
      (exponent)=(exponent)-1
      (mantissa)=integer part of (abs-real)
   if (real) < 0
      (mantissa)= -(mantissa)
   send-packed-integer: (mantissa)
   send-packed-integer: (exponent)

Procedure Send-integer-array: (length),(int-array)
   send-packed-integer: (length)
   (count)=0
   while (count) < (length)
      increment (count)
      send-packed-integer: (int-array(count))

Procedure Send-char-array: (length),(char-array)
   send-packed-integer: (length)
   (count)=0
   while (count) < (length)
      increment (count)
      send-character: (char-array(count))

Procedure Send-xy-array: (length),(x-array),(y-array)
   send-packed-integer: (length)
   (count)=0
   while (count) < (length)
      increment (count)
      send-xy: (x-array(count)),(y-array(count))
```

TERMINAL REPORTS

In addition to packing and sending parameters, your programs must receive and parse reports. Terminal reports are analogous to, but differ from the various types of parameters: i.e. each type of parameter has a corresponding type of report.

You can cause a terminal to report by using one of the following commands:

- REPORT-COLORHARDCOPY-STATUS
- REPORT-DEVICE-STATUS
- REPORT-ERRORS
- REPORT-GIN-POINT
- REPORT-PORT-STATUS
- REPORT-SEGMENT-STATUS
- REPORT-TERMINAL-SETTINGS

The syntax of these commands and the reports they return are detailed in the *4110 Series Command Reference Manual.*

CAUTION

Be careful using the REPORT-TERMINAL-SETTINGS command. Some commands return no parameter reports, while others return reports with different values than those that were last set. See the 4110 Series Command Reference Manual *for details on each command for which you need a report.*

The following algorithms illustrate how to parse the various reports returned by the terminal.

Parsing an *int* or *intc* Report

```
Procedure Input-integer: (int)
   input-character: (char1)
   input-character: (char2)
   input-character: (char3)
   (int)=((char1)-32)*1024 + ((char2)-32)*16 + (char3)
   modulo 16
   if (char3) < 48
      negate (int)

Procedure Input-integer-c: (int)
   global-reference: (report-length)
   (int)=0
   for (counter) = 1 to (report-length)-1
      input-character: (char)
      (int)=(int)*64 + (char)-32
   input-character: (char)
   (int)=(int)*16 + (char) modulo 16
   if (char) < 48
      negate (int)
```

Parsing an *xy* Report

```
Procedure Input-xy: (x),(y),(type)
   global-reference: (coord-mode)
   if (coord-mode) = 0
      input-character: (char1)
      input-character: (char2)
      input-character: (char3)
      input-character: (char4)
      if (type) = 1 !4110 12-bit type!
         input-character: (char5)
         (x)=((char4)-32)*128 + ((char5)-32)*4
               + (char2) modulo 4
         (y)=((char1)-32)*128 + ((char3)-32)*4
               + (char2)/2 modulo 4
      if (type) = 2 !4010 10-bit type!
         (x)=((char1)-32)*128 + ((char2)-32)*4
         (y)=((char3)-32)*128 + ((char4)-32)*4
      else
         input-integer-c: (x)
         input-integer-c: (y)
```

Parsing Array Reports

```
Procedure Input-integer-array: (length),(int-array)
   input-integer: (length)
   (count)=0
   while (count) < (length)
      increment (count)
      input-integer: (int-array(count))

Procedure Input-xy-array: (length),(x-array),(y-array)
   input-integer: (length)
   (count)=0
   while (count) < (length)
      increment (count)
      input-xy: (x-array(count)),(y-array(count))

Procedure Input-char-array: (length),(char-array)
   input-integer: (length)
   (count)=0
   while (count) < (length)
      increment (count)
      input-character: (char-array(count))
```

TROUBLESHOOTING HINTS

Whenever you are having trouble getting the terminal to respond as you think it should, you should have a checklist of common problems and their causes. The following list is by no means complete, but should serve as a beginning for your own. The most baffling problems are often the most obvious.

- Input queue size — If the input queue overflows, it can cause baffling failures. Try setting the queue larger.
- EOM-frequency — Set to frequent: less frequent can cause puzzling problems with GIN
- Segment-writing mode — If you set to XOR, it overstrikes existing graphics. Alphatext over existing graphics can cause quite a few problems.
- Pixel operations — Remember that the BEGIN-PIXEL-OPERATIONS command sets a lot of parameters. If you need anything but the defaults, set them with this command.
- EOF-string — If you set this to a character that is used in *xy* parameters, it can look like the line is picking up noise.
- Dialog scroll buffer — If you set the scroll buffer smaller than the number of dialog area lines, the display moves strangely.
- Invisible text and graphics — Are they a visible color? Are they the background color?
- Silly as it may seem — Is the terminal plugged in, connected, with the communications set right?

MACROS AND MACRO EXPANSION

PREVIEW

- Host macros
- Byte macros
- Key macros
- The key-execute character

CONCEPTS AND DEFINITIONS

A *macro* is a single instruction that stands for a sequence of instructions. 4110 Series terminals support three types of macros: host, byte, and key.

To define or delete macros, use the DEFINE-MACRO command. 4110 Series terminal macro types differ only in (1) the range of macro numbers they occupy and (2) how the terminal expands them.

Host Macros

A *host macro* is a macro that can be expanded by the EXPAND-MACRO command. Since all 4110 Series macros can be expanded by the EXPAND-MACRO command (including byte and key macros) all macros are host macros.

Byte Macros

A *byte macro* is a macro that the terminal expands when it encounters that character in the terminal's input queue. You can think of a byte macro as a character which carries an implicit EXPAND-MACRO command. Byte macros occupy numbers −32768 through −32742, −32740 through −32737, and −32608 through −32513. To get the ADE value of the character which is interpreted as containing an EXPAND-MACRO command, add 32768 to the macro number.

Key Macros

A *Key macro* is a macro that the terminal expands when the operator presses the keyboard key (or key combination) that corresponds to that macro number. The key number is the ADE (ASCII Decimal Equivalent) generated by the keyboard. You program the terminal's function keys by defining key macros.

Key macros are numbers 0 through 143. Numbers 0 through 127 are the 7-bit ASCII character set; numbers 128 through 143 are the keyboard function keys. Section 5, *The Operator Interface*, discusses the keyboard and the ADEs generated by the function keys.

Making the Terminal Execute a Key Macro

When the terminal expands a key macro, it normally places the resulting character stream in the output queue for transmission to the host. If you want to program a function key to give a command to the terminal rather than a character stream to the host, you must use the *key-execute-character* to toggle the output of the macro expander from the terminal output buffer into the terminal's input buffer. You can send a $^{C}_{R}$ to the host at the end of the key macro to indicate the end of the macro.

The key-execute character toggles the output of a key macro between normal expansion and responding to the macro as though its contents were sent by the host.

Expanding Macros

When the terminal expands a macro, several actions take place:

1. The terminal's macro expander marks the macro as being expanded, and if it is a byte macro, removes the byte from the input queue.
2. The macro expander places the macro content into the input queue for processing by whatever parser is currently selected.
3. Nesting macros is allowed; the macro contents may include commands to expand other macros. However, if a macro is marked as being expanded, the command to expand it is ignored, and if it is a byte macro, the character is passed on unchanged.
5. When the terminal completes the macro expansion, it removes the mark from the macro; the macro expander can then expand the macro if it should encounter the macro again.

TERMINAL PARSERS

PREVIEW

- Overview
- Snoopy mode
- The TEK parser
  - Implicit Command Modes (Alpha, Vector, and Marker)
  - Command parsing states
  - Parameter parsing states
- Modes that affect parsing
- The ANSI parser

OVERVIEW

4110 Series terminals contain either two or three parsers to interpret host and terminal commands. DVST terminals contain two parsers: a TEK parser to interpret host commands and a Setup mode parser to interpret commands from the keyboard in Setup mode. Raster display terminals contain three parsers: the TEK and Setup parsers and an ANSI parser to respond to a subset of the ANSI X3.64 commands. You choose between the TEK and ANSI parsers with the SELECT-CODE command.

This section contains a description of the TEK and ANSI parsers. The operators manual for your terminal contains information on Setup mode.

Appendix E contains a table that shows how the TEK and ANSI parsers interpret incoming characters in various modes and states. You can use this table to trace the terminal through its various states for debugging terminal driving programs.

SNOOPY MODE

Snoopy mode intercepts characters from the input buffer before they are routed to the parsers. The terminal displays non-printing ASCII characters as 2-character mnemonic equivalents in one character cell; I.E. the character "escape" becomes ^{E}C in Snoopy mode. The terminal does not execute any commands except the character ^{C}R, which is both displayed and executed as $^{C}R^{L}F$.

You can put the terminal into Snoopy mode from the host with the SET-SNOOPY-MODE command or from Setup mode with the command SNOOPY YES.

You can only remove the terminal from Snoopy mode from the terminal keyboard. You can use the Setup command SNOOPY NO or press the CANCEL key to bring the terminal out of Snoopy mode.

CAUTION

The host programmer should use Snoopy mode for debugging programs. It allows you to see just what the host is sending. It is inadvisable for your program to put the terminal into Snoopy mode, as it must depend on the terminal operator to get it out.

THE TEK PARSER

The Tek parser is a flag and table driven state machine that interprets characters as commands to the terminal. The parser uses flags to remember the parser mode or state. Most flags are binary and represent a yes-no condition for a particular state. One flag, the implicit mode flag, is trinary. Another, the parameter parsing flag is multi-valued to indicate what kind of parameter is being parsed.

The three implicit command modes — Alpha, Vector, and Marker — are available. Each mode implies a different command as implicit to that mode: in Alpha mode the implicit command is PRINT-CHARACTER, in Vector mode the implicit command is either MOVE or DRAW, depending on the move/draw flag, and in Marker mode the implicit command is DRAW-MARKER. In each of these modes, each character received by the parser is interpreted as a parameter to the current implicit command.

The TEK parser is always in one (and only one) of the implicit command modes, even when it is in one of the explicit command states. While an explicit command is being parsed, the implicit command mode is inactive but remembered. The parser reenters the implicit command mode after it terminates the explicit command.

Figure 4-1 illustrates the relationship between the implicit command modes and the explicit command states of the TEK parser. Figure 4-2 gives a simplified state diagram of how the TEK parser moves between the explicit command states while parsing a command.

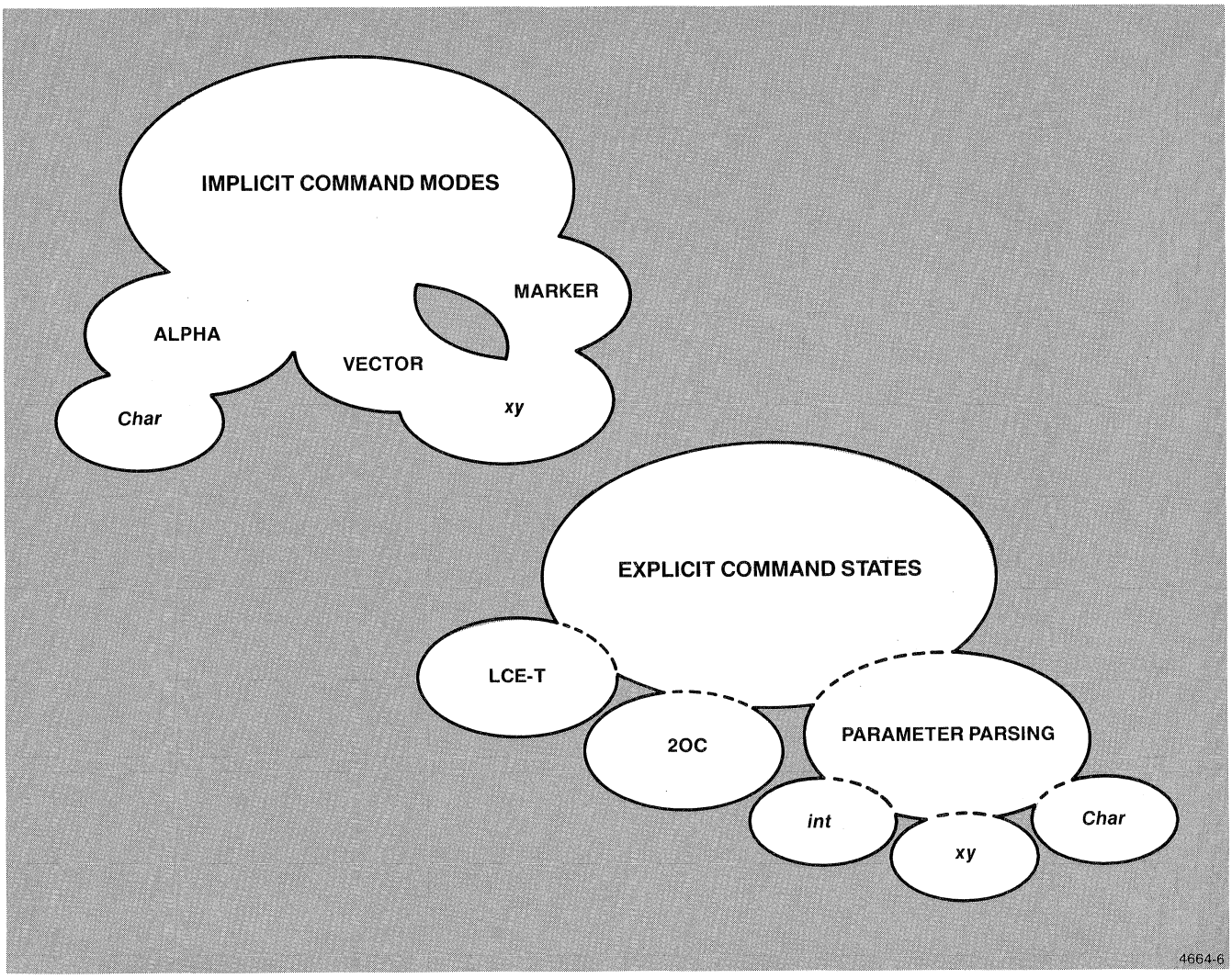


Figure 4-1. Implicit Command Modes and Explicit Command States.

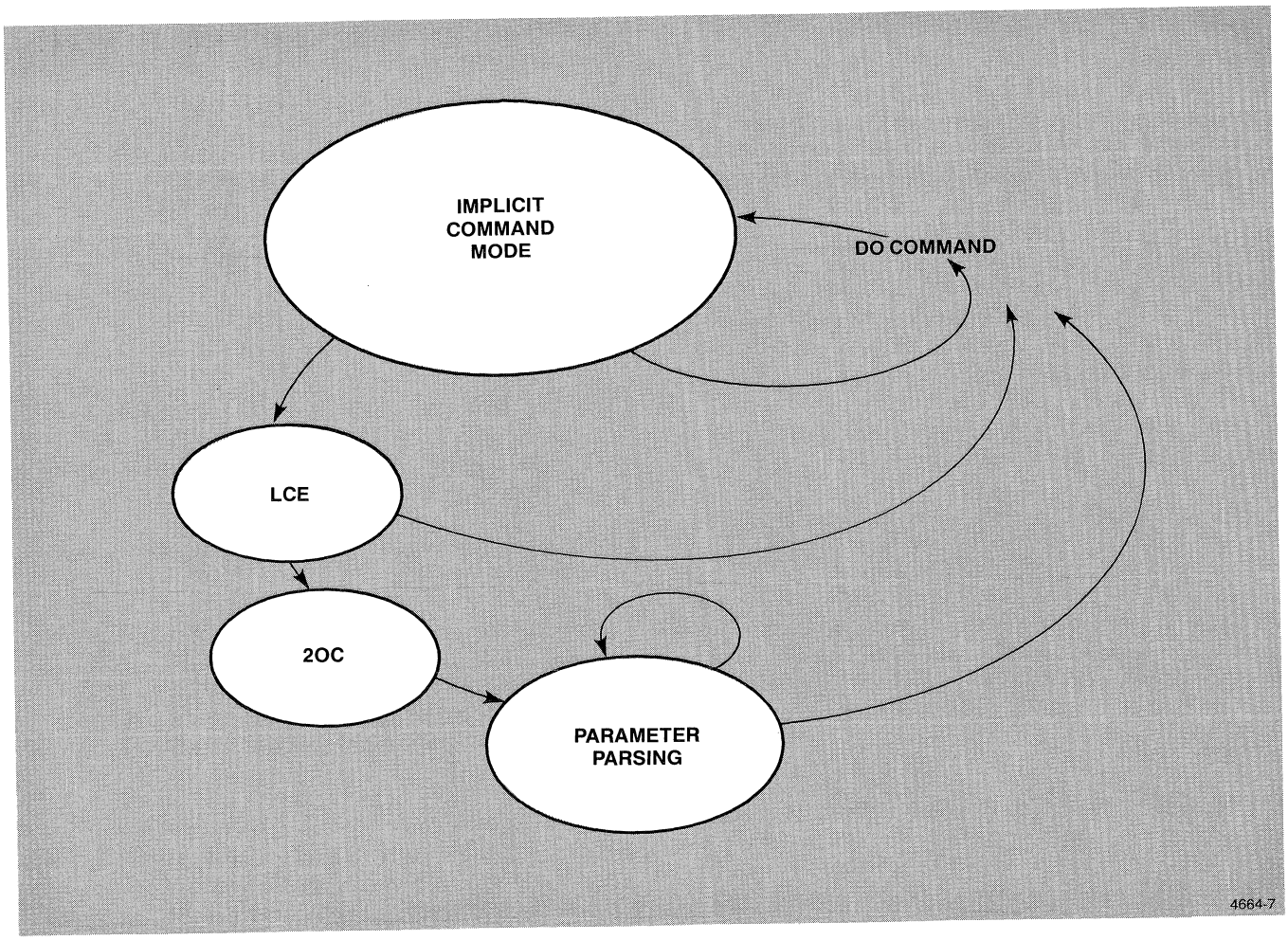


Figure 4-2. Explicit Command States.

THE IMPLICIT COMMAND MODES

Alpha Mode

Entering Alpha Mode. You enter Alpha mode with the command ENTER-ALPHA-MODE. This command is the single character ^{U}S. This character is available on the keyboard as "control-_" or "control-shift-o".

Alpha mode is the default mode from a reset, and CANCEL key or command. The terminal will also enter Alpha mode after a PAGE or ^{C}R with the dialog area disabled. The terminal will also enter Alpha mode after leaving one of the 4010 GIN emulation modes.

Leaving Alpha Mode. You can leave Alpha mode by entering either Vector mode or Marker mode.

Effects of Alpha Mode. In Alpha mode, the terminal interprets each printable ASCII character as an implicit command to print that character as alphatext.

Alpha mode is the default operating mode. When the terminal is in Alpha mode, you can give graphics primitives as explicit commands.

Vector Mode

Entering Vector Mode. To enter Vector mode, send the command ENTER-VECTOR-MODE. This command is the single character ^{G}S (available from the keyboard as "control-[" or "control-shift-m"). You can enter Vector mode from Alpha mode only.

Leaving Vector Mode. You can leave Vector mode by entering either Alpha mode or Marker mode.

Effects of Vector Mode. Vector mode is used for efficient communication of graphics information. When the terminal is in Vector Mode, it interprets incoming characters other than control characters as encoded *xy* coordinates. Vector mode ignores control characters other than ^{B}L, ^{E}C, ^{F}S, ^{G}S or ^{U}S.

As each complete coordinate is assembled, the terminal executes the implied MOVE or DRAW to that coordinate.

The terminal interprets the first *xy* coordinate after ^{G}S (the ENTER-VECTOR-MODE command) as a move to that position. If you follow ^{G}S with ^{B}L, the ^{B}L causes the terminal bell to ring and the terminal to interpret each subsequent *xy* coordinate as a DRAW to that position. As in explicit MOVEs and DRAWs, the graphics beam is left where the MOVE or DRAW ends.

Marker Mode

Entering Marker Mode. You can enter Marker mode with the command ENTER-MARKER-MODE. This command is the single character ^{F}S and is available from the keyboard as control-\ or control-shift-L.

Leaving Marker Mode. The only way to leave Marker mode is to enter Alpha mode.

Effects of Marker Mode. Marker mode is a mode to efficiently draw a number of markers. When the terminal is in Marker mode, it interprets incoming characters other than control characters as encoded *xy* coordinates. Marker mode ignores control characters other than ^{B}L, ^{E}C, ^{F}S, ^{G}S or ^{U}S.

As soon as the coordinate is complete, the terminal executes the implied DRAW-MARKER command and draws a marker of the current type at that coordinate.

EXPLICIT COMMAND STATES

When the TEK parser receives an explicit command, it first enters LCE-T state. From LCE-T state, the parser moves to either do a single op-code command, or to 2OC state if the command is a two op-code command. The parser enters the parameter parsing state prior to doing the command for any command that takes parameters.

Commands can be terminated early by one of the terminator characters: ^{E}C, ^{G}S, ^{U}S, or ^{F}S. When a command that takes parameters is terminated early, the terminal assigns default values to those parameters.

LCE-T State

The TEK parser enters LCE-T State whenever it receives the ^{E}C character.

The parser leaves LCE-T State immediately after it receives a character other than ^{N}L, ^{C}R, ^{L}F, or ^{D}T. The terminal ignores those four characters and remains in LCE-T state. If the character is an upper-case alphabetic character from I through Z, the parser enters 2OC state. Any other character completes a one op-code command (many are no-ops).

2OC State

2OC (two op-code) state is the state the TEK parser enters from LCE-T state after receiving a character in the range from I to Z, such as the "L" in ^{E}CLE.

The parser leaves 2OC state when it receives a character it does not ignore. The parser ignores control characters other than terminators in 2OC state.

In 2OC state, the terminal has received the first character after the ^{E}C of a two op-code command. It is waiting for the second character to complete the command. After the command is complete, the terminal executes those commands that take no parameters, or goes into one of the parameter parsing states to complete the command. The terminal uses a table to determine what parameters must be parsed.

Commands that require parameters are completed when the terminal either receives all the parameters, or receives a terminator character.

PARAMETER PARSING STATES

4110 Series terminals parse three main types of parameters: *xy*, *int*, and *char*. Arrays are composed of a series of *xy*s, *int*s, or *char*s preceded by an *int* count. *Real* parameters are expressed as two *int*s.

int Parsing State

The TEK parser moves into *int* parsing state when a command requires an *int* parameter and out when the parameter is complete. When the parser parses complex parameters, it may actually enter this state several times.

While in *int* Parsing state, the terminal interprets all characters except control characters as parts of *int* parameters. It ignores control characters other than terminators.

When the parser encounters a character whose ADE is from 64 to 127, it interprets this character as a HiI value and remains in *int* parsing state. When the parser encounters a character whose ADE is from 32 to 63, the parser interprets this as a LoI value and completes the *int* parameter it is currently parsing.

xy Parsing State

The TEK parser moves into *xy* Parsing state when a command requires an *xy* parameter and out when the parameter is complete. It ignores control characters other than terminators. In parsing an *xy-array* the parser actually enters this state once for each *xy* value in the array.

When the parser encounters characters whose ADE is from 32 through 63, it interprets these characters as HiX or HiY values and remains in *xy* parsing state. When the parser encounters characters whose ADE is from 96 through 127, it interprets these characters as Loy or ExLoy values and remains in *xy* parsing state. When the parser encounters characters whose ADE is from 64 through 95, it interprets these characters as LoX values and completes the *xy* parameter it is currently parsing.

char Parsing State

Entering and Leaving *char* Parsing State

The TEK parser enters *char* parsing state when a command is expecting a *char* parameter and leaves when the parameter is complete. In parsing a *char-array* or *string*, the terminal actually enters this state for each *char*.

In *char* Parsing state, the parser ignores control characters other than terminators. It interprets characters with ADE values from 32 through 127 as the characters themselves.

MODES THAT AFFECT PARSING

Some modes alter the way the TEK parser interprets incoming characters. Ignore Deletes mode changes the interpretation of the $^{D}_{L}$ character and the 4115 coordinate modes choose whether a 4115 will parse coordinates as *xy*s or *int*s.

IGNORE DELETES MODE

Entering Ignore Deletes Mode

You enter Ignore Deletes mode by issuing the command IGNORE-DELETES with a parameter of 1.

Leaving Ignore Deletes Mode

You can leave Ignore Deletes mode by issuing the command IGNORE-DELETES with a parameter of 0.

Effects of Ignore Deletes Mode

In Ignore Deletes mode, the parser ignores any $^{D}_{L}$ (RUBOUT) characters sent by the host. When you must send a significant $^{D}_{L}$ as in an *int* or *xy*, substitute the sequence $^{E}_{C}$? for each $^{D}_{L}$.

4115 COORDINATE MODES

Changing Coordinate Modes

You switch between 12-bit and 32-bit coordinate modes with the SET-COORDINATE-MODE command. The parameter to the command selects the terminal coordinate mode.

12-Bit Coordinate Mode

12-bit Coordinate mode is common to all 4110 Series terminals. When in 12-Bit Coordinate mode, *xy* parameters are sent as 12-bit *xy*s and *xy-report*s are returned as 12-bit *xy-report*s; *4010-xy-report*s are returned in their 10-bit format.

32-Bit Coordinate Mode

When the terminal is in 32-bit Coordinate mode, *xy* coordinates are expressed in 32-bit *xy* format. All *xy-reports*, including *4010-xy-reports* are sent as a pair of *int-reports*.

THE ANSI PARSER

Only raster display terminals contain an ANSI parser. The ANSI parser allows a 4110 Series raster display terminal to respond to a subset of the ANSI X3.64 command set. Section 5, *The Operator Interface*, contains a discussion of the commands supported by the ANSI parser.

The ANSI parser modes are all automatically entered and left by the parser as it receives characters.

ANSI ALPHA MODE

Alpha mode is the implicit mode for the ANSI parser. When not parsing a command, the ANSI parser is in Alpha mode. When it has finished parsing a command it returns to Alpha mode.

In Alpha mode, the ANSI parser interprets incoming printable characters as the characters themselves. It ignores control characters other than $^{B}_{L}$, $^{B}_{S}$, $^{H}_{T}$, $^{L}_{F}$, $^{V}_{T}$, $^{F}_{F}$, $^{C}_{R}$, $^{S}_{O}$, $^{S}_{I}$, or $^{E}_{C}$.

LCE-A STATE

The ANSI parser enters LCE-A (last character escape ANSI) state after receiving an $^{E}_{C}$ character. It leaves after the next character. The ANSI parser exits to Alpha mode if the character completes a command, or enters CSI state if the second character completes the control sequence identifier.

In LCE-A state, the parser is waiting for the next character in a command.

CSI STATE

The ANSI parser enters CSI state from LCE-A state when the next character is [. The parser leaves CSI state when it receives a terminator character.

In CSI state, the ANSI parser parses characters as parameters, separators, or terminators. When the command is terminated, the parser returns to Alpha mode.

Parameter characters include the digits (ADE 48 through 57) and the characters: <, =, >, and ? (ADE 60 through 63). The only separator character is : (ADE 58). The terminators are: @ through ~ (ADE through 126).

Section 5

THE OPERATOR INTERFACE

INTRODUCTION

4110 Series terminals are efficient and easy for an operator to use. The individual operator's manuals for each terminal detail the features available to the operator. This section contains information for the host programmer who is writing programs that communicate with the operator.

PREVIEW

This section contains discussions on:

- The dialog area
- ANSI mode
- The keyboard
- The display

THE DIALOG AREA

PREVIEW CONCEPTS AND DEFINITIONS

4110 Series terminals are designed for two main uses: alphanumeric communication with a host and displaying graphics to the operator. In many cases, these two uses are incompatible, dialog from the host might interfere with or be obscured by the graphics on the terminal screen. In order to solve this problem, 4110 series terminals feature a *dialog area*, programmable in size and location, to display host messages and the operator responses.

In DVST terminals, the dialog area is displayed in *refresh*, the image is not stored on the display screen. In raster display terminals, the dialog area is placed on a surface (discussed in Section 8, *Raster Display Graphics*). In the 4112 and 4113, the dialog area can be on any of the surfaces, including those used for graphics. In the 4115, the dialog area is assigned to a surface, but occupies a different hardware overlay. The result is: the dialog area on the 4115 can be assigned to any surface but does not interfere with graphics on that surface. If you are programming for a 4112 or 4113 and have enough bit planes, you should put the dialog area on a separate surface.

Text sent to the dialog area is stored in the *dialog scroll*, a buffer whose size you can set from the host. The terminal operator can use the vertical terminal thumbwheel to scroll through the contents of the dialog scroll buffer.

The dialog area parameters, including visibility and presence are retained in nonvolatile memory. On power-up the terminal comes up with the dialog area in the same condition in which it was turned off.

The terminal allocates memory for the scroll buffer when it is first made visible. When you change dialog area parameters, the changes are not made until the dialog area is next made visible.

Controlling the Dialog Area

ENABLE-DIALOG-AREA. A parameter of 1 with this command directs alphatext to the dialog area scroll buffer. A parameter of 0 directs alphatext to the graphics area (useful for emulating 4010 Series terminals). This command does not affect the dialog area visibility.

SET-DIALOG-AREA-VISIBILITY. You can use this command to make the dialog area visible or invisible. A parameter of 0 makes the dialog area invisible, a parameter of 1 makes it visible. The terminal operator can change the dialog area visibility from the keyboard by pressing the DIALOG key. This command does not affect the destination of alphatext.

CLEAR-DIALOG-SCROLL. Use this command to erase the scroll buffer. The terminal operator can also clear the scroll buffer with the CLEAR key.

Dialog Area Parameters

The following commands change the various parameters of the dialog area.

SET-ALPHATEXT-SIZE. This command is valid on DVST terminals and the 4115 only. It allows you to specify the size of characters on DVST terminals or choose between two sizes of alphatext on the 4115.

SET-4014-ALPHATEXT-SIZE. This command is valid on the 4115 (where it selects between two character sizes) and DVST terminals.

SET-DIALOG-AREA-CHARS. Set the number of characters displayed on each line of the dialog area with this command. This command changes the width of the dialog area to correspond to the number of characters you will display. This command also reduces the width of the lines stored in the scroll buffer. You can reduce the memory needed by the scroll buffer by reducing the number of characters per line.

SET-DIALOG-AREA-LINES. Use this command to set the number of lines to display in the dialog area.

SET-DIALOG-AREA-BUFFER-SIZE. You can set the number of lines saved in the scroll buffer from 2 to 32767 with this command.

SET-DIALOG-AREA-POSITION. This command moves the lower left corner of the dialog area as close as possible to the specified position without reducing the number of dialog area lines or characters.

SET-DIALOG-AREA-WRITING-MODE. This command determines whether or not characters written over existing characters will overstrike or replace existing characters. The 4115 has limited overstrike capability, you can overstrike only with the underscore or blank characters.

SET-DIALOG-AREA-SURFACE. This command is valid on raster display terminals only. On the 4115, this command specifies which surface's color map to use on the dialog area.

SET-DIALOG-AREA-INDEX. This command is valid on raster display terminals only. Use this command to set the color index for the characters, the color index for the background, and the *wipe index*, the color index used to erase the dialog area. (On the 4115, the wipe index is always 0.)

ANSI MODE

INTRODUCTION

4110 Series raster display terminals can be used for screen editing by placing them in ANSI mode. In ANSI mode, terminal's ANSI parser responds to a subset of the ANSI X3.64 command set; the terminal no longer recognizes the 4110 Series command set. DVST terminals do not contain an ANSI parser.

You can put a 4110 Series raster terminal into ANSI mode from TEK mode or TEK mode from ANSI mode with the SELECT-CODE command. You specify the mode to enter by the parameter; 0 specifies TEK mode, and 1 specifies ANSI mode.

Most screen editors include a terminal initialization file. The terminal initialization file should include the TEK mode commands to make the dialog area visible and set the dialog area buffer to the correct size to work with your screen editor. It is not necessary to enable the dialog area, since the ANSI parser automatically directs all commands, including alphatext, to the dialog area.

PREVIEW

This discussion includes:

- Screen editor basics
- Cursor positioning commands
- Tabulation commands
- Editing commands
  - Inserting
  - Deleting
  - Erasing
- Display control
- Terminal control
- Communication
- ANSI sub-modes

CONCEPTS AND DEFINITIONS

Screen Editors

A *screen editor* is a host program that displays a copy of the text that the operator is working on. The operator has the illusion of directly entering and editing the text. Most screen editors do not totally redraw the display each time the operator changes something. Instead, they maintain a copy of the terminal display in a buffer which they update in the same way the terminal updates its display in response to the operator. The only communication between the host and the terminal is the transmission of the commands and characters that are changed.

4110 Series terminals implement a large enough subset of the ANSI X3.64 standard commands to allow the use of most purchased screen editors.

Cursor Positioning

The *cursor* is the marker used by the terminal to show where the next editing command will take place. The *cursor position* is the line and column number that define the position of the cursor where (0,0) is the upper left corner of the dialog area.

The cursor positioning commands are:

- CURSOR-FORWARD (CUF) — move the cursor *n* columns to the right on the current line
- CURSOR-BACKWARD (CUB) — move the cursor *n* columns to the left on the current line
- CURSOR-UP (CUU) — move the cursor up the specified number of lines
- CURSOR-DOWN (CUD) — move the cursor down the specified number of lines
- CURSOR-POSITION (CUP) — move the cursor to the specified line and column
- HORIZONTAL-AND-VERTICAL-POSITION (HVP) — nearly identical to CURSOR-POSITION
- NEXT-LINE (NEL) — move the cursor to the beginning of the next line
- INDEX (IND) move the cursor down one line and keep the current column position
- REVERSE-INDEX (RI) — move the cursor up one line and keep the current column position
- $^{C}_{R}$ character — obeys the CRLF setting
- $^{F}_{F}$ character — nearly identical to INDEX
- $^{L}_{F}$ character — obeys the LFCR setting
- $^{V}_{T}$ character — nearly identical to INDEX
- $^{H}_{T}$ character — move the cursor to the next horizontal tab stop no wrap-around
- CURSOR-BACKWARD-TAB (CBT) — moves the cursor backward *n* tab stops on current line with no wrap-around
- CURSOR-HORIZONTAL-TAB (CHT) — moves the cursor forward *n* tab stops with no wrap around

Tabulation Commands

Tabulation commands move the cursor between tab stops and allow you to set and delete tab stops. 4110 Series terminals support only horizontal tabs.

- HORIZONTAL-TAB-SET (HTS) — set a tab stop at the current cursor position
- TABULATION-CLEAR (TBC) — clear tab stops according to the parameter setting: 1 — clear the tab stop at current position 2 — clear all tab stops in the active line (same as 3) 3 — same as 2
- $^{H}_{T}$ character — move the cursor to the next tab stop in current line
- CURSOR-BACKWARD-TAB (CBT) — moves the cursor backward *n* tab stops on current line with no wrap-around
- CURSOR-HORIZONTAL-TAB (CHT) — moves the cursor forward *n* tab stops no wrap around

Editing Commands

Editing commands are of three types: insertion, deletion, and erase. Insertion commands, as their name implies insert blanks at and to the right of the cursor. Text after the cursor is moved to give room for the insertion. Deletion commands remove text from the vicinity of the cursor, and remaining text is pulled in to close the gap. Erasing commands are similar to deletion commands, but the gap is not closed. Erased text is replaced by blanks.

The 4110 ANSI mode editing commands are:

- INSERT-CHARACTER (ICH) — shifts characters at and to the right of the cursor *n* positions to the right
- INSERT-LINE — inserts *n* blank lines at the cursor position
- DELETE-CHARACTER (DCH) — deletes *n* characters from the line to the right of the cursor
- DELETE-LINE (DL) — deletes *n* lines beginning at the cursor
- ERASE-CHARACTER (ECH) — erase *n* characters leaving blanks does not move text wraps to next line
- ERASE-IN-DISPLAY (ED) — deletes characters according to the parameter:

  0 — from cursor through end of scroll, including cursor

  1 — from beginning through and including cursor

  2 — erases entire scroll buffer

Display Control Commands

- SAVE-CURSOR (TEKSC) — terminal saves current cursor position and *graphic rendition*, or text style
- RESTORE-CURSOR (TEKRC) — Restores cursor position and graphic rendition saved with SAVE-CURSOR command
- SCROLL-DOWN (SD) — moves scroll buffer down *n* lines within dialog area does not move cursor — cursor may move out of view
- SCROLL-UP (SU) — moves scroll buffer up *n* lines within dialog area does not move cursor — cursor may move out of view
- SELECT-CHARACTER-SET (SCS) — no-op, included for compatibility with other terminals
- SELECT-GRAPHIC-RENDITION (SGR) — selects the style(s) for displaying text characters

Terminal Control Commands

- ANSI/VT52-MODE (TECKANM) — 4110 Series terminals do not include this mode. The command is a no-op for compatibility with some editors.
- DISABLE-MANUAL-INPUT (DMI) — locks the terminal keyboard and prevents operator from sending any characters
- ENABLE-MANUAL-INPUT (EMI) — unlocks terminal keyboard
- SET-MODE (SM) — sets the terminal mode selected by the parameter(s)
- RESET-MODE (RM) — resets the terminal mode selected by the parameter(s)
- RESET-TO-INITIAL-STATE (RIS) — causes terminal to perform a power-up reset. The effect is the same as pressing the MASTER RESET key or issuing the T4100 RESET command.

Communications

- DEVICE-STATUS-REPORT (DSR) — causes the terminal to send a device status report of the current cursor position

ANSI Sub-Modes

The various ANSI sub-modes are set and reset by the commands:

- SET-MODE (SM) — sets the terminal mode selected by the parameter(s)
- RESET-MODE (RM) — resets the terminal mode selected by the parameter(s)

The modes affected by the SET-MODE and RESET-MODE commands are:

- Keyboard action
- Insertion/Replacement
- Send/Receive
- Linefeed/Newline
- Overstrike/Replace
- Auto-wrap
- Auto-repeat

THE KEYBOARD

The terminal keyboard generates the full 128 character ASCII codes, ADE 0 through 127. In addition, the terminal has eight programmable function keys that generate sixteen (eight unshifted and eight shifted) codes from 128 through 143. Table 5-1 lists the codes generated by the unshifted and shifted function keys.

You can program these function keys, or most of the keyboard keys, by defining a Key macro for the ADE of that key. Section 4, *Using 4110 Series Terminals* discusses macros and macro expansion.

Table 5-1
FUNCTION KEY CODES

Code	Function Key	Code	Shifted Function Key
128	F1	136	S1 (SHIFT-F1)
129	F2	137	S2 (SHIFT-F2)
130	F3	138	S3 (SHIFT-F3)
131	F4	139	S4 (SHIFT-F4)
132	F5	140	S5 (SHIFT-F5)
133	F6	141	S6 (SHIFT-F6)
134	F7	142	S7 (SHIFT-F7)
135	F8	143	S8 (SHIFT-F8)

THE DISPLAY

Many commands control the display from the host. Some commands cause an immediate effect on the display, while others control how the terminal reacts to future conditions — such as filling the page screen with text, or reporting errors.

DISPLAY CONTROL

The following commands directly control the terminal display:

- PAGE
- RENEW-VIEW
- HARDCOPY
- 4010-HARDCOPY

CONTROLLING TERMINAL RESPONSES

The following commands change the terminal's response to situations that may occur later. The immediate effect is usually not visible.

CRLF

Some host systems send only a carriage return at the end of a line. If you need the terminal to generate a ^{L}F with each ^{C}R, you can cause it to generate one with this command.

LFCR

If you need the terminal to assume a ^{C}R for each ^{L}F it receives, you can cause it to do so with this command.

SET-MARGINS

DVST display terminals can set margins within the display area in order to display more than the normal number of lines of alphatext. This command has no affect on raster display terminals. See Section 6, *Graphics Primitives*, for an explanation of alphatext.

SET-PAGE-FULL-ACTION

When the terminal tries to write beyond the last character position on the screen, it is in a page-full condition. You can determine what action the terminal will take when this occurs with the parameter you send with this command. A list of the parameters and their action is contained in the *4110 Series Command Reference Manual* under the discussion of this command.

SET-ECHO

This command determines whether characters typed on the terminal keyboard appear on the terminal screen. In many environments, the host or modem, not the terminal, echoes characters the terminal sends.

SET-ERROR-THRESHOLD

You can control when the terminal displays error messages with this command. When you are communicating with an operator, you should set the error threshold very high, so as not to confuse the operator with the error messages. When you are developing a program from the terminal, set the error threshold at a level that causes the terminal to display errors you want to see.

This command does not affect the terminal's error reporting system. All errors are stored in the terminal's error queue and reported when you give the REPORT-ERRORS command.

Section 6

GRAPHICS PRIMITIVES

INTRODUCTION

Graphics primitives are the graphics elements that the terminal displays in response to a graphics primitive command. In addition to a general discussion of graphics primitives and graphics primitive commands, this section discusses the following:

- Vectors and Vector Mode (MOVE and DRAW)
- Markers and Marker Mode (DRAW-MARKER)
- Text in the Graphics Area (alphatext and graphtext)
- Panels (raster display terminals only)

GRAPHICS PRIMITIVES AND PRIMITIVE COMMANDS

CONCEPTS AND DEFINITIONS

Graphics primitives are the fundamental units of a display. A graphics primitive is drawn in response to a *graphics primitive* command. You can combine a series of graphics primitives (or more precisely graphics primitive commands) into a secondary construct called a *segment*, which you can manipulate as a unit. For example, you can combine a series of MOVEs and DRAWs into a rectangle that, as a segment, you can later reposition, scale, and rotate without repeating the primitive commands that first drew it. This section discusses graphics primitives and how to use them; Section 7 discusses segments.

You can control the appearance of a graphics primitive by selecting its attributes. For example you can choose a vector's color, width, and style (solid, dotted, dashed, etc.).

The *graphics beam* position is the position a terminal uses as a starting point when executing a graphics primitive command.

Explicit and Implicit Commands

An *explicit* command is one that you send using the complete escape sequence and parameters of the command. All commands except alphatext are available as explicit commands. In addition, certain commands may be sent as *implicit* commands; not directly stated but implied by the terminal mode when you send only the parameters of the command.

4110 Series terminals have three implicit command modes: Alpha, Vector and Marker. Each of these modes allows you to send a different implicit command. In Alpha mode the terminal interprets characters as alphatext, while in Vector and Marker mode the terminal interprets the characters (from ^{S}P through ^{D}T only) as encoded *xy* coordinates.

When you put the terminal into Vector or Marker mode, each complete coordinate implies an associated command; in Vector mode each coordinate implies a MOVE or DRAW command, while in Marker mode each coordinate implies a DRAW-MARKER command. By freeing you of the necessity of explicitly sending the escape sequence form or these commands, Vector and Marker modes reduce communication traffic from the host to the terminal.

VECTORS

PREVIEW

- The explicit commands MOVE and DRAW position the graphic beam and draw vectors.
- In Vector mode, the terminal interprets characters as implied MOVE and DRAW commands.
- A vector or a series of vectors draws a line.
- Lines have a number of attributes:
  - *Line style* is an attribute that specifies one of eight dotted and dashed line patterns.
  - *Line index* is an attribute used in raster display terminals to specify line color or shade of gray.
  - *Line width* is an attribute used by DVST terminals to specify one of two different line widths.

CONCEPTS AND DEFINITIONS

A *vector* is a straight line drawn between two points; it is a graphics primitive. The position of the graphics beam defines the starting point of the vector; the position sent as the parameter of the DRAW command forms the end point.

To display a vector on the terminal screen, move the graphics beam to the starting point with a MOVE command, then draw a vector with a DRAW command. To display a complete drawing, give the terminal a sequence of MOVEs and DRAWs. You can draw curves with a series of short straight lines.

Line Attributes

Prior to drawing vectors, select the line attributes to draw the exact type of line you want. Line attributes remain set until you change them; however, on DVST terminals, if the dialog area is disabled, sending a PAGE command, the $^{C}_{R}$ character, or pressing the PAGE key resets the line-style to solid and the line-width to narrow.

Line Style. *Line style* is the attribute that determines the pattern (solid, dashed, dotted, etc.) in which the terminal will draw vectors. You can choose one of eight different line styles with the SET-LINE-STYLE command. Figure 6-1 shows the line styles available.

Line Index. *Line index* is the color index in which a raster display terminal draws vectors and markers. You can set the line index with the SET-LINE-INDEX command. You can also set the index of the dash gap (the space between dots or dashes within dashed lines) with SET-BACKGROUND-INDICES command. The background index also affects the background behind alphatext or string precision graphtext in the graphics area.

Line Width. *Line width* is an attribute used in DVST terminals only. You may choose one of two widths in which to draw subsequent vectors. Set the line width with the SET-LINE-WIDTH command.

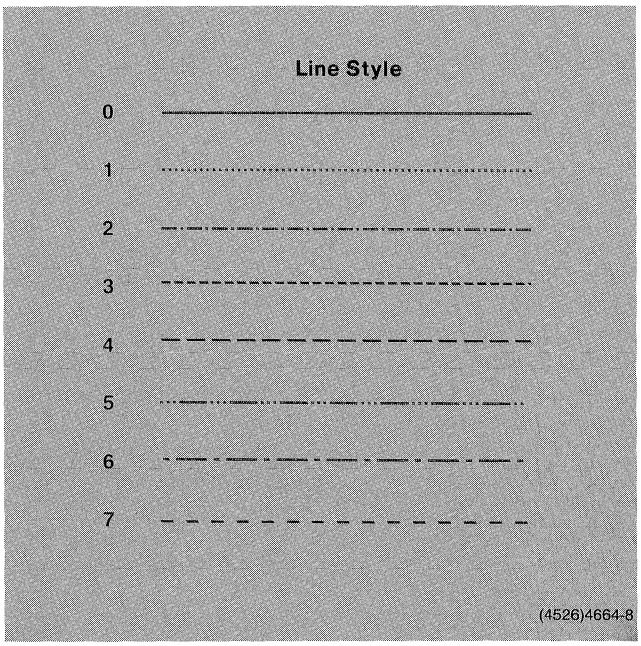


Figure 6-1. Line Styles.

Explicit MOVE and DRAW Commands

The explicit MOVE and DRAW commands are escape sequence commands that do not depend on or change the implicit command modes. The explicit MOVE command is ^{E}cLF xy; the explicit DRAW command is ^{E}cLG xy.

Since the explicit form of these commands takes additional overhead, you should use this form only if your system has trouble sending the ^{G}S or ^{U}S characters, or if the additional overhead does not matter.

Implicit MOVE and DRAW Commands

You can place the terminal in Vector mode by sending the command ENTER-VECTOR-MODE (the single character ^{G}S). In Vector mode, the first xy coordinate following the ^{G}S character is an implicit MOVE to that location (unless it is preceded by a ^{B}L). Subsequent coordinates are implicit DRAWs.

If you want the first implicit command after the ENTER-VECTOR-MODE command to be a DRAW rather than a MOVE, send the sequence ^{G}S^{B}L. The terminal bell will ring, and the following xy coordinate will be an implicit DRAW.

The terminal must be in Alpha or Vector mode in order to enter Vector mode. If the terminal is in Marker mode, it ignores the ^{G}S character.

Leaving Vector Mode

You can leave Vector mode by entering either Alpha or Marker mode. There are other ways to leave Vector mode; see the hint that follows.

HINT

When you use implicit commands, your software must keep track of the implicit command mode of the terminal. Several commands other than ENTER-ALPHA-MODE (^{U}S) can put the terminal into Alpha mode: PAGE, ^{C}R, ENABLE-4010-GIN, and ENABLE-4953-GIN, as well as pressing the PAGE key, all put the terminal in Alpha mode if the dialog area is disabled; RESET will put the terminal into Alpha mode at any time.

MARKERS

PREVIEW

- A marker is a character-like symbol used to identify a point in a drawing.
- The DRAW-MARKER command is the explicit graphics primitive command that locates and draws markers.
- In Marker mode, an xy coordinate is the implicit DRAW-MARKER command.
- Terminals have eleven different marker types.
- A marker's index is the same as the current line index.

CONCEPTS AND DEFINITIONS

Markers

A *marker* is a character-like symbol drawn at a given coordinate. Markers are usually used to mark important points on a drawing; for example, you can use markers to identify data points on a graph or cities on a map.

The terminal always draws markers with solid lines, and the size of the marker on the screen is independent of the current window. A marker will not change size when you zoom the display.

Marker Types

Use the SET-MARKER-TYPE command to choose one of 11 marker types. The command uses only one parameter — an integer from 0 through 10, which identifies the marker type. The *4110 Series Command Reference Manual* contains an illustration of the marker types.

THE EXPLICIT DRAW-MARKER COMMAND

The explicit DRAW-MARKER command is an escape sequence that does not depend on or change the implicit command modes. The explicit DRAW-MARKER command is ^{E}cLH xy.

THE IMPLICIT DRAW-MARKER COMMAND

You can place the terminal in Marker mode by sending the command ENTER-MARKER-MODE (the single character ^{F}S). In Marker mode, each *xy* coordinate following the ^{F}S character is an implicit DRAW-MARKER command.

USES OF MARKERS

You can use markers anywhere in a drawing that you need to accurately identify an important point. Some examples of use are:

- Identifying towns on a map
- Identifying important points on a graph
- Registration points on graphics overlays

HINTS

Markers are more efficient than combinations of MOVEs and DRAWs, but they cannot be scaled or rotated in segments.

In early 4112 and 4113 terminals, the SET-GRAPHICS-WRITING-MODE setting of REPLACE caused markers to wipe an area beneath them with the current text-background-index; to avoid this, use the OVERSTRIKE writing mode.

TEXT IN THE GRAPHICS AREA

PREVIEW

- Text in the graphics area is a graphics primitive.
- Two types of text are available, alphatext and graphtext.
- Alphatext is a graphics primitive only if the dialog area is disabled.
- There are two types of graphtext: string precision and stroke precision.
- String precision graphtext is similar to alphatext; it uses alphatext attributes.
- Stroke precision graphtext is generated (and stored in a segment) as a series of MOVEs and DRAWs.
- Multiple predefined stroke graphtext fonts are available.
- You can define your own stroke graphtext fonts.

CONCEPTS AND DEFINITIONS

Alphatext

In Alpha mode, with the dialog area disabled, the terminal interprets each printable ASCII character it receives as an implicit command to draw that character in the graphics area. This type of text display is called *alphatext*.

NOTE

Alphatext is similar to the text display used in the Tektronix 4010 Series terminals. 4110 series terminals interpret alphatext similar to the way that 4010 Series terminals interpret text.

Alphatext Attributes

Keyboard Option Attribute. Terminals with Option 4E have an APL font as well as the standard ASCII font (APL is not available on the 4115). You should select the font with the command SET-ALPHATEXT-FONT before using alphatext on these terminals. (Other Option 4 keyboards automatically select the font to match the keyboard; the Katakana keyboard is controlled by $^{S}I/^{S}O$.)

DVST Alphatext Attributes. DVST alphatext attributes include character size, character spacing, and line spacing. You can set these attributes with the commands: SET-ALPHATEXT-SIZE, SET-4010-ALPHATEXT-SIZE, and SET-ALPHATEXT-SIZE-GROUP.

Raster Alphatext Attributes. Raster display alphatext attributes include the text foreground index, the text background index, and the *graphics area writing mode*. The graphics area writing mode determines whether characters either overstrike or replace the pixels under them. You can set these attributes with the commands: SET-TEXT-INDEX, SET-BACKGROUND-INDICES, and SET-GRAPHICS-AREA-WRITING-MODE.

Graphtext

When alphatext is not suitable for your display, or you have the dialog area enabled, you can use *graphtext*. To display graphtext, send the GRAPHIC-TEXT command followed by your text as the *string* parameter. (String precision graphtext looks like alphatext, but is sent with the explicit GRAPHIC-TEXT command.)

Graphtext Precision

Graphtext has an attribute called *precision* which controls the type of graphtext. With the command SET-GRAPHTEXT-PRECISION, you choose either *string precision* (alphatext-like) or *stroke precision* (generated by MOVEs and DRAWs) graphtext.

String Precision Graphtext

String precision graphtext uses the same attributes as alphatext. You cannot scale, slant, or rotate string precision graphtext.

Stroke Precision Graphtext

Stroke precision graphtext is generated one vector at a time. Stroke precision graphtext has the following attributes:

- Font — select with SET-GRAPHTEXT-FONT
- Text index — select with SET-TEXT-INDEX
- Size — select with SET-GRAPHTEXT-SIZE
- Slant — select with SET-GRAPHTEXT-SLANT
- Rotation — select with SET-GRAPHTEXT-ROTATION

Graphtext Fonts. You can choose one of several predefined or user defined fonts. You will find a discussion on how to define graphtext fonts later in this section.

Graphtext Size, Slant and Rotation. When the terminal draws graphtext it first scales the text, then slants it and finally rotates it.

DEFINING A GRAPHTEXT FONT

You can define your own fonts for stroke precision graphtext. The process of defining a font consists of initialization, opening a character definition, defining the character, closing the character definition, opening another character definition and so forth until the font is complete.

You need not define all the characters in a graphtext font; those that you do not explicitly define use the default font. To define a font with only one special symbol, you need to define only that symbol.

Initialization

You should first delete the font number that you intend to define. (The terminal gives a level 1 error for deleting a nonexistent font, but gives a level 2 error if you attempt to open a definition for a font that already exists.) To delete a font, use the command DELETE-GRAPHTEXT-CHARACTER for character number −1 of that font.

To initialize the terminal for defining an graphtext font, you must then issue two commands: SET-GRAPHTEXT-FONT-GRID, and SET-PIVOT-POINT.

SET-GRAPHTEXT-FONT-GRID. This command opens the graphtext font definition, and sets up the *graphtext font grid*. The *graphtext font grid* is a rectangle extending above and to the right of the pivot point. The graphtext font grid defines the character cell the terminal uses when drawing graphtext. You can use the graphtext font grid as a reference when you define your characters, but you need not draw the character within it.

SET-PIVOT-POINT. This command sets the origin (lower left corner of the graphtext font grid) for the graphtext character. The terminal saves all coordinates used when you define your graphtext characters as offsets from the pivot point. The pivot point is discussed further in Section 7, *Segments*.

Graphtext Character Definition

Open each character definition with the command BEGIN-GRAPHTEXT-CHARACTER. Use a sequence of MOVEs and DRAWs to draw the character. You can extend vectors beyond the bounds of the cell, such as descenders on lower case characters.

When your character is complete, give the command END-GRAPHTEXT-CHARACTER to close the character definition.

To delete either a single character or an entire graphtext font, use the DELETE-GRAPHTEXT-CHARACTER command.

HINTS

Eliminating Character Definition Display

When you are defining a character, it appears on the screen as you are defining it. If you want to eliminate this display, two methods are:

1. Define the character or font inside an invisible segment.
   a. Set the segment visibility for segment −2 to invisible.
   b. Open a segment definition.
   c. Define the character or font.
   d. Close the segment definition.
   e. Delete the segment.
2. Position the cell and definition vectors outside the visible area. By keeping the y-coordinates of the pivot point and all definition vectors greater than 3200, the definition will remain off the screen.

Saving a Graphtext Font on Disk

If you want to save a graphtext font on disk:

1. Define an EOF string.
2. Send the command COPY HO: TO *font-filename*.
3. Define the font normally.
4. Send the EOF string.
5. Reset the EOF string to the empty string.

Once the disk file with the font has been created, you can load the font from the disk by issuing the LOAD command from your program.

PANELS

PREVIEW

- A panel is a graphics primitive.
- Panels are available on raster display terminals only.
- A panel is a closed figure bounded by one or more panel boundaries.
- You can fill panels with a solid color or pattern.
- You can key patterns to the panel, view, or screen.
- Panels can replace or overstrike other graphics on the screen.
- You can use the predefined patterns or define your own.
- The 4115 can draw and fill rectangles, a special type of panel, very quickly.

CONCEPTS AND DEFINITIONS

Panels

A *Panel* is the set of pixels on a surface which lies inside a closed Panel Boundary. A *panel definition* is a series of MOVEs and DRAWs which form a panel boundary. You define a panel by first opening the panel definition, then drawing the panel boundary, and finally closing the panel definition. As soon as you close the panel definition, the terminal fills the panel with a solid color or pattern selected by the SET-PANEL-FILLING-MODE command.

The 4115 can also define and fill *rectangles*, a special case of panels. You specify rectangles by giving an array of pairs of *xy* coordinates as the parameters of the DRAW-RECTANGLE command. Each pair of coordinates defines a rectangular panel.

Panel Boundary

A *Panel Boundary* is the closed set of lines defined after the terminal receives the BEGIN-PANEL-BOUNDARY command and before it receives the END-PANEL command. You need not completely close the panel boundary because the terminal closes the remaining gap when you close the panel definition. You must choose to draw or not draw the panel boundary when you issue the BEGIN-PANEL-BOUNDARY command.

A point or pixel on the picture plane is defined as *inside* a panel if an imaginary line drawn from that point to any point outside the terminal space crosses an odd number of panel boundaries. If the line crosses an even number of panel boundaries or no panel boundaries, the point is *outside* the panel. You can use multiple boundaries to define a panel, and you can create panels in any shape.

Figure 6-2 illustrates several panels with the inside of each panel filled with a pattern.

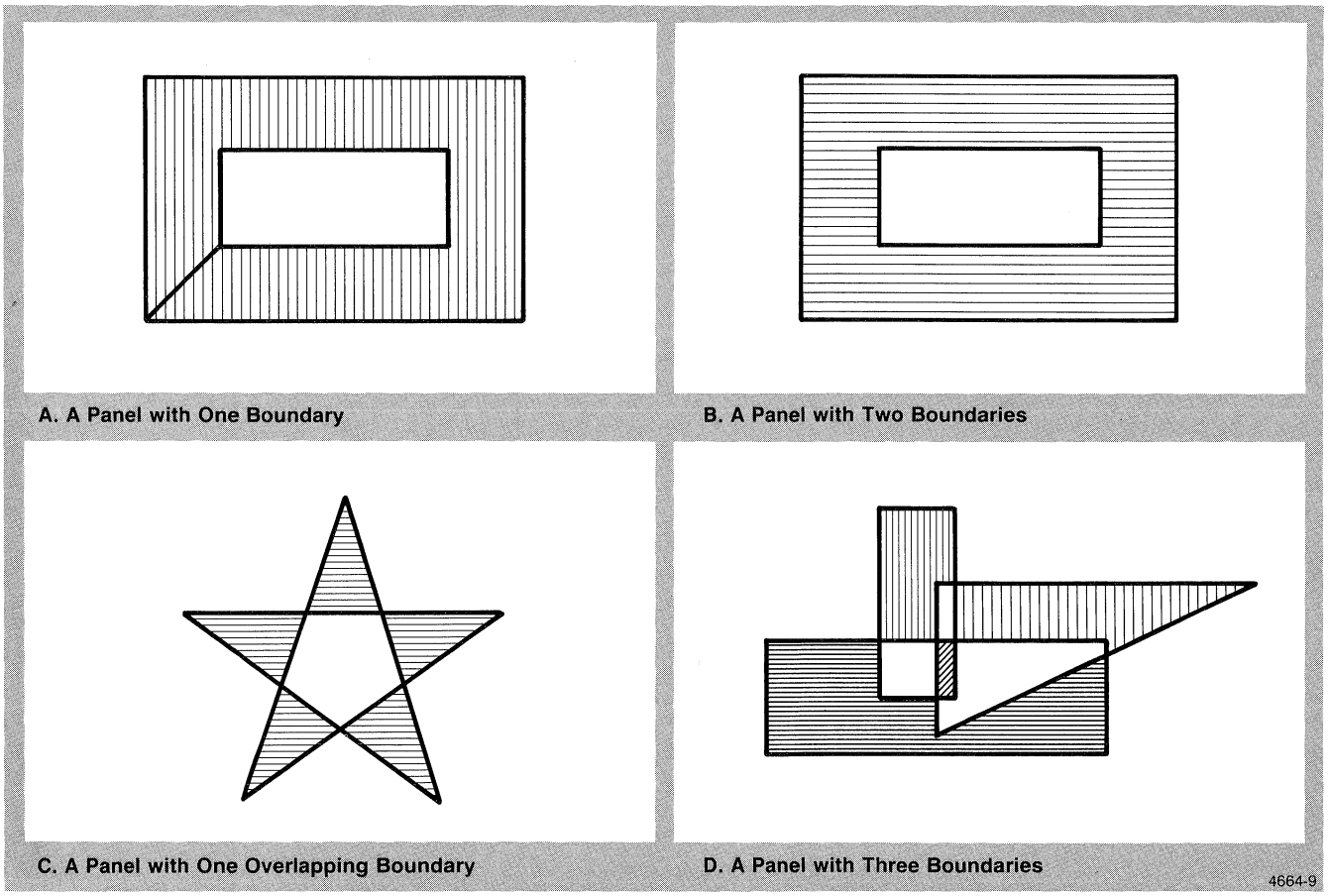


Figure 6-2. Inside of Panels Filled With a Pattern.

Rectangle Boundaries

When you define rectangles on the 4115 with the DRAW-RECTANGLE command, the terminal supplies the vectors that define the panel boundary of a rectangle. Each pair of coordinates closes the previous rectangle definition and opens a new one. Overlapping rectangles appear as one on top of the other.

When you define a rectangle inside a panel definition, the DRAW-RECTANGLE command closes the current panel boundary before beginning the first rectangle boundary, but does not close the last rectangle boundary. As a result, overlapping rectangles have a different appearance when drawn outside a panel definition than when drawn inside a panel definition. Figure 6-3 shows rectangles drawn outside and inside a panel definition.

Panel Attributes

The panel attributes are:

- Overstrike/replace — set by SET-PANEL-FILLING-MODE
- Boundary cover — set by SET-PANEL-FILLING-MODE
- Pattern keying — set by SET-PANEL-FILLING-MODE
- Panel boundary visibility — set by BEGIN-PANEL-BOUNDARY
- Rectangle boundary visibility — set by SET-DRAW-BOUNDARY-MODE
- Panel fill-pattern — set by SELECT-FILL-PATTERN

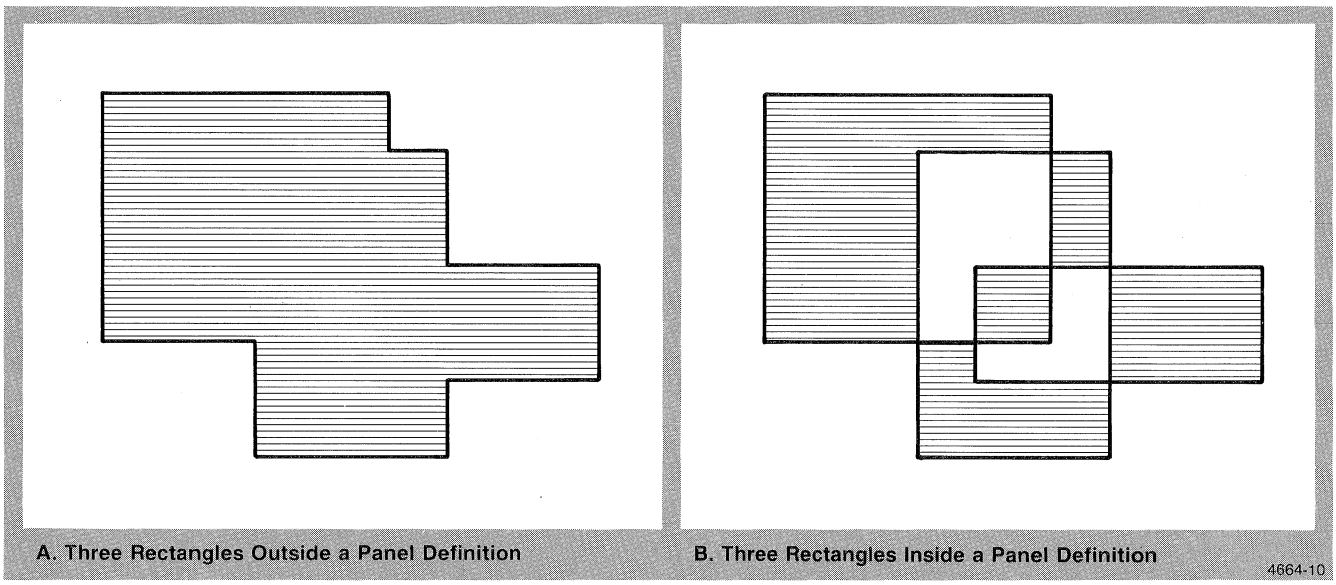


Figure 6-3. Rectangles Drawn Inside and Outside a Panel Definition.

Section 7

SEGMENTS

INTRODUCTION

One of the most powerful features of the TEKTRONIX 4110 Series graphics terminals is the ability to define and use segments. This section explains what a segment is, some uses for segments, and how to define a segment. In this section you will find the following discussions:

- An introduction to segments
- How to build segments
- Segment classes and matching classes

AN INTRODUCTION TO SEGMENTS

PREVIEW

- A segment is a reusable collection of graphics primitives and primitive attributes stored in the terminal's memory.
- The terminal can manipulate a segment as a single object.
- Using segments reduces communications time with the host.
- Segments are named by integers from 1 through 32767.
- Segment numbers 0, −1, −2, and −3 each have a special purpose.
- Segment attributes affect how the terminal draws the segment.
- You can change a segment's dynamic attributes after the segment has been defined.
- A segment's static attribute (the pivot point) cannot be changed after the segment definition has begun.

CONCEPTS AND DEFINITIONS

What Is a Segment?

A *segment* is a collection of graphics primitives and their attributes that the terminal a treats as a single object. A segment can contain nothing, as little as a single graphics primitive, or as much as an entire display. The terminal can draw, translate, rotate, or scale segments.

The terminal stores segments in its memory as a list of graphics primitives and changes to the primitive attributes. Once a segment is defined, you cannot change the graphics primitives or their attributes.

You will find a discussion on how to define segments later in this section.

Retained and Non Retained Segments

When you define a segment, the terminal stores the graphics primitives in a segment definition as a *retained segment* — or more simply, a *segment*. The terminal displays these graphics primitives as you send them, unless you use the fixup level or segment visibility to suppress the display.

The terminal also displays the graphics primitives that you send outside of a segment definition. Once these primitives are executed, however, the terminal can no longer access them. These sequences of graphics primitives that are not part of a segment are sometimes called *non retained segments*. In this manual, whenever the term *segment* is used, it means a retained segment: one that is stored in the terminal's memory and is accessible to the terminal.

Segment Numbering

Each segment must have a unique name, an integer from 1 through 32767. All segment commands require a segment number to identify the segment. Three segment commands (BEGIN HIGHER SEGMENT, BEGIN LOWER SEGMENT, and END SEGMENT) use an implicit segment number from, or calculated from, an ongoing segment definition. The remaining segment commands require an explicit segment number. This segment number can be a normal segment number, or in the case of some commands, a special segment number.

Segment numbers 0, −1, −2, and −3 have special meaning and can be used as parameters for some segment commands. You should check the *4110 Series Command Reference Manual* to see if a special segment number is an allowed parameter for a particular command. The meaning of each special segment number is as follows:

- *Segment 0* is the crosshair graphics input cursor. The crosshair cursor is generated by circuitry inside the terminal. You cannot manipulate it in all the ways that you can manipulate other segments. You can only position it, set its visibility, or report on it.

  For example, although you can move the crosshair cursor position by using the SET-SEGMENT-POSITION command, you cannot scale or rotate the crosshair cursor. This means that segment number 0 is not allowed as a parameter in the SET-SEGMENT-IMAGE-TRANSFORM command.

- *Segment −1* means "all segments currently defined." This includes defined segments from 1 through 32767. Segment 0 is not included in Segment −1.

  Segment −1 can be used in commands that change the dynamic segment attributes. For example, if you want to make all currently defined segments invisible, you can use −1 as the segment number in the SET-SEGMENT-VISIBILITY command.

- *Segment −2* means "all future segments." For example, if you do not want to see segments as they are being defined, make Segment −2 invisible. (You must make individual segments visible before the terminal will display them.)

- *Segment −3* means "all segments, from Segment 1 through Segment 32767, that match the current segment matching class." Using −3 as a segment number allows you to manipulate a class of segments simultaneously. A discussion on segment classes and matching classes is included later in this section.

SEGMENT ATTRIBUTES

A retained segment has a large number of *attributes*: definable characteristics that affect the visible display of the segment. One attribute is set before you define the segment and cannot be changed. Other attributes can be altered to change the appearance of the segment.

The Pivot Point and the Segment Origin

The *pivot point* is the coordinate specified by the SET-PIVOT-POINT command. In an untransformed segment, the pivot point is the origin of the segment. When you define a segment, the graphics primitive commands are stored relative to the segment's pivot point. If you transform a segment, the rotation is about the segment origin and scaling is centered on it.

If you define segments after changing the location of future segments by issuing the SET-SEGMENT-POSITION or SET-SEGMENT-IMAGE-TRANSFORM commands for Segment −2 (all future segments), the origin of newly defined segments is defined as though you had moved the pivot point with the SET-PIVOT-POINT command. The pivot point, however, remains at the coordinates set by the SET-PIVOT-POINT command.

When you move a segment, the terminal moves the segment origin to the location you specify; then redraws the segment by relative MOVEs and DRAWs about that location. Thus, each time the terminal draws a segment, the terminal begins with an absolute MOVE to the segment location, then draws the segment relative to that location.

Dynamic Segment Attributes

A segment's *dynamic attributes* are those attributes that can be changed after the segment is defined. These attributes are:

- Position
- Scaling
- Rotation
- Visibility
- Writing mode
- Highlighting
- Detectability
- Display priority
- Segment class

Position, Scaling, and Rotation. The command SET-SEGMENT-POSITION changes the location of the segment's origin in terminal space. On raster display terminals, you may not see a change in the display after moving a segment, depending on the fixup level. On DVST terminals, you will see segments — that are displayed in Storage mode — displayed in their new position, while the old image remains on the screen.

In addition to simply positioning a segment, you might want to change the size or rotation of a segment. You can scale, rotate, and position a segment with the SET-SEGMENT-IMAGE-TRANSFORM command.

On a raster display terminal you can make segments visible or invisible with the SET-SEGMENT-VISIBILITY command. A visible segment is retained and displayed; an invisible segment is retained but not displayed.

The SET-SEGMENT-WRITING-MODE command determines just how a terminal draws a segment on the display screen. This command affects raster display and DVST terminals differently.

When a raster display terminal displays a segment, it writes a pattern into the raster memory. All raster terminals can draw the segment in one of two ways: *Set mode* and *XOR mode*. The 4115, in addition, can draw in *AND mode* and *OR mode*.

In Set mode, the terminal writes the color index for each pixel in the segment into the raster memory.

In XOR mode, the terminal writes in raster memory the result of a bit-by-bit exclusive OR between the index in the raster memory cell and the color index of the affected pixel. The terminal can erase a segment by redrawing it in XOR mode.

A disadvantage of XOR mode is that segment overlaps can be drawn in the color index that is the result of the XOR between the two color indices. When it is important that the segment look correct at an overlap, you should either use Set mode, or set the color map such that the XORed index will be a color you want. When you want to be able to remove the segment by redrawing it, you should use XOR mode.

The 4115 AND and OR modes are bit-by-bit logical AND or OR operations between the color index of the pixel and the raster memory cell of the affected pixel.

When a DVST terminal displays a segment, it displays in either storage or refresh mode. Storage mode allows you to write the segment to the screen, where it remains until the screen is erased. Refresh mode allows you to delete a segment from the display without erasing the screen. (Erasing the screen uses terminal processor time.)

You can *highlight* — that is, cause a segment to blink between invisible and visible — with the SET-SEGMENT-HIGHLIGHTING command. DVST terminals display highlighted segments in refresh and alternately make them visible and invisible.

The *detectability* of a segment determines whether or not the segment can be picked by the GIN Pick function. You can set the segment detectability with the command SET-SEGMENT-DETECTABILITY. For more information see Section 9, *Graphics Input*.

The *display priority* of a segment determines the order in which the terminal draws segments on the display screen and the order in which the terminal scans segments during a GIN Pick. You can change the segment display priority with the SET-SEGMENT-DISPLAY-PRIORITY command. The display priority for a segment may be any integer from −32768 through +32767.

When a raster display terminal redraws segments, it draws those segments with a higher display priority later than those with a lower display priority. If segments overlap, the segments drawn later appear to be closer to the viewer. Figure 7-1 shows how you can change the appearance of a picture by interchanging the display priority of two segments.

When the terminal is doing a GIN Pick, it examines segments in priority order. Thus, if parts of several segments fall in the pick aperture, the highest priority segment is picked. See Section 9, *Graphics Input* for details on the Pick function.

Segment Class. Each segment contains a 64-bit segment class field. The contents of this field can be changed after the segment is defined. You will find a discussion on segment classes later in this section.

USE OF SEGMENTS

The primary reason to use segments is to reduce communication with the host computer. You do not have to send the entire group of graphics primitives to draw your display each time you change it. Rather, the terminal remembers and redraws the visible segments to form the display. When the terminal does a pan or zoom, it draws the new display from the segments which define the old display.

You can also use segments to replicate images on the display, or use a segment as a GIN cursor.

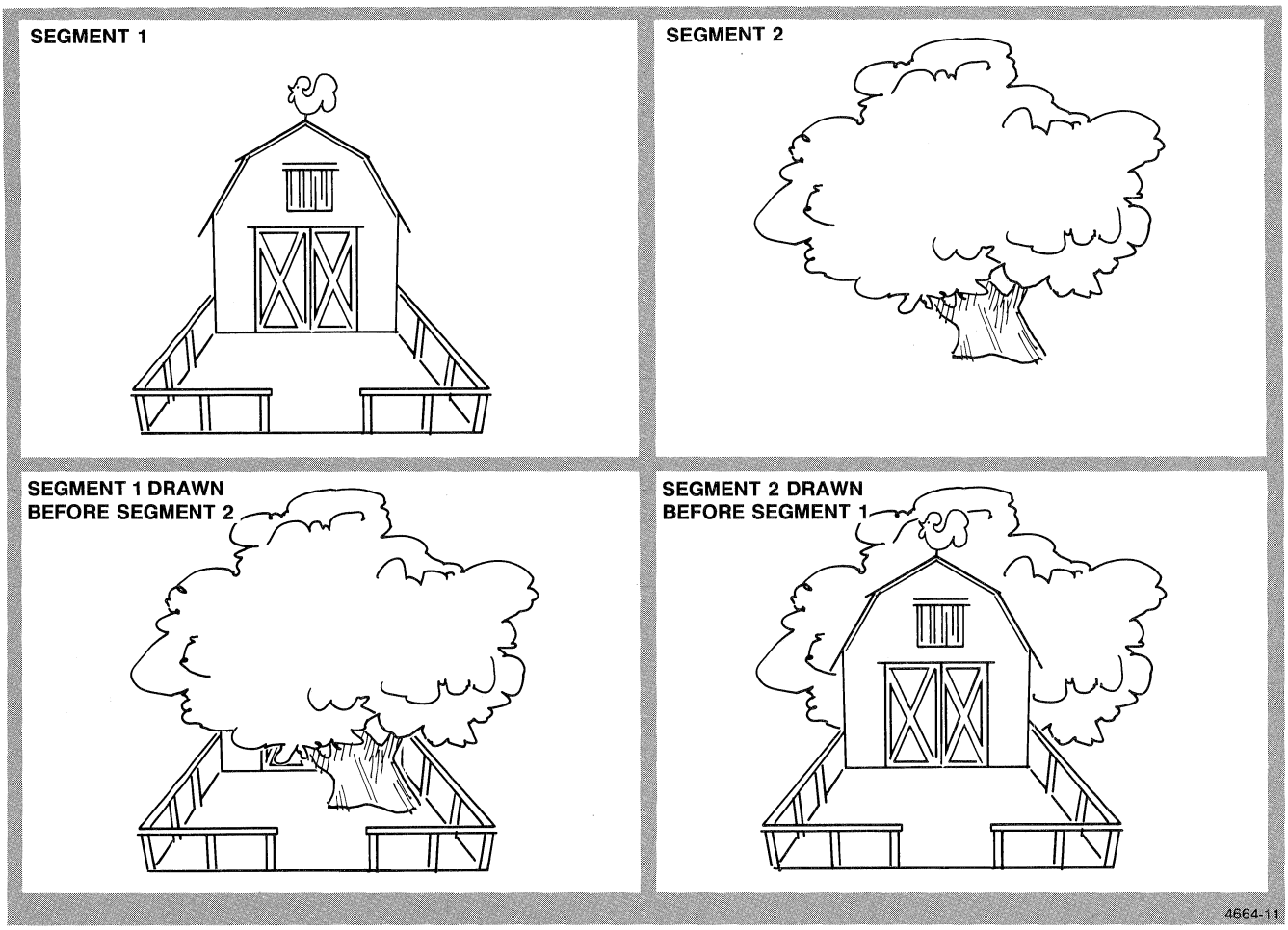


Figure 7-1. Interchanging the Display Priority of Two Segments.

BUILDING SEGMENTS

PREVIEW

- You build a segment by:
  1. Setting the attributes for future segments
  2. Opening a segment definition
  3. Sending the graphics primitives and primitive attributes
  4. Closing the segment definition
- You set attributes for future segments by using −2 as the segment-number parameter
- You can open a segment definition with the commands:

  BEGIN-SEGMENT
  BEGIN-NEW-SEGMENT
  BEGIN-HIGHER-SEGMENT
  BEGIN-LOWER-SEGMENT
- The body of a segment is one or more graphics primitives and associated primitive attributes.
- You can include the segment body of an existing segment in a new segment definition.
- You can close a segment definition with the commands:

  END-SEGMENT
  BEGIN-NEW-SEGMENT
  BEGIN-HIGHER-SEGMENT
  BEGIN-LOWER-SEGMENT
- You can delete segments with the command DELETE-SEGMENT

THE SEGMENT DEFINITION

Setting Attributes for Future Segments

To set the attributes for the segments that have not yet been defined (that is, *future segments*), set the various segment attributes using −2 for the segment-number parameter. These attributes take effect when the segment definition is opened, with the exception of segment highlighting, segment detectability, and segment class. The latter attributes require the segment to be defined and the segment definition closed before they take effect.

You do not need to send a complete set of future segment attributes each time you open a segment definition. Each time you set the future-segment attributes, these attributes remain in effect until you change them. On power up, the terminal sets each attribute to a default value.

Set the future segment's origin with the command SET-PIVOT-POINT before you open the segment definition. (You can also use the commands SET-SEGMENT-POSITION or SET-SEGMENT-IMAGE-TRANSFORM for Segment −2.) Once the segment definition is opened, the origin cannot be changed.

Set the dynamic segment attributes you want to change with the commands:

- SET-SEGMENT-POSITION
- SET-SEGMENT-IMAGE-TRANSFORM
- SET-SEGMENT-VISIBILITY
- SET-SEGMENT-WRITING-MODE
- SET-SEGMENT-HIGHLIGHTING
- SET-SEGMENT-DETECTABILITY
- SET-SEGMENT-DISPLAY-PRIORITY
- SET-SEGMENT-CLASS

Opening the Segment Definition

The commands that you can use to open a segment definition are:

- BEGIN-SEGMENT
- BEGIN-NEW-SEGMENT
- BEGIN-HIGHER-SEGMENT
- BEGIN-LOWER-SEGMENT

When you define a single segment or the first segment in a sequence of segments, use the BEGIN-SEGMENT or BEGIN-NEW-SEGMENT command. The other two segment opening commands require you to have a currently open segment definition.

When you are defining a series of segments, use the commands BEGIN-HIGHER-SEGMENT and BEGIN-LOWER-SEGMENT. These commands close the current segment and open the next one, which saves communications time and host overhead.

The commands BEGIN-SEGMENT and BEGIN-NEW-SEGMENT must have an explicit segment number as a parameter.

The commands BEGIN-HIGHER-SEGMENT and BEGIN-LOWER-SEGMENT open segments implicitly numbered either one higher or one lower than the segment number of the currently open segment definition.

NOTE

The commands BEGIN-NEW-SEGMENT, BEGIN-HIGHER-SEGMENT, and BEGIN-LOWER-SEGMENT all create segments whose origin is at the current graphics beam position. Use these commands for segments that will remain in the same position. Use the BEGIN-SEGMENT command for defining segments that you will move and transform.

Contents of a Segment Definition. Segments can contain any graphics primitives and their associated attributes such as:

- Vectors
- Markers
- Text
- Panels (raster display terminals only)

Including an Existing Segment in a Segment Definition. When you are defining a segment, you can include a copy of the contents of a currently defined segment with the INCLUDE-COPY-OF-SEGMENT command. This command causes an image of the specified segment's contents to be duplicated in the new segment definition. The primitive attributes are copied from the old segment as well as the graphics primitives. After the copy, the terminal's graphic beam position and primitive attributes return to the values they had before the copy.

How the Terminal Stores a Segment Definition. When the terminal builds a segment definition, it constructs a data structure as follows:

1. The terminal reserves a number of internal memory blocks when it opens a segment definition, then builds a segment header based on the future-segment attributes. The terminal adds an absolute move to the segment origin, then adds graphics primitives as it receives them.
2. The terminal translates the absolute xy coordinates from the graphics primitives it receives into coordinates relative to its pivot point. (This simplifies later transformations about the pivot point, mapping the pivot point onto address points, or mapping the pivot point onto the GIN cursor.)
3. When the terminal closes a segment definition, it puts an *end-of-segment* mark on the final block of memory that it used and frees the remaining unused memory.

Closing the Segment Definition

Close the segment definition with one of the following commands:

- END-SEGMENT
- BEGIN-NEW-SEGMENT
- BEGIN-HIGHER-SEGMENT
- BEGIN-LOWER-SEGMENT

Use the END-SEGMENT command to close a single segment or the last segment in a sequence of segments. Use one of the other commands if you are defining more segments in a series.

Appearance of the Display When Defining a Segment

A raster display terminal will, depending on the fixup level, display graphics primitives as it receives them. If the future segment transform is other than either a scale of 1 or a rotation of 0, or if the future segment position is other than the pivot point, the terminal transforms the segments as it receives them and displays the transformed segments. (A discussion on the fixup level can be found in Section 8, *Raster Graphics*.)

In DVST terminals, graphics primitives are displayed only if they are not transformed or moved. If the future segment is transformed or moved, the segment will be transformed, moved, and displayed when the segment is closed.

Commands That Are Not Part of a Segment Definition

If the terminal receives commands that are not part of a segment definition (such as pixel commands) while a segment definition is open, the terminal simply executes these commands. You can change future segment attributes while a segment definition is open, but these future segment attributes will not affect the segment currently being defined.

SEGMENT CLASSES AND MATCHING CLASSES

PREVIEW

- You can manipulate entire classes of segments with commands that allow −3 as the segment-number parameter.
- Each segment contains a 64-bit segment class field.
- The terminal examines the segment class field to determine whether the segment matches the current matching class.
- You can control the matching operation by setting the segment class field and the segment's current matching-class.

CONCEPTS AND DEFINITIONS

In some applications, you might want to manipulate groups of segments with a single command. For example, in a complex picture you might want to highlight all segments that have some feature in common. Or you might want to have some means of selecting a segment or group of segments out of a large collection of segments without knowing each segment number.

In this discussion you will find some examples of how to use segment classes and matching classes. The choice of segment class meanings in these examples is not meant to be representative of a real application, but to illustrate how to manipulate segment classes and matching classes.

Segment Class Field

Within each segment definition is a 64-bit field called the *segment class field*. An individual bit of the segment class field is sometimes called a *segment class* and referenced by its bit number. By storing 1's and 0's in this field, you can create up to 2^{64} different bit patterns. By defining subfields in the 64-bit segment class field, you can express many different relationships between segments.

You store values in the segment class field using the SET-SEGMENT-CLASS command. The SET-SEGMENT-CLASS command takes three parameters: the segment being acted upon, a removal array, and an addition array.

The Removal Array. The removal array is the second parameter of the SET-SEGMENT-CLASS command. This parameter is an array of integers between 1 and 64 or an array consisting of the single integer −1. Each integer stands for a bit position in the segment class field. If you put an integer between 1 and 64 in the removal array, you set that bit of the segment class field to 0. If you put the integer −1 in the removal array, you set all bits in the segment class field to 0.

The Addition Array. The addition array is the third parameter of the SET-SEGMENT-CLASS command. As with the removal array, this parameter is an array of integers between 1 and 64 or an array consisting of the single integer −1. Each positive integer between 1 and 64 that you include in the addition array sets that bit in the segment class field to 1. If you put the integer −1 in the addition array, you set all bits in the segment class field to 1.

A Simple Example. Assume that you will be drawing a mechanical assembly with a lot of parts. You can use some of the bits in the segment class field to indicate part types and others to indicate membership in a subassembly. If you use Bits 1 through 10 for a a part-type subfield, you can have 1024 different part types. If you use Bits 11 through 15 as a subassembly subfield, you can have 32 different subassemblies. The other bits of the segment class field are available for other uses.

Assume that you have defined Segment 50 as a component in your drawing. You should first set the segment class field to all 0's (use −1 in the removal array). Then calculate which bits in the addition array should be set to 1. Assuming Part Number 27 and Subassembly Number 12, your addition array would include 5, 4, 2, and 1 (for 27) and 14 and 13 (for 12) The entire addition array would then be:

14,13,5,4,2,1

Current Matching Class

Definition. The *current matching class* is defined by a pair of 64-bit terminal registers, the inclusion class register and the exclusion class register. These registers are defined by two arrays sent as parameters to the SET-CURRENT-MATCHING-CLASS command. When you use −3 as the segment-number parameter in a segment manipulation command, the terminal compares every segment's segment class field with the inclusion and exclusion class registers. Each segment that passes the matching operation is acted upon by the segment command. By defining the matching class registers properly, you can cause a segment manipulation command to affect any group of segments.

The Inclusion Array. This parameter of the SET-CURRENT-MATCHING-CLASS command is an array of integers between 1 and 64 or the single integer −1. Bits in the inclusion class register are set to 1 if that integer is included in the array and to 0 if that integer is not included in the array. If the array consists of the single integer −1, all bits in the inclusion class register are set to 1. To set bits 13 and 14 (Subassembly 12 from our earlier example) in the inclusion class register to 1, your inclusion array should be:

14,13

The Exclusion Array. This parameter of the SET-CURRENT-MATCHING-CLASS command is an array of integers between 1 and 64 or the single integer −1. Bits in the exclusion class register are set to 1 if that integer is included in the array and to 0 if that integer is not included in the array. If the array consists of the single integer −1, all bits in the exclusion class register are set to 1. If you want to set Bits 1, 2, 4, and 5 (Part Number 27 in our previous example) to 1, your exclusion array should be:

5,4,2,1

The Matching Operation. When you have a segment number of −3 for a segment command, the terminal performs a bitwise comparison between the segment class field of each segment and the inclusion and exclusion class registers in the terminal. If the result of this comparison is true, the operation is performed for that segment.

The comparison is true when each bit of the segment class field ANDed with the inclusion class register yields the inclusion class register and each bit of the segment class field ANDed with the exclusion class register yields a field of zeros. If SCF is a bit in the segment class field, ICR is a bit in the inclusion class register, and ECR is a bit in the exclusion class register, the comparison is true when, for each SCF, ICR, and ECR, the following expression is true:

(SCF AND ICR = ICR) AND (SCF AND ECR = 0)

AN EXAMPLE USING SEGMENT CLASSES

The following example uses an encoding method chosen to illustrate the use of both inclusion and exclusion arrays.

The Scenario

You have a program that draws widgets. Each widget may be one of 45 colors. Widgets may be made of glass, plastic, copper, or tin. Widgets are made by companies A, B, and C in Great Britain; by company L and M in Sri Lanka; and companies V, W, X, and Y in the United States.

Defining the Segment Class Subfields

First, partition the segment class field into subfields to represent the country of origin, the manufacturer, the color, the material, and the product.

You could assign bit positions to the subfields like this:

- Country — Bits 1 and 2
- Manufacturer — Bits 3 through 6
- Color — Bits 7 through 12
- Material — Bits 13 and 14
- Widget — Bit 15

Figure 7-2 shows how, in this example, the first 15 bits of the segment class field are grouped into subfields.

Setting the Segment Class Field

You must set the segment class field for each widget. Assuming that Segment 43 is a blue tin widget made in Sri Lanka by company L we might have the following conditions:

- Sri Lanka is Country Number 3
- Company L is Company Number 4
- Blue is Color Number 12
- Tin is Material Number 1
- Segment 43 is a widget

In this case, the first 15 bits of the segment class field for Segment 43 should look like Figure 7-3.

To set the segment class field to this pattern, use the SET-SEGMENT-CLASS command and send the addition array:

15,14,10,9,4,1,2

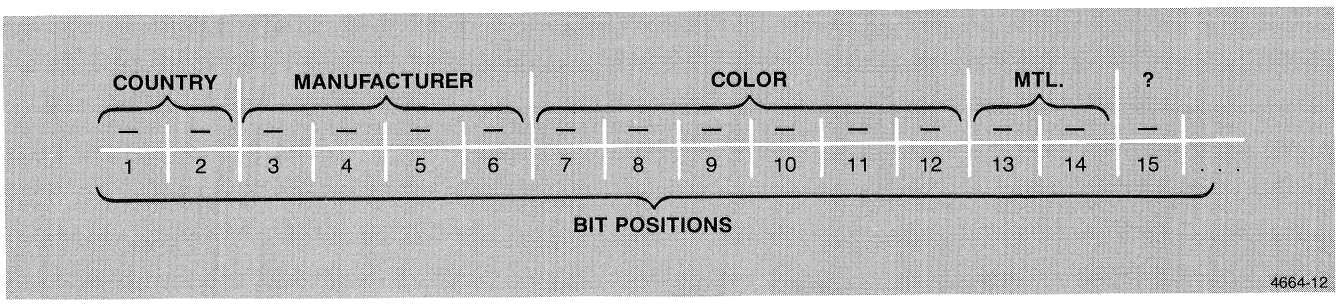


Figure 7-2. The First 15 Bits of the Segment Class Field.

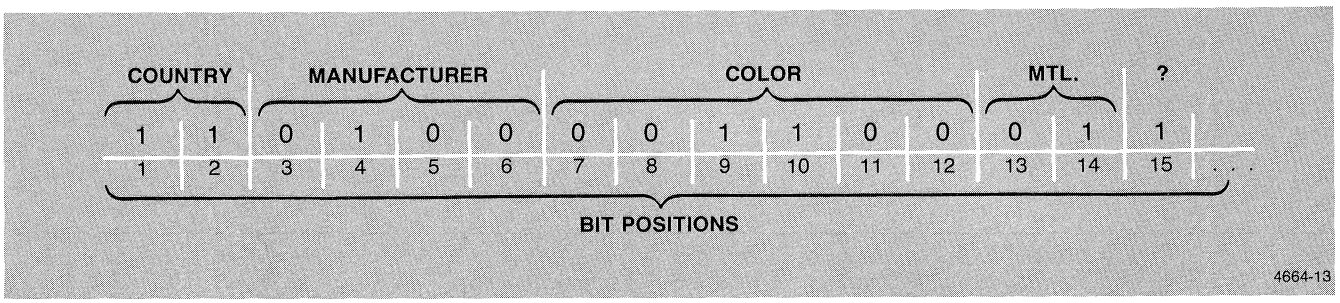


Figure 7-3 . The Segment Class Field for Segment 43.

Using the Current Matching Class

After you have set the segment class fields for several segments, you can manipulate groups of segments using the current matching class and a segment number of −3. If you want to see only the widgets from Sri Lanka, first make all segments invisible:

```
SET-SEGMENT-VISIBILITY −1,0
```

Then make just the widgets from Sri Lanka visible.

```
SET-CURRENT-MATCHING-CLASS (1,2,15)()
SET-SEGMENT-VISIBILITY −3,1
```

In this case you need only specify the bits you want to find as set in the segment class field by putting the bit number in the inclusion array. To check that a bit is 0, put its number in the exclusion array. For example, if you want to show all segments from Sri Lanka that are not widgets, make all the segments invisible, then set the current matching class to exclude Bit 15, the widget subfield:

```
SET-CURRENT-MATCHING-CLASS (1,2)(15)
SET-SEGMENT-VISIBILITY −3,1
```

Remember, the comparison is a bitwise comparison. You could inadvertently include or exclude undesired segments if you are not cautious in allocating your subfields. For example, if Great Britain were country 1 (Bit 2 set) excluding Great Britain would also exclude Sri Lanka, which has both Bits 1 and 2 set.

Section 8

RASTER DISPLAY GRAPHICS

INTRODUCTION

Tektronix raster display graphics terminals combine most DVST terminal capabilities with raster technology. This section discusses raster functions and how to use them. You will find discussions on:

- Raster display hardware.
- Surfaces
- Color and gray scales
- Pixel operations
- Terminal spaces
- Views and view clusters

THE RASTER DISPLAY

INTRODUCTION

Monochrome and color raster display terminals share similar display hardware. The following discussion gives a simplified overview of raster display terminal hardware.

CONCEPTS AND DEFINITIONS

The smallest screen element that a terminal can address is called a *pixel*. The terminal draws each individual pixel as a uniform color. When you look at a raster display, your eyes blend pixels and give the illusion of a continuous form. For example, pixels arranged as in Figure 8-1 give the illusion of a straight diagonal line.

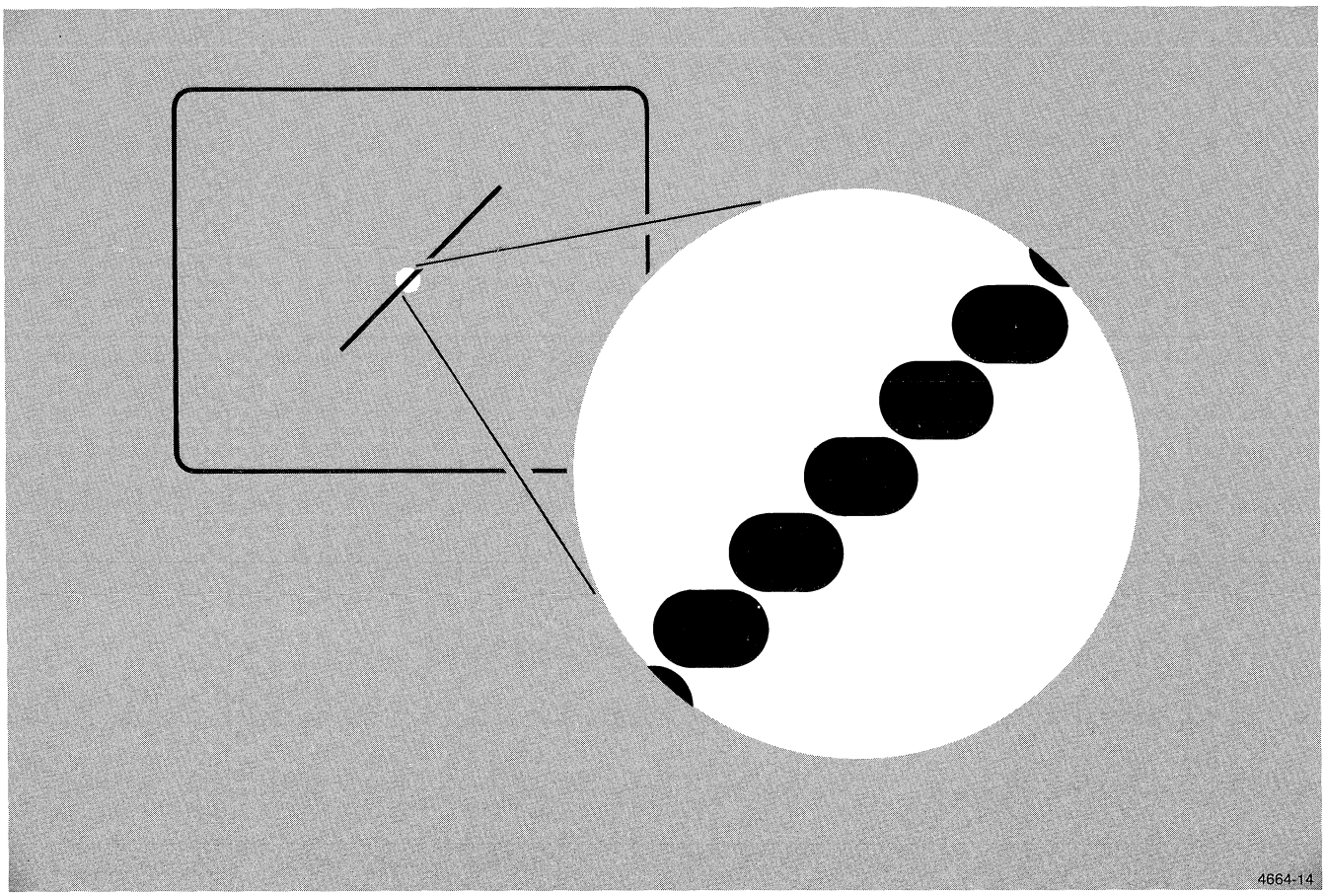


Figure 8-1. Magnified View of Pixels in a Line.

A *raster* display is similar to a television picture. The terminal repeatedly covers the screen with lines drawn by a beam of electrons. The mechanism inside the CRT that generates the electron beam is called an *electron gun.* When the electrons impact the screen they cause the screen to give off light. By changing the intensity of the electron gun, the terminal can cause a spot to glow more, or less, brightly. Figure 8-2 illustrates the electron gun tracing the raster.

A color raster display consists of a special screen and three electron guns. Although we may speak of a single electron beam in a color display, the beam is actually made up of three independent beams that move together. Each gun emits an electron beam that hits one screen color red, green, or blue. By independently controlling the intensity of these three colors, the terminal controls the color and brightness of a single pixel. Your eye blends the three colors in the pixel into a single color.

As the three-part electron beam moves across the screen, it varies in intensity. Each time the beam reaches a new pixel on the screen, the terminal adjusts the intensity of the three electron beams to color that pixel. By refreshing the entire screen 60 times each second, the terminal gives the illusion of a continuously illuminated screen. (In areas of the world using 50 Hz current, the screen is refreshed at 50 Hz.)

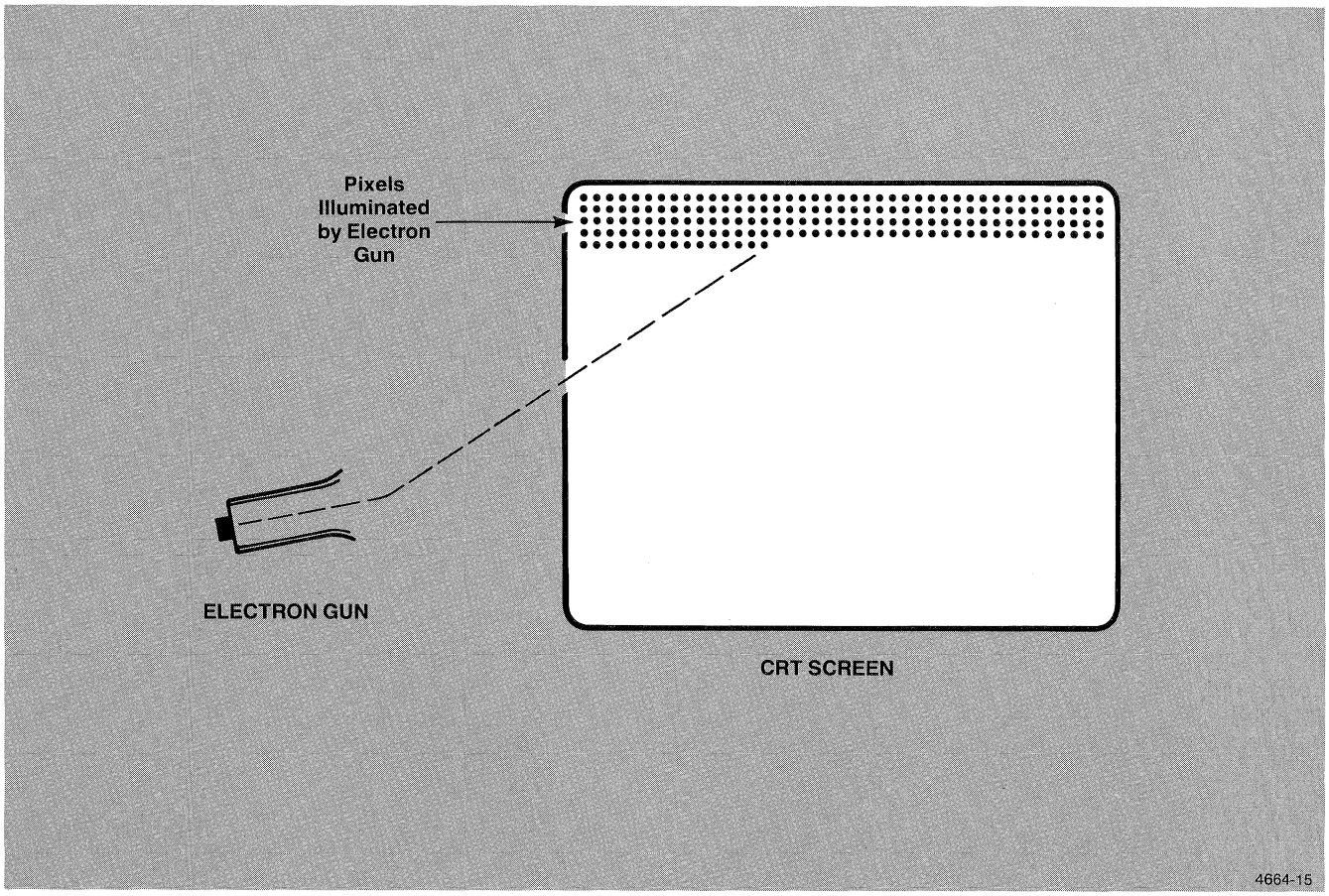


Figure 8-2. An Electron Gun Generating a Raster.

Raster Memory Buffer

Each pixel on the screen is paired with a memory location in a special memory area called the *raster memory buffer*, or *frame buffer*, or simply *raster memory*. As the terminal scans across the raster memory buffer, it reads the number stored in that location and uses the number as an index into a table called the *color map*. The terminal converts the values stored in the color map to gun intensities which, in turn, control the electron guns. The intensity of the electron guns then determine the brightness of the corresponding pixel on the screen. Thus, the screen display is a visual copy of the contents of the raster memory buffer.

You can visualize the raster memory buffer as a three-dimensional array of bits. The height and width of this array correspond to the dimensions of the screen in pixels. The depth of the screen in bits is the number of *bit planes* present in the terminal. The discussion on surfaces, which follows this discussion, shows how you can use groups of bit planes. Figure 8-3 shows how screen pixels correspond to the raster memory array.

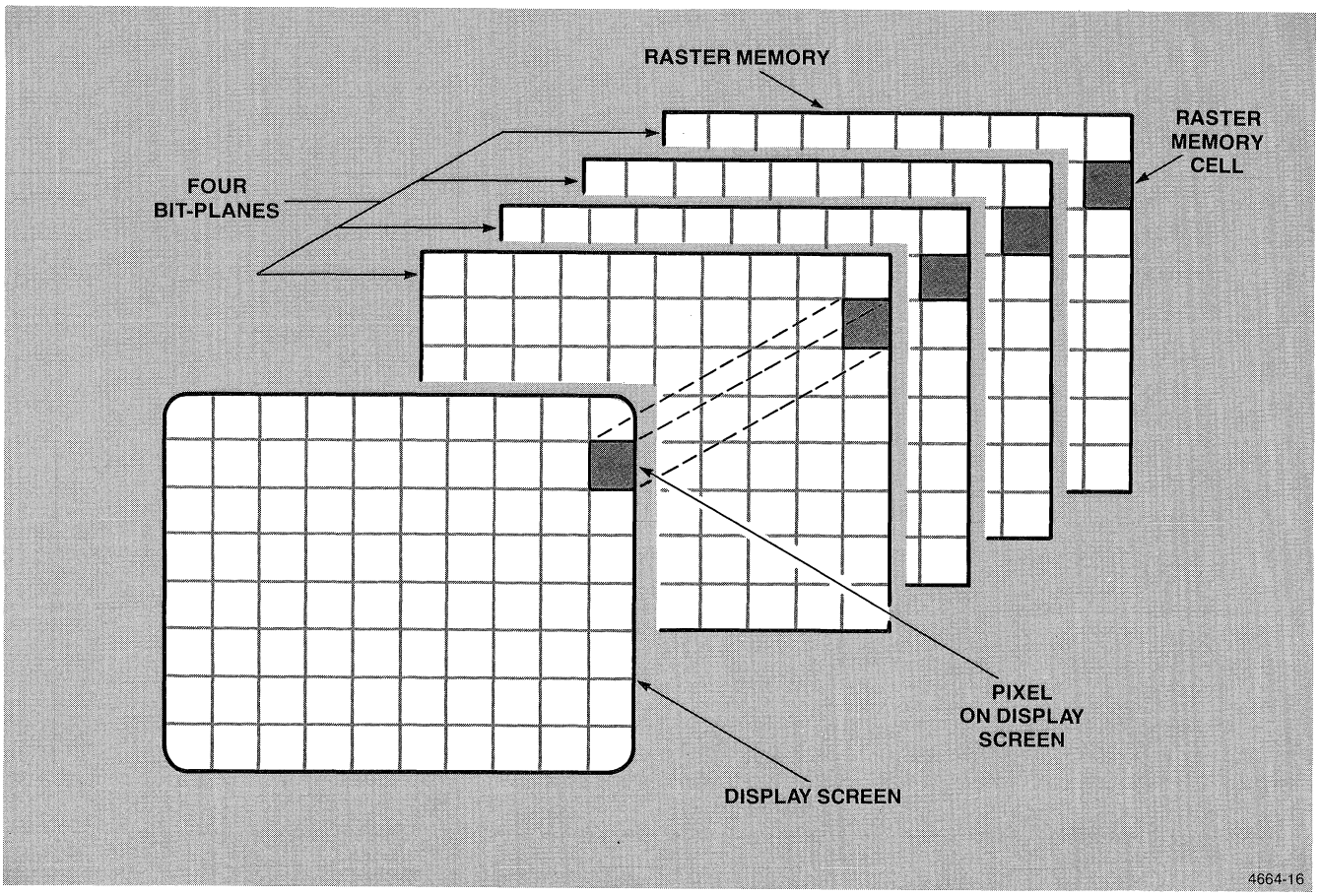


Figure 8-3. Screen Pixels and Raster Memory.

Color Indices

The number stored for each pixel in the raster memory buffer is called a *color index*; an index into the color map. The terminal reads color indices from raster memory, looks in the color map for the entry corresponding to that index, and displays the pixel with the color mix stored in the color map.

You can change the image on the screen by changing the contents of either raster memory (via pixel operations) or the contents of the color map, or both.

Figure 8-4 shows a simplified block diagram of a monochrome raster display system and Figure 8-5 shows a simplified block diagram of a color raster display system.

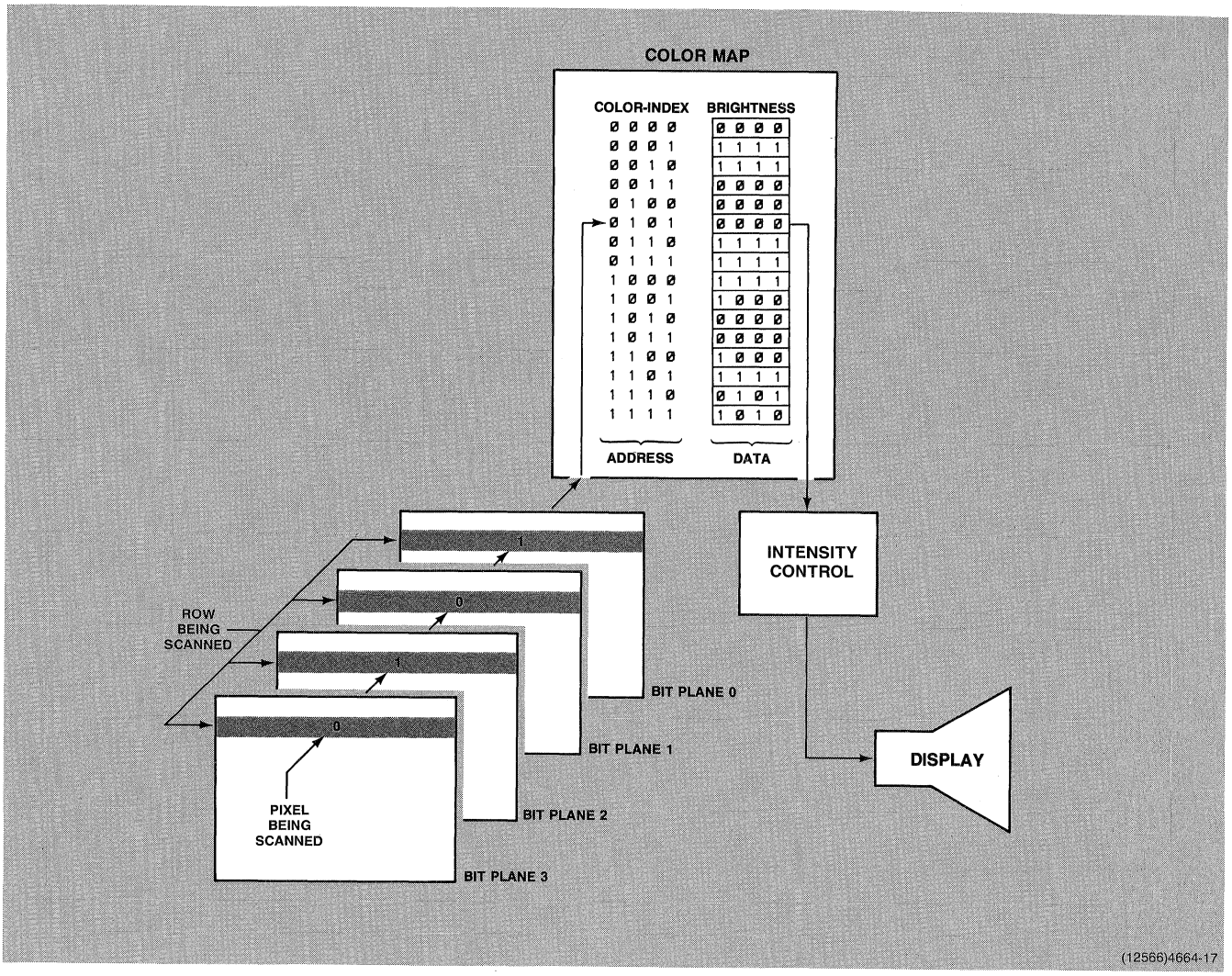


Figure 8-4. Block Diagram of a Monochrome Raster System.

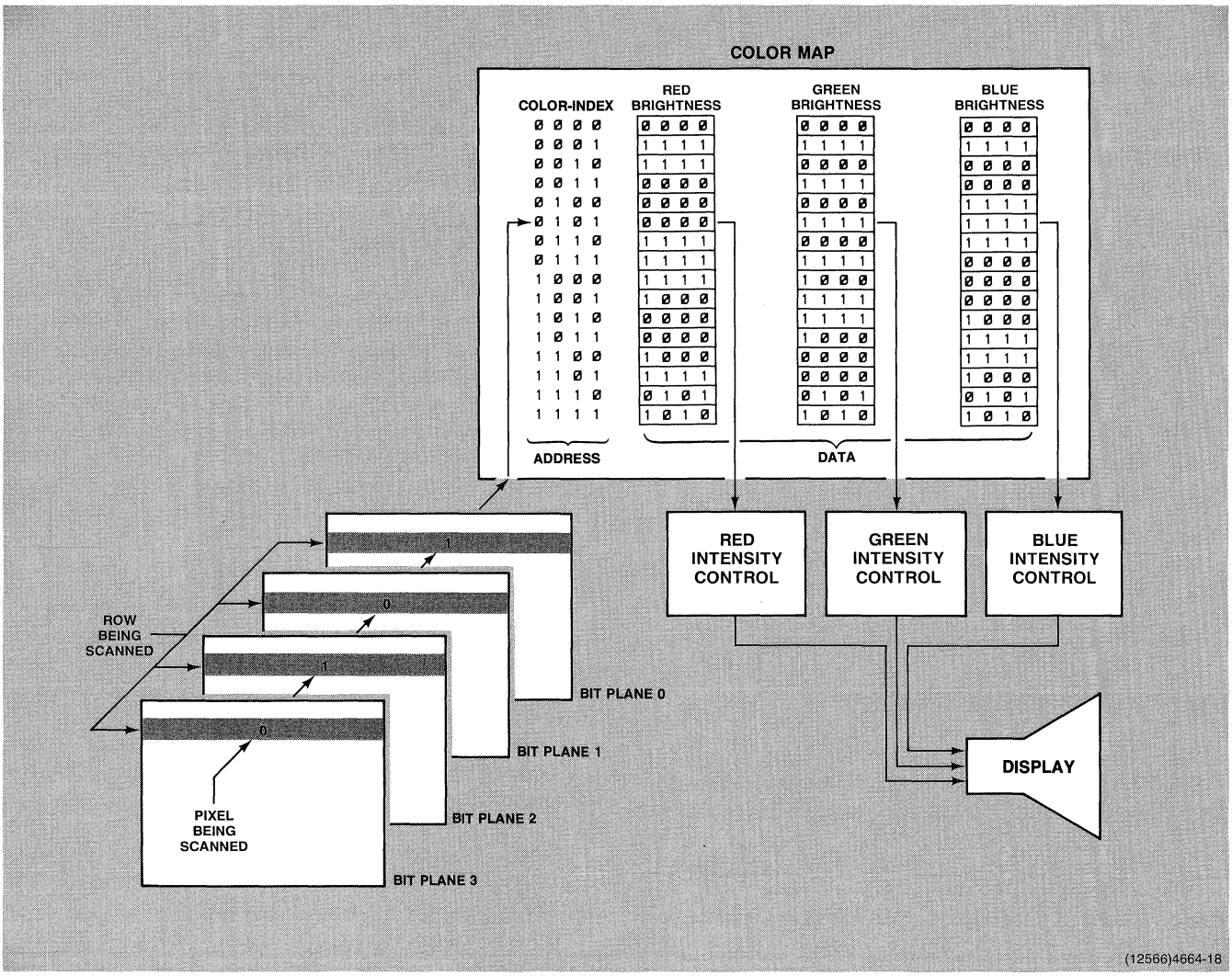


Figure 8-5. Block Diagram of a Color Raster System.

SURFACES

PREVIEW

- 4110 Series raster graphics are drawn on a *surface*, a group of bit planes.
- You can think of a surface as a transparent display screen.
- The display is a composite picture that combines all the visible surfaces over a background.
- Each surface is independent. You can include or exclude it from the composite picture without affecting other surfaces.
- The number of available surfaces depends on the number of bit planes in the terminal. (Each bit plane can be in only one surface at a time.)
- Each additional bit plane allocated to a surface doubles the number of color indices available on the surface.
- You can write to all defined surfaces, by writing to the *super surface.*

CONCEPTS AND DEFINITIONS

Raster memory is a three dimensional bit array, made up of one or more two dimensional bit arrays called *bit planes.* Each bit in a bit plane maps to a single pixel on the display.

A *surface* is a subset of raster memory; it is a group of zero or more contiguous bit planes treated as a unit. A *cell* is the group of bits on a surface that all address the same pixel.

Think of a surface as a transparent sheet the size and shape of your display screen, on which you will draw pictures.

Writing on a Surface

When you write in terminal space, the terminal transforms the information from terminal space into pixel space and stores index numbers into the cells of the surface. (You can also address each cell on a surface directly with *pixel operations*, described later in this section.)

Defining a Surface

The terminal defines one surface containing all available bit planes at power-up. You can define other surfaces by allocating available bit planes to surface numbers with the SET-SURFACE-DEFINITIONS command. You can define multiple surfaces and draw different pictures on each. When you define a surface, you can allocate from zero to the number of bit planes in the terminal to that surface. You cannot use a bit plane on more than one surface.

Displaying Surfaces

You can make surfaces visible or invisible, or make them blink with the SET-SURFACE-VISIBILITY command. A visible surface is displayed on the screen, an invisible surface is totally transparent, and a blinking surface is alternately visible and invisible. Your view of the screen is a composite picture made of all visible surfaces (see Figure 8-6). Making a surface visible or invisible does not alter the contents of it, other surfaces, or the background.

Making a surface visible is similar to adding an overlay to a composite picture. Making a surface invisible is like removing one overlay from the composite picture.

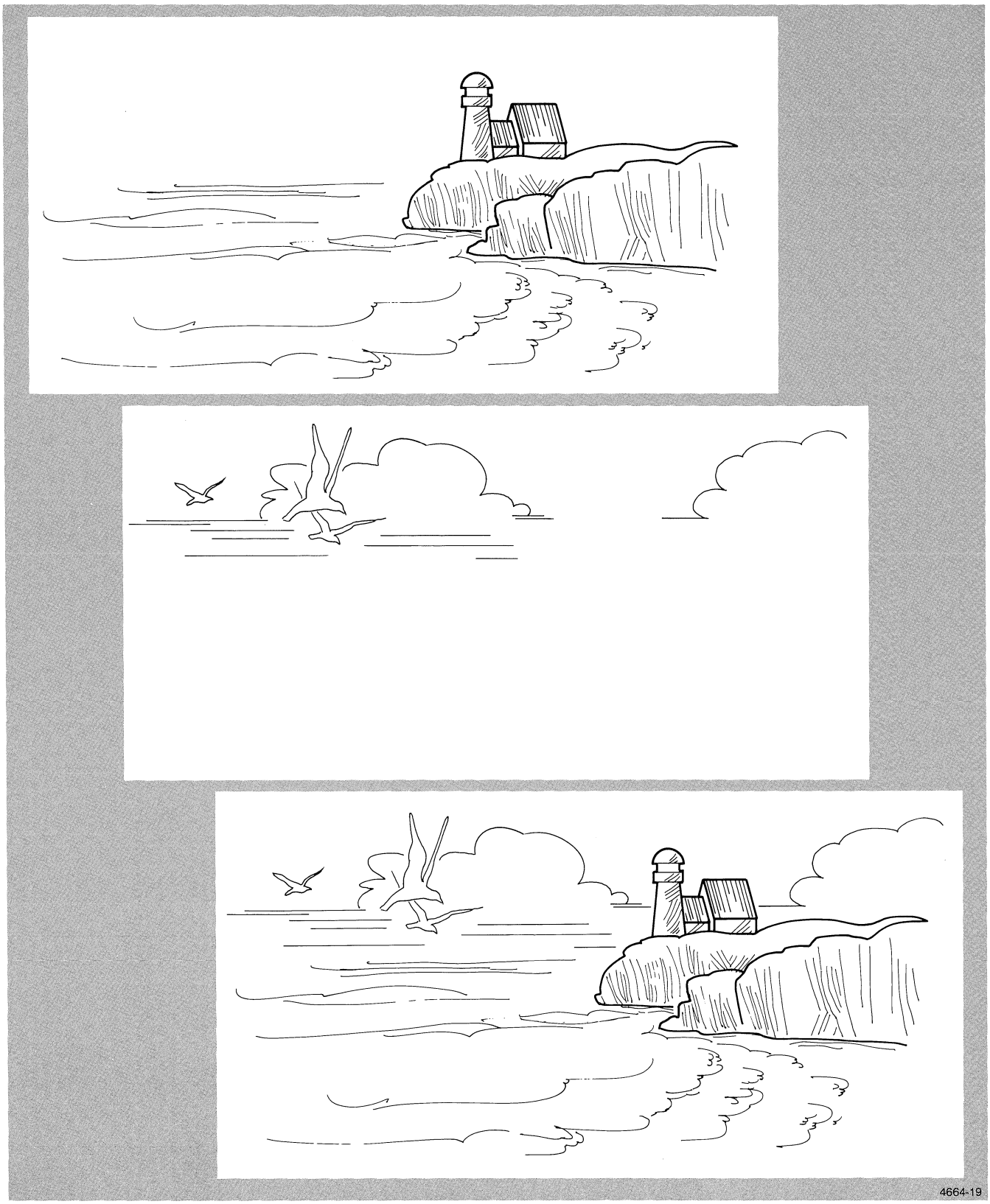


Figure 8-6. Graphics Drawn on Two Surfaces.

Surface Priority

Surface priority determines two things: First, it determines the order in which the terminal displays surfaces and, second, it determines how the terminal displays colors when different colors overlap on different surfaces. This discussion is limited to the order in which the terminal displays surfaces. The discussion on color overlap is in this section under the discussion of color.

By default, the terminal displays visible surfaces as though the lowest numbered surface were nearest the viewer. The lowest numbered surface therefore has the highest surface priority. Surface priority is analogous to the order in which overlays are laid down in a composite picture.

You can redefine surface priority by using the SET-SURFACE-PRIORITY command. Figure 8-7 shows the result of reordering surface priorities.

USING SURFACES

Surfaces are particularly useful for applications such as cartography or circuit board design, where you combine several overlays to form a composite picture. By grouping related graphics information on a surface you can let the operator select individual surfaces or combinations of surfaces appropriate to the task at hand. By allowing the operator to zero-in on the level of detail for a task, you can help the operator achieve greater accuracy and productivity.

As an example, you might have a program that designs floor plans for a building. It would create a display of the floor plan of each floor on separate surfaces. Your program can make these surfaces visible or invisible under program control, or by programmed function keys. The operator can then make a single surface visible to arrange the layout of a single office, or view a combination of surfaces to check the layout of elevator shafts.

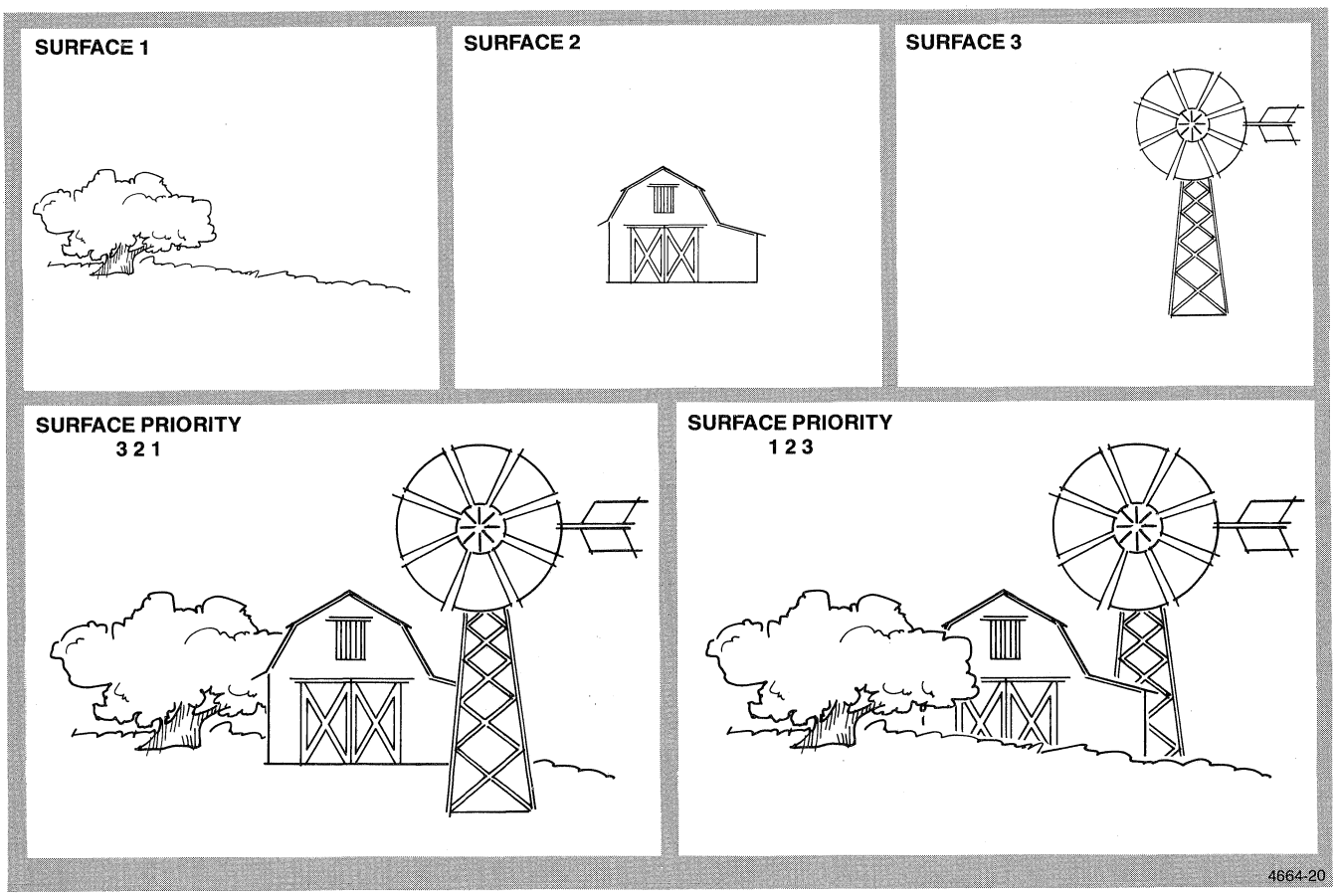


Figure 8-7. Surface Priorities.

You can also use surfaces for *double buffering*, a technique in which you create several independent screens of information on separate surfaces. You can then make these surfaces visible or invisible under program control.

Surfaces and the Dialog Area

In the 4112 and 4113, the dialog area is drawn on one of the terminal's surfaces. The 4115, however, has a separate dialog area surface. This separation of the dialog and graphics areas allows the dialog area to be superimposed on the graphics and scrolled or removed without disturbing the graphics on the other surfaces. Therefore, if you are working with a 4112 or 4113, you should use separate surfaces for graphics and the dialog area if you have enough bit planes.

Number of Surfaces

The number of displayable surfaces possible on a terminal depends upon the number of bit planes in the terminal's raster memory.

Although you can define surfaces with no bit planes assigned to them, you are limited in the number of surfaces that you can define on each terminal. A terminal can define only as many surfaces as it can contain bit planes.

If you define a surface with zero bit planes, you can use it like any other surface. You will not be able to display anything on this surface, however, until you allocate some bit planes to it.

For example, with a 4113 containing 4 bit planes, you can define one to four surfaces with from 0 to 4 bit planes on each. Note that defining a surface with 0 bit planes is not the same as not defining that surface.

The Super Surface

The *super surface* is Surface −1. Some terminal commands allow a surface number of −1 as a parameter. Surface −1 means all bit planes in the terminal are treated as a single surface. When you write to the super surface, cells in *all* surfaces are affected.

For example, consider a 4113 with four bit planes arranged in two surfaces of two bit planes each. A particular screen pixel will map to one cell, two bits deep, on each of these two surfaces. This same screen pixel will map to the same four bits arranged as a cell four bits deep on the super surface. The color index of the super surface is then made up of the color indices of the individual surfaces. If Surface 1 has a color index of 2 (binary 10) and Surface 2 has a color index of 1 (binary 01), the super surface has a color index of 9 (binary 1001). Figure 8-8 shows how the super surface is related to two surfaces in such a case. Appendix D of the *4110 Series Command Reference Manual* contains more information on the super surface.

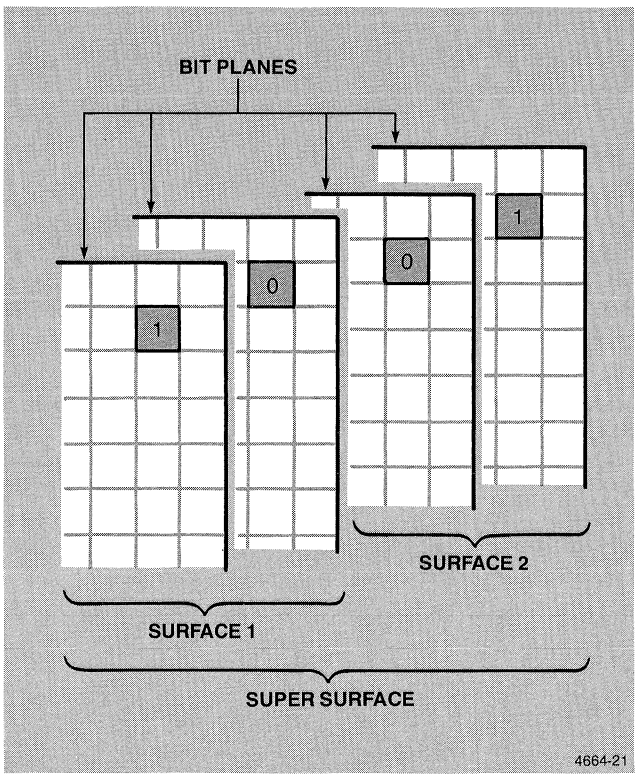


Figure 8-8. The Super Surface.

COLOR

PREVIEW

- Color indices control line, text, and fill pattern color.
- The background color is controlled separately.
- The terminal stores color index numbers in the segment definition.
- The terminal assigns colors to color indices through the color map. You can dynamically change this mapping, thereby changing the appearance of the display.
- The number of bit planes in the raster memory determines the maximum number of color indices the terminal can display.
- The terminal can define colors in three color systems: HLS, RGB, or CMY.
- The 4115 has an additional color system, *Machine RGB*. Machine RGB allows you to define over 16 million different colors.
- You can set color interaction between overlayed surfaces to *Opaque*, *Additive*, or *Subtractive*.

INTRODUCTION

Raster display terminals can display several colors simultaneously. (On monochrome display terminals these colors are different shades of gray.) You can specify the color of a graphics primitive or of dialog text by using color indices in conjunction with a color map.

When you draw a picture, you specify the index for lines, text, and fill patterns by index numbers. Each index is used as a pointer into a color map that the terminal uses to translate an index into a color. Color indices, not the displayable colors, are stored as part of the segment definition. You can remap different colors to the individual index numbers and dynamically alter the appearance of the picture.

CONCEPTS AND DEFINITIONS

Color Indices

Tektronix 4110 Series raster display terminals can display a large number of colors. This potential color range is greater than the number of colors that a terminal can display simultaneously. For maximum compatibility between 4110 Series terminals, and for maximum flexibility in the use of colors, colors are not directly specified. They are, instead, referenced by numbers called *color indices*. These index numbers are attributes of the graphics primitives and stored as part of the segment definition.

Color Map

Each color index is a pointer into the *color map*, a table used by the terminal to control the display. You can change the color map with the SET-SURFACE-COLOR-MAP and the SET-SURFACE-GRAY-LEVELS commands. Figure 8-9 shows how you can control the appearance of a display by changing the color map.

The Erase Index. Index 0 is predefined as the *erase index*. You can think of the erase index as a transparent invisible color. Graphics drawn in index 0 allow the background or graphics on other surfaces to show through unaltered. (The SET-SURFACE-COLOR-MAP command cannot define Index 0, it uses that entry to specify the background color.)

Maximum Number of Indices. The maximum number of unique color indices that can be used on a given surface depends on the number of bit planes assigned to the surface. When n is the number of bit planes assigned to the surface, there are 2^n-1 unique indices including the erase index available. When you use an index number that is larger than 2^n-1, the terminal uses the index 2^n-1. For example, if you specify Index 15 on a surface with 2 bit planes, the terminal will use Index 3.

Selecting Color Coordinate Systems

A color raster display terminal generates colors on the display screen by mixing the three additive primaries: red, green, and blue. However, the terminal allows the user to specify colors in one of three systems: HLS (hue, lightness, saturation), RGB (red, green, blue), or CMY (cyan, magenta, yellow). In addition, the 4115 can specify a greater range of colors with Machine RGB. You choose the color coordinate system with the SET-COLOR-MODE command.

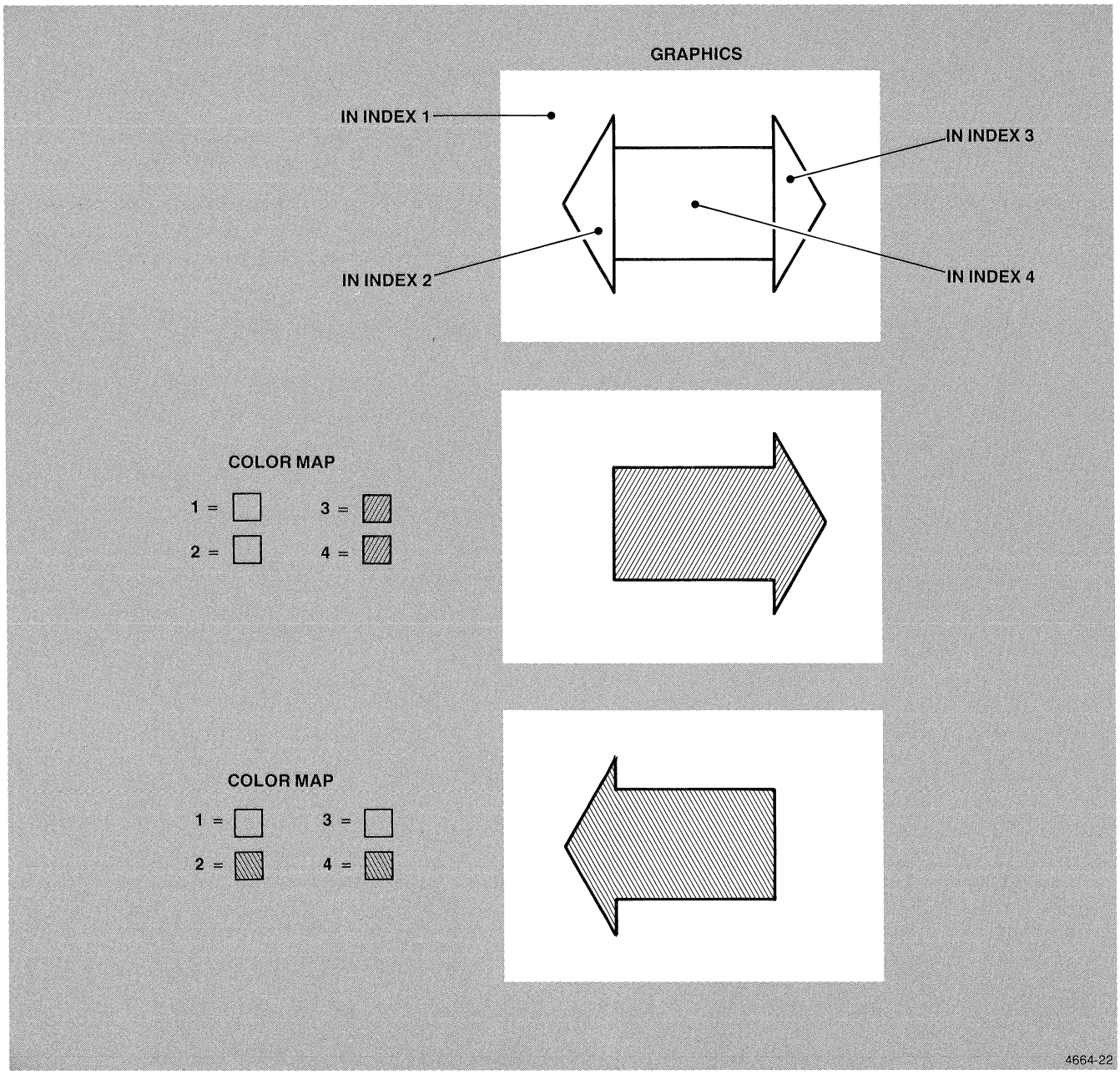


Figure 8-9. The Effect of Changing the Color Map.

The HLS System. You can visualize the HLS system by looking at the color cone in Appendix D. A particular color is a point in the volume of the cone defined by hue, lightness, and saturation. Figure 8-10 illustrates the HLS system color cone.

Hue is the angle formed by rotating a vector around the axis of the double ended cone with blue as the reference. Hue is the basic sensation we think of as color. A hue of 0° (or 360°) corresponds to blue, 120° to red and 240° to green, with intermediate shades corresponding to intermediate rotations. You specify hue as an integer in the range −32768 to +32767 degrees. Integers less than 0 or more than 359 are converted to the range 0 to 359 by a modulo function.

Lightness is the position of a vector along the axis of the cone. Lightness is how bright or dull a color appears; it is how much light is emitted by the color. A lightness of 0% is black and a lightness of 100% is white. (At lightness 0% or 100%, saturation and hue are irrelevant.) You specify lightness by an integer percentage in the range 0 to 100%.

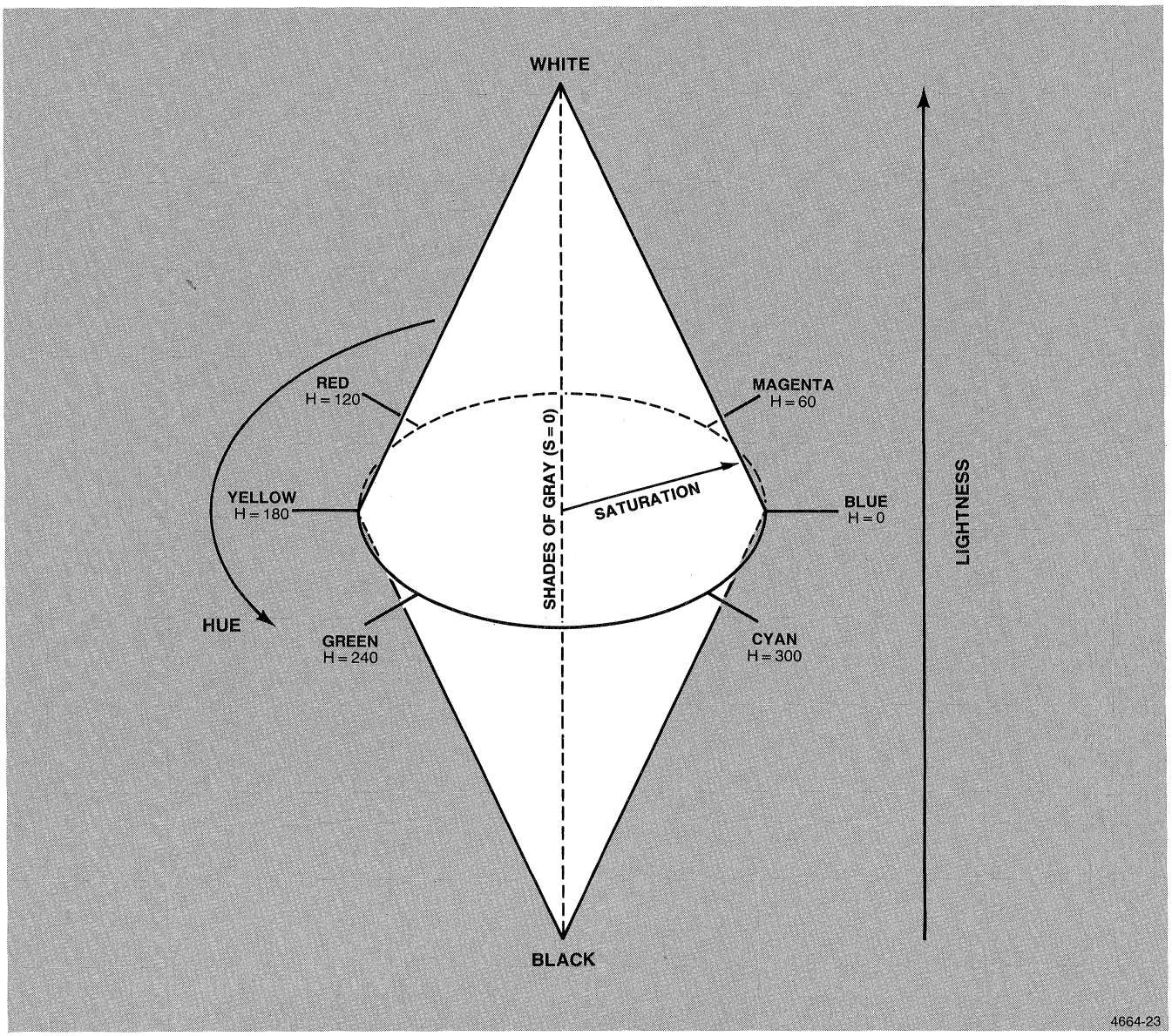


Figure 8-10. The HLS System Color Cone.

Saturation is the radial distance of the vector from the cone axis. Saturation is the intensity of a color. A saturated color is very intense, while a less saturated color is one that appears grayed or muted. A saturation of 0% is simply a shade of gray, while a saturation of 100% gives the most intense possible color having that hue and brightness. You specify saturation as an integer percentage in the range 0 to 100%.

The HLS system is the default system for the 4110 Series color raster display terminals. The HLS system gives a good intuitive "feel" for a programmer or operator when attempting to specify colors. HLS allows you to specify approximately 3,600,000 colors.

The RGB System. The RGB system, also called the *additive color system*, defines colors as mixtures of the three additive color primaries: red, green, and blue. You specify colors in the RGB system as integer percentages from 0 to 100%.

For example, you can define various shades of yellow by mixing equal amounts of red and green with a smaller amount of (or no) blue. You control the overall brightness of the color with the brightness of the primaries you use.

More specifically, a yellow with the proportions (90,90,0) is much brighter than a (30,30,0) yellow, while a (100,100,20) yellow is a still brighter washed-out light yellow.

In additive color mixing, adding equal amounts of the three primaries gives a gray. (50,50,50) is a medium gray, (20,20,20) a dark gray, (100,100,100) is white, and (0,0,0) is black.

You can visualize the color space defined by the RGB system as a cube. The three axes are the three primaries, while the origin is black. Each color is a volume within the RGB color cube. Figure 8-11 shows the RGB color cube.

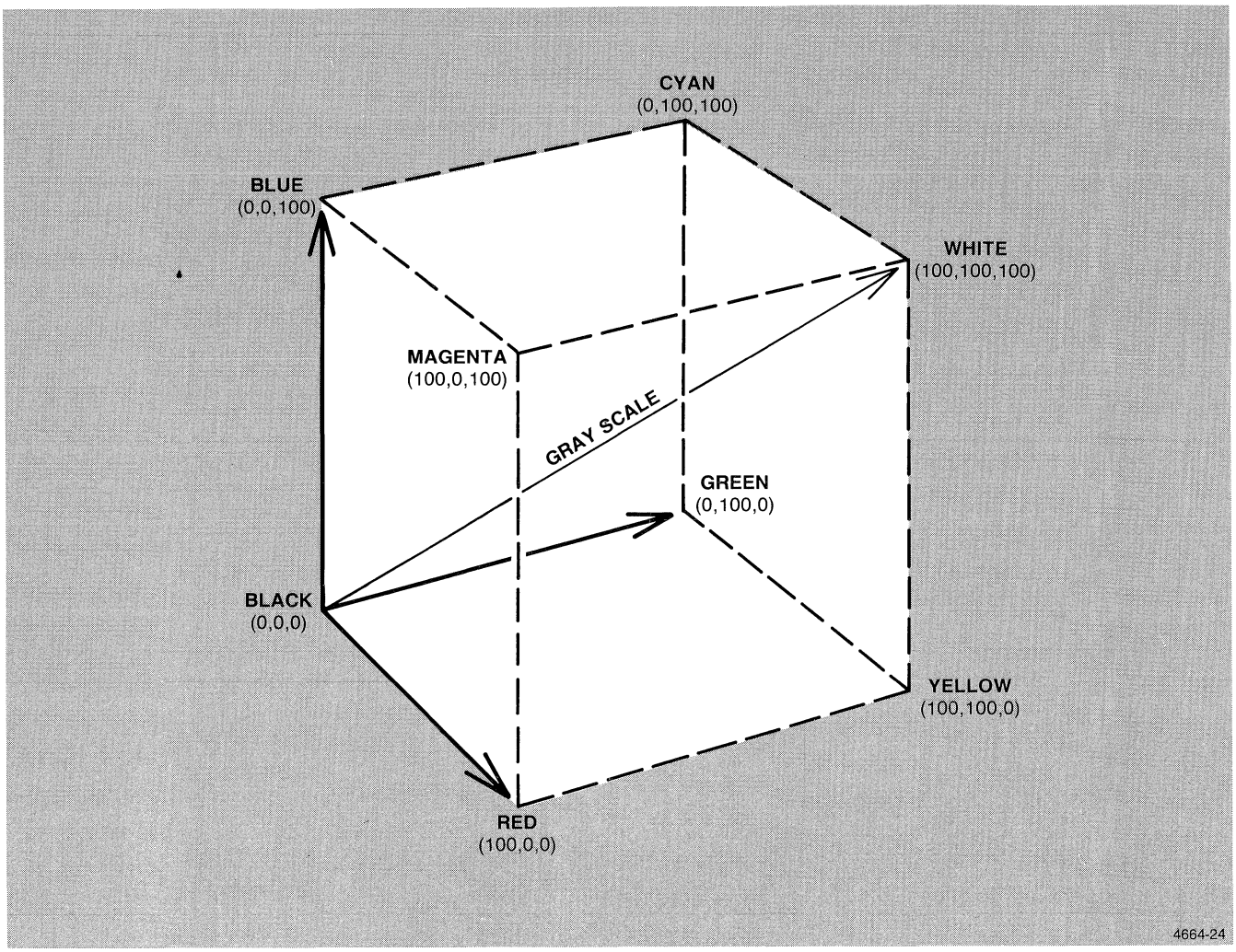


Figure 8-11. The RGB Color Cube.

Machine RGB in the 4115. The 4115 terminal is capable of 256 intensities for each color gun. To specify all the colors that the terminal can display, you can use Machine RGB as the color-specifying mode. Machine RGB allows you to specify 256^3 (over 16 million) different colors.

The CMY System. The CMY system, also called the *subtractive color system*, defines colors as a mixture of the three subtractive primaries: cyan, magenta, and yellow. As in the RGB system, you specify colors with an integral percent (from 0 to 100) of the three primaries.

In the CMY system, adding a greater percentage of a primary reduces the lightness of the displayed color. The gray scale in CMY is the range of equal mixtures from (0,0,0) to (100,100,100) for white to black. Figure 8-12 shows a color cube for the CMY system analogous to the RGB color cube.

Gray Levels. The number of different shades of gray that a terminal can display is limited by the terminal hardware. The 4112, for example, can display 15 distinct levels of gray, while the 4113 can display 16 levels and the 4115 can display 256.

You can cause a color terminal to operate only in gray levels with the third parameter of the SET-COLOR-MODE command. When you are operating only with gray levels, it is more efficient to use the shorter gray-level commands to define color indices and background levels.

Color Interaction on Overlaying Surfaces. When you have colors from different surfaces that overlap on the screen, you can control the color of the area where they overlay. You can choose the color mix of that area to be opaque, additive, or subtractive.

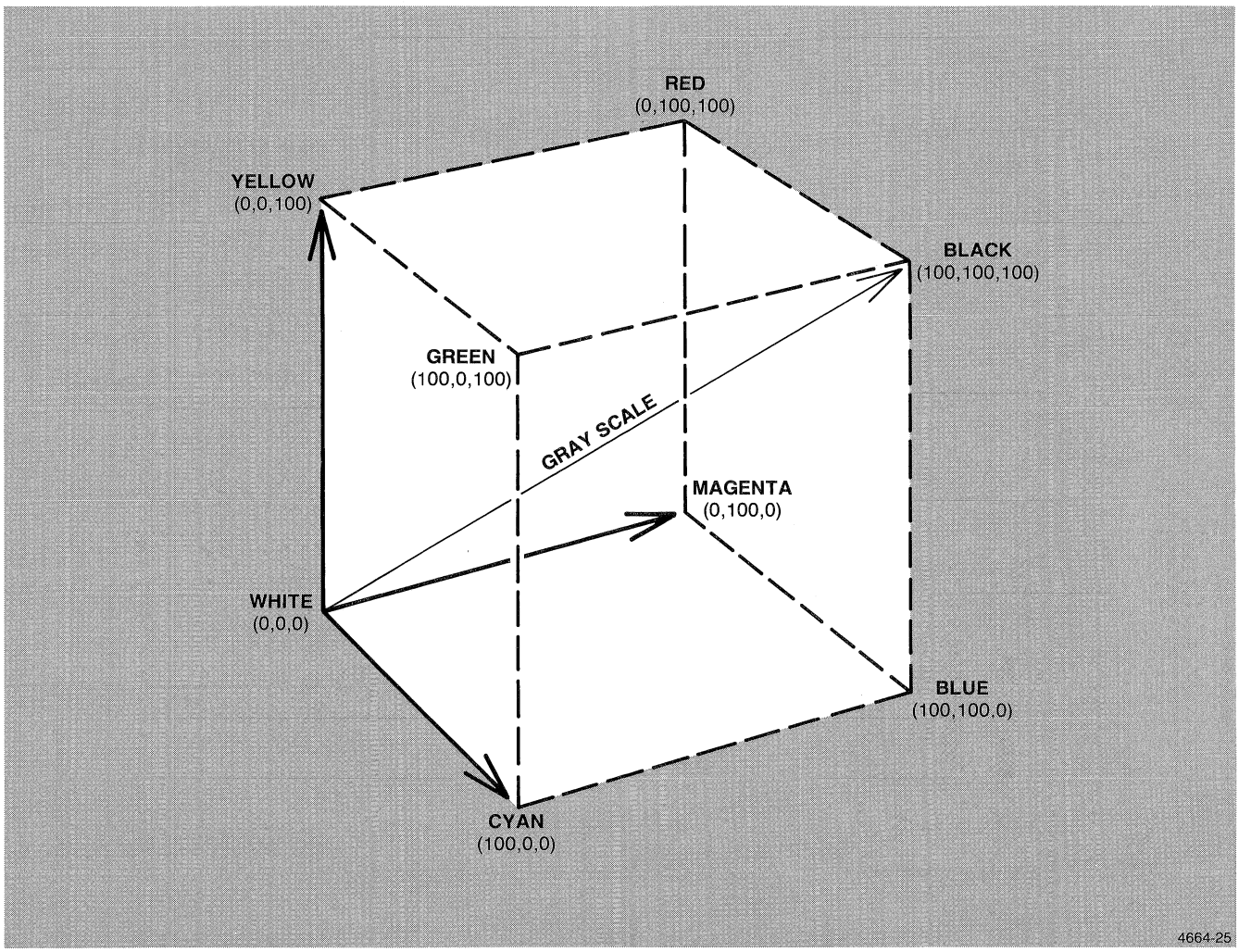


Figure 8-12. The CMY Color Cube.

If you define the color mix between different surfaces as opaque, a colored area on a surface with a higher surface display priority completely replaces any area on a surface with a lower surface display priority that intersects it.

If you define the intersurface color mix as additive, overlayed colored areas from different surfaces will mix by addition, as in the RGB system.

If you define the intersurface color mix as subtractive, the intersection of colored areas from different surfaces will mix by subtraction, as in the CMY system.

When you define colors as either additive or subtractive, the intersection of two overlayed colors may result in a color that is not in the color map because the mixture is performed by the hardware between the color map and the display guns.

For example, Figure 8-13A shows two figures with the intersection shaded. Shape A is red, (100,0,0) in RGB. Shape B is green, (0,100,0) in RGB. Shape A is on Surface 1 with surface display priority 1 and Shape B is on Surface 2 with surface display priority 2.

Figure 8-13B shows that, with opaque color interaction, Shape A obscures a portion of Shape B.

Figure 8-13C shows that, with additive color interaction, the intersection of the shapes becomes yellow.

Figure 8-13D shows that, with subtractive color interaction, the intersection of the shapes becomes black.

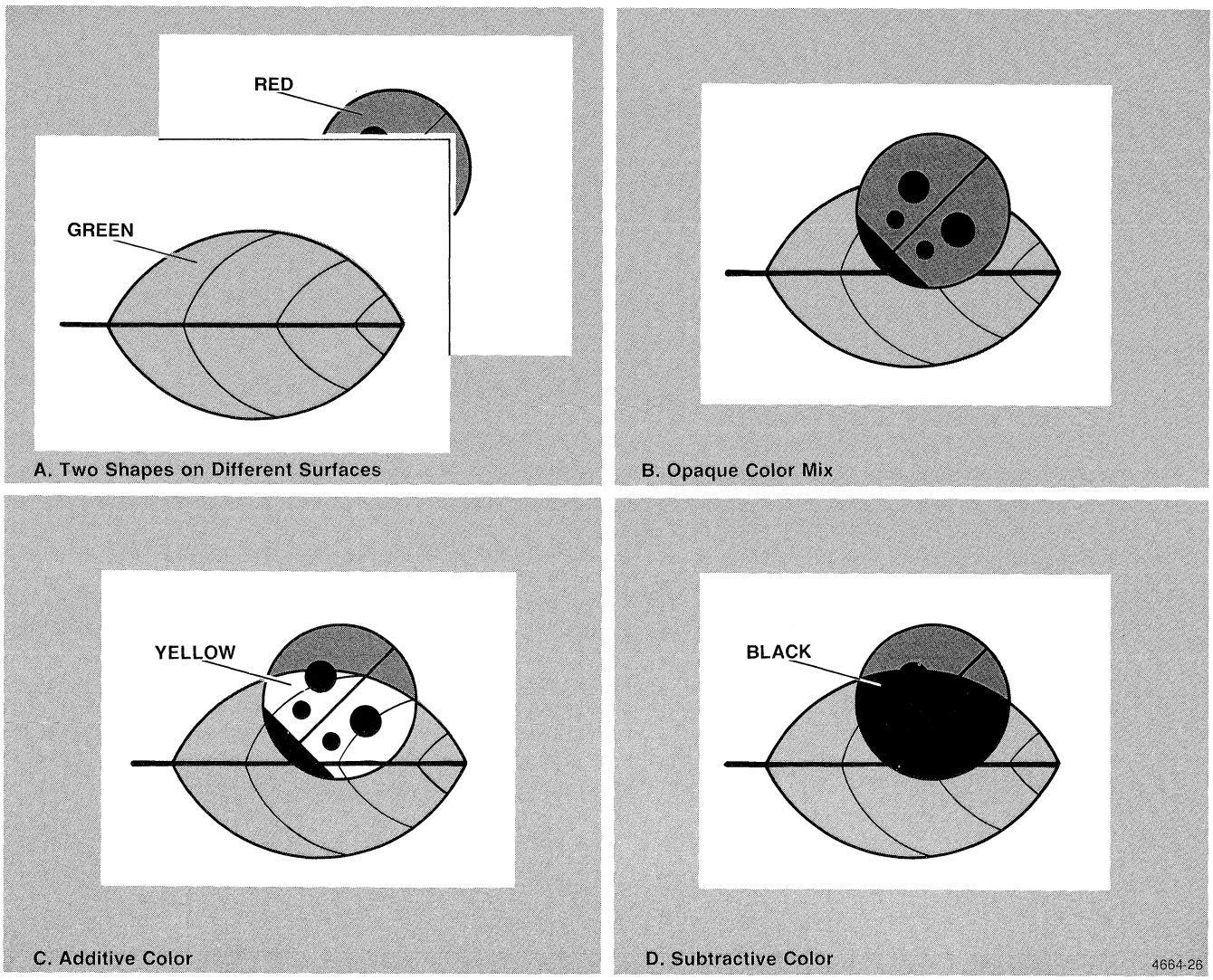


Figure 8-13. Interaction of Colors on Different Surfaces.

Background Colors

Although Index 0 is the erase index, you use it with the SET-SURFACE-COLOR-MAP command to set the background index.

Two other commands also control the background color: SET-BACKGROUND-GRAY-LEVEL and SET-BACKGROUND-COLOR. These commands do not set a color index, you specify them in the current color system parameters.

Background Indices

You can specify the background index for text in the graphics area and the color of line gaps in dotted and dashed lines with the SET-BACKGROUND-INDICES command.

Hint

If you are using a black and white Hard Copy Unit, such as the TEKTRONIX 4632, with a color terminal, you can preview the hard copy in black and white by using the SET-COLOR-MODE command.

PIXELS

PREVIEW

- Pixel operations set the color index of pixels on a raster display terminal.
- Pixel operations are not graphics primitives and *cannot* be included in a segment definition. (The terminal will accept and execute them while a segment is open, but will not save them.)
- Pixel operations are used to define fill patterns for panels.

CONCEPTS AND DEFINITIONS

Pixel Operations

You can control the content of raster memory cells by using pixel operations. Unlike graphics primitives, you work directly with raster memory; you specify which cells are affected and what their contents will be. You can perform pixel operations on a single surface, or on the super surface.

The BEGIN-PIXEL-OPERATIONS command sets the surface number for future pixel operations, the ALU (arithmetic logic unit) mode, and the bits-per-pixel value. The ALU mode controls just how the pixels you write affect the display; you use the bits-per-pixel value with the RASTER-WRITE and RUNLENGTH-WRITE commands.

Pixel Viewport and Pixel Beam Position

RASTER-WRITE and RUNLENGTH-WRITE operate within a *pixel viewport*, which is a rectangular area of the screen addressed in pixel units.

The *pixel beam position* is the position in the pixel viewport at which the RASTER-WRITE and RUNLENGTH-WRITE operations will begin. You control the placement of the pixel beam with the command SET-PIXEL-BEAM-POSITION. The pixel beam position is relative to the pixel viewport. The same coordinates will address a different pixel if the lower-left corner of the pixel viewport is moved.

The RASTER-WRITE Command

When you want to directly enter index numbers into pixels, you can use the RASTER-WRITE command. This command takes two parameters: (1) the number of pixels that you are encoding, and (2) a string of ASCII characters into which you have encoded the index values of the pixels.

When the terminal executes the RASTER-WRITE command, it begins at the current pixel beam position, fills that pixel, advances to the right one pixel, fills that pixel, and repeats until it reaches the the end of the list of indices. When the terminal reaches the right edge of the pixel viewport, it wraps to the left edge of the pixel viewport one pixel down. If the terminal has reached the bottom of the pixel viewport, it wraps back around to the top.

If the special character " ` " (left single quote, or accent grave) is in the ASCII string, the terminal fills the rest of the current pixel line with index 0. The terminal then wraps back around just as if it had normally filled that line.

On the 4115 the command SET-PIXEL-WRITING-FACTORS controls how many actual pixels the terminal writes for each pixel sent, and the direction the pixel beam moves.

Encoding The RASTER-WRITE *character-array*. The encoding for the RASTER-WRITE *character-array* is called *bit packing*. The bit packing used for this command is the same basic algorithm used in Block mode bit packing. The following algorithm shows how to send a complete RASTER-WRITE command. This algorithm supports any positive number of bits-per-pixel, but does not include the use of " ` " (accent grave).

```
Procedure Send-raster-write: (number-of-pixels),(index-array)
    global-reference: (bits-per-pixel)
    send-character: (ESC)
    send-character: (R)
    send-character: (P)
    send-packed-integer: (number-of-pixels)
    (number-of-characters)=integer of
    ((number-of-pixels)*(bits-per-pixel)+5)/6
    send-packed-integer: (number-of-characters)
    (index-pointer)=0
    (register)=0
    (bits-in-register)=0
    until (index-pointer)=(number-of-pixels):
        increment (index-pointer)
        (index)=(index-array(index-pointer))
        shift (register) left (bits-per-pixel)
        increment (register) by (index) modulo (bits-per-pixel)
        increment (bits-in-register) by (bits-per-pixel)
        while (bits-in-register) => 6:
            send-character: ((register) modulo 64)+32
            shift (register) right 6
            decrement (bits-in-register) by 6
    if (bits-in-register) > 0:
        shift (register) left 6-(bits-in-register)
        send-character: (register)+32
```

The RUNLENGTH-WRITE Command

The RUNLENGTH-WRITE command is similar to the RASTER-WRITE command. Use it in preference to the RASTER-WRITE command when many pixels in a row have the same index. When most of the pixels on a line will be set to the same index, the RUNLENGTH-WRITE command reduces the number of characters you need to specify the pixel data. The parameter of the RUNLENGTH-WRITE command is an array of runcodes (single integers). Packed into each runcode are two numbers: the run length and the pixel index. The runcode is an integer computed as follows:

Where N is the number of bits-per-pixel from the BEGIN-PIXEL-OPERATIONS or BEGIN-FILL-PATTERN command, L is the length of the contiguous run of identical pixels, and I is the gray or color index, the runcode R is given by:

$$R = 2^{N} \cdot L + I$$

Encoding a RUNLENGTH-WRITE Command. The following sequence shows how to encode the RUNLENGTH-WRITE runcodes.

```
Procedure Send-runlength-write: (number-of-pixels),
(index-array)
    global-reference: (bits-per-pixel),(terminal-model)
    local-array: (code-array)
    send-character: (ESC)
    send-character: (R)
    send-character: (L)
    (code-count)=0
    (index-pointer)=1
    (multiplier)=2**(bits-per-pixel)
    (index)=(index-array(1))
    (index-count)=1
    (max-index-count)=integer of 65535/(multiplier)
    if (terminal-model)=4115
        (max-index-count)=integer of 2147483647/(multiplier)
    until (index-pointer) = (number-of-pixels)
        increment (index-pointer)
        if (index) <> (index-array(index-pointer))
        or (index-count) = (max-index-count)
            increment (code-count)
            if (index) => (multiplier)
            (index)=(multiplier)-1
            (code-array(code-count))=(multiplier)*
            (index-count)+(index)
            (index)=(index-array(index-pointer))
            (index-count)=1
        else
            increment (index-count)
    send-packed-integer: (code-count)
    for (counter)=1 to (code-count)
        send-packed-integer: (code-array(counter))
```

ALU Modes

The ALU mode parameter determines how the terminal will modify raster memory when it writes color indices into the pixel cells of raster memory. Refer to the *4100 Series Command Reference Manual* for details of the ALU modes.

The RECTANGLE-FILL Command

If you want to fill a rectangular area of the screen with a single color index, you can use the RECTANGLE-FILL command. This command works on the current surface, but need not be within the pixel viewport.

The PIXEL-COPY Command

If you want to copy a rectangular area of raster space onto another rectangular area of raster space, use the command PIXEL-COPY. You can also copy the screen to a disk file to save an image and copy from the disk file to the screen to restore it.

USER-DEFINED FILL PATTERNS

You can define your own fill patterns for panel filling. The fill pattern definition uses the RASTER-WRITE and RUNLENGTH-WRITE commands.

The BEGIN-FILL-PATTERN Command

To open a fill pattern definition, use the command BEGIN-FILL-PATTERN. This command opens a numbered fill pattern, defines the height and width of the fill pattern, and establishes the bits-per-pixel value used by the RASTER-WRITE and RUNLENGTH-WRITE commands.

The number for your fill pattern can be the same as the number of a predefined fill pattern. You can delete your fill pattern by defining it to have 0 height.

The fill pattern is a rectangular array of color indices. The height of the fill pattern can be any value from 0 to the maximum Y-dimension of the screen in pixels. If you specify a height of 0, you will delete the fill pattern definition.

The width of the fill pattern is also defined in pixels, and should be 1, 2, 4, 8, 16, or 32 pixels for the 4112 or 4113 (the 4115 allows a fill pattern to extend across the entire screen.) Other widths in this range will not give a terminal error, but the terminal always generates the fill pattern width as a power of 2. If you give a width that is not a power of 2, the terminal will complete the fill pattern with Index 0 to the next larger power of 2.

The parameter that gives the number of bits per pixel controls the number of color indices you can transmit in the pixel encoding commands RUNLENGTH-WRITE and RASTER-WRITE. The number of bits per pixel determines the number of color indices you can use in your fill pattern just as the number of bit planes on a surface determines the number of color indices you can define. For example, with one bit per pixel, you can only define one color index; with three bits per pixel, you can define seven color indices. Using six bits per pixel is often convenient as this number fits each color index in a RASTER-WRITE command into one ASCII character.

If you want to open a fill pattern definition for Pattern 32 and give it a width of four pixels, a height of four pixels, and six bits per pixel for encoding, use the sequence:

^{E}**cMD** 32,4,4,6

The END-FILL-PATTERN Command

The command END-FILL-PATTERN closes the fill pattern definition. If you have filled the entire fill pattern, you do not need to use this command. If you have not completely filled the pattern, the END-FILL-PATTERN command writes the remaining cells of the fill pattern with Index 0.

An Example of a User-Defined Fill Pattern

As a simple example of a user-defined fill pattern, let us define the pattern shown in Figure 8-14. You can fill a panel with a uniform mixture of two colors if you arrange two indices as shown.

Choosing Pattern number 32 for our fill pattern, and Indices 1 and 2 for colors, the pattern is two pixels high and two pixels wide. Using six bits per pixel to encode the indices, we begin the fill pattern definition.

1. Open the pattern definition with the sequence:

   ^{E}c**MD** 32 2 2 6

2. Encode the pixel indices for the RASTER-WRITE command:

   Index 1 encodes to the ASCII character "1"

   Index 2 encodes to the ASCII character "2"

4. Send four pixel definitions with the RASTER-WRITE command:

   ^{E}c**RP** 4,4,1,2,2,1

5. Ending the fill pattern definition with the END-FILL-PATTERN command is not necessary, since we have filled the entire fill pattern.

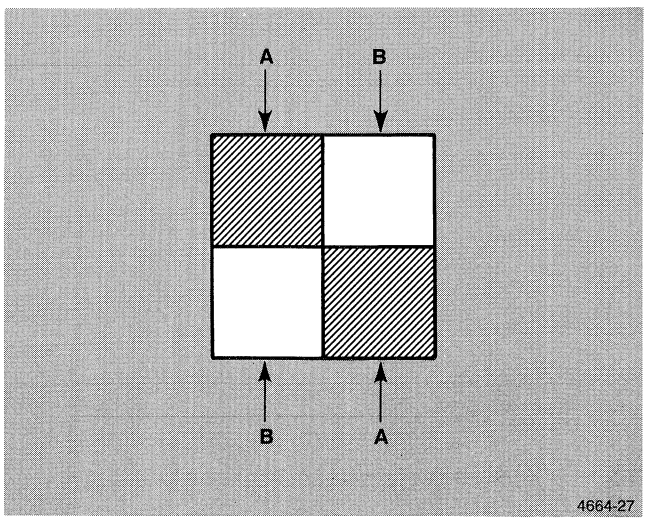


Figure 8-14 . A User-Defined Fill Pattern.

VIEWS

PREVIEW

- Terminals work in many spaces. These include:
  - Terminal space (12-bit space)
  - Extended terminal space (32-bit space)
  - Pixel space (terminal dependent)
  - Normalized screen space (12-bit space)
  - GIN space (12-bit space)

- A view is a collection of attributes, identified by an integer from 1 to 64, that define a transformation from a terminal space window to a viewport.
- Each view has its own set of independent attributes.
- A terminal can remember up to 64 views and can switch from one to another.
- You can address only one view at a time.
- You can manipulate a group of views as a view cluster.
- You can select the circumstances under which the terminal renews the display by setting the fixup level.

CONCEPTS AND DEFINITIONS

Space

A *space* is a coordinate system used by the terminal for a graphics operation. Each space has a coordinate system that bounds the maximum and minimum values that you can pass to the terminal's graphics commands.

Terminal Space. *Terminal space*, also called world or window-coordinate space, is an imaginary plane bounded by the terminal's addressing limits. The concept of terminal space originated with DVST terminals where graphics are drawn directly on the face of the viewing screen. DVST terminals can typically address the display with a resolution of 4096 x 4096 points, thus the X and Y coordinates can range from 0 to 4095. As this is the resolution that can be expressed in a twelve bit binary number, terminal space is also referred to as *12-bit space.*

Extended Terminal Space. The 4115 can address a much larger terminal space than 4096 by 4096; its coordinates can range from -2^{31} to $2^{31} - 1$ on each axis. This *extended terminal space* is also referred to as *32-bit space.*

Pixel Space. *Pixel space* is the addressable space on the display screen. The terms *pixel space*, *raster memory space*, and *raster space* interchangeable. The size of pixel space depends upon the terminal. Table 8-1 summarizes the pixel space on 4110 Series terminals.

Table 8-1

PIXEL SPACE FOR VARIOUS TERMINALS

Terminal	X-Axis Address Range	Y-Axis Address Range
4112	0 — 639	0 — 479
4113	0 — 639	0 — 479
4115	0 — 1279	0 — 1023

Normalized Screen Space. *Normalized screen space* is the coordinate set of terminal space mapped onto pixel space. When you use normalized screen space, you are operating in pixel space, but giving the coordinates in terms of 12-bit terminal space. The terminal performs the conversion from 12-bit to pixel coordinates.

Normalized screen space is 4096 x 4096. Using normalized screen space allows you to address areas on the viewing screen without needing to convert to actual pixel numbers for different terminals.

GIN Space. Terminals perform GIN (graphics input) functions in a 12-bit space similar to terminal space. See Section 9, *GIN*, for more information. The 4115 also performs GIN in 32-bit space.

Views

A *view* is a set of attributes that define a transformation from terminal space to pixel space. The terminal stores the set of attributes that defines the view. This stored information includes:

- *view number*: an integer from 1 to 64
- *window*: a rectangular area in terminal space
- *viewport*: a rectangular area in normalized screen space
- *view surface*: the surface that will store the results of the transform
- *wipe index*: the color index the viewport will be set to when the view is renewed before the window contents are displayed
- *border index*: the color index of the border when it is visible
- *Border visibility*: an attribute that determines whether or not the border is visible.

Window

A *window* is a rectangular area in terminal space that you define with the SET-WINDOW command. A window can cover all or part of terminal space. Graphics primitives, whether or not they are part of segments, that lie partially or wholly within the bounds of the window are mapped into the viewport by the *window-viewport transform.*

The window-viewport transform scales and clips graphics primitives within the window, then writes the result into pixel space. When the terminal displays a graphics primitive that it maps into the viewport, the affected pixels on the view's surface are changed to the primitive's color index.

Your program can zoom and pan on segment graphics by moving the size and location of the window. You can scale or distort the image in the viewport by changing the aspect ratio of the window. However, if you set either the height or the width of the window to zero, the terminal will automatically set the aspect ratio of the window equal to the aspect ratio of the viewport. Figure 8-15 shows several windows in terminal space.

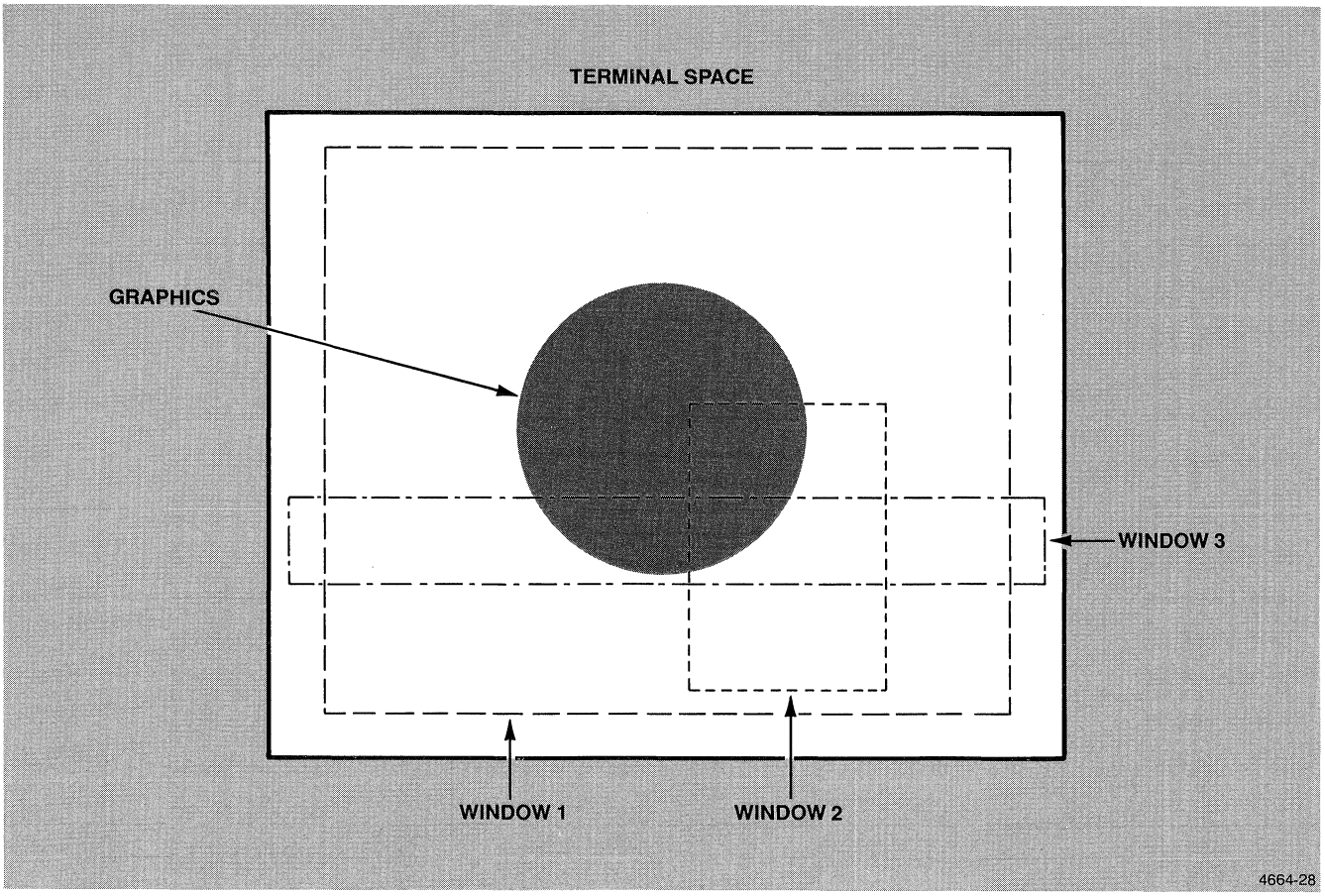


Figure 8-15. Windows in Terminal Space.

Viewport

A *viewport* is a rectangular area of a surface that can occupy either all or part of that surface.

The SET-VIEWPORT command defines a viewport. Although viewports exist in pixel space, you don't give the viewport coordinates in pixel space coordinates, but rather as normalized screen space coordinates. The terminal creates the viewport on the current view's surface.

If you want several views on the screen at one time, allot each one a portion of the screen with the SET-VIEWPORT command.

NOTE

You can define overlapping viewports on the same surface, but when the terminal renews one view, it will destroy graphics on any other view where they overlap.

You can create a view on any defined surface. If a surface is visible, all views on that surface are visible in the composite display.

Figure 8-16 shows two viewports on the screen mapped to different windows in terminal space.

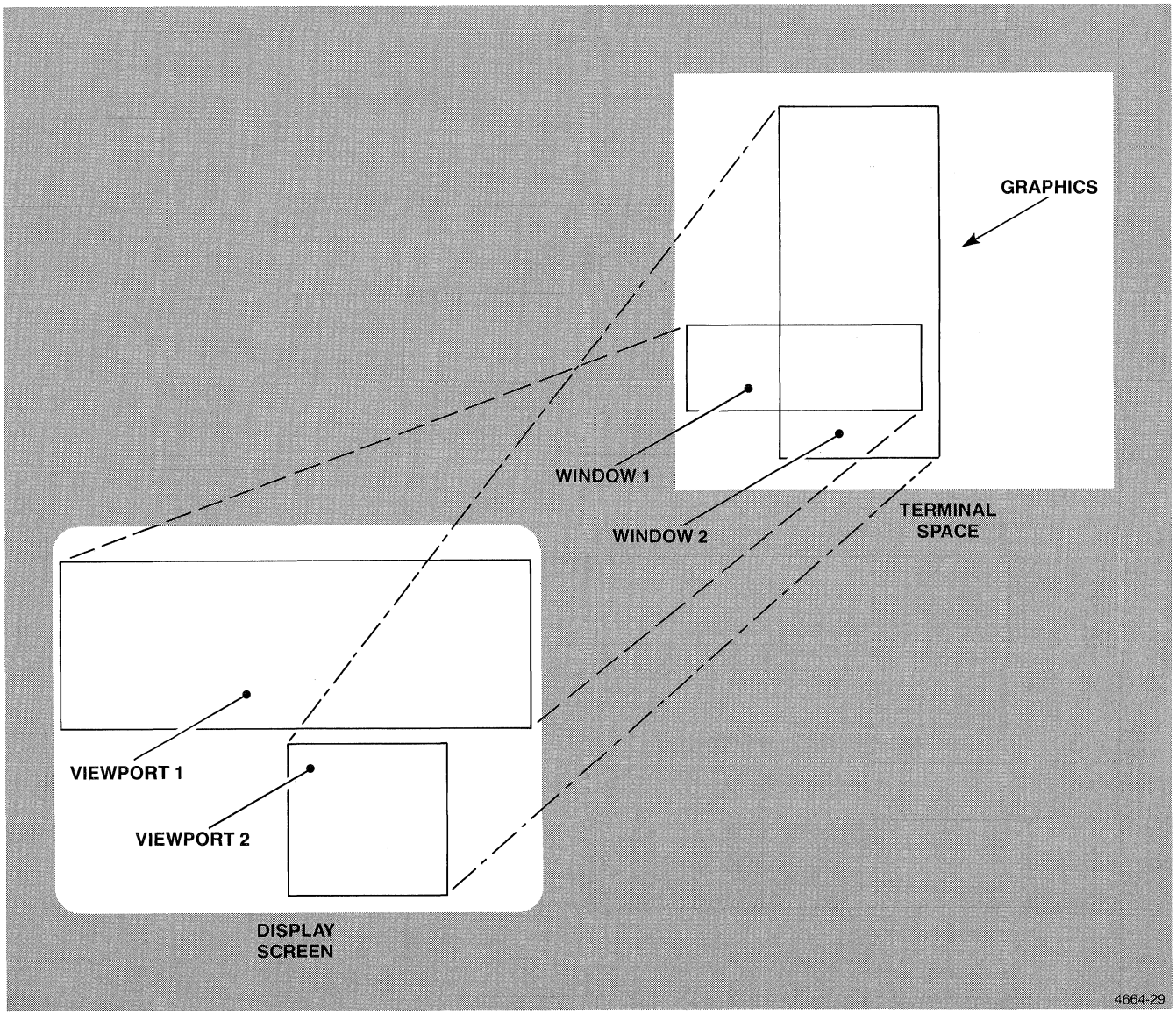


Figure 8-16. Two Viewports Mapped to Two Windows.

CREATING AND SELECTING VIEWS

You create or select views with the SELECT-VIEW command. When you select a view that does not exist, that view is created with all the attributes of the current view. To change the attributes of a new view you must use the SET-WINDOW, SET-VIEWPORT, and SET-VIEW-ATTRIBUTES commands.

Using Views

You can use views wherever you need multiple drawing areas on the screen. For example, you might find views useful for presenting separate perspectives of the same picture, such as different rotations of a mechanical insert or a large overview map and a detailed insert.

The Terminal Viewing Keys

The operator can change some view attributes and select views with the terminal viewing keys. See the terminal's operators manual for details on these keys and how they operate.

You can prevent the operator from changing views or view attributes by issuing the LOCK-VIEWING-KEYS command.

The Overview and Home Position in 4115 Terminals

The 4115 *overview*, the overall display of terminal space, differs from other raster display terminals. The other raster display terminals display normalized screen coordinate space in response to pressing the OVERVIEW key and display all of terminal space in response to pressing the CTRL-OVERVIEW key. The 4115, since it can cover a much greater range of sizes, allows you to set an *overview window* with the SET-OVERVIEW-WINDOW command.

The 4115 overview window is a rectangular area of terminal space just like any other window in terminal space, and is analogous to terminal space in other raster display terminals. When you define an overview window, you also define a *partialview* window. The partialview window is approximately the lower four-fifths of the overview window. It has the same relationship the overview window as normalized screen space has to terminal space in other raster display terminals.

When the 4115 operator presses the CTRL-OVERVIEW key, the terminal displays the overview window in the current screen viewport. When the operator presses the OVERVIEW key, the terminal displays the partialview window in the current viewport.

In addition to modifying the action of the OVERVIEW key, when you change the size of the overview window, you change the coordinates of the cursor HOME position. The 4115 HOME position for the cursor is always the upper left corner of the partialview window.

Fixup Level

The *fixup level* determines whether graphics primitives are displayed as they are received, and whether a segment's image is removed when it is deleted or repositioned. The fixup levels are thresholds, any level exceeding a given threshold will cause action at that fixup level action below to take place.

Hint

If you have a large segment and want to delete it quickly, set the fixup level to 0, delete the segment, restore the fixup level to its original value, and finally give the RENEW-VIEW command.

View Display Cluster

A *view display cluster* is a group of views, which are defined by the SET-VIEW-DISPLAY-CLUSTER command. When you change a window or renew a view on any member of a view display cluster, the operation is performed on all members of the cluster.

To allow the operator to ZOOM and PAN a display that is made up of several overlayed views, you should make every view in the display a member of the view display cluster. Zoom and pan operations then affect the entire cluster and not just a single view on a single surface.

A view can belong to only one view display cluster. It must be explicitly set as a member of a view display cluster by the SET-VIEW-DISPLAY-CLUSTER command. You can assign views that have not yet been created to view display clusters, these views will then be members of the cluster as soon as you create them.

Section 9

GRAPHICS INPUT

INTRODUCTION

In many graphics applications, a terminal operator must enter graphics coordinates for the host program. Graphics Input (GIN) is a means for an operator to quickly enter graphics data without typing in numeric coordinates.

Using GIN from a host program is quite simple. Your program must enable GIN for the device and function you want the operator to use. The operator performs the GIN functions and the terminal returns GIN function reports to the host. The host program then parses the GIN function reports and uses that information.

This section discusses the various types of GIN available on the 4110 series terminals, how to enable them from the host, and how to parse the returned GIN function reports.

PREVIEW

- Gin translates operator action to *xy* coordinates that the terminal transmits to the host in a GIN function report.
- The terminal supports three different GIN devices:
  - The pair of keyboard thumbwheels, which is the standard GIN device.
  - A graphics tablet, which is an optional GIN device.
  - A TEKTRONIX 4662 or 4663 Plotter connected to an optional 3PPI port, which can serve as a GIN device.
- The terminal supports three GIN functions: Locate, Pick, and Stroke.
- You choose the GIN device and GIN function with a *device-function-code* when you enable GIN.
- You can parse all three GIN function reports with the same algorithm.
- You can enable all GIN devices at the same time.
- You can select signature characters to identify the reports from each device and function you use.
- You can provide visible feedback to the operator by selecting inking, rubberbanding, or a user defined cursor as the GIN cursor.
- You can control the possible GIN locations by using gridding.
- You can map areas on GIN devices to windows in terminal space.

CONCEPTS AND DEFINITIONS

A *GIN-Event* is an action that causes the terminal to send a GIN-Report. To cause a GIN event from a terminal, the operator presses a keyboard key; from a tablet, the operator presses the pen to the tablet or a key on the puck; from a plotter, the operator presses a plotter button. You can cause a GIN-event from the host with the REPORT-GIN-POINT command.

A *GIN device* is a physical device that can be used to generate graphics input to the terminal. The standard GIN device is a pair of thumbwheels located on the terminal keyboard. The operator uses the thumbwheels to move the GIN cursor on the display screen and then presses a key to cause a GIN event, which sends a GIN report to the host.

An optional GIN device is a graphics tablet connected to the terminal. While the operator moves a puck or pen around the surface of the graphics tablet, the terminal continuously monitors the position of the puck or pen and updates the graphics cursor on the screen. With Locate and Pick functions, one GIN event occurs each time the operator presses the puck button or presses the pen against the tablet. With the Stroke function, GIN events occur as long as the pen or puck button is held down, at a rate determined by the current time and distance filters.

Although it is not as convenient as a tablet or the thumbwheels, you can use a TEKTRONIX 4662 or 4663 Plotter as a GIN device. Using the plotter's joystick, the operator moves the plotter stylus to the desired position and presses a button on the plotter. The plotter sends a plotter GIN report to the terminal, which the terminal translates into 4110 format and transmits to the host. The terminal's GIN cursor moves only when the plotter sends a GIN report, since it does not track the plotter stylus motion.

GIN FUNCTIONS

4110 Series terminals support three different types of GIN: Locate, Pick, and Stroke. When you enable GIN with the command ENABLE-GIN, you choose (1) a number of GIN-events and (2) a GIN device and a GIN function. You specify the GIN device and function with a device-function-code.

Locate Function

The GIN Locate function is, as the name suggests, used for locating individual points. The operator moves the GIN device, then signals a GIN event. In response, the terminal sends a GIN Locate report to the host that contains the *xy* location of the GIN cursor and the code for the key that was pressed to signal the GIN event. You can use Locate function from any GIN device.

Pick Function

The GIN Pick function is used to select a segment. The terminal sends a GIN Pick report in response to a GIN event. In addition to the *xy* location of the GIN cursor and key code, the GIN Pick report contains the segment number and *pick ID number* of the part of the first detectable segment that is near the graphics cursor.

By adjusting the size of the *pick aperture*, you can control how near the graphics cursor must be to a segment to pick it. The pick aperture is a programmable-size square in normalized screen space centered at the location of the graphics cursor.

Before a segment can be Picked, several conditions must be met.

- The segment must be detectable
- The segment must be visible
- The segment must be in the current view
- A portion of the segment must be inside the pick aperture
- The operator must cause a GIN event

You can use any GIN device with the Pick function.

Stroke Function

The GIN Stroke function sends a stream of GIN Stroke reports to the host as the operator moves the puck or pen over the graphics tablet. You can use the stroke function for such things as tracing hand-drawn graphics for computer entry.

Each movement of the puck or pen can cause a large number of GIN Stroke reports. You can *filter* Stroke function GIN by time, distance, or both to reduce the number of GIN Stroke reports the terminal sends. A *time filter* suppresses GIN Stroke reports until a certain amount of time has elapsed. A *distance filter* suppresses GIN Stroke reports until the pen or puck has moved a minimum distance.

You cannot enable the GIN Stroke function on the terminal thumbwheels or on a plotter, you can use the GIN Stroke function only on a graphics tablet.

GIN Space

Each GIN device maps into a virtual 12-bit (4096 x 4096) GIN space. GIN devices that do not have a square working area map their working area into the lower portion of GIN space. Figure 9-1 shows how a graphics tablet, a plotter, and a terminal's thumbwheels map into GIN space. The position of the GIN device is always defined in terms of GIN space.

GIN location is the point in terminal space that corresponds to the location of the GIN device. It is this *xy* coordinate that is returned by a GIN function report. This report is given in 12-bit format, 10-bit format, or 32-bit format depending on the terminal mode.

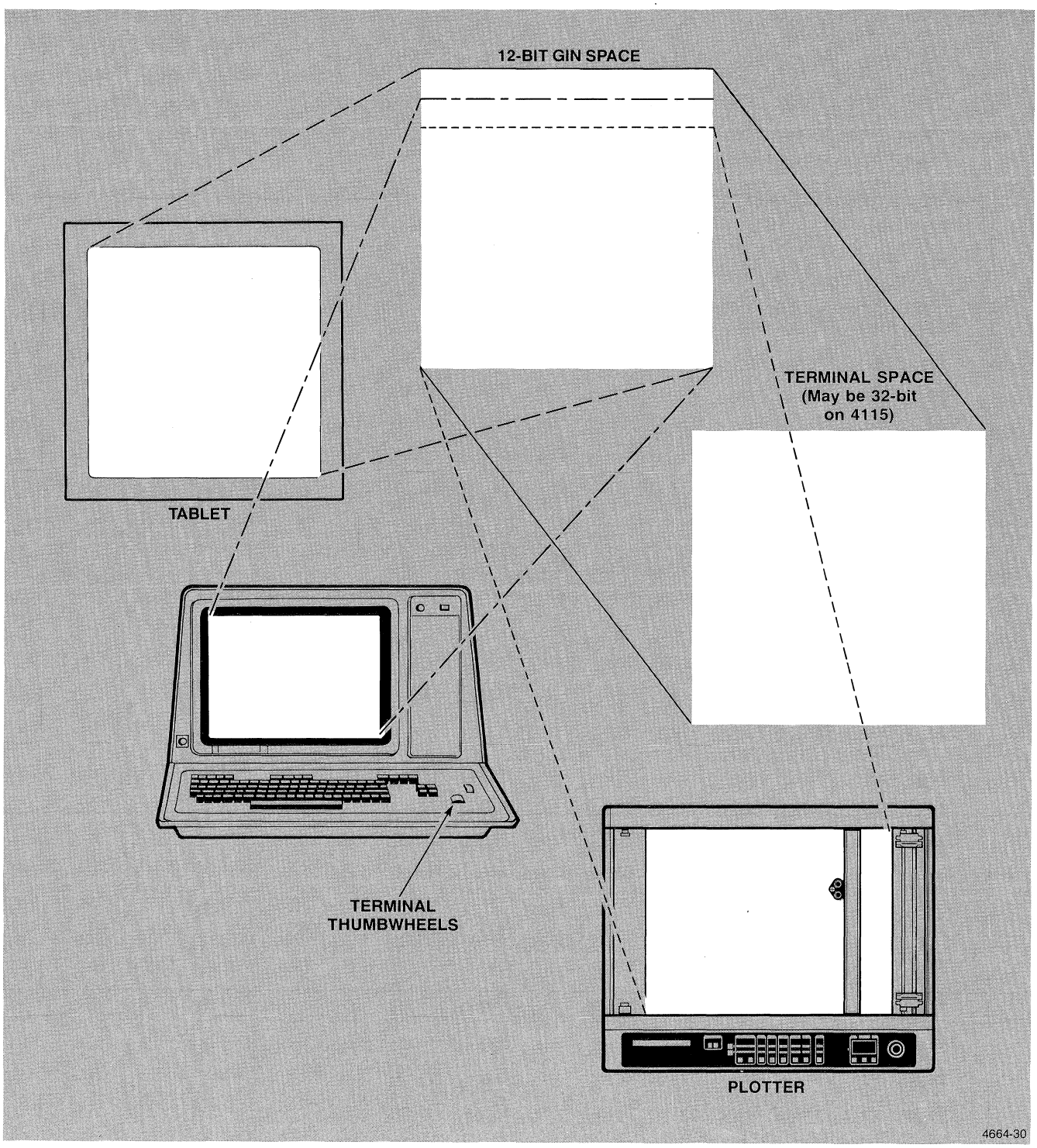


Figure 9-1. GIN Devices Mapped into GIN Space.

GIN Windows and Areas

You can control the mapping of GIN space into terminal space with the two commands SET-GIN-WINDOW and SET-GIN-AREA. The command SET-GIN-WINDOW establishes a rectangular GIN window (similar to a window for a view) in terminal space. The command SET-GIN-AREA establishes a rectangular area (similar to a viewport) in GIN space and links it to a GIN window. The terminal maps points from the GIN area into either the GIN window or the window associated with the viewport specified in the SET-GIN-AREA command. The terminal sends GIN reports to the host in terms of terminal space.

You can define multiple GIN areas on tablets and plotters. The terminal retains GIN areas until you completely cover them with a new GIN area. GIN that is not within a GIN area is mapped into the default 4095 x 4095 GIN window. Figure 9-2 shows how several GIN areas map into windows in terminal space.

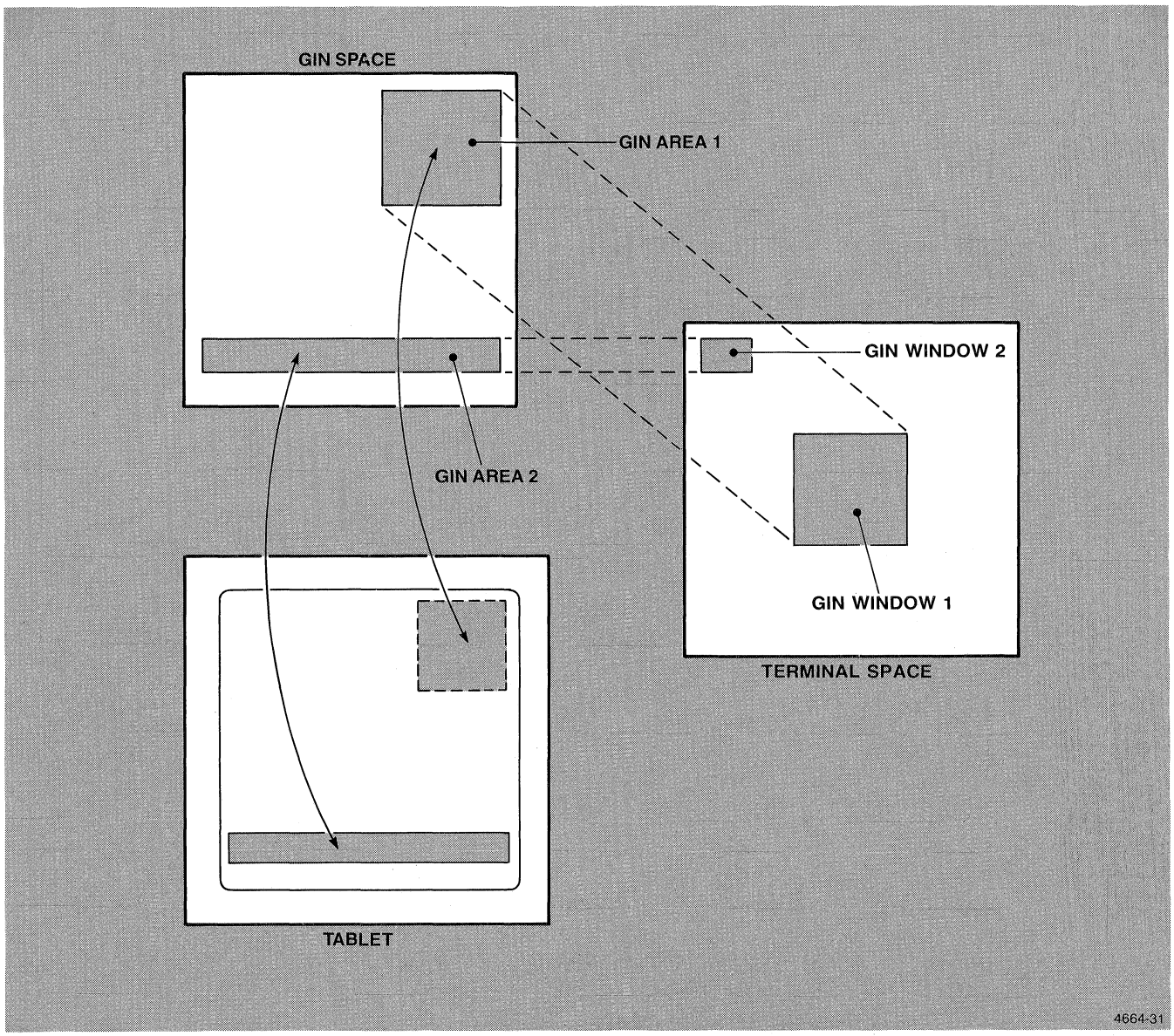


Figure 9-2. GIN Areas Mapped on GIN Windows.

When the operator causes a GIN event, the terminal compares the GIN device location with all active GIN areas for that device and function and translates it into terminal space accordingly. If GIN areas overlap, the terminal uses the most recently defined GIN area for the transformation into terminal space. The terminal then updates the GIN cursor, and sends a GIN function report.

GIN Function Reports

Each GIN event causes the terminal to send a GIN function report to the host. The format of all three GIN function reports is very similar. The GIN-Locate-report and GIN Stroke report have the same format, while the GIN-Pick-report also returns the segment number and Pick ID of the Picked segment. GIN function reports include:

1. The signature-character (if used) for that device and function
2. The key-character (the ASCII code for the key pressed or button pushed)
3. The GIN location as an *xy-report*
4. The segment number that was picked as an *int-report* (GIN-Pick-report only)
5. The pick ID of the picked segment as an *int-report* (GIN-Pick-report only)
6. The EOM indicator (if used)

This sequence is sent for each GIN function report until the number of events for which you enabled GIN has been reached, or the device-function is disabled with a DISABLE-GIN command, CANCEL command, or CANCEL keystroke. The final GIN function report sends a normal GIN report and appends:

7. The terminal-signature-character (if used)
8. The EOM indicator

Parsing GIN Function Reports

You can use the same algorithm for parsing all three GIN function reports. A general-purpose algorithm is:

```
Input-character: (key-char)
Input-xy: (x),(y)
If (function) is (pick)
    Input-int: (segment)
    Input-int: (pickID)
```

(function) may be an input argument to this procedure or could be determined from the signature-character

GIN COMMANDS

The names of most GIN commands suggest their action. The following list of GIN commands includes notes on when and how you might use these commands. See the *4100 Series Command Reference Manual* for the syntax and another discussion of these commands.

Enabling and Disabling GIN

ENABLE-GIN. This command turns on GIN. The parameters of this command (1) enable GIN for a number of GIN-Events and (2) enable a particular device for a single function. While you can enable each device for only one function at a time, you can enable more than one device at one time.

DISABLE-GIN. This command disables GIN. You can disable a specific device and function with a device-function code, or you can disable all GIN with a parameter of −1. When you disable a GIN device, it always sends a terminating-GIN report.

REPORT-GIN-POINT. This command forces the terminal to return a GIN report. If you want the current graphics beam position rather than the current GIN position, you can use a device-function-code of −2. If the device-function is not enabled, the device will send a terminating-GIN report.

Emulating Earlier Terminals. The commands that allow you to emulate a 4010 Series terminal with a 4110 series terminal are:

- ENABLE-4010-GIN
- ENABLE-4953-TABLET-GIN
- DISABLE-4953-TABLET-GIN

Setting GIN Parameters

SET-GIN-AREA. This command establishes a mapping from GIN space to terminal space for a device function. Many such mappings for each device function can be defined at the same time. A mapping is deleted when its GIN area is totally covered by a new GIN area. All mappings are deleted when you set the GIN area to full GIN space.

A GIN area is associated with a GIN window (not associated with any viewport and set by the SET-GIN-WINDOW command), or associated with a window that in turn is associated with a viewport that you specify as a parameter to the SET-GIN-AREA command.

SET-GIN-WINDOW. This command establishes a rectangular area in terminal space as the GIN window. You can use this window with the SET-GIN-AREA command.

SET-GIN-CURSOR. This command selects a segment for use as the graphics cursor for a particular device-function-code. You may want to choose different shaped segments for each device and function you use to help the operator differentiate between the device functions.

SET-GIN-GRIDDING. When you want to restrict the set of possible GIN positions for Locate or Pick functions, you can use GIN-gridding. You specify the spacing of an invisible grid in terminal space. The GIN device location is always translated to the next smaller intersection of the permissible GIN grid positions.

SET-GIN-INKING. You can use inking in Locate or Stroke functions so the operator can see where the GIN device has moved. When the operator causes a GIN event, the terminal draws a line from the location of the last GIN event to the current GIN location. You can start inking after the first point is entered if you enable inking with a parameter of 1, or start inking immediately from the GIN-Display-Start-Point if you enable inking with a parameter of 2.

SET-GIN-RUBBERBANDING. You can use rubberbanding with the Locate function to draw an elastic line between the current cursor position and the location of the GIN-Display-Start-Point or last GIN event. As with inking, you can control rubberbanding with parameters to the command. The terminal draws the elastic line in the current line index and style. Raster display terminals draw the elastic line in XOR mode while DVST terminals draw it in refresh mode.

SET-GIN-DISPLAY-START-POINT. Use this command to set an initial point for GIN inking or GIN rubberbanding.

SET-GIN-STROKE-FILTERING. When you are using Stroke function with a graphics tablet, you might want to restrict the volume of GIN Stroke reports that the terminal generates. You can specify (1) the minimum distance the tablet device moves before sending a report with a distance filter and (2) the minimum time between reports with a time filter. The two filters act as thresholds and must both be exceeded before the terminal will send a GIN Stroke report.

SET-PICK-APERTURE. This command sets the size of the aperture, or acceptance area around the graphics cursor. The pick aperture is a square in normalized screen space centered on the GIN location. To pick a segment, the pick aperture must cover at least a portion of the segment you want to pick.

Hint. You can make the pick aperture visible to the operator by defining a GIN cursor segment as a box the same size as the pick aperture, although the segment image can change size if the terminal zooms. If a pick aperture of approximately 40 units on a side is usable, you can use a marker (box shape), as the graphics cursor. Markers do not change size when zoomed.

HINTS AND EXAMPLES

Picking and Dragging

You can let an operator choose a shape from a menu and move it to a desired location by following this sequence:

1. Prepare a menu of segment shapes for the operator to select. Be sure that the segments are detectable when you define them.
2. Give the command ENABLE-GIN for one pick function from the thumbwheels. Instruct the operator to move the graphics cursor to point to the selection and press a key.
4. When the operator causes the GIN event, make the selected segment the graphics cursor. Instruct the operator to move the shape to the desired location.
5. The operator can now move the shape with the thumbwheels. When the shape is in position and the operator causes a GIN event, leave the segment there. Your program then could copy the segment and return the shape to its position in the menu.

Hint. Don't set the pick aperture too small. If the aperture is too small, the operator will find it almost impossible to put the aperture on a a segment.

Hint. Keep segments fairly simple if you plan to pick and drag them. If they get too complex, segments take too long to redraw.

Section 10

THE TERMINAL FILE SYSTEM

INTRODUCTION

4110 Series terminals use a file transfer system to move data between various ports and devices connected to the terminal. This section discusses the terminal file system and how to use it from the host program. This section discusses the following:

- The terminal file system and the devices and commands available on a standard terminal.
- Local disk storage (Options 42, 43, and 45).
- Option 10, the three port peripheral interface (3PPI).
- The color copier interface (Option 09).
- The Direct Memory Access interface (Option 03)

THE TERMINAL FILE SYSTEM

PREVIEW

- The terminal file system transfers files between devices.
- Devices are identified by a device name in the format *XX:*, followed by a parameter.
- Device name parameters can be an empty string, a file name, a string, or an integer, depending on the device.
- File name parameters in the format *FILENAME.EXT* identify files on file-structured devices.
- The full identification of a file on a file-structured device is its file name appended to the device in the format *XX:FILENAME.EXT*.
- The standard terminal device is *HO:*, the host communication port.
- The standard terminal file commands are:
  - COPY
  - SPOOL
  - STOP-SPOOLING
  - SAVE
  - LOAD
  - REPORT-DEVICE-STATUS

CONCEPTS AND DEFINITIONS

The terminal file system transfers files between the host computer, the terminal, and options attached to the terminal. These files may contain terminal commands, alphanumeric data, or binary images.

Devices and Device Names

Devices. A *device* (identified by a device name and parameter) is a source or destination for files.

Device Names. A *device name* is a three character sequence, two alphanumeric characters followed by a colon, which uniquely identifies a device. For example:

- H0: identifies the host port
- F0: identifies disk drive 0
- P1: identifies port one on the three port peripheral interface.

If a device name is not given with a file name parameter, the default device F0: is assumed. Thus, MYFILE.DAT identifies a file on device F0: and is equivalent to F0:MYFILE.DAT.

Device Parameters. A *device parameter* is a string or integer following the device name. The permissible device parameters depend on the particular device in use. In general, file-structured devices take a file name as a parameter, while format-dependent devices take an integer as a parameter.

Types of Devices

Physical Devices. A *physical device* has a tangible mechanism associated with it. An example of a physical device is a disk drive.

Physical devices can be port devices such as H0:, or file-structured devices such as F0:. Only file-structured devices use file names; for example, you could use F0:AB.DAT, but not H0:AB.DAT.

Pseudo Devices. A *pseudo device* has no tangible mechanism associated with it. For example, in a terminal equipped with the DMA interface, the pseudo device SG: is the segment list. Some pseudo devices take a parameter after the device name to specify a particular data type. For example, the pseudo device DM:, which is the DMA interface, allows a parameter of 0 or 1 to indicate the type of data transfer.

Source and Destination Devices. Some devices originate data only, some receive data only, and some transfer data in either direction. You should know whether a particular device can function as a source or destination or both.

File Names

On file-structured devices, each file is identified by a unique file name. A file name is a sequence of up to eight characters followed by an optional extension. The extension, if present, consists of a period (.) followed by up to three characters. Characters allowed in file names are printable ASCII characters from ! to ~, excluding: ^{S}P, *, ., :, ;, <, =, >, ?, [,], and _.

File names appear in the format FILENAME.EXT. When you want to specify a file on a particular device, append the file name to the device name. For example, to specify a file called MYFILE.SEG on Drive 1, you would use F1:MYFILE.SEG.

General File Transfer Operations

The terminal begins a file transfer operation by opening the source and destination devices. The terminal reads the source device into an internal file block, then writes the contents of that block to the destination device, and repeats the process until it encounters an end-of-file mark. The terminal then closes both the source and destination devices.

The terminal operator can abort any file transfer operation (except a SPOOL) by pressing the CANCEL key.

Your program cannot abort a file transfer (except a SPOOL) because all data from the host is placed in the input queue and examined only after the transfer is complete. You can abort a SPOOL with the STOP-SPOOLING command.

DEVICES AND COMMANDS IN THE STANDARD TERMINAL

Standard Terminal Devices

The only device on a standard terminal is HO:, the host communication port. HO: is a physical device that can both send and receive data. HO: is a port structured device and takes no parameters during file transfers. The communications parameters that control HO: are discussed in Section 3, *Communications*.

Standard Terminal Commands

The following commands are normally used with terminal options. These commands are available to the operator in Setup mode, or to the host programmer as escape sequences.

COPY. Use this command to transfer a file from one device to another. You can copy an entire disk to another by specifying only the device names such as COPY F0: TO F1:.

SPOOL. Use this command to move a file from one device to another without visibly affecting the operation of the terminal. The data transfer is carried on in the background of normal terminal operation. You can only do one spool operation at a time.

STOP-SPOOLING. Abort a spooling operation with this command.

SAVE. Use this command to save data from the terminal's memory to a device. You can save each of the following:

- One or more macro definitions
- One or more segments
- Pixels from the current pixel viewport (raster display terminals only)

LOAD. This command transfers a file from a device to the terminal command processor, so that all commands in the file are executed as if they had come from the host. Although you can load a file from HO:, it has no advantages, and since keystrokes are queued until the operation is over, it has the disadvantage of preventing operator interaction.

REPORT-DEVICE-STATUS. Use this command to force the terminal to send a device-status report. You can use the device-status report to determine whether the terminal recognizes a device.

HINTS

When you transfer files to and from the host computer, several problems can arise:

- Problem — The data contains the bypass control character in the data. Since the terminal sends files to the host using the report system, if the host echoes the bypass control character, it will bring the terminal out of Bypass mode and enter the file characters in the terminal input queue. This can lead to several potentially disastrous situations.

  If the host and terminal are using flagging, the terminal will flag the host when the input queue is full. The terminal then waits to complete the file transfer, and the host waits for a flag to resume sending. In addition, the terminal in Prompt mode has the transmission gate closed, and it cannot open it until it gets a prompt from the host.

- Problem — The lines the terminal sends are too long for the host's input buffer.

- Problem — The host can send and receive only 7-bit characters, but the device you want to transfer a file to or from uses 8-bit characters.

Many different solutions to these problems exist, but they require close attention to the communications environment. Some possible solutions are:

- Do all file transfers in Block mode
- If possible, disable the remote echo from the host
- Send files to the host with Prompt mode disabled. Set the EOM string to $^{C}_{R}{}^{L}_{F}$ and set the transmit delay large enough to allow the host input system to prepare for the input.
- In Prompt mode, change the bypass control character with your host program to a character that is not in the data. Send the bypass control character to the terminal after the host receives each line of data.
- Use the SET-REPORT-LINE-LENGTH command with a parameter of 0 if your file has line terminators such as $^{C}_{R}$ embedded within the data (such as a text file). If your file has infrequent line terminators (such as the result of a SAVE command) set the report line length smaller than your host's input buffer.
- To send 8-bit characters to or from a 7-bit only host you can sometimes use DATA parity if you can control and read control bit 8 (the parity bit).

As an example, assume you are sending a text file from the terminal to the host with full duplex communications in Prompt mode. The host echoes the terminal and the remote echo cannot be disabled. You must consider the following:

- Wait after setting the prompt string for about 1/4 second so the terminal can process the command and recognize the prompt string.
- Set the EOM string to the host terminator characters. Usually this is $^{C}_{R}$ or $^{C}_{R}{}^{N}_{U}$.
- Set the bypass cancel character to the last character the host echoes on each line. This is frequently $^{L}_{F}$ or $^{C}_{R}$. (If the host does not echo, set the bypass control character to $^{N}_{U}$.)
- With a text file containing embedded host terminator characters ($^{C}_{R}$) set the report max line length to 0.
- Set the transmit delay long enough to allow the host to be ready to accept the input.

LOCAL DISK STORAGE

PREVIEW

- Disks are used to store frequently used data.
- Disk utility commands permit formatting disks, and renaming, deleting, and protecting files.
- The DIRECTORY command lists the names of the files on the disk, the amount of used space, and the amount of available space.
- Files are automatically created as needed.
- Disk drives are available with Options 42, 43, and 45.

DISK DEVICES AND COMMANDS

Devices and Device Names

Option 42. Option 42 gives one flexible disk drive, named F0:.

Option 43. Option 43 gives two flexible disk drives. The right drive is F0: and the left drive is F1:.

Option 45. Option 45 adds devices with names in the format *Xn:*. The first character is a member of the set S, T, U, V, W, X, Y. This letter associates the device with a particular controller, such as a hard disk or flexible disk controller on the Option 45 bus. The address of a controller (usually set by hardware straps) determines the letter associated with that controller; address 0 with letter S, 1 with T, and so forth. The second character is a member of the set 0, 1, 2, 3 and specifies a particular hardware device associated with that controller.

Although valid Option 45 device names range from S0: through Y4:, the terminal file system will only accept those device names that are present. To determine whether an Option 45 device is present, use the REPORT-DEVICE-STATUS command for the device in question.

Disk Commands

In addition to the general file transfer commands, the terminal contains several commands that pertain only to the disk options. These commands are:

FORMAT. Normally used by the operator in SETUP mode, the FORMAT command formats a disk for later use.

NOTE

Terminals with version 6.0 or later software use a format that is totally incompatible with earlier versions. You cannot interchange disks with different formats. You must update older format disks with a utility program in order to preserve older files.

DELETE-FILE. This command lets you delete a named file from a specified device.

DIRECTORY. This command transfers the directory file of an entire disk or of a single file to another device, or to the terminal screen if a destination device is not given.

For example, to send the directory of F0: to the host port, use the escape sequence:

^{E}c**JD** F0: TO HO:

THREE-PORT PERIPHERAL INTERFACE (3PPI)

PREVIEW

- The 3PPI adds three port-devices: P0:, P1:, and P2:
- Each port has an independent set of communications parameters.
- The 3PPI has port-protocol identifiers for Tektronix peripherals.
- Some plotter port-protocol identifiers will change pens on a multipen plotter to match a color index.
- Some plotter port-protocol identifiers take GIN information from the plotter joystick and passes it to the terminal.
- The printer port-protocol identifier allows the terminal to print information on a local printer.
- A general purpose port-protocol identifier allows the terminal file system to communicate with any RS-232C, full-duplex peripheral.
- The terminal file system allows two-way file transfers between host and peripheral ports or between two peripheral ports.

DEVICES AND COMMANDS

Option 10, the 3PPI, adds three RS-232 ports, P0:, P1:, and P2:. to the terminal file system. Each port is independent of the other ports. To each port, you can assign its own port-protocol identifier, port baud rate, port EOF string, port EOL string, port flagging mode, port parity, and port stop bits.

The end-of-file mark for the peripheral ports is the port-EOF-string.

3PPI Commands

PORT-ASSIGN. This command assigns a port-protocol identifier to a port device. The port-protocol identifiers and their actions are summarized in the *4110 Series Command Reference Manual*. All port-protocol identifiers except PPORT are output only, so a port with other than PPORT as a driver is a destination only device.

MAP-INDEX-TO-PEN. TEKTRONIX 4663 Plotters have two pens and 4662 Plotters with Option 31 have eight. This command assigns a color index to a pen number. After this assignment, the plotter will select the pen you specify when plotting in that index. You must repeat this command for each color index assignment.

Be aware that this command does not color sort; the plotter changes pens each time the line index changes. Most multi-color plotting can be done much faster by completing a plot in each color before moving to the next.

PLOT. This command saves all currently visible segments (in the current view on a raster display terminal) to any valid device.

PORT-COPY. This command allows a bidirectional file transfer between two ports when both can be used either as source or destination, such as HO: and P0: or P1: and P2:.

This command is most useful when you want to drive a plotter directly. You can issue plotter-format commands and receiving plotter-format reports as if the plotter were connected directly to the host. The port-protocol identifier must be PPORT, since the other port-protocol identifiers are not valid sources.

Either device can send an EOF mark to end the file transfer.

REPORT-PORT-STATUS. This command returns the current parameter settings for a particular port device. For details of the syntax of the report, see the PORT-STATUS-REPORT message type in the *4110 Series Command Reference Manual.*

Commands to Set 3PPI Communications Parameters. You can set the various port communications parameters with these commands:

- SET-PORT-BAUD-RATE
- SET-PORT-EOF-STRING
- SET-PORT-EOL-STRING
- SET-PORT-FLAGGING-MODE
- SET-PORT-PARITY
- SET-PORT-STOP-BITS

THE COLOR HARD COPIER

PREVIEW

- Option 09 adds two devices:
  - SC:, a pseudo device, is valid only as a source for the COPY command.
  - HC:, a physical port device, is valid only as a destination for the COPY and SPOOL commands.

COMMANDS AND DEVICES

Option 09 does not add any commands to the terminal file system. The only two commands that can be used with this option are COPY and SPOOL.

SC:

SC: is a pseudo device that translates information from the display screen to a color hard copy format file. This file causes the color hard copy unit to reproduce the contents of the current pixel viewport. You can use SC: only as a source device for the COPY command.

HC:

HC: is the color hard copy port. A TEKTRONIX 4691 Color Hard Copy unit must be connected and functioning for a file transfer to occur.

HC: is a destination-only device and is only valid for the commands COPY and SPOOL. You can control how the copy will appear by appending a parameter to the HC: device. HC:0 will make a copy in which a black screen background will print white and "HC:1" will make a copy where the black screen background will print black.

You can copy from HO: directly to HC:. HC:, however, requires 8-bit data in a special format.

HINTS

The file transfer from SC: to HC: takes several minutes if the pixel viewport is large. To retain use of the terminal, you should COPY SC: to a file on a disk, and then SPOOL from the disk file to HC:.

THE DMA INTERFACE

Option 3A, the DMA Interface, gives the host computer access to the terminal's internal memory via an extremely high-speed transmission route.

PREVIEW

- The DMA interface adds no new commands.
- The DMA interface adds five new devices.

COMMANDS AND DEVICES

Option 3A, the DMA (Direct Memory Access) Interface adds one physical device and four pseudo devices.

The four pseudo devices require special data formats. For details on these data formats, see the *4115 Option 3A Instruction Manual.*

DM:. DM: is a physical port device, the actual DMA interface. DM: is valid as a source and destination for all commands.

DS:. The pseudo device DS: is the vector display list. DS: is valid as a destination for the command COPY only.

SG:. The pseudo device SG: is the retained segment list. SG: is both a source and destination device for the COPY command.

PX:. The pseudo device PX: is the current pixel viewport. It is both a source and destination device for the COPY command.

CM:. The pseudo device CM: is the color map. It is both a source and destination for the COPY command.

USING DMA FROM THE HOST PROGRAM

DMA is a very high-speed file transfer between the host and the terminal. To use DMA from the host, ready the host and give a file transfer command through normal communications.

Section 11

SOFTWARE COMPATIBILITY

USING 4010 PROGRAMS WITH 4110 SERIES TERMINALS

Most software written for earlier Textronix terminals (4010 Series) will run with little or no modification on 4110 Series terminals. You can connect a 4110 Series terminal, put the terminal into Setup mode, set the necessary parameters, and run the 4010 Series software.

You must do the following to run 4110 Series software on a 4110 Series terminal:

- Set the communications parameters.
- Disable the dialog area.
- Set the window size, if necessary.
- For the 4115, set the overview window.
- For GIN, use the command SET-TABLET-HEADER-CHARACTERS.
- For GIN, use the command SET-TABLET-STATUS-STRAP.

See the appropriate sections of this manual and the *4110 Series Command Reference Manual* for details.

A 4110 Series terminal running 4010 software executes 4010 commands in nearly the same way as a 4010 Series terminal does. 4110 Series alphatext sizes are not the same as 4010 Series alphatext size; the display will have a different appearance.

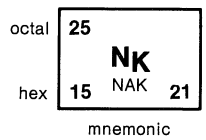
Appendix A

ASCII CHART

B7 B6 B5 / BITS B4 B3 B2 B1	ø ø ø	ø ø 1	ø 1 ø	ø 1 1	1 ø ø	1 ø 1	1 1 ø	1 1 1
	CONTROL		NUMBERS SYMBOLS		UPPERCASE SYMBOLS		LOWERCASE SYMBOLS	
ø ø ø ø	0 NU NUL 0	20 DL DLE 16	40 SP 32	60 0 48	100 @ 64	120 P 80	140 ` 96	160 p 112
ø ø ø 1	1 SH SOH 1	21 D1 DC1 17	41 ! 33	61 1 49	101 A 65	121 Q 81	141 a 97	161 q 113
ø ø 1 ø	2 SX STX 2	22 D2 DC2 18	42 " 34	62 2 50	102 B 66	122 R 82	142 b 98	162 r 114
ø ø 1 1	3 EX ETX 3	23 D3 DC3 19	43 # 35	63 3 51	103 C 67	123 S 83	143 c 99	163 s 115
ø 1 ø ø	4 ET EOT 4	24 D4 DC4 20	44 $ 36	64 4 52	104 D 68	124 T 84	144 d 100	164 t 116
ø 1 ø 1	5 EQ ENQ 5	25 NK NAK 21	45 % 37	65 5 53	105 E 69	125 U 85	145 e 101	165 u 117
ø 1 1 ø	6 AK ACK 6	26 SY SYN 22	46 & 38	66 6 54	106 F 70	126 V 86	146 f 102	166 v 118
ø 1 1 1	7 BL BEL 7	27 EB ETB 23	47 ' 39	67 7 55	107 G 71	127 W 87	147 g 103	167 w 119
1 ø ø ø	10 BS 8	30 CN CAN 24	50 (40	70 8 56	110 H 72	130 X 88	150 h 104	170 x 120
1 ø ø 1	11 HT 9	31 EM 25	51) 41	71 9 57	111 I 73	131 Y 89	151 i 105	171 y 121
1 ø 1 ø	12 LF A 10	32 SB SUB 1A 26	52 * 42	72 : 58	112 J 74	132 Z 90	152 j 106	172 z 122
1 ø 1 1	13 VT B 11	33 EC ESC 1B 27	53 + 43	73 ; 59	113 K 75	133 [91	153 k 107	173 { 123
1 1 ø ø	14 FF C 12	34 FS 1C 28	54 , 44	74 < 60	114 L 76	134 \ 92	154 l 108	174 | * 124
1 1 ø 1	15 CR D 13	35 GS 1D 29	55 - 45	75 = 61	115 M 77	135] 93	155 m 109	175 } 125
1 1 1 ø	16 SO E 14	36 RS 1E 30	56 . 46	76 > 62	116 N 78	136 ^ 4	156 n 110	176 ~ 126
1 1 1 1	17 SI F 15	37 US 1F 31	57 / 47	77 UNL ? 63	117 O 79	137 UNT _ 95	157 o 111	177 DT DEL RUBOUT 7F 127

* | on some keyboards or systems

KEY



3806-60

Appendix B

INT PARAMETERS

You can manually convert integers to *int* parameters by either sucessive division or looking values up in a table. The *4110 Series Command Reference Manual* contains a set of tables for looking up arbitrary *int* values. This appendix contains a method for calculating *int* values and a list of integers from —4049 through 4049 and their corresponding *int* values.

CALCULATING *INT* VALUES

An *int* value consists of from 1 to 6 ASCII characters. The rightmost character is called the *LoI* character and the remaining characters (if present) are called *HiI* characters. The LoI character contains the sign value.

To convert an integer to an *int* value, calculate the characters from right to left as follows:

1. Divide the absolute value of the integer by 16, reserving the quotient. Use the remainder to find the LoI.
   a. If the integer is positive, add 48 to the remainder. This is the ADE (ASCII decimal equivalent) of the LoI.
   b. If the integer is negative, add 32 to the remainder. This is the ADE (ASCII decimal equivalent) of the LoI.
2. Divide the absolute value of the quotient by 64, reserving the quotient if it is greater than zero. Add 64 to the remainder to obtain the ADE of the HiI.
3. Repeat Step 2 as long as any values remain.

As each value is calculated, look up the corresponding ASCII character and write it down from right to left.

Example 1:

What *int* parameter represents +31416?

$\frac{3146}{16}$ = 1963 with remainder 8

(positive integer) 8 + 48 = 56

56 = ADE of "8" on ASCII chart

LoI = **8**

$\frac{1963}{64}$ = 30 with remainder 43

43 + 64 = 107

107 = ADE of "k"

HiI = **k**

$\frac{30}{64}$ = 0 with remainder 30

30 + 64 = 94

94 = ADE of "^"

HiI = ^

31416 = the *int* parameter ^**k8**.

Example 2:

What *int* parameter represents –1024?

$\frac{1024}{16}$ = 64 with remainder 0

(negative) 0 + 32 = 32

32 = ADE of ^{S}P

LoI = ^{S}P

$\frac{64}{64}$ = 1 with remainder 0

0 + 64 = 64

64 = ADE of "@"

HiI = @

$\frac{1}{64}$ = 0 with remainder 1

1 + 64 = 65

65 = ADE of "A"

HiI = **A**

–1024 = the *int* parameter **A@**^{S}P.

Table B-1

REPRESENTING NUMBERS AS *INT* PARAMETERS

n	*int*: n	−n	*int*: −n
0	0	−0	S_P
0	1	−1	!
2	2	−2	"
3	3	−3	#
4	4	−4	$
5	5	−5	%
6	6	−6	&
7	7	−7	'
8	8	−8	(
9	9	−9	)
10	:	−10	*
11	;	−11	+
12	<	−12	,
13	=	−13	−
14	>	−14	.
15	?	−15	/
16	A0	−16	A^{S}_{P}
17	A1	−17	A!
18	A2	−18	A"
19	A3	−19	A#
20	A4	−20	A$
21	A5	−21	A%
22	A6	−22	A&
23	A7	−23	A'
24	A8	−24	A(
25	A9	−25	A)
26	A:	−26	A*
27	A;	−27	A+
28	A<	−28	A,
29	A=	−29	A−
30	A>	−30	A.
31	A?	−31	A/
32	B0	−32	B^{S}_{P}
33	B1	−33	B!
34	B2	−34	B"
35	B3	−35	B#
36	B4	−36	B$
37	B5	−37	B%
38	B6	−38	B&
39	B7	−39	B'
40	B8	−40	B(
41	B9	−41	B)
42	B:	−42	B*
43	B;	−43	B+
44	B<	−44	B,
45	B=	−45	B−
46	B>	−46	B.
47	B?	−47	B/
48	C0	−48	C^{S}_{P}
49	C1	−49	C!
50	C2	−50	C"
51	C3	−51	C#
52	C4	−52	C$
53	C5	−53	C%
54	C6	−54	C&
55	C7	−55	C'
56	C8	−56	C(
57	C9	−57	C)
58	C:	−58	C*
59	C;	−59	C+
60	C<	−60	C,
61	C=	−61	C−
62	C>	−62	C.
63	C?	−63	C/
64	D0	−64	D^{S}_{P}
65	D1	−65	D!
66	D2	−66	D"
67	D3	−67	D#
68	D4	−68	D$
69	D5	−69	D%
70	D6	−70	D&
71	D7	−71	D'
72	D8	−72	D(
73	D9	−73	D)
74	D:	−74	D*
75	D;	−75	D+
76	D<	−76	D,
77	D=	−77	D−
78	D>	−78	D.
79	D?	−79	D/
80	E0	−80	E^{S}_{P}
81	E1	−81	E!
82	E2	−82	E"
83	E3	−83	E#
84	E4	−84	E$
85	E5	−85	E%
86	E6	−86	E&
87	E7	−87	E'
88	E8	−88	E(
89	E9	−89	E)
90	E:	−90	E*
91	E;	−91	E+
92	E<	−92	E,
93	E=	−93	E−
94	E>	−94	E.
95	E?	−95	E/
96	F0	−96	F^{S}_{P}
97	F1	−97	F!
98	F2	−98	F"
99	F3	−99	F#
100	F4	−100	F$
101	F5	−101	F%
102	F6	−102	F&
103	F7	−103	F'
104	F8	−104	F(
105	F9	−105	F)
106	F:	−106	F*
107	F;	−107	F+
108	F<	−108	F,
109	F=	−109	F−
110	F>	−110	F.
111	F?	−111	F/
112	G0	−112	G^{S}_{P}
113	G1	−113	G!
114	G2	−114	G"
115	G3	−115	G#
116	G4	−116	G$
117	G5	−117	G%
118	G6	−118	G&
119	G7	−119	G'
120	G8	−120	G(
121	G9	−121	G)
122	G:	−122	G*
123	G;	−123	G+
124	G<	−124	G,
125	G=	−125	G−
126	G>	−126	G.
127	G?	−127	G/
128	H0	−128	H^{S}_{P}
129	H1	−129	H!
130	H2	−130	H"
131	H3	−131	H#
132	H4	−132	H$
133	H5	−133	H%
134	H6	−134	H&
135	H7	−135	H'
136	H8	−136	H(
137	H9	−137	H)
138	H:	−138	H*
139	H;	−139	H+
140	H<	−140	H,
141	H=	−141	H−
142	H>	−142	H.
143	H?	−143	H/
144	I0	−144	I^{S}_{P}
145	I1	−145	I!
146	I2	−146	I"
147	I3	−147	I#
148	I4	−148	I$
149	I5	−149	I%

Table B-1 (cont)

REPRESENTING NUMBERS AS *INT* PARAMETERS

n	*int*: n	-n	*int*: -n
150	I6	−150	I&
151	I7	−151	I'
152	I8	−152	I(
153	I9	−153	I)
154	I:	−154	I*
155	I;	−155	I+
156	I<	−156	I,
157	I=	−157	I−
158	I>	−158	I.
159	I?	−159	I/
160	J0	−160	J^{SP}
161	J1	−161	J!
162	J2	−162	J"
163	J3	−163	J#
164	J4	−164	J$
165	J5	−165	J%
166	J6	−166	J&
167	J7	−167	J'
168	J8	−168	J(
169	J9	−169	J)
170	J:	−170	J*
171	J;	−171	J+
172	J<	−172	J,
173	J=	−173	J−
174	J>	−174	J.
175	J?	−175	J/
176	K0	−176	K^{SP}
177	K1	−177	K!
178	K2	−178	K"
179	K3	−179	K#
180	K4	−180	K$
181	K5	−181	K%
182	K6	−182	K&
183	K7	−183	K'
184	K8	−184	K(
185	K9	−185	K)
186	K:	−186	K*
187	K;	−187	K+
188	K<	−188	K,
189	K=	−189	K−
190	K>	−190	K.
191	K?	−191	K/
192	L0	−192	L^{SP}
193	L1	−193	L!
194	L2	−194	L"
195	L3	−195	L#
196	L4	−196	L$
197	L5	−197	L%
198	L6	−198	L&
199	L7	−199	L'
200	L8	−200	L(
201	L9	−201	L)
202	L:	−202	L*
203	L;	−203	L+
204	L<	−204	L,
205	L=	−205	L−
206	L>	−206	L.
207	L?	−207	L/
208	M0	−208	M^{SP}
209	M1	−209	M!
210	M2	−210	M"
211	M3	−211	M#
212	M4	−212	M$
213	M5	−213	M%
214	M6	−214	M&
215	M7	−215	M'
216	M8	−216	M(
217	M9	−217	M)
218	M:	−218	M*
219	M;	−219	M+
220	M<	−220	M,
221	M=	−221	M−
222	M>	−222	M.
223	M?	−223	M/
224	N0	−224	N^{SP}
225	N1	−225	N!
226	N2	−226	N"
227	N3	−227	N#
228	N4	−228	N$
229	N5	−229	N%
230	N6	−230	N&
231	N7	−231	N'
232	N8	−232	N(
233	N9	−233	N)
234	N:	−234	N*
235	N;	−235	N+
236	N<	−236	N,
237	N=	−237	N−
238	N>	−238	N.
239	N?	−239	N/
240	O0	−240	O^{SP}
241	O1	−241	O!
242	O2	−242	O"
243	O3	−243	O#
244	O4	−244	O$
245	O5	−245	O%
246	O6	−246	O&
247	O7	−247	O'
248	O8	−248	O(
249	O9	−249	O)
250	O:	−250	O*
251	O;	−251	O+
252	O<	−252	O,
253	O=	−253	O−
254	O>	−254	O.
255	O?	−255	O/
256	P0	−256	P^{SP}
257	P1	−257	P!
258	P2	−258	P"
259	P3	−259	P#
260	P4	−260	P$
261	P5	−261	P%
262	P6	−262	P&
263	P7	−263	P'
264	P8	−264	P(
265	P9	−265	P)
266	P:	−266	P*
267	P;	−267	P+
268	P<	−268	P,
269	P=	−269	P−
270	P>	−270	P.
271	P?	−271	P/
272	Q0	−272	Q^{SP}
273	Q1	−273	Q!
274	Q2	−274	Q"
275	Q3	−275	Q#
276	Q4	−276	Q$
277	Q5	−277	Q%
278	Q6	−278	Q&
279	Q7	−279	Q'
280	Q8	−280	Q(
281	Q9	−281	Q)
282	Q:	−282	Q*
283	Q;	−283	Q+
284	Q<	−284	Q,
285	Q=	−285	Q−
286	Q>	−286	Q.
287	Q?	−287	Q/
288	R0	−288	R^{SP}
289	R1	−289	R!
290	R2	−290	R"
291	R3	−291	R#
292	R4	−292	R$
293	R5	−293	R%
294	R6	−294	R&
295	R7	−295	R'
296	R8	−296	R(
297	R9	−297	R)
298	R:	−298	R*
299	R;	−299	R+

Table B-1 (cont)

REPRESENTING NUMBERS AS *INT* PARAMETERS

n	*int*: n	-n	*int*: -n
300	R<	−300	R,
301	R=	−301	R−
302	R>	−302	R.
303	R?	−303	R/
304	S0	−304	S^{SP}
305	S1	−305	S!
306	S2	−306	S"
307	S3	−307	S#
308	S4	−308	S$
309	S5	−309	S%
310	S6	−310	S&
311	S7	−311	S'
312	S8	−312	S(
313	S9	−313	S)
314	S:	−314	S*
315	S;	−315	S+
316	S<	−316	S,
317	S=	−317	S−
318	S>	−318	S.
319	S?	−319	S/
320	T0	−320	T^{SP}
321	T1	−321	T!
322	T2	−322	T"
323	T3	−323	T#
324	T4	−324	T$
325	T5	−325	T%
326	T6	−326	T&
327	T7	−327	T'
328	T8	−328	T(
329	T9	−329	T)
330	T:	−330	T*
331	T;	−331	T+
332	T<	−332	T,
333	T=	−333	T−
334	T>	−334	T.
335	T?	−335	T/
336	U0	−336	U^{SP}
337	U1	−337	U!
338	U2	−338	U"
339	U3	−339	U#
340	U4	−340	U$
341	U5	−341	U%
342	U6	−342	U&
343	U7	−343	U'
344	U8	−344	U(
345	U9	−345	U)
346	U:	−346	U*
347	U;	−347	U+
348	U<	−348	U,
349	U=	−349	U−
350	U>	−350	U.
351	U?	−351	U/
352	V0	−352	V^{SP}
353	V1	−353	V!
354	V2	−354	V"
355	V3	−355	V#
356	V4	−356	V$
357	V5	−357	V%
358	V6	−358	V&
359	V7	−359	V'
360	V8	−360	V(
361	V9	−361	V)
362	V:	−362	V*
363	V;	−363	V+
364	V<	−364	V,
365	V=	−365	V−
366	V>	−366	V.
367	V?	−367	V/
368	W0	−368	W^{SP}
369	W1	−369	W!
370	W2	−370	W"
371	W3	−371	W#
372	W4	−372	W$
373	W5	−373	W%
374	W6	−374	W&
375	W7	−375	W'
376	W8	−376	W(
377	W9	−377	W)
378	W:	−378	W*
379	W;	−379	W+
380	W<	−380	W,
381	W=	−381	W−
382	W>	−382	W.
383	W?	−383	W/
384	X0	−384	X^{SP}
385	X1	−385	X!
386	X2	−386	X"
387	X3	−387	X#
388	X4	−388	X$
389	X5	−389	X%
390	X6	−390	X&
391	X7	−391	X'
392	X8	−392	X(
393	X9	−393	X)
394	X:	−394	X*
395	X;	−395	X+
396	X<	−396	X,
397	X=	−397	X−
398	X>	−398	X.
399	X?	−399	X/
400	Y0	−400	Y^{SP}
401	Y1	−401	Y!
402	Y2	−402	Y"
403	Y3	−403	Y#
404	Y4	−404	Y$
405	Y5	−405	Y%
406	Y6	−406	Y&
407	Y7	−407	Y'
408	Y8	−408	Y(
409	Y9	−409	Y)
410	Y:	−410	Y*
411	Y;	−411	Y+
412	Y<	−412	Y,
413	Y=	−413	Y−
414	Y>	−414	Y.
415	Y?	−415	Y/
416	Z0	−416	Z^{SP}
417	Z1	−417	Z!
418	Z2	−418	Z"
419	Z3	−419	Z#
420	Z4	−420	Z$
421	Z5	−421	Z%
422	Z6	−422	Z&
423	Z7	−423	Z'
424	Z8	−424	Z(
425	Z9	−425	Z)
426	Z:	−426	Z*
427	Z;	−427	Z+
428	Z<	−428	Z,
429	Z=	−429	Z−
430	Z>	−430	Z.
431	Z?	−431	Z/
432	[0	−432	$[^{SP}$
433	[1	−433	[!
434	[2	−434	["
435	[3	−435	[#
436	[4	−436	[$
437	[5	−437	[%
438	[6	−438	[&
439	[7	−439	['
440	[8	−440	[(
441	[9	−441	[)
442	[:	−442	[*
443	[;	−443	[+
444	[<	−444	[,
445	[=	−445	[−
446	[>	−446	[.
447	[?	−447	[/
448	\0	−448	$\backslash^{SP}$
449	\1	−449	\!

Table B-1 (cont)

REPRESENTING NUMBERS AS *INT* PARAMETERS

n	*int*: n	−n	*int*: −n	n	*int*: n	−n	*int*: −n	n	*int*: n	−n	*int*: −n
450	\2	−450	\"	500	_4	−500	_$	550	b6	−550	b&
451	\3	−451	\#	501	_5	−501	_%	551	b7	−551	b'
452	\4	−452	\$	502	_6	−502	_&	552	b8	−552	b(
453	\5	−453	\%	503	_7	−503	_'	553	b9	−553	b)
454	\6	−454	\&	504	_8	−504	_(	554	b:	−554	b*
455	\7	−455	\'	505	_9	−505	_)	555	b;	−555	b+
456	\8	−456	\(	506	_:	−506	_*	556	b<	−556	b,
457	\9	−457	\)	507	_;	−507	_+	557	b=	−557	b−
458	\:	−458	*	508	_<	−508	_,	558	b>	−558	b.
459	\;	−459	\+	509	_=	−509	_−	559	b?	−559	b/
460	\<	−460	\,	510	_>	−510	_.	560	c0	−560	c^{SP}
461	\=	−461	\−	511	_?	−511	_/	561	c1	−561	c!
462	\>	−462	\.	512	'0	−512	$'^{SP}$	562	c2	−562	c"
463	\?	−463	\/	513	'1	−513	'!	563	c3	−563	c#
464	]0	−464	$]^{SP}$	514	'2	−514	'"	564	c4	−564	c$
465	]1	−465	]!	515	'3	−515	'#	565	c5	−565	c%
466	]2	−466	]"	516	'4	−516	'$	566	c6	−566	c&
467	]3	−467	]#	517	'5	−517	'%	567	c7	−567	c'
468	]4	−468	]$	518	'6	−518	'&	568	c8	−568	c(
469	]5	−469	]%	519	'7	−519	'	569	c9	−569	c)
470	]6	−470	]&	520	'8	−520	'(	570	c:	−570	c*
471	]7	−471	]'	521	'9	−521	')	571	c;	−571	c+
472	]8	−472	](	522	':	−522	'*	572	c<	−572	c,
473	]9	−473	])	523	';	−523	'+	573	c=	−573	c−
474	]:	−474	]*	524	'<	−524	',	574	c>	−574	c.
475	];	−475	]+	525	'=	−525	'−	575	c?	−575	c/
476	]<	−476	],	526	'>	−526	'.	576	d0	−576	d^{SP}
477	]=	−477	]−	527	'?	−527	'/	577	d1	−577	d!
478	]>	−478	].	528	a0	−528	a^{SP}	578	d2	−578	d"
479	]?	−479	]/	529	a1	−529	a!	579	d3	−579	d#
480	∧0	−480	$∧^{SP}$	530	a2	−530	a"	580	d4	−580	d$
481	∧1	−481	∧!	531	a3	−531	a#	581	d5	−581	d%
482	∧2	−482	∧"	532	a4	−532	a$	582	d6	−582	d&
483	∧3	−483	∧#	533	a5	−533	a%	583	d7	−583	d'
484	∧4	−484	∧$	534	a6	−534	a&	584	d8	−584	d(
485	∧5	−485	∧%	535	a7	−535	a'	585	d9	−585	d)
486	∧6	−486	∧&	536	a8	−536	a(	586	d:	−586	d*
487	∧7	−487	∧'	537	a9	−537	a)	587	d;	−587	d+
488	∧8	−488	∧(	538	a:	−538	a*	588	d<	−588	d,
489	∧9	−489	∧)	539	a;	−539	a+	589	d=	−589	d−
490	∧:	−490	∧*	540	a<	−540	a,	590	d>	−590	d.
491	∧;	−491	∧+	541	a=	−541	a−	591	d?	−591	d/
492	∧<	−492	∧,	542	a>	−542	a.	592	e0	−592	e^{SP}
493	∧=	−493	∧−	543	a?	−543	a/	593	e1	−593	e!
494	∧>	−494	∧.	544	b0	−544	b^{SP}	594	e2	−594	e"
495	∧?	−495	∧/	545	b1	−545	b!	595	e3	−595	e#
496	_0	−496	$_^{SP}$	546	b2	−546	b"	596	e4	−596	e$
497	_1	−497	_!	547	b3	−547	b#	597	e5	−597	e%
498	_2	−498	_"	548	b4	−548	b$	598	e6	−598	e&
499	_3	−499	_#	549	b5	−549	b%	599	e7	−599	e'

Table B-1 (cont)

REPRESENTING NUMBERS AS *INT* PARAMETERS

n	*int*: n	-n	*int*: -n
600	e8	−600	e(
601	e9	−601	e)
602	e:	−602	e*
603	e;	−603	e+
604	e<	−604	e,
605	e=	−605	e−
606	e>	−606	e.
607	e?	−607	e/
608	f0	−608	f^{SP}
609	f1	−609	f!
610	f2	−610	f"
611	f3	−611	f#
612	f4	−612	f$
613	f5	−613	f%
614	f6	−614	f&
615	f7	−615	f'
616	f8	−616	f(
617	f9	−617	f)
618	f:	−618	f*
619	f;	−619	f+
620	f<	−620	f,
621	f=	−621	f−
622	f>	−622	f.
623	f?	−623	f/
624	g0	−624	g^{SP}
625	g1	−625	g!
626	g2	−626	g"
627	g3	−627	g#
628	g4	−628	g$
629	g5	−629	g%
630	g6	−630	g&
631	g7	−631	g'
632	g8	−632	g(
633	g9	−633	g)
634	g:	−634	g*
635	g;	−635	g+
636	g<	−636	g,
637	g=	−637	g−
638	g>	−638	g.
639	g?	−639	g/
640	h0	−640	h^{SP}
641	h1	−641	h!
642	h2	−642	h"
643	h3	−643	h#
644	h4	−644	h$
645	h5	−645	h%
646	h6	−646	h&
647	h7	−647	h'
648	h8	−648	h(
649	h9	−649	h)
650	h:	−650	h*
651	h;	−651	h+
652	h<	−652	h,
653	h=	−653	h−
654	h>	−654	h.
655	h?	−655	h/
656	i0	−656	i^{SP}
657	i1	−657	i!
658	i2	−658	i"
659	i3	−659	i#
660	i4	−660	i$
661	i5	−661	i%
662	i6	−662	i&
663	i7	−663	i'
664	i8	−664	i(
665	i9	−665	i)
666	i:	−666	i*
667	i;	−667	i+
668	i<	−668	i,
669	i=	−669	i−
670	i>	−670	i.
671	i?	−671	i/
672	j0	−672	j^{SP}
673	j1	−673	j!
674	j2	−674	j"
675	j3	−675	j#
676	j4	−676	j$
677	j5	−677	j%
678	j6	−678	j&
679	j7	−679	j'
680	j8	−680	j(
681	j9	−681	j)
682	j:	−682	j*
683	j;	−683	j+
684	j<	−684	j,
685	j=	−685	j−
686	j>	−686	j.
687	j?	−687	j/
688	k0	−688	k^{SP}
689	k1	−689	k!
690	k2	−690	k"
691	k3	−691	k#
692	k4	−692	k$
693	k5	−693	k%
694	k6	−694	k&
695	k7	−695	k'
696	k8	−696	k(
697	k9	−697	k)
698	k:	−698	k*
699	k;	−699	k+
700	k<	−700	k,
701	k=	−701	k−
702	k>	−702	k.
703	k?	−703	k/
704	l0	−704	l^{SP}
705	l1	−705	l!
706	l2	−706	l"
707	l3	−707	l#
708	l4	−708	l$
709	l5	−709	l%
710	l6	−710	l&
711	l7	−711	l'
712	l8	−712	l(
713	l9	−713	l)
714	l:	−714	l*
715	l;	−715	l+
716	l<	−716	l,
717	l=	−717	l−
718	l>	−718	l.
719	l?	−719	l/
720	m0	−720	m^{SP}
721	m1	−721	m!
722	m2	−722	m"
723	m3	−723	m#
724	m4	−724	m$
725	m5	−725	m%
726	m6	−726	m&
727	m7	−727	m'
728	m8	−728	m(
729	m9	−729	m)
730	m:	−730	m*
731	m;	−731	m+
732	m<	−732	m,
733	m=	−733	m−
734	m>	−734	m.
735	m?	−735	m/
736	n0	−736	n^{SP}
737	n1	−737	n!
738	n2	−738	n"
739	n3	−739	n#
740	n4	−740	n$
741	n5	−741	n%
742	n6	−742	n&
743	n7	−743	n'
744	n8	−744	n(
745	n9	−745	n)
746	n:	−746	n*
747	n;	−747	n+
748	n<	−748	n,
749	n=	−749	n−

Table B-1 (cont)

REPRESENTING NUMBERS AS *INT* PARAMETERS

n	*int*: n	-n	*int*: -n
750	n>	−750	n.
751	n?	−751	n/
752	o0	−752	o^{SP}
753	o1	−753	o!
754	o2	−754	o"
755	o3	−755	o#
756	o4	−756	o$
757	o5	−757	o%
758	o6	−758	o&
759	o7	−759	o'
760	o8	−760	o(
761	o9	−761	o)
762	o:	−762	o*
763	o;	−763	o+
764	o<	−764	o,
765	o=	−765	o−
766	o>	−766	o.
767	o?	−767	o/
768	p0	−768	p^{SP}
769	p1	−769	p!
770	p2	−770	p"
771	p3	−771	p#
772	p4	−772	p$
773	p5	−773	p%
774	p6	−774	p&
775	p7	−775	p'
776	p8	−776	p(
777	p9	−777	p)
778	p:	−778	p*
779	p;	−779	p+
780	p<	−780	p,
781	p=	−781	p−
782	p>	−782	p.
783	p?	−783	p/
784	q0	−784	q^{SP}
785	q1	−785	q!
786	q2	−786	q"
787	q3	−787	q#
788	q4	−788	q$
789	q5	−789	q%
790	q6	−790	q&
791	q7	−791	q'
792	q8	−792	q(
793	q9	−793	q)
794	q:	−794	q*
795	q;	−795	q+
796	q<	−796	q,
797	q=	−797	q−
798	q>	−798	q.
799	q?	−799	q/
800	r0	−800	r^{SP}
801	r1	−801	r!
802	r2	−802	r"
803	r3	−803	r#
804	r4	−804	r$
805	r5	−805	r%
806	r6	−806	r&
807	r7	−807	r'
808	r8	−808	r(
809	r9	−809	r)
810	r:	−810	r*
811	r;	−811	r+
812	r<	−812	r,
813	r=	−813	r−
814	r>	−814	r.
815	r?	−815	r/
816	s0	−816	s^{SP}
817	s1	−817	s!
818	s2	−818	s"
819	s3	−819	s#
820	s4	−820	s$
821	s5	−821	s%
822	s6	−822	s&
823	s7	−823	s'
824	s8	−824	s(
825	s9	−825	s)
826	s:	−826	s*
827	s;	−827	s+
828	s<	−828	s,
829	s=	−829	s−
830	s>	−830	s.
831	s?	−831	s/
832	t0	−832	t^{SP}
833	t1	−833	t!
834	t2	−834	t"
835	t3	−835	t#
836	t4	−836	t$
837	t5	−837	t%
838	t6	−838	t&
839	t7	−839	t'
840	t8	−840	t(
841	t9	−841	t)
842	t:	−842	t*
843	t;	−843	t+
844	t<	−844	t,
845	t=	−845	t−
846	t>	−846	t.
847	t?	−847	t/
848	u0	−848	u^{SP}
849	u1	−849	u!
850	u2	−850	u"
851	u3	−851	u#
852	u4	−852	u$
853	u5	−853	u%
854	u6	−854	u&
855	u7	−855	u'
856	u8	−856	u(
857	u9	−857	u)
858	u:	−858	u*
859	u;	−859	u+
860	u<	−860	u,
861	u=	−861	u−
862	u>	−862	u.
863	u?	−863	u/
864	v0	−864	v^{SP}
865	v1	−865	v!
866	v2	−866	v"
867	v3	−867	v#
868	v4	−868	v$
869	v5	−869	v%
870	v6	−870	v&
871	v7	−871	v'
872	v8	−872	v(
873	v9	−873	v)
874	v:	−874	v*
875	v;	−875	v+
876	v<	−876	v,
877	v=	−877	v−
878	v>	−878	v.
879	v?	−879	v/
880	w0	−880	w^{SP}
881	w1	−881	w!
882	w2	−882	w"
883	w3	−883	w#
884	w4	−884	w$
885	w5	−885	w%
886	w6	−886	w&
887	w7	−887	w'
888	w8	−888	w(
889	w9	−889	w)
890	w:	−890	w*
891	w;	−891	w+
892	w<	−892	w,
893	w=	−893	w−
894	w>	−894	w.
895	w?	−895	w/
896	x0	−896	x^{SP}
897	x1	−897	x!
898	x2	−898	x"
899	x3	−899	x#

Table B-1 (cont)

REPRESENTING NUMBERS AS *INT* PARAMETERS

n	*int*: n	−n	*int*: −n
900	x4	−900	x$
901	x5	−901	x%
902	x6	−902	x&
903	x7	−903	x'
904	x8	−904	x(
905	x9	−905	x)
906	x:	−906	x*
907	x;	−907	x+
908	x<	−908	x,
909	x=	−909	x−
910	x>	−910	x.
911	x?	−911	x/
912	y0	−912	y$_P$
913	y1	−913	y!
914	y2	−914	y"
915	y3	−915	y#
916	y4	−916	y$
917	y5	−917	y%
918	y6	−918	y&
919	y7	−919	y'
920	y8	−920	y(
921	y9	−921	y)
922	y:	−922	y*
923	y;	−923	y+
924	y<	−924	y,
925	y=	−925	y−
926	y>	−926	y.
927	y?	−927	y/
928	z0	−928	z$_P$
929	z1	−929	z!
930	z2	−930	z"
931	z3	−931	z#
932	z4	−932	z$
933	z5	−933	z%
934	z6	−934	z&
935	z7	−935	z'
936	z8	−936	z(
937	z9	−937	z)
938	z:	−938	z*
939	z;	−939	z+
940	z<	−940	z,
941	z=	−941	z−
942	z>	−942	z.
943	z?	−943	z/
944	{0	−944	{$_P$
945	{1	−945	{!
946	{2	−946	{"
947	{3	−947	{#
948	{4	−948	{$
949	{5	−949	{%

n	*int*: n	−n	*int*: −n
950	{6	−950	{&
951	{7	−951	{'
952	{8	−952	{(
953	{9	−953	{)
954	{:	−954	{*
955	{;	−955	{+
956	{<	−956	{,
957	{=	−957	{−
958	{>	−958	{.
959	{?	−959	{/
960	|0	−960	|$_P$
961	|1	−961	|!
962	|2	−962	|"
963	|3	−963	|#
964	|4	−964	|$
965	|5	−965	|%
966	|6	−966	|&
967	|7	−967	|'
968	|8	−968	|(
969	|9	−969	|)
970	|:	−970	|*
971	|;	−971	|+
972	|<	−972	|,
973	|=	−973	|−
974	|>	−974	|.
975	|?	−975	|/
976	}0	−976	}$_P$
977	}1	−977	}!
978	}2	−978	}"
979	}3	−979	}#
980	}4	−980	}$
981	}5	−981	}%
982	}6	−982	}&
983	}7	−983	}'
984	}8	−984	}(
985	}9	−985	})
986	}:	−986	}*
987	};	−987	}+
988	}<	−988	},
989	}=	−989	}−
990	}>	−990	}.
991	}?	−991	}/
992	~0	−992	~$_P$
993	~1	−993	~!
994	~2	−994	~"
995	~3	−995	~#
996	~4	−996	~$
997	~5	−997	~%
998	~6	−998	~&
999	~7	−999	~'

n	*int*: n	−n	*int*: −n
1000	~8	−1000	~(
1001	~9	−1001	~)
1002	~:	−1002	~*
1003	~;	−1003	~+
1004	~<	−1004	~,
1005	~=	−1005	~−
1006	~>	−1006	~.
1007	~?	−1007	~/
1008	^D_T0	−1008	$^D_T$$^S_P$
1009	^D_T1	−1009	^D_T!
1010	^D_T2	−1010	^D_T"
1011	^D_T3	−1011	^D_T#
1012	^D_T4	−1012	$^D_T$$
1013	^D_T5	−1013	^D_T%
1014	^D_T6	−1014	^D_T&
1015	^D_T7	−1015	^D_T'
1016	^D_T8	−1016	^D_T(
1017	^D_T9	−1017	^D_T)
1018	^D_T:	−1018	^D_T*
1019	^D_T;	−1019	^D_T+
1020	^D_T<	−1020	^D_T,
1021	^D_T=	−1021	^D_T−
1022	^D_T>	−1022	^D_T.
1023	^D_T?	−1023	^D_T/
1024	A@0	−1024	A@^S_P
1025	A@1	−1025	A@!
1026	A@2	−1026	A@"
1027	A@3	−1027	A@#
1028	A@4	−1028	A@$
1029	A@5	−1029	A@%
1030	A@6	−1030	A@&
1031	A@7	−1031	A@'
1032	A@8	−1032	A@(
1033	A@9	−1033	A@)
1034	A@:	−1034	A@*
1035	A@;	−1035	A@+
1036	A@<	−1036	A@,
1037	A@=	−1037	A@−
1038	A@>	−1038	A@.
1039	A@?	−1039	A@/
1040	AA0	−1040	AA^S_P
1041	AA1	−1041	AA!
1042	AA2	−1042	AA"
1043	AA3	−1043	AA#
1044	AA4	−1044	AA$
1045	AA5	−1045	AA%
1046	AA6	−1046	AA&
1047	AA7	−1047	AA'
1048	AA8	−1048	AA(
1049	AA9	−1049	AA)

Table B-1 (cont)

REPRESENTING NUMBERS AS *INT* PARAMETERS

n	*int*: n	-n	*int*: -n
1050	AA:	−1050	AA*
1051	AA;	−1051	AA+
1052	AA<	−1052	AA,
1053	AA=	−1053	AA−
1054	AA>	−1054	AA.
1055	AA?	−1055	AA/
1056	AB0	−1056	AB^{SP}
1057	AB1	−1057	AB!
1058	AB2	−1058	AB"
1059	AB3	−1059	AB#
1060	AB4	−1060	AB$
1061	AB5	−1061	AB%
1062	AB6	−1062	AB&
1063	AB7	−1063	AB'
1064	AB8	−1064	AB(
1065	AB9	−1065	AB)
1066	AB:	−1066	AB*
1067	AB;	−1067	AB+
1068	AB<	−1068	AB,
1069	AB=	−1069	AB−
1070	AB>	−1070	AB.
1071	AB?	−1071	AB/
1072	AC0	−1072	AC^{SP}
1073	AC1	−1073	AC!
1074	AC2	−1074	AC"
1075	AC3	−1075	AC#
1076	AC4	−1076	AC$
1077	AC5	−1077	AC%
1078	AC6	−1078	AC&
1079	AC7	−1079	AC'
1080	AC8	−1080	AC(
1081	AC9	−1081	AC)
1082	AC:	−1082	AC*
1083	AC;	−1083	AC+
1084	AC<	−1084	AC,
1085	AC=	−1085	AC−
1086	AC>	−1086	AC.
1087	AC?	−1087	AC/
1088	AD0	−1088	AD^{SP}
1089	AD1	−1089	AD!
1090	AD2	−1090	AD"
1091	AD3	−1091	AD#
1092	AD4	−1092	AD$
1093	AD5	−1093	AD%
1094	AD6	−1094	AD&
1095	AD7	−1095	AD'
1096	AD8	−1096	AD(
1097	AD9	−1097	AD)
1098	AD:	−1098	AD*
1099	AD;	−1099	AD+
1100	AD<	−1100	AD,
1101	AD=	−1101	AD−
1102	AD>	−1102	AD.
1103	AD?	−1103	AD/
1104	AE0	−1104	AE^{SP}
1105	AE1	−1105	AE!
1106	AE2	−1106	AE"
1107	AE3	−1107	AE#
1108	AE4	−1108	AE$
1109	AE5	−1109	AE%
1110	AE6	−1110	AE&
1111	AE7	−1111	AE'
1112	AE8	−1112	AE(
1113	AE9	−1113	AE)
1114	AE:	−1114	AE*
1115	AE;	−1115	AE+
1116	AE<	−1116	AE,
1117	AE=	−1117	AE−
1118	AE>	−1118	AE.
1119	AE?	−1119	AE/
1120	AF0	−1120	AF^{SP}
1121	AF1	−1121	AF!
1122	AF2	−1122	AF"
1123	AF3	−1123	AF#
1124	AF4	−1124	AF$
1125	AF5	−1125	AF%
1126	AF6	−1126	AF&
1127	AF7	−1127	AF'
1128	AF8	−1128	AF(
1129	AF9	−1129	AF)
1130	AF:	−1130	AF*
1131	AF;	−1131	AF+
1132	AF<	−1132	AF,
1133	AF=	−1133	AF−
1134	AF>	−1134	AF.
1135	AF?	−1135	AF/
1136	AG0	−1136	AG^{SP}
1137	AG1	−1137	AG!
1138	AG2	−1138	AG"
1139	AG3	−1139	AG#
1140	AG4	−1140	AG$
1141	AG5	−1141	AG%
1142	AG6	−1142	AG&
1143	AG7	−1143	AG'
1144	AG8	−1144	AG(
1145	AG9	−1145	AG)
1146	AG:	−1146	AG*
1147	AG;	−1147	AG+
1148	AG<	−1148	AG,
1149	AG=	−1149	AG−
1150	AG>	−1150	AG.
1151	AG?	−1151	AG/
1152	AH0	−1152	AH^{SP}
1153	AH1	−1153	AH!
1154	AH2	−1154	AH"
1155	AH3	−1155	AH#
1156	AH4	−1156	AH$
1157	AH5	−1157	AH%
1158	AH6	−1158	AH&
1159	AH7	−1159	AH'
1160	AH8	−1160	AH(
1161	AH9	−1161	AH)
1162	AH:	−1162	AH*
1163	AH;	−1163	AH+
1164	AH<	−1164	AH,
1165	AH=	−1165	AH−
1166	AH>	−1166	AH.
1167	AH?	−1167	AH/
1168	AI0	−1168	AI^{SP}
1169	AI1	−1169	AI!
1170	AI2	−1170	AI"
1171	AI3	−1171	AI#
1172	AI4	−1172	AI$
1173	AI5	−1173	AI%
1174	AI6	−1174	AI&
1175	AI7	−1175	AI'
1176	AI8	−1176	AI(
1177	AI9	−1177	AI)
1178	AI:	−1178	AI*
1179	AI;	−1179	AI+
1180	AI<	−1180	AI,
1181	AI=	−1181	AI−
1182	AI>	−1182	AI.
1183	AI?	−1183	AI/
1184	AJ0	−1184	AJ^{SP}
1185	AJ1	−1185	AJ!
1186	AJ2	−1186	AJ"
1187	AJ3	−1187	AJ#
1188	AJ4	−1188	AJ$
1189	AJ5	−1189	AJ%
1190	AJ6	−1190	AJ&
1191	AJ7	−1191	AJ'
1192	AJ8	−1192	AJ(
1193	AJ9	−1193	AJ)
1194	AJ:	−1194	AJ*
1195	AJ;	−1195	AJ+
1196	AJ<	−1196	AJ,
1197	AJ=	−1197	AJ−
1198	AJ>	−1198	AJ.
1199	AJ?	−1199	AJ/

Table B-1 (cont)

REPRESENTING NUMBERS AS *INT* PARAMETERS

n	*int*: n	-n	*int*: -n
1200	AK0	−1200	AK^{SP}
1201	AK1	−1201	AK!
1202	AK2	−1202	AK"
1203	AK3	−1203	AK#
1204	AK4	−1204	AK$
1205	AK5	−1205	AK%
1206	AK6	−1206	AK&
1207	AK7	−1207	AK'
1208	AK8	−1208	AK(
1209	AK9	−1209	AK)
1210	AK:	−1210	AK*
1211	AK;	−1211	AK+
1212	AK<	−1212	AK,
1213	AK=	−1213	AK−
1214	AK>	−1214	AK.
1215	AK?	−1215	AK/
1216	AL0	−1216	AL^{SP}
1217	AL1	−1217	AL!
1218	AL2	−1218	AL"
1219	AL3	−1219	AL#
1220	AL4	−1220	AL$
1221	AL5	−1221	AL%
1222	AL6	−1222	AL&
1223	AL7	−1223	AL'
1224	AL8	−1224	AL(
1225	AL9	−1225	AL)
1226	AL:	−1226	AL*
1227	AL;	−1227	AL+
1228	AL<	−1228	AL,
1229	AL=	−1229	AL−
1230	AL>	−1230	AL.
1231	AL?	−1231	AL/
1232	AM0	−1232	AM^{SP}
1233	AM1	−1233	AM!
1234	AM2	−1234	AM"
1235	AM3	−1235	AM#
1236	AM4	−1236	AM$
1237	AM5	−1237	AM%
1238	AM6	−1238	AM&
1239	AM7	−1239	AM'
1240	AM8	−1240	AM(
1241	AM9	−1241	AM)
1242	AM:	−1242	AM*
1243	AM;	−1243	AM+
1244	AM<	−1244	AM,
1245	AM=	−1245	AM−
1246	AM>	−1246	AM.
1247	AM?	−1247	AM/
1248	AN0	−1248	AN^{SP}
1249	AN1	−1249	AN!
1250	AN2	−1250	AN"
1251	AN3	−1251	AN#
1252	AN4	−1252	AN$
1253	AN5	−1253	AN%
1254	AN6	−1254	AN&
1255	AN7	−1255	AN'
1256	AN8	−1256	AN(
1257	AN9	−1257	AN)
1258	AN:	−1258	AN*
1259	AN;	−1259	AN+
1260	AN<	−1260	AN,
1261	AN=	−1261	AN−
1262	AN>	−1262	AN.
1263	AN?	−1263	AN/
1264	AO0	−1264	AO^{SP}
1265	AO1	−1265	AO!
1266	AO2	−1266	AO"
1267	AO3	−1267	AO#
1268	AO4	−1268	AO$
1269	AO5	−1269	AO%
1270	AO6	−1270	AO&
1271	AO7	−1271	AO'
1272	AO8	−1272	AO(
1273	AO9	−1273	AO)
1274	AO:	−1274	AO*
1275	AO;	−1275	AO+
1276	AO<	−1276	AO,
1277	AO=	−1277	AO−
1278	AO>	−1278	AO.
1279	AO?	−1279	AO/
1280	AP0	−1280	AP^{SP}
1281	AP1	−1281	AP!
1282	AP2	−1282	AP"
1283	AP3	−1283	AP#
1284	AP4	−1284	AP$
1285	AP5	−1285	AP%
1286	AP6	−1286	AP&
1287	AP7	−1287	AP'
1288	AP8	−1288	AP(
1289	AP9	−1289	AP)
1290	AP:	−1290	AP*
1291	AP;	−1291	AP+
1292	AP<	−1292	AP,
1293	AP=	−1293	AP−
1294	AP>	−1294	AP.
1295	AP?	−1295	AP/
1296	AQ0	−1296	AQ^{SP}
1297	AQ1	−1297	AQ!
1298	AQ2	−1298	AQ"
1299	AQ3	−1299	AQ#
1300	AQ4	−1300	AQ$
1301	AQ5	−1301	AQ%
1302	AQ6	−1302	AQ&
1303	AQ7	−1303	AQ'
1304	AQ8	−1304	AQ(
1305	AQ9	−1305	AQ)
1306	AQ:	−1306	AQ*
1307	AQ;	−1307	AQ+
1308	AQ<	−1308	AQ,
1309	AQ=	−1309	AQ−
1310	AQ>	−1310	AQ.
1311	AQ?	−1311	AQ/
1312	AR0	−1312	AR^{SP}
1313	AR1	−1313	AR!
1314	AR2	−1314	AR"
1315	AR3	−1315	AR#
1316	AR4	−1316	AR$
1317	AR5	−1317	AR%
1318	AR6	−1318	AR&
1319	AR7	−1319	AR'
1320	AR8	−1320	AR(
1321	AR9	−1321	AR)
1322	AR:	−1322	AR*
1323	AR;	−1323	AR+
1324	AR<	−1324	AR,
1325	AR=	−1325	AR−
1326	AR>	−1326	AR.
1327	AR?	−1327	AR/
1328	AS0	−1328	AS^{SP}
1329	AS1	−1329	AS!
1330	AS2	−1330	AS"
1331	AS3	−1331	AS#
1332	AS4	−1332	AS$
1333	AS5	−1333	AS%
1334	AS6	−1334	AS&
1335	AS7	−1335	AS'
1336	AS8	−1336	AS(
1337	AS9	−1337	AS)
1338	AS:	−1338	AS*
1339	AS;	−1339	AS+
1340	AS<	−1340	AS,
1341	AS=	−1341	AS−
1342	AS>	−1342	AS.
1343	AS?	−1343	AS/
1344	AT0	−1344	AT^{SP}
1345	AT1	−1345	AT!
1346	AT2	−1346	AT"
1347	AT3	−1347	AT#
1348	AT4	−1348	AT$
1349	AT5	−1349	AT%

Table B-1 (cont)

REPRESENTING NUMBERS AS *INT* PARAMETERS

n	*int*: n	-n	*int*: -n
1350	AT6	−1350	AT&
1351	AT7	−1351	AT'
1352	AT8	−1352	AT(
1353	AT9	−1353	AT)
1354	AT:	−1354	AT*
1355	AT;	−1355	AT+
1356	AT<	−1356	AT,
1357	AT=	−1357	AT−
1358	AT>	−1358	AT.
1359	AT?	−1359	AT/
1360	AU0	−1360	AU^{SP}
1361	AU1	−1361	AU!
1362	AU2	−1362	AU"
1363	AU3	−1363	AU#
1364	AU4	−1364	AU$
1365	AU5	−1365	AU%
1366	AU6	−1366	AU&
1367	AU7	−1367	AU'
1368	AU8	−1368	AU(
1369	AU9	−1369	AU)
1370	AU:	−1370	AU*
1371	AU;	−1371	AU+
1372	AU<	−1372	AU,
1373	AU=	−1373	AU−
1374	AU>	−1374	AU.
1375	AU?	−1375	AU/
1376	AV0	−1376	AV^{SP}
1377	AV1	−1377	AV!
1378	AV2	−1378	AV"
1379	AV3	−1379	AV#
1380	AV4	−1380	AV$
1381	AV5	−1381	AV%
1382	AV6	−1382	AV&
1383	AV7	−1383	AV'
1384	AV8	−1384	AV(
1385	AV9	−1385	AV)
1386	AV:	−1386	AV*
1387	AV;	−1387	AV+
1388	AV<	−1388	AV,
1389	AV=	−1389	AV−
1390	AV>	−1390	AV.
1391	AV?	−1391	AV/
1392	AW0	−1392	AW^{SP}
1393	AW1	−1393	AW!
1394	AW2	−1394	AW"
1395	AW3	−1395	AW#
1396	AW4	−1396	AW$
1397	AW5	−1397	AW%
1398	AW6	−1398	AW&
1399	AW7	−1399	AW'
1400	AW8	−1400	AW(
1401	AW9	−1401	AW)
1402	AW:	−1402	AW*
1403	AW;	−1403	AW+
1404	AW<	−1404	AW,
1405	AW=	−1405	AW−
1406	AW>	−1406	AW.
1407	AW?	−1407	AW/
1408	AX0	−1408	AX^{SP}
1409	AX1	−1409	AX!
1410	AX2	−1410	AX"
1411	AX3	−1411	AX#
1412	AX4	−1412	AX$
1413	AX5	−1413	AX%
1414	AX6	−1414	AX&
1415	AX7	−1415	AX'
1416	AX8	−1416	AX(
1417	AX9	−1417	AX)
1418	AX:	−1418	AX*
1419	AX;	−1419	AX+
1420	AX<	−1420	AX,
1421	AX=	−1421	AX−
1422	AX>	−1422	AX.
1423	AX?	−1423	AX/
1424	AY0	−1424	AY^{SP}
1425	AY1	−1425	AY!
1426	AY2	−1426	AY"
1427	AY3	−1427	AY#
1428	AY4	−1428	AY$
1429	AY5	−1429	AY%
1430	AY6	−1430	AY&
1431	AY7	−1431	AY'
1432	AY8	−1432	AY(
1433	AY9	−1433	AY)
1434	AY:	−1434	AY*
1435	AY;	−1435	AY+
1436	AY<	−1436	AY,
1437	AY=	−1437	AY−
1438	AY>	−1438	AY.
1439	AY?	−1439	AY/
1440	AZ0	−1440	AZ^{SP}
1441	AZ1	−1441	AZ!
1442	AZ2	−1442	AZ"
1443	AZ3	−1443	AZ#
1444	AZ4	−1444	AZ$
1445	AZ5	−1445	AZ%
1446	AZ6	−1446	AZ&
1447	AZ7	−1447	AZ'
1448	AZ8	−1448	AZ(
1449	AZ9	−1449	AZ)
1450	AZ:	−1450	AZ*
1451	AZ;	−1451	AZ+
1452	AZ<	−1452	AZ,
1453	AZ=	−1453	AZ−
1454	AZ>	−1454	AZ.
1455	AZ?	−1455	AZ/
1456	A[0	−1456	A[^{SP}
1457	A[1	−1457	A[!
1458	A[2	−1458	A["
1459	A[3	−1459	A[#
1460	A[4	−1460	A[$
1461	A[5	−1461	A[%
1462	A[6	−1462	A[&
1463	A[7	−1463	A['
1464	A[8	−1464	A[(
1465	A[9	−1465	A[)
1466	A[:	−1466	A[*
1467	A[;	−1467	A[+
1468	A[<	−1468	A[,
1469	A[=	−1469	A[−
1470	A[>	−1470	A[.
1471	A[?	−1471	A[/
1472	A\0	−1472	A\$^{SP}$
1473	A\1	−1473	A\!
1474	A\2	−1474	A\"
1475	A\3	−1475	A\#
1476	A\4	−1476	A\$
1477	A\5	−1477	A\%
1478	A\6	−1478	A\&
1479	A\7	−1479	A\'
1480	A\8	−1480	A\(
1481	A\9	−1481	A\)
1482	A\:	−1482	A*
1483	A\;	−1483	A\+
1484	A\<	−1484	A\,
1485	A\=	−1485	A\−
1486	A\>	−1486	A\.
1487	A\?	−1487	A\/
1488	A]0	−1488	A]^{SP}
1489	A]1	−1489	A]!
1490	A]2	−1490	A]"
1491	A]3	−1491	A]#
1492	A]4	−1492	A]$
1493	A]5	−1493	A]%
1494	A]6	−1494	A]&
1495	A]7	−1495	A]'
1496	A]8	−1496	A](
1497	A]9	−1497	A])
1498	A]:	−1498	A]*
1499	A];	−1499	A]+

Table B-1 (cont)

REPRESENTING NUMBERS AS *INT* PARAMETERS

n	*int*: n	-n	*int*: -n	n	*int*: n	-n	*int*: -n	n	*int*: n	-n	*int*: -n
1500	A]<	−1500	A],	1550	A'>	−1550	A'.	1600	Ad0	−1600	Ad^{SP}
1501	A]=	−1501	A]−	1551	A'?	−1551	A'/	1601	Ad1	−1601	Ad!
1502	A]>	−1502	A].	1552	Aa0	−1552	Aa^{SP}	1602	Ad2	−1602	Ad"
1503	A]?	−1503	A]/	1553	Aa1	−1553	Aa!	1603	Ad3	−1603	Ad#
1504	A∧0	−1504	A∧^{SP}	1554	Aa2	−1554	Aa"	1604	Ad4	−1604	Ad$
1505	A∧1	−1505	A∧!	1555	Aa3	−1555	Aa#	1605	Ad5	−1605	Ad%
1506	A∧2	−1506	A∧"	1556	Aa4	−1556	Aa$	1606	Ad6	−1606	Ad&
1507	A∧3	−1507	A∧#	1557	Aa5	−1557	Aa%	1607	Ad7	−1607	Ad'
1508	A∧4	−1508	A∧$	1558	Aa6	−1558	Aa&	1608	Ad8	−1608	Ad(
1509	A∧5	−1509	A∧%	1559	Aa7	−1559	Aa'	1609	Ad9	−1609	Ad)
1510	A∧6	−1510	A∧&	1560	Aa8	−1560	Aa(	1610	Ad:	−1610	Ad*
1511	A∧7	−1511	A∧'	1561	Aa9	−1561	Aa)	1611	Ad;	−1611	Ad+
1512	A∧8	−1512	A∧(	1562	Aa:	−1562	Aa*	1612	Ad<	−1612	Ad,
1513	A∧9	−1513	A∧)	1563	Aa;	−1563	Aa+	1613	Ad=	−1613	Ad−
1514	A∧:	−1514	A∧*	1564	Aa<	−1564	Aa,	1614	Ad>	−1614	Ad.
1515	A∧;	−1515	A∧+	1565	Aa=	−1565	Aa−	1615	Ad?	−1615	Ad/
1516	A∧<	−1516	A∧,	1566	Aa>	−1566	Aa.	1616	Ae0	−1616	Ae^{SP}
1517	A∧=	−1517	A∧−	1567	Aa?	−1567	Aa/	1617	Ae1	−1617	Ae!
1518	A∧>	−1518	A∧.	1568	Ab0	−1568	Ab^{SP}	1618	Ae2	−1618	Ae"
1519	A∧?	−1519	A∧/	1569	Ab1	−1569	Ab!	1619	Ae3	−1619	Ae#
1520	A_0	−1520	A_^{SP}	1570	Ab2	−1570	Ab"	1620	Ae4	−1620	Ae$
1521	A_1	−1521	A_!	1571	Ab3	−1571	Ab#	1621	Ae5	−1621	Ae%
1522	A_2	−1522	A_"	1572	Ab4	−1572	Ab$	1622	Ae6	−1622	Ae&
1523	A_3	−1523	A_#	1573	Ab5	−1573	Ab%	1623	Ae7	−1623	Ae'
1524	A_4	−1524	A_$	1574	Ab6	−1574	Ab&	1624	Ae8	−1624	Ae(
1525	A_5	−1525	A_%	1575	Ab7	−1575	Ab'	1625	Ae9	−1625	Ae)
1526	A_6	−1526	A_&	1576	Ab8	−1576	Ab(	1626	Ae:	−1626	Ae*
1527	A_7	−1527	A_'	1577	Ab9	−1577	Ab)	1627	Ae;	−1627	Ae+
1528	A_8	−1528	A_(	1578	Ab:	−1578	Ab*	1628	Ae<	−1628	Ae,
1529	A_9	−1529	A_)	1579	Ab;	−1579	Ab+	1629	Ae=	−1629	Ae−
1530	A_:	−1530	A_*	1580	Ab<	−1580	Ab,	1630	Ae>	−1630	Ae.
1531	A_;	−1531	A_+	1581	Ab=	−1581	Ab−	1631	Ae?	−1631	Ae/
1532	A_<	−1532	A_,	1582	Ab>	−1582	Ab.	1632	Af0	−1632	Af^{SP}
1533	A_=	−1533	A_−	1583	Ab?	−1583	Ab/	1633	Af1	−1633	Af!
1534	A_>	−1534	A_.	1584	Ac0	−1584	Ac^{SP}	1634	Af2	−1634	Af"
1535	A_?	−1535	A_/	1585	Ac1	−1585	Ac!	1635	Af3	−1635	Af#
1536	A'0	−1536	A'^{SP}	1586	Ac2	−1586	Ac"	1636	Af4	−1636	Af$
1537	A'1	−1537	A'!	1587	Ac3	−1587	Ac#	1637	Af5	−1637	Af%
1538	A'2	−1538	A'"	1588	Ac4	−1588	Ac$	1638	Af6	−1638	Af&
1539	A'3	−1539	A'#	1589	Ac5	−1589	Ac%	1639	Af7	−1639	Af'
1540	A'4	−1540	A'$	1590	Ac6	−1590	Ac&	1640	Af8	−1640	Af(
1541	A'5	−1541	A'%	1591	Ac7	−1591	Ac'	1641	Af9	−1641	Af)
1542	A'6	−1542	A'&	1592	Ac8	−1592	Ac(	1642	Af:	−1642	Af*
1543	A'7	−1543	A'	1593	Ac9	−1593	Ac)	1643	Af;	−1643	Af+
1544	A'8	−1544	A'(	1594	Ac:	−1594	Ac*	1644	Af<	−1644	Af,
1545	A'9	−1545	A')	1595	Ac;	−1595	Ac+	1645	Af=	−1645	Af−
1546	A':	−1546	A'*	1596	Ac<	−1596	Ac,	1646	Af>	−1646	Af.
1547	A';	−1547	A'+	1597	Ac=	−1597	Ac−	1647	Af?	−1647	Af/
1548	A'<	−1548	A',	1598	Ac>	−1598	Ac.	1648	Ag0	−1648	Ag^{SP}
1549	A'=	−1549	A'−	1599	Ac?	−1599	Ac/	1649	Ag1	−1649	Ag!

Table B-1 (cont)

REPRESENTING NUMBERS AS *INT* PARAMETERS

n	*int*: n	-n	*int*: -n
1650	**Ag2**	−1650	**Ag"**
1651	**Ag3**	−1651	**Ag#**
1652	**Ag4**	−1652	**Ag$**
1653	**Ag5**	−1653	**Ag%**
1654	**Ag6**	−1654	**Ag&**
1655	**Ag7**	−1655	**Ag'**
1656	**Ag8**	−1656	**Ag(**
1657	**Ag9**	−1657	**Ag)**
1658	**Ag:**	−1658	**Ag***
1659	**Ag;**	−1659	**Ag+**
1660	**Ag<**	−1660	**Ag,**
1661	**Ag=**	−1661	**Ag−**
1662	**Ag>**	−1662	**Ag.**
1663	**Ag?**	−1663	**Ag/**
1664	**Ah0**	−1664	**Ah$_P$**
1665	**Ah1**	−1665	**Ah!**
1666	**Ah2**	−1666	**Ah"**
1667	**Ah3**	−1667	**Ah#**
1668	**Ah4**	−1668	**Ah$**
1669	**Ah5**	−1669	**Ah%**
1670	**Ah6**	−1670	**Ah&**
1671	**Ah7**	−1671	**Ah'**
1672	**Ah8**	−1672	**Ah(**
1673	**Ah9**	−1673	**Ah)**
1674	**Ah:**	−1674	**Ah***
1675	**Ah;**	−1675	**Ah+**
1676	**Ah<**	−1676	**Ah,**
1677	**Ah=**	−1677	**Ah−**
1678	**Ah>**	−1678	**Ah.**
1679	**Ah?**	−1679	**Ah/**
1680	**Ai0**	−1680	**Ai$_P$**
1681	**Ai1**	−1681	**Ai!**
1682	**Ai2**	−1682	**Ai"**
1683	**Ai3**	−1683	**Ai#**
1684	**Ai4**	−1684	**Ai$**
1685	**Ai5**	−1685	**Ai%**
1686	**Ai6**	−1686	**Ai&**
1687	**Ai7**	−1687	**Ai'**
1688	**Ai8**	−1688	**Ai(**
1689	**Ai9**	−1689	**Ai)**
1690	**Ai:**	−1690	**Ai***
1691	**Ai;**	−1691	**Ai+**
1692	**Ai<**	−1692	**Ai,**
1693	**Ai=**	−1693	**Ai−**
1694	**Ai>**	−1694	**Ai.**
1695	**Ai?**	−1695	**Ai/**
1696	**Aj0**	−1696	**Aj$_P$**
1697	**Aj1**	−1697	**Aj!**
1698	**Aj2**	−1698	**Aj"**
1699	**Aj3**	−1699	**Aj#**
1700	**Aj4**	−1700	**Aj$**
1701	**Aj5**	−1701	**Aj%**
1702	**Aj6**	−1702	**Aj&**
1703	**Aj7**	−1703	**Aj'**
1704	**Aj8**	−1704	**Aj(**
1705	**Aj9**	−1705	**Aj)**
1706	**Aj:**	−1706	**Aj***
1707	**Aj;**	−1707	**Aj+**
1708	**Aj<**	−1708	**Aj,**
1709	**Aj=**	−1709	**Aj−**
1710	**Aj>**	−1710	**Aj.**
1711	**Aj?**	−1711	**Aj/**
1712	**Ak0**	−1712	**Ak$_P$**
1713	**Ak1**	−1713	**Ak!**
1714	**Ak2**	−1714	**Ak"**
1715	**Ak3**	−1715	**Ak#**
1716	**Ak4**	−1716	**Ak$**
1717	**Ak5**	−1717	**Ak%**
1718	**Ak6**	−1718	**Ak&**
1719	**Ak7**	−1719	**Ak'**
1720	**Ak8**	−1720	**Ak(**
1721	**Ak9**	−1721	**Ak)**
1722	**Ak:**	−1722	**Ak***
1723	**Ak;**	−1723	**Ak+**
1724	**Ak<**	−1724	**Ak,**
1725	**Ak=**	−1725	**Ak−**
1726	**Ak>**	−1726	**Ak.**
1727	**Ak?**	−1727	**Ak/**
1728	**Al0**	−1728	**Al$_P$**
1729	**Al1**	−1729	**Al!**
1730	**Al2**	−1730	**Al"**
1731	**Al3**	−1731	**Al#**
1732	**Al4**	−1732	**Al$**
1733	**Al5**	−1733	**Al%**
1734	**Al6**	−1734	**Al&**
1735	**Al7**	−1735	**Al'**
1736	**Al8**	−1736	**Al(**
1737	**Al9**	−1737	**Al)**
1738	**Al:**	−1738	**Al***
1739	**Al;**	−1739	**Al+**
1740	**Al<**	−1740	**Al,**
1741	**Al=**	−1741	**Al−**
1742	**Al>**	−1742	**Al.**
1743	**Al?**	−1743	**Al/**
1744	**Am0**	−1744	**Am$_P$**
1745	**Am1**	−1745	**Am!**
1746	**Am2**	−1746	**Am"**
1747	**Am3**	−1747	**Am#**
1748	**Am4**	−1748	**Am$**
1749	**Am5**	−1749	**Am%**
1750	**Am6**	−1750	**Am&**
1751	**Am7**	−1751	**Am'**
1752	**Am8**	−1752	**Am(**
1753	**Am9**	−1753	**Am)**
1754	**Am:**	−1754	**Am***
1755	**Am;**	−1755	**Am+**
1756	**Am<**	−1756	**Am,**
1757	**Am=**	−1757	**Am−**
1758	**Am>**	−1758	**Am.**
1759	**Am?**	−1759	**Am/**
1760	**An0**	−1760	**An$_P$**
1761	**An1**	−1761	**An!**
1762	**An2**	−1762	**An"**
1763	**An3**	−1763	**An#**
1764	**An4**	−1764	**An$**
1765	**An5**	−1765	**An%**
1766	**An6**	−1766	**An&**
1767	**An7**	−1767	**An'**
1768	**An8**	−1768	**An(**
1769	**An9**	−1769	**An)**
1770	**An:**	−1770	**An***
1771	**An;**	−1771	**An+**
1772	**An<**	−1772	**An,**
1773	**An=**	−1773	**An−**
1774	**An>**	−1774	**An.**
1775	**An?**	−1775	**An/**
1776	**Ao0**	−1776	**Ao$_P$**
1777	**Ao1**	−1777	**Ao!**
1778	**Ao2**	−1778	**Ao"**
1779	**Ao3**	−1779	**Ao#**
1780	**Ao4**	−1780	**Ao$**
1781	**Ao5**	−1781	**Ao%**
1782	**Ao6**	−1782	**Ao&**
1783	**Ao7**	−1783	**Ao'**
1784	**Ao8**	−1784	**Ao(**
1785	**Ao9**	−1785	**Ao)**
1786	**Ao:**	−1786	**Ao***
1787	**Ao;**	−1787	**Ao+**
1788	**Ao<**	−1788	**Ao,**
1789	**Ao=**	−1789	**Ao−**
1790	**Ao>**	−1790	**Ao.**
1791	**Ao?**	−1791	**Ao/**
1792	**Ap0**	−1792	**Ap$_P$**
1793	**Ap1**	−1793	**Ap!**
1794	**Ap2**	−1794	**Ap"**
1795	**Ap3**	−1795	**Ap#**
1796	**Ap4**	−1796	**Ap$**
1797	**Ap5**	−1797	**Ap%**
1798	**Ap6**	−1798	**Ap&**
1799	**Ap7**	−1799	**Ap'**

Table B-1 (cont)

REPRESENTING NUMBERS AS *INT* PARAMETERS

n	*int*: n	-n	*int*: -n
1800	Ap8	−1800	Ap(
1801	Ap9	−1801	Ap)
1802	Ap:	−1802	Ap*
1803	Ap;	−1803	Ap+
1804	Ap<	−1804	Ap,
1805	Ap=	−1805	Ap−
1806	Ap>	−1806	Ap.
1807	Ap?	−1807	Ap/
1808	Aq0	−1808	Aq^{SP}
1809	Aq1	−1809	Aq!
1810	Aq2	−1810	Aq"
1811	Aq3	−1811	Aq#
1812	Aq4	−1812	Aq$
1813	Aq5	−1813	Aq%
1814	Aq6	−1814	Aq&
1815	Aq7	−1815	Aq'
1816	Aq8	−1816	Aq(
1817	Aq9	−1817	Aq)
1818	Aq:	−1818	Aq*
1819	Aq;	−1819	Aq+
1820	Aq<	−1820	Aq,
1821	Aq=	−1821	Aq−
1822	Aq>	−1822	Aq.
1823	Aq?	−1823	Aq/
1824	Ar0	−1824	Ar^{SP}
1825	Ar1	−1825	Ar!
1826	Ar2	−1826	Ar"
1827	Ar3	−1827	Ar#
1828	Ar4	−1828	Ar$
1829	Ar5	−1829	Ar%
1830	Ar6	−1830	Ar&
1831	Ar7	−1831	Ar'
1832	Ar8	−1832	Ar(
1833	Ar9	−1833	Ar)
1834	Ar:	−1834	Ar*
1835	Ar;	−1835	Ar+
1836	Ar<	−1836	Ar,
1837	Ar=	−1837	Ar−
1838	Ar>	−1838	Ar.
1839	Ar?	−1839	Ar/
1840	As0	−1840	As^{SP}
1841	As1	−1841	As!
1842	As2	−1842	As"
1843	As3	−1843	As#
1844	As4	−1844	As$
1845	As5	−1845	As%
1846	As6	−1846	As&
1847	As7	−1847	As'
1848	As8	−1848	As(
1849	As9	−1849	As)
1850	As:	−1850	As*
1851	As;	−1851	As+
1852	As<	−1852	As,
1853	As=	−1853	As−
1854	As>	−1854	As.
1855	As?	−1855	As/
1856	At0	−1856	At^{SP}
1857	At1	−1857	At!
1858	At2	−1858	At"
1859	At3	−1859	At#
1860	At4	−1860	At$
1861	At5	−1861	At%
1862	At6	−1862	At&
1863	At7	−1863	At'
1864	At8	−1864	At(
1865	At9	−1865	At)
1866	At:	−1866	At*
1867	At;	−1867	At+
1868	At<	−1868	At,
1869	At=	−1869	At−
1870	At>	−1870	At.
1871	At?	−1871	At/
1872	Au0	−1872	Au^{SP}
1873	Au1	−1873	Au!
1874	Au2	−1874	Au"
1875	Au3	−1875	Au#
1876	Au4	−1876	Au$
1877	Au5	−1877	Au%
1878	Au6	−1878	Au&
1879	Au7	−1879	Au'
1880	Au8	−1880	Au(
1881	Au9	−1881	Au)
1882	Au:	−1882	Au*
1883	Au;	−1883	Au+
1884	Au<	−1884	Au,
1885	Au=	−1885	Au−
1886	Au>	−1886	Au.
1887	Au?	−1887	Au/
1888	Av0	−1888	Av^{SP}
1889	Av1	−1889	Av!
1890	Av2	−1890	Av"
1891	Av3	−1891	Av#
1892	Av4	−1892	Av$
1893	Av5	−1893	Av%
1894	Av6	−1894	Av&
1895	Av7	−1895	Av'
1896	Av8	−1896	Av(
1897	Av9	−1897	Av)
1898	Av:	−1898	Av*
1899	Av;	−1899	Av+
1900	Av<	−1900	Av,
1901	Av=	−1901	Av−
1902	Av>	−1902	Av.
1903	Av?	−1903	Av/
1904	Aw0	−1904	Aw^{SP}
1905	Aw1	−1905	Aw!
1906	Aw2	−1906	Aw"
1907	Aw3	−1907	Aw#
1908	Aw4	−1908	Aw$
1909	Aw5	−1909	Aw%
1910	Aw6	−1910	Aw&
1911	Aw7	−1911	Aw'
1912	Aw8	−1912	Aw(
1913	Aw9	−1913	Aw)
1914	Aw:	−1914	Aw*
1915	Aw;	−1915	Aw+
1916	Aw<	−1916	Aw,
1917	Aw=	−1917	Aw−
1918	Aw>	−1918	Aw.
1919	Aw?	−1919	Aw/
1920	Ax0	−1920	Ax^{SP}
1921	Ax1	−1921	Ax!
1922	Ax2	−1922	Ax"
1923	Ax3	−1923	Ax#
1924	Ax4	−1924	Ax$
1925	Ax5	−1925	Ax%
1926	Ax6	−1926	Ax&
1927	Ax7	−1927	Ax'
1928	Ax8	−1928	Ax(
1929	Ax9	−1929	Ax)
1930	Ax:	−1930	Ax*
1931	Ax;	−1931	Ax+
1932	Ax<	−1932	Ax,
1933	Ax=	−1933	Ax−
1934	Ax>	−1934	Ax.
1935	Ax?	−1935	Ax/
1936	Ay0	−1936	Ay^{SP}
1937	Ay1	−1937	Ay!
1938	Ay2	−1938	Ay"
1939	Ay3	−1939	Ay#
1940	Ay4	−1940	Ay$
1941	Ay5	−1941	Ay%
1942	Ay6	−1942	Ay&
1943	Ay7	−1943	Ay'
1944	Ay8	−1944	Ay(
1945	Ay9	−1945	Ay)
1946	Ay:	−1946	Ay*
1947	Ay;	−1947	Ay+
1948	Ay<	−1948	Ay,
1949	Ay=	−1949	Ay−

Table B-1 (cont)

REPRESENTING NUMBERS AS *INT* PARAMETERS

n	*int*: n	-n	*int*: -n	n	*int*: n	-n	*int*: -n	n	*int*: n	-n	*int*: -n
1950	Ay>	−1950	Ay.	2000	A}0	−2000	A}^{S_P}	2050	B@2	−2050	B@"
1951	Ay?	−1951	Ay/	2001	A}1	−2001	A}!	2051	B@3	−2051	B@#
1952	Az0	−1952	Az^{S_P}	2002	A}2	−2002	A}"	2052	B@4	−2052	B@$
1953	Az1	−1953	Az!	2003	A}3	−2003	A}#	2053	B@5	−2053	B@%
1954	Az2	−1954	Az"	2004	A}4	−2004	A}$	2054	B@6	−2054	B@&
1955	Az3	−1955	Az#	2005	A}5	−2005	A}%	2055	B@7	−2055	B@'
1956	Az4	−1956	Az$	2006	A}6	−2006	A}&	2056	B@8	−2056	B@(
1957	Az5	−1957	Az%	2007	A}7	−2007	A}'	2057	B@9	−2057	B@)
1958	Az6	−1958	Az&	2008	A}8	−2008	A}(	2058	B@:	−2058	B@*
1959	Az7	−1959	Az'	2009	A}9	−2009	A})	2059	B@;	−2059	B@+
1960	Az8	−1960	Az(	2010	A}:	−2010	A}*	2060	B@<	−2060	B@,
1961	Az9	−1961	Az)	2011	A};	−2011	A}+	2061	B@=	−2061	B@−
1962	Az:	−1962	Az*	2012	A}<	−2012	A},	2062	B@>	−2062	B@.
1963	Az;	−1963	Az+	2013	A}=	−2013	A}−	2063	B@?	−2063	B@/
1964	Az<	−1964	Az,	2014	A}>	−2014	A}.	2064	BA0	−2064	BA^{S_P}
1965	Az=	−1965	Az−	2015	A}?	−2015	A}/	2065	BA1	−2065	BA!
1966	Az>	−1966	Az.	2016	A~0	−2016	A~^{S_P}	2066	BA2	−2066	BA"
1967	Az?	−1967	Az/	2017	A~1	−2017	A~!	2067	BA3	−2067	BA#
1968	A{0	−1968	A{^{S_P}	2018	A~2	−2018	A~"	2068	BA4	−2068	BA$
1969	A{1	−1969	A{!	2019	A~3	−2019	A~#	2069	BA5	−2069	BA%
1970	A{2	−1970	A{"	2020	A~4	−2020	A~$	2070	BA6	−2070	BA&
1971	A{3	−1971	A{#	2021	A~5	−2021	A~$	2071	BA7	−2071	BA'
1972	A{4	−1972	A{$	2022	A~6	−2022	A~&	2072	BA8	−2072	BA(
1973	A{5	−1973	A{%	2023	A~7	−2023	A~'	2073	BA9	−2073	BA)
1974	A{6	−1974	A{&	2024	A~8	−2024	A~(	2074	BA:	−2074	BA*
1975	A{7	−1975	A{'	2025	A~9	−2025	A~)	2075	BA;	−2075	BA+
1976	A{8	−1976	A{(	2026	A~:	−2026	A~*	2076	BA<	−2076	BA,
1977	A{9	−1977	A{)	2027	A~;	−2027	A~+	2077	BA=	−2077	BA−
1978	A{:	−1978	A{*	2028	A~<	−2028	A~,	2078	BA>	−2078	BA.
1979	A{;	−1979	A{+	2029	A~=	−2029	A~−	2079	BA?	−2079	BA/
1980	A{<	−1980	A{,	2030	A~>	−2030	A~.	2080	BB0	−2080	BB^{S_P}
1981	A{=	−1981	A{−	2031	A~?	−2031	A~/	2081	BB1	−2081	BB!
1982	A{>	−1982	A{.	2032	A^{D_T}0	−2032	A$^{D_T S_P}$	2082	BB2	−2082	BB"
1983	A{?	−1983	A{/	2033	A^{D_T}1	−2033	A^{D_T}!	2083	BB3	−2083	BB#
1984	A|0	−1984	A|^{S_P}	2034	A^{D_T}2	−2034	A^{D_T}"	2084	BB4	−2084	BB$
1985	A|1	−1985	A|!	2035	A^{D_T}3	−2035	A^{D_T}#	2085	BB5	−2085	BB%
1986	A|2	−1986	A|"	2036	A^{D_T}4	−2036	A^{D_T}$	2086	BB6	−2086	BB&
1987	A|3	−1987	A|#	2037	A^{D_T}5	−2037	A^{D_T}%	2087	BB7	−2087	BB'
1988	A|4	−1988	A|$	2038	A^{D_T}6	−2038	A^{D_T}&	2088	BB8	−2088	BB(
1989	A|5	−1989	A|%	2039	A^{D_T}7	−2039	A^{D_T}'	2089	BB9	−2089	BB)
1990	A|6	−1990	A|&	2040	A^{D_T}8	−2040	A^{D_T}(	2090	BB:	−2090	BB*
1991	A|7	−1991	A|'	2041	A^{D_T}9	−2041	A^{D_T})	2091	BB;	−2091	BB+
1992	A|8	−1992	A|(	2042	A^{D_T}:	−2042	A^{D_T}*	2092	BB<	−2092	BB,
1993	A|9	−1993	A|)	2043	A^{D_T};	−2043	A^{D_T}+	2093	BB=	−2093	BB−
1994	A|:	−1994	A|*	2044	A^{D_T}<	−2044	A^{D_T},	2094	BB>	−2094	BB.
1995	A|;	−1995	A|+	2045	A^{D_T}=	−2045	A^{D_T}−	2095	BB?	−2095	BB/
1996	A|<	−1996	A|,	2046	A^{D_T}>	−2046	A^{D_T}.	2096	BC0	−2096	BC^{S_P}
1997	A|=	−1997	A|−	2047	A^{D_T}?	−2047	A^{D_T}/	2097	BC1	−2097	BC!
1998	A|>	−1998	A|.	2048	B@0	−2048	B@^{S_P}	2098	BC2	−2098	BC"
1999	A|?	−1999	A|/	2049	B@1	−2049	B@!	2099	BC3	−2099	BC#

Table B-1 (cont)

REPRESENTING NUMBERS AS *INT* PARAMETERS

n	*int*: n	−n	*int*: −n
2100	BC4	−2100	BC$
2101	BC5	−2101	BC%
2102	BC6	−2102	BC&
2103	BC7	−2103	BC'
2104	BC8	−2104	BC(
2105	BC9	−2105	BC)
2106	BC:	−2106	BC*
2107	BC;	−2107	BC+
2108	BC<	−2108	BC,
2109	BC=	−2109	BC−
2110	BC>	−2110	BC.
2111	BC?	−2111	BC/
2112	BD0	−2112	BD^{SP}
2113	BD1	−2113	BD!
2114	BD2	−2114	BD"
2115	BD3	−2115	BD#
2116	BD4	−2116	BD$
2117	BD5	−2117	BD%
2118	BD6	−2118	BD&
2119	BD7	−2119	BD'
2120	BD8	−2120	BD(
2121	BD9	−2121	BD)
2122	BD:	−2122	BD*
2123	BD;	−2123	BD+
2124	BD<	−2124	BD,
2125	BD=	−2125	BD−
2126	BD>	−2126	BD.
2127	BD?	−2127	BD/
2128	BE0	−2128	BE^{SP}
2129	BE1	−2129	BE!
2130	BE2	−2130	BE"
2131	BE3	−2131	BE#
2132	BE4	−2132	BE$
2133	BE5	−2133	BE%
2134	BE6	−2134	BE&
2135	BE7	−2135	BE'
2136	BE8	−2136	BE(
2137	BE9	−2137	BE)
2138	BE:	−2138	BE*
2139	BE;	−2139	BE+
2140	BE<	−2140	BE,
2141	BE=	−2141	BE−
2142	BE>	−2142	BE.
2143	BE?	−2143	BE/
2144	BF0	−2144	BF^{SP}
2145	BF1	−2145	BF!
2146	BF2	−2146	BF"
2147	BF3	−2147	BF#
2148	BF4	−2148	BF$
2149	BF5	−2149	BF%
2150	BF6	−2150	BF&
2151	BF7	−2151	BF'
2152	BF8	−2152	BF(
2153	BF9	−2153	BF)
2154	BF:	−2154	BF*
2155	BF;	−2155	BF+
2156	BF<	−2156	BF,
2157	BF=	−2157	BF−
2158	BF>	−2158	BF.
2159	BF?	−2159	BF/
2160	BG0	−2160	BG^{SP}
2161	BG1	−2161	BG!
2162	BG2	−2162	BG"
2163	BG3	−2163	BG#
2164	BG4	−2164	BG$
2165	BG5	−2165	BG%
2166	BG6	−2166	BG&
2167	BG7	−2167	BG'
2168	BG8	−2168	BG(
2169	BG9	−2169	BG)
2170	BG:	−2170	BG*
2171	BG;	−2171	BG+
2172	BG<	−2172	BG,
2173	BG=	−2173	BG−
2174	BG>	−2174	BG.
2175	BG?	−2175	BG/
2176	BH0	−2176	BH^{SP}
2177	BH1	−2177	BH!
2178	BH2	−2178	BH"
2179	BH3	−2179	BH#
2180	BH4	−2180	BH$
2181	BH5	−2181	BH%
2182	BH6	−2182	BH&
2183	BH7	−2183	BH'
2184	BH8	−2184	BH(
2185	BH9	−2185	BH)
2186	BH:	−2186	BH*
2187	BH;	−2187	BH+
2188	BH<	−2188	BH,
2189	BH=	−2189	BH−
2190	BH>	−2190	BH.
2191	BH?	−2191	BH/
2192	BI0	−2192	BI^{SP}
2193	BI1	−2193	BI!
2194	BI2	−2194	BI"
2195	BI3	−2195	BI#
2196	BI4	−2196	BI$
2197	BI5	−2197	BI%
2198	BI6	−2198	BI&
2199	BI7	−2199	BI'
2200	BI8	−2200	BI(
2201	BI9	−2201	BI)
2202	BI:	−2202	BI*
2203	BI;	−2203	BI+
2204	BI<	−2204	BI,
2205	BI=	−2205	BI−
2206	BI>	−2206	BI.
2207	BI?	−2207	BI/
2208	BJ0	−2208	BJ^{SP}
2209	BJ1	−2209	BJ!
2210	BJ2	−2210	BJ"
2211	BJ3	−2211	BJ#
2212	BJ4	−2212	BJ$
2213	BJ5	−2213	BJ%
2214	BJ6	−2214	BJ&
2215	BJ7	−2215	BJ'
2216	BJ8	−2216	BJ(
2217	BJ9	−2217	BJ)
2218	BJ:	−2218	BJ*
2219	BJ;	−2219	BJ+
2220	BJ<	−2220	BJ,
2221	BJ=	−2221	BJ−
2222	BJ>	−2222	BJ.
2223	BJ?	−2223	BJ/
2224	BK0	−2224	BK^{SP}
2225	BK1	−2225	BK!
2226	BK2	−2226	BK"
2227	BK3	−2227	BK#
2228	BK4	−2228	BK$
2229	BK5	−2229	BK%
2230	BK6	−2230	BK&
2231	BK7	−2231	BK'
2232	BK8	−2232	BK(
2233	BK9	−2233	BK)
2234	BK:	−2234	BK*
2235	BK;	−2235	BK+
2236	BK<	−2236	BK,
2237	BK=	−2237	BK−
2238	BK>	−2238	BK.
2239	BK?	−2239	BK/
2240	BL0	−2240	BL^{SP}
2241	BL1	−2241	BL!
2242	BL2	−2242	BL"
2243	BL3	−2243	BL#
2244	BL4	−2244	BL$
2245	BL5	−2245	BL%
2246	BL6	−2246	BL&
2247	BL7	−2247	BL'
2248	BL8	−2248	BL(
2249	BL9	−2249	BL)

Table B-1 (cont)

REPRESENTING NUMBERS AS *INT* PARAMETERS

n	*int*: n	-n	*int*: -n
2250	BL:	−2250	BL*
2251	BL;	−2251	BL+
2252	BL<	−2252	BL,
2253	BL=	−2253	BL−
2254	BL>	−2254	BL.
2255	BL?	−2255	BL/
2256	BM0	−2256	BM^{S_P}
2257	BM1	−2257	BM!
2258	BM2	−2258	BM"
2259	BM3	−2259	BM#
2260	BM4	−2260	BM$
2261	BM5	−2261	BM%
2262	BM6	−2262	BM&
2263	BM7	−2263	BM'
2264	BM8	−2264	BM(
2265	BM9	−2265	BM)
2266	BM:	−2266	BM*
2267	BM;	−2267	BM+
2268	BM<	−2268	BM,
2269	BM=	−2269	BM−
2270	BM>	−2270	BM.
2271	BM?	−2271	BM/
2272	BN0	−2272	BN^{S_P}
2273	BN1	−2273	BN!
2274	BN2	−2274	BN"
2275	BN3	−2275	BN#
2276	BN4	−2276	BN$
2277	BN5	−2277	BN%
2278	BN6	−2278	BN&
2279	BN7	−2279	BN'
2280	BN8	−2280	BN(
2281	BN9	−2281	BN)
2282	BN:	−2282	BN*
2283	BN;	−2283	BN+
2284	BN<	−2284	BN,
2285	BN=	−2285	BN−
2286	BN>	−2286	BN.
2287	BN?	−2287	BN/
2288	BO0	−2288	BO^{S_P}
2289	BO1	−2289	BO!
2290	BO2	−2290	BO"
2291	BO3	−2291	BO#
2292	BO4	−2292	BO$
2293	BO5	−2293	BO%
2294	BO6	−2294	BO&
2295	BO7	−2295	BO'
2296	BO8	−2296	BO(
2297	BO9	−2297	BO)
2298	BO:	−2298	BO*
2299	BO;	−2299	BO+
2300	BO<	−2300	BO,
2301	BO=	−2301	BO−
2302	BO>	−2302	BO.
2303	BO?	−2303	BO/
2304	BP0	−2304	BP^{S_P}
2305	BP1	−2305	BP!
2306	BP2	−2306	BP"
2307	BP3	−2307	BP#
2308	BP4	−2308	BP$
2309	BP5	−2309	BP%
2310	BP6	−2310	BP&
2311	BP7	−2311	BP'
2312	BP8	−2312	BP(
2313	BP9	−2313	BP)
2314	BP:	−2314	BP*
2315	BP;	−2315	BP+
2316	BP<	−2316	BP,
2317	BP=	−2317	BP−
2318	BP>	−2318	BP.
2319	BP?	−2319	BP/
2320	BQ0	−2320	BQ^{S_P}
2321	BQ1	−2321	BQ!
2322	BQ2	−2322	BQ"
2323	BQ3	−2323	BQ#
2324	BQ4	−2324	BQ$
2325	BQ5	−2325	BQ%
2326	BQ6	−2326	BQ&
2327	BQ7	−2327	BQ'
2328	BQ8	−2328	BQ(
2329	BQ9	−2329	BQ)
2330	BQ:	−2330	BQ*
2331	BQ;	−2331	BQ+
2332	BQ<	−2332	BQ,
2333	BQ=	−2333	BQ−
2334	BQ>	−2334	BQ.
2335	BQ?	−2335	BQ/
2336	BR0	−2336	BR^{S_P}
2337	BR1	−2337	BR!
2338	BR2	−2338	BR"
2339	BR3	−2339	BR#
2340	BR4	−2340	BR$
2341	BR5	−2341	BR%
2342	BR6	−2342	BR&
2343	BR7	−2343	BR'
2344	BR8	−2344	BR(
2345	BR9	−2345	BR)
2346	BR:	−2346	BR*
2347	BR;	−2347	BR+
2348	BR<	−2348	BR,
2349	BR=	−2349	BR−
2350	BR>	−2350	BR.
2351	BR?	−2351	BR/
2352	BS0	−2352	BS^{S_P}
2353	BS1	−2353	BS!
2354	BS2	−2354	BS"
2355	BS3	−2355	BS#
2356	BS4	−2356	BS$
2357	BS5	−2357	BS%
2358	BS6	−2358	BS&
2359	BS7	−2359	BS'
2360	BS8	−2360	BS(
2361	BS9	−2361	BS)
2362	BS:	−2362	BS*
2363	BS;	−2363	BS+
2364	BS<	−2364	BS,
2365	BS=	−2365	BS−
2366	BS>	−2366	BS.
2367	BS?	−2367	BS/
2368	BT0	−2368	BT^{S_P}
2369	BT1	−2369	BT!
2370	BT2	−2370	BT"
2371	BT3	−2371	BT#
2372	BT4	−2372	BT$
2373	BT5	−2373	BT%
2374	BT6	−2374	BT&
2375	BT7	−2375	BT'
2376	BT8	−2376	BT(
2377	BT9	−2377	BT)
2378	BT:	−2378	BT*
2379	BT;	−2379	BT+
2380	BT<	−2380	BT,
2381	BT=	−2381	BT−
2382	BT>	−2382	BT.
2383	BT?	−2383	BT/
2384	BU0	−2384	BU^{S_P}
2385	BU1	−2385	BU!
2386	BU2	−2386	BU"
2387	BU3	−2387	BU#
2388	BU4	−2388	BU$
2389	BU5	−2389	BU%
2390	BU6	−2390	BU&
2391	BU7	−2391	BU'
2392	BU8	−2392	BU(
2393	BU9	−2393	BU)
2394	BU:	−2394	BU*
2395	BU;	−2395	BU+
2396	BU<	−2396	BU,
2397	BU=	−2397	BU−
2398	BU>	−2398	BU.
2399	BU?	−2399	BU/

Table B-1 (cont)

REPRESENTING NUMBERS AS *INT* PARAMETERS

n	*int*: n	-n	*int*: -n
2400	**BV0**	−2400	**BV^{SP}**
2401	**BV1**	−2401	**BV!**
2402	**BV2**	−2402	**BV"**
2403	**BV3**	−2403	**BV#**
2404	**BV4**	−2404	**BV$**
2405	**BV5**	−2405	**BV%**
2406	**BV6**	−2406	**BV&**
2407	**BV7**	−2407	**BV'**
2408	**BV8**	−2408	**BV(**
2409	**BV9**	−2409	**BV)**
2410	**BV:**	−2410	**BV***
2411	**BV;**	−2411	**BV+**
2412	**BV<**	−2412	**BV,**
2413	**BV=**	−2413	**BV−**
2414	**BV>**	−2414	**BV.**
2415	**BV?**	−2415	**BV/**
2416	**BW0**	−2416	**BW^{SP}**
2417	**BW1**	−2417	**BW!**
2418	**BW2**	−2418	**BW"**
2419	**BW3**	−2419	**BW#**
2420	**BW4**	−2420	**BW$**
2421	**BW5**	−2421	**BW%**
2422	**BW6**	−2422	**BW&**
2423	**BW7**	−2423	**BW'**
2424	**BW8**	−2424	**BW(**
2425	**BW9**	−2425	**BW)**
2426	**BW:**	−2426	**BW***
2427	**BW;**	−2427	**BW+**
2428	**BW<**	−2428	**BW,**
2429	**BW=**	−2429	**BW−**
2430	**BW>**	−2430	**BW.**
2431	**BW?**	−2431	**BW/**
2432	**BX0**	−2432	**BX^{SP}**
2433	**BX1**	−2433	**BX!**
2434	**BX2**	−2434	**BX"**
2435	**BX3**	−2435	**BX#**
2436	**BX4**	−2436	**BX$**
2437	**BX5**	−2437	**BX%**
2438	**BX6**	−2438	**BX&**
2439	**BX7**	−2439	**BX'**
2440	**BX8**	−2440	**BX(**
2441	**BX9**	−2441	**BX)**
2442	**BX:**	−2442	**BX***
2443	**BX;**	−2443	**BX+**
2444	**BX<**	−2444	**BX,**
2445	**BX=**	−2445	**BX−**
2446	**BX>**	−2446	**BX.**
2447	**BX?**	−2447	**BX/**
2448	**BY0**	−2448	**BY^{SP}**
2449	**BY1**	−2449	**BY!**
2450	**BY2**	−2450	**BY"**
2451	**BY3**	−2451	**BY#**
2452	**BY4**	−2452	**BY$**
2453	**BY5**	−2453	**BY%**
2454	**BY6**	−2454	**BY&**
2455	**BY7**	−2455	**BY'**
2456	**BY8**	−2456	**BY(**
2457	**BY9**	−2457	**BY)**
2458	**BY:**	−2458	**BY***
2459	**BY;**	−2459	**BY+**
2460	**BY<**	−2460	**BY,**
2461	**BY=**	−2461	**BY−**
2462	**BY>**	−2462	**BY.**
2463	**BY?**	−2463	**BY/**
2464	**BZ0**	−2464	**BZ^{SP}**
2465	**BZ1**	−2465	**BZ!**
2466	**BZ2**	−2466	**BZ"**
2467	**BZ3**	−2467	**BZ#**
2468	**BZ4**	−2468	**BZ$**
2469	**BZ5**	−2469	**BZ%**
2470	**BZ6**	−2470	**BZ&**
2471	**BZ7**	−2471	**BZ'**
2472	**BZ8**	−2472	**BZ(**
2473	**BZ9**	−2473	**BZ)**
2474	**BZ:**	−2474	**BZ***
2475	**BZ;**	−2475	**BZ+**
2476	**BZ<**	−2476	**BZ,**
2477	**BZ=**	−2477	**BZ−**
2478	**BZ>**	−2478	**BZ.**
2479	**BZ?**	−2479	**BZ/**
2480	**B[0**	−2480	**B[^{SP}**
2481	**B[1**	−2481	**B[!**
2482	**B[2**	−2482	**B["**
2483	**B[3**	−2483	**B[#**
2484	**B[4**	−2484	**B[$**
2485	**B[5**	−2485	**B[%**
2486	**B[6**	−2486	**B[&**
2487	**B[7**	−2487	**B['**
2488	**B[8**	−2488	**B[(**
2489	**B[9**	−2489	**B[)**
2490	**B[:**	−2490	**B[***
2491	**B[;**	−2491	**B[+**
2492	**B[<**	−2492	**B[,**
2493	**B[=**	−2493	**B[−**
2494	**B[>**	−2494	**B[.**
2495	**B[?**	−2495	**B[/**
2496	**B\0**	−2496	**B\$^{SP}$**
2497	**B\1**	−2497	**B\!**
2498	**B\2**	−2498	**B\"**
2499	**B\3**	−2499	**B\#**
2500	**B\4**	−2500	**B\$**
2501	**B\5**	−2501	**B\%**
2502	**B\6**	−2502	**B\&**
2503	**B\7**	−2503	**B\'**
2504	**B\8**	−2504	**B\(**
2505	**B\9**	−2505	**B\)**
2506	**B\:**	−2506	**B***
2507	**B\;**	−2507	**B\+**
2508	**B\<**	−2508	**B\,**
2509	**B\=**	−2509	**B\−**
2510	**B\>**	−2510	**B\.**
2511	**B\?**	−2511	**B\/**
2512	**B]0**	−2512	**B]^{SP}**
2513	**B]1**	−2513	**B]!**
2514	**B]2**	−2514	**B]"**
2515	**B]3**	−2515	**B]#**
2516	**B]4**	−2516	**B]$**
2517	**B]5**	−2517	**B]%**
2518	**B]6**	−2518	**B]&**
2519	**B]7**	−2519	**B]'**
2520	**B]8**	−2520	**B](**
2521	**B]9**	−2521	**B])**
2522	**B]:**	−2522	**B]***
2523	**B];**	−2523	**B]+**
2524	**B]<**	−2524	**B],**
2525	**B]=**	−2525	**B]−**
2526	**B]>**	−2526	**B].**
2527	**B]?**	−2527	**B]/**
2528	**B∧0**	−2528	**B∧^{SP}**
2529	**B∧1**	−2529	**B∧!**
2530	**B∧2**	−2530	**B∧"**
2531	**B∧3**	−2531	**B∧#**
2532	**B∧4**	−2532	**B∧$**
2533	**B∧5**	−2533	**B∧%**
2534	**B∧6**	−2534	**B∧&**
2535	**B∧7**	−2535	**B∧'**
2536	**B∧8**	−2536	**B∧(**
2537	**B∧9**	−2537	**B∧)**
2538	**B∧:**	−2538	**B∧***
2539	**B∧;**	−2539	**B∧+**
2540	**B∧<**	−2540	**B∧,**
2541	**B∧=**	−2541	**B∧−**
2542	**B∧>**	−2542	**B∧.**
2543	**B∧?**	−2543	**B∧/**
2544	**B_0**	−2544	**B_^{SP}**
2545	**B_1**	−2545	**B_!**
2546	**B_2**	−2546	**B_"**
2547	**B_3**	−2547	**B_#**
2548	**B_4**	−2548	**B_$**
2549	**B_5**	−2549	**B_%**

Table B-1 (cont)

REPRESENTING NUMBERS AS *INT* PARAMETERS

n	*int*: n	-n	*int*: -n
2550	B_6	−2550	B_&
2551	B_7	−2551	B_'
2552	B_8	−2552	B_(
2553	B_9	−2553	B_)
2554	B_:	−2554	B_*
2555	B_;	−2555	B_+
2556	B_<	−2556	B_,
2557	B_=	−2557	B_−
2558	B_>	−2558	B_.
2559	B_?	−2559	B_/
2560	B'0	−2560	B'^{SP}
2561	B'1	−2561	B'!
2562	B'2	−2562	B'"
2563	B'3	−2563	B'#
2564	B'4	−2564	B'$
2565	B'5	−2565	B'%
2566	B'6	−2566	B'&
2567	B'7	−2567	B'
2568	B'8	−2568	B'(
2569	B'9	−2569	B')
2570	B':	−2570	B'*
2571	B';	−2571	B'+
2572	B'<	−2572	B',
2573	B'=	−2573	B'−
2574	B'>	−2574	B'.
2575	B'?	−2575	B'/
2576	Ba0	−2576	Ba^{SP}
2577	Ba1	−2577	Ba!
2578	Ba2	−2578	Ba"
2579	Ba3	−2579	Ba#
2580	Ba4	−2580	Ba$
2581	Ba5	−2581	Ba%
2582	Ba6	−2582	Ba&
2583	Ba7	−2583	Ba'
2584	Ba8	−2584	Ba(
2585	Ba9	−2585	Ba)
2586	Ba:	−2586	Ba*
2587	Ba;	−2587	Ba+
2588	Ba<	−2588	Ba,
2589	Ba=	−2589	Ba−
2590	Ba>	−2590	Ba.
2591	Ba?	−2591	Ba/
2592	Bb0	−2592	Bb^{SP}
2593	Bb1	−2593	Bb!
2594	Bb2	−2594	Bb"
2595	Bb3	−2595	Bb#
2596	Bb4	−2596	Bb$
2597	Bb5	−2597	Bb%
2598	Bb6	−2598	Bb&
2599	Bb7	−2599	Bb'
2600	Bb8	−2600	Bb(
2601	Bb9	−2601	Bb)
2602	Bb:	−2602	Bb*
2603	Bb;	−2603	Bb+
2604	Bb<	−2604	Bb,
2605	Bb=	−2605	Bb−
2606	Bb>	−2606	Bb.
2607	Bb?	−2607	Bb/
2608	Bc0	−2608	Bc^{SP}
2609	Bc1	−2609	Bc!
2610	Bc2	−2610	Bc"
2611	Bc3	−2611	Bc#
2612	Bc4	−2612	Bc$
2613	Bc5	−2613	Bc%
2614	Bc6	−2614	Bc&
2615	Bc7	−2615	Bc'
2616	Bc8	−2616	Bc(
2617	Bc9	−2617	Bc)
2618	Bc:	−2618	Bc*
2619	Bc;	−2619	Bc+
2620	Bc<	−2620	Bc,
2621	Bc=	−2621	Bc−
2622	Bc>	−2622	Bc.
2623	Bc?	−2623	Bc/
2624	Bd0	−2624	Bd^{SP}
2625	Bd1	−2625	Bd!
2626	Bd2	−2626	Bd"
2627	Bd3	−2627	Bd#
2628	Bd4	−2628	Bd$
2629	Bd5	−2629	Bd%
2630	Bd6	−2630	Bd&
2631	Bd7	−2631	Bd'
2632	Bd8	−2632	Bd(
2633	Bd9	−2633	Bd)
2634	Bd:	−2634	Bd*
2635	Bd;	−2635	Bd+
2636	Bd<	−2636	Bd,
2637	Bd=	−2637	Bd−
2638	Bd>	−2638	Bd.
2639	Bd?	−2639	Bd/
2640	Be0	−2640	Be^{SP}
2641	Be1	−2641	Be!
2642	Be2	−2642	Be"
2643	Be3	−2643	Be#
2644	Be4	−2644	Be$
2645	Be5	−2645	Be%
2646	Be6	−2646	Be&
2647	Be7	−2647	Be'
2648	Be8	−2648	Be(
2649	Be9	−2649	Be)
2650	Be:	−2650	Be*
2651	Be;	−2651	Be+
2652	Be<	−2652	Be,
2653	Be=	−2653	Be−
2654	Be>	−2654	Be.
2655	Be?	−2655	Be/
2656	Bf0	−2656	Bf^{SP}
2657	Bf1	−2657	Bf!
2658	Bf2	−2658	Bf"
2659	Bf3	−2659	Bf#
2660	Bf4	−2660	Bf$
2661	Bf5	−2661	Bf%
2662	Bf6	−2662	Bf&
2663	Bf7	−2663	Bf'
2664	Bf8	−2664	Bf(
2665	Bf9	−2665	Bf)
2666	Bf:	−2666	Bf*
2667	Bf;	−2667	Bf+
2668	Bf<	−2668	Bf,
2669	Bf=	−2669	Bf−
2670	Bf>	−2670	Bf.
2671	Bf?	−2671	Bf/
2672	Bg0	−2672	Bg^{SP}
2673	Bg1	−2673	Bg!
2674	Bg2	−2674	Bg"
2675	Bg3	−2675	Bg#
2676	Bg4	−2676	Bg$
2677	Bg5	−2677	Bg%
2678	Bg6	−2678	Bg&
2679	Bg7	−2679	Bg'
2680	Bg8	−2680	Bg(
2681	Bg9	−2681	Bg)
2682	Bg:	−2682	Bg*
2683	Bg;	−2683	Bg+
2684	Bg<	−2684	Bg,
2685	Bg=	−2685	Bg−
2686	Bg>	−2686	Bg.
2687	Bg?	−2687	Bg/
2688	Bh0	−2688	Bh^{SP}
2689	Bh1	−2689	Bh!
2690	Bh2	−2690	Bh"
2691	Bh3	−2691	Bh#
2692	Bh4	−2692	Bh$
2693	Bh5	−2693	Bh%
2694	Bh6	−2694	Bh&
2695	Bh7	−2695	Bh'
2696	Bh8	−2696	Bh(
2697	Bh9	−2697	Bh)
2698	Bh:	−2698	Bh*
2699	Bh;	−2699	Bh+

Table B-1 (cont)

REPRESENTING NUMBERS AS *INT* PARAMETERS

n	*int*: n	-n	*int*: -n
2700	Bh<	−2700	Bh,
2701	Bh=	−2701	Bh−
2702	Bh>	−2702	Bh.
2703	Bh?	−2703	Bh/
2704	Bi0	−2704	Bi^{SP}
2705	Bi1	−2705	Bi!
2706	Bi2	−2706	Bi"
2707	Bi3	−2707	Bi#
2708	Bi4	−2708	Bi$
2709	Bi5	−2709	Bi%
2710	Bi6	−2710	Bi&
2711	Bi7	−2711	Bi'
2712	Bi8	−2712	Bi(
2713	Bi9	−2713	Bi)
2714	Bi:	−2714	Bi*
2715	Bi;	−2715	Bi+
2716	Bi<	−2716	Bi,
2717	Bi=	−2717	Bi−
2718	Bi>	−2718	Bi.
2719	Bi?	−2719	Bi/
2720	Bj0	−2720	Bj^{SP}
2721	Bj1	−2721	Bj!
2722	Bj2	−2722	Bj"
2723	Bj3	−2723	Bj#
2724	Bj4	−2724	Bj$
2725	Bj5	−2725	Bj%
2726	Bj6	−2726	Bj&
2727	Bj7	−2727	Bj'
2728	Bj8	−2728	Bj(
2729	Bj9	−2729	Bj)
2730	Bj:	−2730	Bj*
2731	Bj;	−2731	Bj+
2732	Bj<	−2732	Bj,
2733	Bj=	−2733	Bj−
2734	Bj>	−2734	Bj.
2735	Bj?	−2735	Bj/
2736	Bk0	−2736	Bk^{SP}
2737	Bk1	−2737	Bk!
2738	Bk2	−2738	Bk"
2739	Bk3	−2739	Bk#
2740	Bk4	−2740	Bk$
2741	Bk5	−2741	Bk%
2742	Bk6	−2742	Bk&
2743	Bk7	−2743	Bk'
2744	Bk8	−2744	Bk(
2745	Bk9	−2745	Bk)
2746	Bk:	−2746	Bk*
2747	Bk;	−2747	Bk+
2748	Bk<	−2748	Bk,
2749	Bk=	−2749	Bk−
2750	Bk>	−2750	Bk.
2751	Bk?	−2751	Bk/
2752	Bl0	−2752	Bl^{SP}
2753	Bl1	−2753	Bl!
2754	Bl2	−2754	Bl"
2755	Bl3	−2755	Bl#
2756	Bl4	−2756	Bl$
2757	Bl5	−2757	Bl%
2758	Bl6	−2758	Bl&
2759	Bl7	−2759	Bl'
2760	Bl8	−2760	Bl(
2761	Bl9	−2761	Bl)
2762	Bl:	−2762	Bl*
2763	Bl;	−2763	Bl+
2764	Bl<	−2764	Bl,
2765	Bl=	−2765	Bl−
2766	Bl>	−2766	Bl.
2767	Bl?	−2767	Bl/
2768	Bm0	−2768	Bm^{SP}
2769	Bm1	−2769	Bm!
2770	Bm2	−2770	Bm"
2771	Bm3	−2771	Bm#
2772	Bm4	−2772	Bm$
2773	Bm5	−2773	Bm%
2774	Bm6	−2774	Bm&
2775	Bm7	−2775	Bm'
2776	Bm8	−2776	Bm(
2777	Bm9	−2777	Bm)
2778	Bm:	−2778	Bm*
2779	Bm;	−2779	Bm+
2780	Bm<	−2780	Bm,
2781	Bm=	−2781	Bm−
2782	Bm>	−2782	Bm.
2783	Bm?	−2783	Bm/
2784	Bn0	−2784	Bn^{SP}
2785	Bn1	−2785	Bn!
2786	Bn2	−2786	Bn"
2787	Bn3	−2787	Bn#
2788	Bn4	−2788	Bn$
2789	Bn5	−2789	Bn%
2790	Bn6	−2790	Bn&
2791	Bn7	−2791	Bn'
2792	Bn8	−2792	Bn(
2793	Bn9	−2793	Bn)
2794	Bn:	−2794	Bn*
2795	Bn;	−2795	Bn+
2796	Bn<	−2796	Bn,
2797	Bn=	−2797	Bn−
2798	Bn>	−2798	Bn.
2799	Bn?	−2799	Bn/
2800	Bo0	−2800	Bo^{SP}
2801	Bo1	−2801	Bo!
2802	Bo2	−2802	Bo"
2803	Bo3	−2803	Bo#
2804	Bo4	−2804	Bo$
2805	Bo5	−2805	Bo%
2806	Bo6	−2806	Bo&
2807	Bo7	−2807	Bo'
2808	Bo8	−2808	Bo(
2809	Bo9	−2809	Bo)
2810	Bo:	−2810	Bo*
2811	Bo;	−2811	Bo+
2812	Bo<	−2812	Bo,
2813	Bo=	−2813	Bo−
2814	Bo>	−2814	Bo.
2815	Bo?	−2815	Bo/
2816	Bp0	−2816	Bp^{SP}
2817	Bp1	−2817	Bp!
2818	Bp2	−2818	Bp"
2819	Bp3	−2819	Bp#
2820	Bp4	−2820	Bp$
2821	Bp5	−2821	Bp%
2822	Bp6	−2822	Bp&
2823	Bp7	−2823	Bp'
2824	Bp8	−2824	Bp(
2825	Bp9	−2825	Bp)
2826	Bp:	−2826	Bp*
2827	Bp;	−2827	Bp+
2828	Bp<	−2828	Bp,
2829	Bp=	−2829	Bp−
2830	Bp>	−2830	Bp.
2831	Bp?	−2831	Bp/
2832	Bq0	−2832	Bq^{SP}
2833	Bq1	−2833	Bq!
2834	Bq2	−2834	Bq"
2835	Bq3	−2835	Bq#
2836	Bq4	−2836	Bq$
2837	Bq5	−2837	Bq%
2838	Bq6	−2838	Bq&
2839	Bq7	−2839	Bq'
2840	Bq8	−2840	Bq(
2841	Bq9	−2841	Bq)
2842	Bq:	−2842	Bq*
2843	Bq;	−2843	Bq+
2844	Bq<	−2844	Bq,
2845	Bq=	−2845	Bq−
2846	Bq>	−2846	Bq.
2847	Bq?	−2847	Bq/
2848	Br0	−2848	Br^{SP}
2849	Br1	−2849	Br!

Table B-1 (cont)

REPRESENTING NUMBERS AS *INT* PARAMETERS

n	*int*: n	-n	*int*: -n
2850	**Br2**	−2850	**Br"**
2851	**Br3**	−2851	**Br#**
2852	**Br4**	−2852	**Br$**
2853	**Br5**	−2853	**Br%**
2854	**Br6**	−2854	**Br&**
2855	**Br7**	−2855	**Br'**
2856	**Br8**	−2856	**Br(**
2857	**Br9**	−2857	**Br)**
2858	**Br:**	−2858	**Br***
2859	**Br;**	−2859	**Br+**
2860	**Br<**	−2860	**Br,**
2861	**Br=**	−2861	**Br−**
2862	**Br>**	−2862	**Br.**
2863	**Br?**	−2863	**Br/**
2864	**Bs0**	−2864	**Bs^{SP}**
2865	**Bs1**	−2865	**Bs!**
2866	**Bs2**	−2866	**Bs"**
2867	**Bs3**	−2867	**Bs#**
2868	**Bs4**	−2868	**Bs$**
2869	**Bs5**	−2869	**Bs%**
2870	**Bs6**	−2870	**Bs&**
2871	**Bs7**	−2871	**Bs'**
2872	**Bs8**	−2872	**Bs(**
2873	**Bs9**	−2873	**Bs)**
2874	**Bs:**	−2874	**Bs***
2875	**Bs;**	−2875	**Bs+**
2876	**Bs<**	−2876	**Bs,**
2877	**Bs=**	−2877	**Bs−**
2878	**Bs>**	−2878	**Bs.**
2879	**Bs?**	−2879	**Bs/**
2880	**Bt0**	−2880	**Bt^{SP}**
2881	**Bt1**	−2881	**Bt!**
2882	**Bt2**	−2882	**Bt"**
2883	**Bt3**	−2883	**Bt#**
2884	**Bt4**	−2884	**Bt$**
2885	**Bt5**	−2885	**Bt%**
2886	**Bt6**	−2886	**Bt&**
2887	**Bt7**	−2887	**Bt'**
2888	**Bt8**	−2888	**Bt(**
2889	**Bt9**	−2889	**Bt)**
2890	**Bt:**	−2890	**Bt***
2891	**Bt;**	−2891	**Bt+**
2892	**Bt<**	−2892	**Bt,**
2893	**Bt=**	−2893	**Bt−**
2894	**Bt>**	−2894	**Bt.**
2895	**Bt?**	−2895	**Bt/**
2896	**Bu0**	−2896	**Bu^{SP}**
2897	**Bu1**	−2897	**Bu!**
2898	**Bu2**	−2898	**Bu"**
2899	**Bu3**	−2899	**Bu#**
2900	**Bu4**	−2900	**Bu$**
2901	**Bu5**	−2901	**Bu%**
2902	**Bu6**	−2902	**Bu&**
2903	**Bu7**	−2903	**Bu'**
2904	**Bu8**	−2904	**Bu(**
2905	**Bu9**	−2905	**Bu)**
2906	**Bu:**	−2906	**Bu***
2907	**Bu;**	−2907	**Bu+**
2908	**Bu<**	−2908	**Bu,**
2909	**Bu=**	−2909	**Bu−**
2910	**Bu>**	−2910	**Bu.**
2911	**Bu?**	−2911	**Bu/**
2912	**Bv0**	−2912	**Bv^{SP}**
2913	**Bv1**	−2913	**Bv!**
2914	**Bv2**	−2914	**Bv"**
2915	**Bv3**	−2915	**Bv#**
2916	**Bv4**	−2916	**Bv$**
2917	**Bv5**	−2917	**Bv%**
2918	**Bv6**	−2918	**Bv&**
2919	**Bv7**	−2919	**Bv'**
2920	**Bv8**	−2920	**Bv(**
2921	**Bv9**	−2921	**Bv)**
2922	**Bv:**	−2922	**Bv***
2923	**Bv;**	−2923	**Bv+**
2924	**Bv<**	−2924	**Bv,**
2925	**Bv=**	−2925	**Bv−**
2926	**Bv>**	−2926	**Bv.**
2927	**Bv?**	−2927	**Bv/**
2928	**Bw0**	−2928	**Bw^{SP}**
2929	**Bw1**	−2929	**Bw!**
2930	**Bw2**	−2930	**Bw"**
2931	**Bw3**	−2931	**Bw#**
2932	**Bw4**	−2932	**Bw$**
2933	**Bw5**	−2933	**Bw%**
2934	**Bw6**	−2934	**Bw&**
2935	**Bw7**	−2935	**Bw'**
2936	**Bw8**	−2936	**Bw(**
2937	**Bw9**	−2937	**Bw)**
2938	**Bw:**	−2938	**Bw***
2939	**Bw;**	−2939	**Bw+**
2940	**Bw<**	−2940	**Bw,**
2941	**Bw=**	−2941	**Bw−**
2942	**Bw>**	−2942	**Bw.**
2943	**Bw?**	−2943	**Bw/**
2944	**Bx0**	−2944	**Bx^{SP}**
2945	**Bx1**	−2945	**Bx!**
2946	**Bx2**	−2946	**Bx"**
2947	**Bx3**	−2947	**Bx#**
2948	**Bx4**	−2948	**Bx$**
2949	**Bx5**	−2949	**Bx%**
2950	**Bx6**	−2950	**Bx&**
2951	**Bx7**	−2951	**Bx'**
2952	**Bx8**	−2952	**Bx(**
2953	**Bx9**	−2953	**Bx)**
2954	**Bx:**	−2954	**Bx***
2955	**Bx;**	−2955	**Bx+**
2956	**Bx<**	−2956	**Bx,**
2957	**Bx=**	−2957	**Bx−**
2958	**Bx>**	−2958	**Bx.**
2959	**Bx?**	−2959	**Bx/**
2960	**By0**	−2960	**By^{SP}**
2961	**By1**	−2961	**By!**
2962	**By2**	−2962	**By"**
2963	**By3**	−2963	**By#**
2964	**By4**	−2964	**By$**
2965	**By5**	−2965	**By%**
2966	**By6**	−2966	**By&**
2967	**By7**	−2967	**By'**
2968	**By8**	−2968	**By(**
2969	**By9**	−2969	**By)**
2970	**By:**	−2970	**By***
2971	**By;**	−2971	**By+**
2972	**By<**	−2972	**By,**
2973	**By=**	−2973	**By-**
2974	**By>**	−2974	**By.**
2975	**By?**	−2975	**By/**
2976	**Bz0**	−2976	**Bz<>**
2977	**Bz1**	−2977	**Bz!**
2978	**Bz2**	−2978	**Bz"**
2979	**Bz3**	−2979	**Bz#**
2980	**Bz4**	−2980	**Bz$**
2981	**Bz5**	−2981	**Bz%**
2982	**Bz6**	−2982	**Bz&**
2983	**Bz7**	−2983	**Bz'**
2984	**Bz8**	−2984	**Bz(**
2985	**Bz9**	−2985	**Bz)**
2986	**Bz:**	−2986	**Bz***
2987	**Bz;**	−2987	**Bz+**
2988	**Bz<**	−2988	**Bz,**
2989	**Bz=**	−2989	**Bz−**
2990	**Bz>**	−2990	**Bz.**
2991	**Bz?**	−2991	**Bz/**
2992	**B{0**	−2992	**B{^{SP}**
2993	**B{1**	−2993	**B{!**
2994	**B{2**	−2994	**B{"**
2995	**B{3**	−2995	**B{#**
2996	**B{4**	−2996	**B{$**
2997	**B{5**	−2997	**B{%**
2998	**B{6**	−2998	**B{&**
2999	**B{7**	−2999	**B{'**

Table B-1 (cont)

REPRESENTING NUMBERS AS *INT* PARAMETERS

n	*int*: n	-n	*int*: -n
3000	B{8	−3000	B{(
3001	B{9	−3001	B{)
3002	B{:	−3002	B{*
3003	B{;	−3003	B{+
3004	B{<	−3004	B{,
3005	B{=	−3005	B{−
3006	B{>	−3006	B{.
3007	B{?	−3007	B{/
3008	B|0	−3008	B|SP
3009	B|1	−3009	B|!
3010	B|2	−3010	B|"
3011	B|3	−3011	B|#
3012	B|4	−3012	B|$
3013	B|5	−3013	B|%
3014	B|6	−3014	B|&
3015	B|7	−3015	B|'
3016	B|8	−3016	B|(
3017	B|9	−3017	B|)
3018	B|:	−3018	B|*
3019	B|;	−3019	B|+
3020	B|<	−3020	B|,
3021	B|=	−3021	B|−
3022	B|>	−3022	B|.
3023	B|?	−3023	B|/
3024	B}0	−3024	B}SP
3025	B}1	−3025	B}!
3026	B}2	−3026	B}"
3027	B}3	−3027	B}#
3028	B}4	−3028	B}$
3029	B}5	−3029	B}%
3030	B}6	−3030	B}&
3031	B}7	−3031	B}'
3032	B}8	−3032	B}(
3033	B}9	−3033	B})
3034	B}:	−3034	B}*
3035	B};	−3035	B}+
3036	B}<	−3036	B},
3037	B}=	−3037	B}−
3038	B}>	−3038	B}.
3039	B}?	−3039	B}/
3040	B~0	−3040	B~SP
3041	B~1	−3041	B~!
3042	B~12	−3042	B~"
3043	B~13	−3043	B~1#
3044	B~14	−3044	B~1$
3045	B~15	−3045	B~1%
3046	B~16	−3046	B~1&
3047	B~17	−3047	B~1'
3048	B~18	−3048	B~1(
3049	B~19	−3049	B~1)

n	*int*: n	-n	*int*: -n
3050	B~1:	−3050	B~1*
3051	B~1;	−3051	B~1+
3052	B~1<	−3052	B~1,
3053	B~1=	−3053	B~1−
3054	B~1>	−3054	B~1.
3055	B~1?	−3055	B~1/
3056	BDT0	−3056	BDTSP
3057	BDT1	−3057	BDT!
3058	BDT2	−3058	BDT"
3059	BDT3	−3059	BDT#
3060	BDT4	−3060	BDT$
3061	BDT5	−3061	BDT%
3062	BDT6	−3062	BDT&
3063	BDT7	−3063	BDT'
3064	BDT8	−3064	BDT(
3065	BDT9	−3065	BDT)
3066	BDT:	−3066	BDT*
3067	BDT;	−3067	BDT+
3068	BDT<	−3068	BDT,
3069	BDT=	−3069	BDT−
3070	BDT>	−3070	BDT.
3071	BDT?	−3071	BDT/
3072	C@0	−3072	C@SP
3073	C@1	−3073	C@!
3074	C@2	−3074	C@"
3075	C@3	−3075	C@#
3076	C@4	−3076	C@$
3077	C@5	−3077	C@%
3078	C@6	−3078	C@&
3079	C@7	−3079	C@'
3080	C@8	−3080	C@(
3081	C@9	−3081	C@)
3082	C@:	−3082	C@*
3083	C@;	−3083	C@+
3084	C@<	−3084	C@,
3085	C@=	−3085	C@−
3086	C@>	−3086	C@.
3087	C@?	−3087	C@/
3088	CA0	−3088	CASP
3089	CA1	−3089	CA!
3090	CA2	−3090	CA"
3091	CA3	−3091	CA#
3092	CA4	−3092	CA$
3093	CA5	−3093	CA%
3094	CA6	−3094	CA&
3095	CA7	−3095	CA'
3096	CA8	−3096	CA(
3097	CA9	−3097	CA)
3098	CA:	−3098	CA*
3099	CA;	−3099	CA+

n	*int*: n	-n	*int*: -n
3100	CA<	−3100	CA,
3101	CA=	−3101	CA−
3102	CA>	−3102	CA.
3103	CA?	−3103	CA/
3104	CB0	−3104	CBSP
3105	CB1	−3105	CB!
3106	CB2	−3106	CB"
3107	CB3	−3107	CB#
3108	CB4	−3108	CB$
3109	CB5	−3109	CB%
3110	CB6	−3110	CB&
3111	CB7	−3111	CB'
3112	CB8	−3112	CB(
3113	CB9	−3113	CB)
3114	CB:	−3114	CB*
3115	CB;	−3115	CB+
3116	CB<	−3116	CB,
3117	CB=	−3117	CB−
3118	CB>	−3118	CB.
3119	CB?	−3119	CB/
3120	CC0	−3120	CCSP
3121	CC1	−3121	CC!
3122	CC2	−3122	CC"
3123	CC3	−3123	CC#
3124	CC4	−3124	CC$
3125	CC5	−3125	CC%
3126	CC6	−3126	CC&
3127	CC7	−3127	CC'
3128	CC8	−3128	CC(
3129	CC9	−3129	CC)
3130	CC:	−3130	CC*
3131	CC;	−3131	CC+
3132	CC<	−3132	CC,
3133	CC=	−3133	CC−
3134	CC>	−3134	CC.
3135	CC?	−3135	CC/
3136	CD0	−3136	CDSP
3137	CD1	−3137	CD!
3138	CD2	−3138	CD"
3139	CD3	−3139	CD#
3140	CD4	−3140	CD$
3141	CD5	−3141	CD%
3142	CD6	−3142	CD&
3143	CD7	−3143	CD'
3144	CD8	−3144	CD(
3145	CD9	−3145	CD)
3146	CD:	−3146	CD*
3147	CD;	−3147	CD+
3148	CD<	−3148	CD,
3149	CD=	−3149	CD−

Table B-1 (cont)

REPRESENTING NUMBERS AS *INT* PARAMETERS

n	*int*: n	-n	*int*: -n
3150	CD>	−3150	CD.
3151	CD?	−3151	CD/
3152	CE0	−3152	CE^{SP}
3153	CE1	−3153	CE!
3154	CE2	−3154	CE"
3155	CE3	−3155	CE#
3156	CE4	−3156	CE$
3157	CE5	−3157	CE%
3158	CE6	−3158	CE&
3159	CE7	−3159	CE'
3160	CE8	−3160	CE(
3161	CE9	−3161	CE)
3162	CE:	−3162	CE*
3163	CE;	−3163	CE+
3164	CE<	−3164	CE,
3165	CE=	−3165	CE−
3166	CE>	−3166	CE.
3167	CE?	−3167	CE/
3168	CF0	−3168	CF^{SP}
3169	CF1	−3169	CF!
3170	CF2	−3170	CF"
3171	CF3	−3171	CF#
3172	CF4	−3172	CF$
3173	CF5	−3173	CF%
3174	CF6	−3174	CF&
3175	CF7	−3175	CF'
3176	CF8	−3176	CF(
3177	CF9	−3177	CF)
3178	CF:	−3178	CF*
3179	CF;	−3179	CF+
3180	CF<	−3180	CF,
3181	CF=	−3181	CF−
3182	CF>	−3182	CF.
3183	CF?	−3183	CF/
3184	CG0	−3184	CG^{SP}
3185	CG1	−3185	CG!
3186	CG2	−3186	CG"
3187	CG3	−3187	CG#
3188	CG4	−3188	CG$
3189	CG5	−3189	CG%
3190	CG6	−3190	CG&
3191	CG7	−3191	CG'
3192	CG8	−3192	CG(
3193	CG9	−3193	CG)
3194	CG:	−3194	CG*
3195	CG;	−3195	CG+
3196	CG<	−3196	CG,
3197	CG=	−3197	CG−
3198	CG>	−3198	CG.
3199	CG?	−3199	CG/
3200	CH0	−3200	CH^{SP}
3201	CH1	−3201	CH!
3202	CH2	−3202	CH"
3203	CH3	−3203	CH#
3204	CH4	−3204	CH$
3205	CH5	−3205	CH%
3206	CH6	−3206	CH&
3207	CH7	−3207	CH'
3208	CH8	−3208	CH(
3209	CH9	−3209	CH)
3210	CH:	−3210	CH*
3211	CH;	−3211	CH+
3212	CH<	−3212	CH,
3213	CH=	−3213	CH−
3214	CH>	−3214	CH.
3215	CH?	−3215	CH/
3216	CI0	−3216	CI^{SP}
3217	CI1	−3217	CI!
3218	CI2	−3218	CI"
3219	CI3	−3219	CI#
3220	CI4	−3220	CI$
3221	CI5	−3221	CI%
3222	CI6	−3222	CI&
3223	CI7	−3223	CI'
3224	CI8	−3224	CI(
3225	CI9	−3225	CI)
3226	CI:	−3226	CI*
3227	CI;	−3227	CI+
3228	CI<	−3228	CI,
3229	CI=	−3229	CI−
3230	CI>	−3230	CI.
3231	CI?	−3231	CI/
3232	CJ0	−3232	CJ^{SP}
3233	CJ1	−3233	CJ!
3234	CJ2	−3234	CJ"
3235	CJ3	−3235	CJ#
3236	CJ4	−3236	CJ$
3237	CJ5	−3237	CJ%
3238	CJ6	−3238	CJ&
3239	CJ7	−3239	CJ'
3240	CJ8	−3240	CJ(
3241	CJ9	−3241	CJ)
3242	CJ:	−3242	CJ*
3243	CJ;	−3243	CJ+
3244	CJ<	−3244	CJ,
3245	CJ=	−3245	CJ−
3246	CJ>	−3246	CJ.
3247	CJ?	−3247	CJ/
3248	CK0	−3248	CK^{SP}
3249	CK1	−3249	CK!
3250	CK2	−3250	CK"
3251	CK3	−3251	CK#
3252	CK4	−3252	CK$
3253	CK5	−3253	CK%
3254	CK6	−3254	CK&
3255	CK7	−3255	CK'
3256	CK8	−3256	CK(
3257	CK9	−3257	CK)
3258	CK:	−3258	CK*
3259	CK;	−3259	CK+
3260	CK<	−3260	CK,
3261	CK=	−3261	CK−
3262	CK>	−3262	CK.
3263	CK?	−3263	CK/
3264	CL0	−3264	CL^{SP}
3265	CL1	−3265	CL!
3266	CL2	−3266	CL"
3267	CL3	−3267	CL#
3268	CL4	−3268	CL$
3269	CL5	−3269	CL%
3270	CL6	−3270	CL&
3271	CL7	−3271	CL'
3272	CL8	−3272	CL(
3273	CL9	−3273	CL)
3274	CL:	−3274	CL*
3275	CL;	−3275	CL+
3276	CL<	−3276	CL,
3277	CL=	−3277	CL−
3278	CL>	−3278	CL.
3279	CL?	−3279	CL/
3280	CM0	−3280	CM^{SP}
3281	CM1	−3281	CM!
3282	CM2	−3282	CM"
3283	CM3	−3283	CM#
3284	CM4	−3284	CM$
3285	CM5	−3285	CM%
3286	CM6	−3286	CM&
3287	CM7	−3287	CM'
3288	CM8	−3288	CM(
3289	CM9	−3289	CM)
3290	CM:	−3290	CM*
3291	CM;	−3291	CM+
3292	CM<	−3292	CM,
3293	CM=	−3293	CM−
3294	CM>	−3294	CM.
3295	CM?	−3295	CM/
3296	CN0	−3296	CN^{SP}
3297	CN1	−3297	CN!
3298	CN2	−3298	CN"
3299	CN3	−3299	CN#

Table B-1 (cont)

REPRESENTING NUMBERS AS *INT* PARAMETERS

n	*int*: n	-n	*int*: -n
3300	CN4	−3300	CN$
3301	CN5	−3301	CN%
3302	CN6	−3302	CN&
3303	CN7	−3303	CN'
3304	CN8	−3304	CN(
3305	CN9	−3305	CN)
3306	CN:	−3306	CN*
3307	CN;	−3307	CN+
3308	CN<	−3308	CN,
3309	CN=	−3309	CN−
3310	CN>	−3310	CN.
3311	CN?	−3311	CN/
3312	CO0	−3312	CO$_P$
3313	CO1	−3313	CO!
3314	CO2	−3314	CO"
3315	CO3	−3315	CO#
3316	CO4	−3316	CO$
3317	CO5	−3317	CO%
3318	CO6	−3318	CO&
3319	CO7	−3319	CO'
3320	CO8	−3320	CO(
3321	CO9	−3321	CO)
3322	CO:	−3322	CO*
3323	CO;	−3323	CO+
3324	CO<	−3324	CO,
3325	CO=	−3325	CO−
3326	CO>	−3326	CO.
3327	CO?	−3327	CO/
3328	CP0	−3328	CP$_P$
3329	CP1	−3329	CP!
3330	CP2	−3330	CP"
3331	CP3	−3331	CP#
3332	CP4	−3332	CP$
3333	CP5	−3333	CP%
3334	CP6	−3334	CP&
3335	CP7	−3335	CP'
3336	CP8	−3336	CP(
3337	CP9	−3337	CP)
3338	CP:	−3338	CP*
3339	CP;	−3339	CP+
3340	CP<	−3340	CP,
3341	CP=	−3341	CP−
3342	CP>	−3342	CP.
3343	CP?	−3343	CP/
3344	CQ0	−3344	CQ$_P$
3345	CQ1	−3345	CQ!
3346	CQ2	−3346	CQ"
3347	CQ3	−3347	CQ#
3348	CQ4	−3348	CQ$
3349	CQ5	−3349	CQ%
3350	CQ6	−3350	CQ&
3351	CQ7	−3351	CQ'
3352	CQ8	−3352	CQ(
3353	CQ9	−3353	CQ)
3354	CQ:	−3354	CQ*
3355	CQ;	−3355	CQ+
3356	CQ<	−3356	CQ,
3357	CQ=	−3357	CQ−
3358	CQ>	−3358	CQ.
3359	CQ?	−3359	CQ/
3360	CR0	−3360	CR$_P$
3361	CR1	−3361	CR!
3362	CR2	−3362	CR"
3363	CR3	−3363	CR#
3364	CR4	−3364	CR$
3365	CR5	−3365	CR%
3366	CR6	−3366	CR&
3367	CR7	−3367	CR'
3368	CR8	−3368	CR(
3369	CR9	−3369	CR)
3370	CR:	−3370	CR*
3371	CR;	−3371	CR+
3372	CR<	−3372	CR,
3373	CR=	−3373	CR−
3374	CR>	−3374	CR.
3375	CR?	−3375	CR/
3376	CS0	−3376	CS$_P$
3377	CS1	−3377	CS!
3378	CS2	−3378	CS"
3379	CS3	−3379	CS#
3380	CS4	−3380	CS$
3381	CS5	−3381	CS%
3382	CS6	−3382	CS&
3383	CS7	−3383	CS'
3384	CS8	−3384	CS(
3385	CS9	−3385	CS)
3386	CS:	−3386	CS*
3387	CS;	−3387	CS+
3388	CS<	−3388	CS,
3389	CS=	−3389	CS−
3390	CS>	−3390	CS.
3391	CS?	−3391	CS/
3392	CT0	−3392	CT$_P$
3393	CT1	−3393	CT!
3394	CT2	−3394	CT"
3395	CT3	−3395	CT#
3396	CT4	−3396	CT$
3397	CT5	−3397	CT%
3398	CT6	−3398	CT&
3399	CT7	−3399	CT'
3400	CT8	−3400	CT(
3401	CT9	−3401	CT)
3402	CT:	−3402	CT*
3403	CT;	−3403	CT+
3404	CT<	−3404	CT,
3405	CT=	−3405	CT−
3406	CT>	−3406	CT.
3407	CT?	−3407	CT/
3408	CU0	−3408	CU$_P$
3409	CU1	−3409	CU!
3410	CU2	−3410	CU"
3411	CU3	−3411	CU#
3412	CU4	−3412	CU$
3413	CU5	−3413	CU%
3414	CU6	−3414	CU&
3415	CU7	−3415	CU'
3416	CU8	−3416	CU(
3417	CU9	−3417	CU)
3418	CU:	−3418	CU*
3419	CU;	−3419	CU+
3420	CU<	−3420	CU,
3421	CU=	−3421	CU−
3422	CU>	−3422	CU.
3423	CU?	−3423	CU/
3424	CV0	−3424	CV$_P$
3425	CV1	−3425	CV!
3426	CV2	−3426	CV"
3427	CV3	−3427	CV#
3428	CV4	−3428	CV$
3429	CV5	−3429	CV%
3430	CV6	−3430	CV&
3431	CV7	−3431	CV'
3432	CV8	−3432	CV(
3433	CV9	−3433	CV)
3434	CV:	−3434	CV*
3435	CV;	−3435	CV+
3436	CV<	−3436	CV,
3437	CV=	−3437	CV−
3438	CV>	−3438	CV.
3439	CV?	−3439	CV/
3440	CW0	−3440	CW$_P$
3441	CW1	−3441	CW!
3442	CW2	−3442	CW"
3443	CW3	−3443	CW#
3444	CW4	−3444	CW$
3445	CW5	−3445	CW%
3446	CW6	−3446	CW&
3447	CW7	−3447	CW'
3448	CW8	−3448	CW(
3449	CW9	−3449	CW)

Table B-1 (cont)

REPRESENTING NUMBERS AS *INT* PARAMETERS

n	*int*: n	–n	*int*: –n
3450	CW:	–3450	CW*
3451	CW;	–3451	CW+
3452	CW<	–3452	CW,
3453	CW=	–3453	CW–
3454	CW>	–3454	CW.
3455	CW?	–3455	CW/
3456	CX0	–3456	CX^{SP}
3457	CX1	–3457	CX!
3458	CX2	–3458	CX"
3459	CX3	–3459	CX#
3460	CX4	–3460	CX$
3461	CX5	–3461	CX%
3462	CX6	–3462	CX&
3463	CX7	–3463	CX'
3464	CX8	–3464	CX(
3465	CX9	–3465	CX)
3466	CX:	–3466	CX*
3467	CX;	–3467	CX+
3468	CX<	–3468	CX,
3469	CX=	–3469	CX–
3470	CX>	–3470	CX.
3471	CX?	–3471	CX/
3472	CY0	–3472	CY^{SP}
3473	CY1	–3473	CY!
3474	CY2	–3474	CY"
3475	CY3	–3475	CY#
3476	CY4	–3476	CY$
3477	CY5	–3477	CY%
3478	CY6	–3478	CY&
3479	CY7	–3479	CY'
3480	CY8	–3480	CY(
3481	CY9	–3481	CY)
3482	CY:	–3482	CY*
3483	CY;	–3483	CY+
3484	CY<	–3484	CY,
3485	CY=	–3485	CY–
3486	CY>	–3486	CY.
3487	CY?	–3487	CY/
3488	CZ0	–3488	CZ^{SP}
3489	CZ1	–3489	CZ!
3490	CZ2	–3490	CZ"
3491	CZ3	–3491	CZ#
3492	CZ4	–3492	CZ$
3493	CZ5	–3493	CZ%
3494	CZ6	–3494	CZ&
3495	CZ7	–3495	CZ'
3496	CZ8	–3496	CZ(
3497	CZ9	–3497	CZ)
3498	CZ:	–3498	CZ*
3499	CZ;	–3499	CZ+
3500	CZ<	–3500	CZ,
3501	CZ=	–3501	CZ–
3502	CZ>	–3502	CZ.
3503	CZ?	–3503	CZ/
3504	C[0	–3504	C[^{SP}
3505	C[1	–3505	C[!
3506	C[2	–3506	C["
3507	C[3	–3507	C[#
3508	C[4	–3508	C[$
3509	C[5	–3509	C[%
3510	C[6	–3510	C[&
3511	C[7	–3511	C['
3512	C[8	–3512	C[(
3513	C[9	–3513	C[)
3514	C[:	–3514	C[*
3515	C[;	–3515	C[+
3516	C[<	–3516	C[,
3517	C[=	–3517	C[–
3518	C[>	–3518	C[.
3519	C[?	–3519	C[/
3520	C\0	–3520	C\$^{SP}$
3521	C\1	–3521	C\!
3522	C\2	–3522	C\"
3523	C\3	–3523	C\#
3524	C\4	–3524	C\$
3525	C\5	–3525	C\%
3526	C\6	–3526	C\&
3527	C\7	–3527	C\'
3528	C\8	–3528	C\(
3529	C\9	–3529	C\)
3530	C\:	–3530	C*
3531	C\;	–3531	C\+
3532	C\<	–3532	C\,
3533	C\=	–3533	C\–
3534	C\>	–3534	C\.
3535	C\?	–3535	C\/
3536	C]0	–3536	C]^{SP}
3537	C]1	–3537	C]!
3538	C]2	–3538	C]"
3539	C]3	–3539	C]#
3540	C]4	–3540	C]$
3541	C]5	–3541	C]%
3542	C]6	–3542	C]&
3543	C]7	–3543	C]'
3544	C]8	–3544	C](
3545	C]9	–3545	C])
3546	C]:	–3546	C]*
3547	C];	–3547	C]+
3548	C]<	–3548	C],
3549	C]=	–3549	C]–
3550	C]>	–3550	C].
3551	C]?	–3551	C]/
3552	C^0	–3552	C^^{SP}
3553	C^1	–3553	C^!
3554	C^2	–3554	C^"
3555	C^3	–3555	C^#
3556	C^4	–3556	C^$
3557	C^5	–3557	C^%
3558	C^6	–3558	C^&
3559	C^7	–3559	C^'
3560	C^8	–3560	C^(
3561	C^9	–3561	C^)
3562	C^:	–3562	C^*
3563	C^;	–3563	C^+
3564	C^<	–3564	C^,
3565	C^=	–3565	C^–
3566	C^>	–3566	C^.
3567	C^?	–3567	C^/
3568	C_0	–3568	C_^{SP}
3569	C_1	–3569	C_!
3570	C_2	–3570	C_"
3571	C_3	–3571	C_#
3572	C_4	–3572	C_$
3573	C_5	–3573	C_%
3574	C_6	–3574	C_&
3575	C_7	–3575	C_'
3576	C_8	–3576	C_(
3577	C_9	–3577	C_)
3578	C_:	–3578	C_*
3579	C_;	–3579	C_+
3580	C_<	–3580	C_,
3581	C_=	–3581	C_–
3582	C_>	–3582	C_.
3583	C_?	–3583	C_/
3584	C‘0	–3584	C‘^{SP}
3585	C‘1	–3585	C‘!
3586	C‘2	–3586	C‘"
3587	C‘3	–3587	C‘#
3588	C‘4	–3588	C‘$
3589	C‘5	–3589	C‘%
3590	C‘6	–3590	C‘&
3591	C‘7	–3591	C‘'
3592	C‘8	–3592	C‘(
3593	C‘9	–3593	C‘)
3594	C‘:	–3594	C‘*
3595	C‘;	–3595	C‘+
3596	C‘<	–3596	C‘,
3597	C‘=	–3597	C‘–
3598	C‘>	–3598	C‘.
3599	C‘?	–3599	C‘/

Table B-1 (cont)

REPRESENTING NUMBERS AS *INT* PARAMETERS

n	*int*: n	-n	*int*: -n
3600	Ca0	−3600	Ca^{SP}
3601	Ca1	−3601	Ca!
3602	Ca2	−3602	Ca"
3603	Ca3	−3603	Ca#
3604	Ca4	−3604	Ca$
3605	Ca5	−3605	Ca%
3606	Ca6	−3606	Ca&
3607	Ca7	−3607	Ca'
3608	Ca8	−3608	Ca(
3609	Ca9	−3609	Ca)
3610	Ca:	−3610	Ca*
3611	Ca;	−3611	Ca+
3612	Ca<	−3612	Ca,
3613	Ca=	−3613	Ca−
3614	Ca>	−3614	Ca.
3615	Ca?	−3615	Ca/
3616	Cb0	−3616	Cb^{SP}
3617	Cb1	−3617	Cb!
3618	Cb2	−3618	Cb"
3619	Cb3	−3619	Cb#
3620	Cb4	−3620	Cb$
3621	Cb5	−3621	Cb%
3622	Cb6	−3622	Cb&
3623	Cb7	−3623	Cb'
3624	Cb8	−3624	Cb(
3625	Cb9	−3625	Cb)
3626	Cb:	−3626	Cb*
3627	Cb;	−3627	Cb+
3628	Cb<	−3628	Cb,
3629	Cb=	−3629	Cb−
3630	Cb>	−3630	Cb.
3631	Cb?	−3631	Cb/
3632	Cc0	−3632	Cc^{SP}
3633	Cc1	−3633	Cc!
3634	Cc2	−3634	Cc"
3635	Cc3	−3635	Cc#
3636	Cc4	−3636	Cc$
3637	Cc5	−3637	Cc%
3638	Cc6	−3638	Cc&
3639	Cc7	−3639	Cc'
3640	Cc8	−3640	Cc(
3641	Cc9	−3641	Cc)
3642	Cc:	−3642	Cc*
3643	Cc;	−3643	Cc+
3644	Cc<	−3644	Cc,
3645	Cc=	−3645	Cc−
3646	Cc>	−3646	Cc.
3647	Cc?	−3647	Cc/
3648	Cd0	−3648	Cd^{SP}
3649	Cd1	−3649	Cd!
3650	Cd2	−3650	Cd"
3651	Cd3	−3651	Cd#
3652	Cd4	−3652	Cd$
3653	Cd5	−3653	Cd%
3654	Cd6	−3654	Cd&
3655	Cd7	−3655	Cd'
3656	Cd8	−3656	Cd(
3657	Cd9	−3657	Cd)
3658	Cd:	−3658	Cd*
3659	Cd;	−3659	Cd+
3660	Cd<	−3660	Cd,
3661	Cd=	−3661	Cd−
3662	Cd>	−3662	Cd.
3663	Cd?	−3663	Cd/
3664	Ce0	−3664	Ce^{SP}
3665	Ce1	−3665	Ce!
3666	Ce2	−3666	Ce"
3667	Ce3	−3667	Ce#
3668	Ce4	−3668	Ce$
3669	Ce5	−3669	Ce%
3670	Ce6	−3670	Ce&
3671	Ce7	−3671	Ce'
3672	Ce8	−3672	Ce(
3673	Ce9	−3673	Ce)
3674	Ce:	−3674	Ce*
3675	Ce;	−3675	Ce+
3676	Ce<	−3676	Ce,
3677	Ce=	−3677	Ce−
3678	Ce>	−3678	Ce.
3679	Ce?	−3679	Ce/
3680	Cf0	−3680	Cf^{SP}
3681	Cf1	−3681	Cf!
3682	Cf2	−3682	Cf"
3683	Cf3	−3683	Cf#
3684	Cf4	−3684	Cf$
3685	Cf5	−3685	Cf%
3686	Cf6	−3686	Cf&
3687	Cf7	−3687	Cf'
3688	Cf8	−3688	Cf(
3689	Cf9	−3689	Cf)
3690	Cf:	−3690	Cf*
3691	Cf;	−3691	Cf+
3692	Cf<	−3692	Cf,
3693	Cf=	−3693	Cf−
3694	Cf>	−3694	Cf.
3695	Cf?	−3695	Cf/
3696	Cg0	−3696	Cg^{SP}
3697	Cg1	−3697	Cg!
3698	Cg2	−3698	Cg"
3699	Cg3	−3699	Cg#
3700	Cg4	−3700	Cg$
3701	Cg5	−3701	Cg%
3702	Cg6	−3702	Cg&
3703	Cg7	−3703	Cg'
3704	Cg8	−3704	Cg(
3705	Cg9	−3705	Cg)
3706	Cg:	−3706	Cg*
3707	Cg;	−3707	Cg+
3708	Cg<	−3708	Cg,
3709	Cg=	−3709	Cg−
3710	Cg>	−3710	Cg.
3711	Cg?	−3711	Cg/
3712	Ch0	−3712	Ch^{SP}
3713	Ch1	−3713	Ch!
3714	Ch2	−3714	Ch"
3715	Ch3	−3715	Ch#
3716	Ch4	−3716	Ch$
3717	Ch5	−3717	Ch%
3718	Ch6	−3718	Ch&
3719	Ch7	−3719	Ch'
3720	Ch8	−3720	Ch(
3721	Ch9	−3721	Ch)
3722	Ch:	−3722	Ch*
3723	Ch;	−3723	Ch+
3724	Ch<	−3724	Ch,
3725	Ch=	−3725	Ch−
3726	Ch>	−3726	Ch.
3727	Ch?	−3727	Ch/
3728	Ci0	−3728	Ci^{SP}
3729	Ci1	−3729	Ci!
3730	Ci2	−3730	Ci"
3731	Ci3	−3731	Ci#
3732	Ci4	−3732	Ci$
3733	Ci5	−3733	Ci%
3734	Ci6	−3734	Ci&
3735	Ci7	−3735	Ci'
3736	Ci8	−3736	Ci(
3737	Ci9	−3737	Ci)
3738	Ci:	−3738	Ci*
3739	Ci;	−3739	Ci+
3740	Ci<	−3740	Ci,
3741	Ci=	−3741	Ci−
3742	Ci>	−3742	Ci.
3743	Ci?	−3743	Ci/
3744	Cj0	−3744	Cj^{SP}
3745	Cj1	−3745	Cj!
3746	Cj2	−3746	Cj"
3747	Cj3	−3747	Cj#
3748	Cj4	−3748	Cj$
3749	Cj5	−3749	Cj%

Table B-1 (cont)

REPRESENTING NUMBERS AS *INT* PARAMETERS

n	*int*: n	-n	*int*: -n
3750	Cj6	−3750	Cj&
3751	Cj7	−3751	Cj'
3752	Cj8	−3752	Cj(
3753	Cj9	−3753	Cj)
3754	Cj:	−3754	Cj*
3755	Cj;	−3755	Cj+
3756	Cj<	−3756	Cj,
3757	Cj=	−3757	Cj−
3758	Cj>	−3758	Cj.
3759	Cj?	−3759	Cj/
3760	Ck0	−3760	Ck$^{S}_{P}$
3761	Ck1	−3761	Ck!
3762	Ck2	−3762	Ck"
3763	Ck3	−3763	Ck#
3764	Ck4	−3764	Ck$
3765	Ck5	−3765	Ck%
3766	Ck6	−3766	Ck&
3767	Ck7	−3767	Ck'
3768	Ck8	−3768	Ck(
3769	Ck9	−3769	Ck)
3770	Ck:	−3770	Ck*
3771	Ck;	−3771	Ck+
3772	Ck<	−3772	Ck,
3773	Ck=	−3773	Ck−
3774	Ck>	−3774	Ck.
3775	Ck?	−3775	Ck/
3776	Cl0	−3776	Cl$^{S}_{P}$
3777	Cl1	−3777	Cl!
3778	Cl2	−3778	Cl"
3779	Cl3	−3779	Cl#
3780	Cl4	−3780	Cl$
3781	Cl5	−3781	Cl%
3782	Cl6	−3782	Cl&
3783	Cl7	−3783	Cl'
3784	Cl8	−3784	Cl(
3785	Cl9	−3785	Cl)
3786	Cl:	−3786	Cl*
3787	Cl;	−3787	Cl+
3788	Cl<	−3788	Cl,
3789	Cl=	−3789	Cl−
3790	Cl>	−3790	Cl.
3791	Cl?	−3791	Cl/
3792	Cm0	−3792	Cm$^{S}_{P}$
3793	Cm1	−3793	Cm!
3794	Cm2	−3794	Cm"
3795	Cm3	−3795	Cm#
3796	Cm4	−3796	Cm$
3797	Cm5	−3797	Cm%
3798	Cm6	−3798	Cm&
3799	Cm7	−3799	Cm'
3800	Cm8	−3800	Cm(
3801	Cm9	−3801	Cm)
3802	Cm:	−3802	Cm*
3803	Cm;	−3803	Cm+
3804	Cm<	−3804	Cm,
3805	Cm=	−3805	Cm−
3806	Cm>	−3806	Cm.
3807	Cm?	−3807	Cm/
3808	Cn0	−3808	Cn$^{S}_{P}$
3809	Cn1	−3809	Cn!
3810	Cn2	−3810	Cn"
3811	Cn3	−3811	Cn#
3812	Cn4	−3812	Cn$
3813	Cn5	−3813	Cn%
3814	Cn6	−3814	Cn&
3815	Cn7	−3815	Cn'
3816	Cn8	−3816	Cn(
3817	Cn9	−3817	Cn)
3818	Cn:	−3818	Cn*
3819	Cn;	−3819	Cn+
3820	Cn<	−3820	Cn,
3821	Cn=	−3821	Cn−
3822	Cn>	−3822	Cn.
3823	Cn?	−3823	Cn/
3824	Co0	−3824	Co$^{S}_{P}$
3825	Co1	−3825	Co!
3826	Co2	−3826	Co"
3827	Co3	−3827	Co#
3828	Co4	−3828	Co$
3829	Co5	−3829	Co%
3830	Co6	−3830	Co&
3831	Co7	−3831	Co'
3832	Co8	−3832	Co(
3833	Co9	−3833	Co)
3834	Co:	−3834	Co*
3835	Co;	−3835	Co+
3836	Co<	−3836	Co,
3837	Co=	−3837	Co−
3838	Co>	−3838	Co.
3839	Co?	−3839	Co/
3840	Cp0	−3840	Cp$^{S}_{P}$
3841	Cp1	−3841	Cp!
3842	Cp2	−3842	Cp"
3843	Cp3	−3843	Cp#
3844	Cp4	−3844	Cp$
3845	Cp5	−3845	Cp%
3846	Cp6	−3846	Cp&
3847	Cp7	−3847	Cp'
3848	Cp8	−3848	Cp(
3849	Cp9	−3849	Cp)
3850	Cp:	−3850	Cp*
3851	Cp;	−3851	Cp+
3852	Cp<	−3852	Cp,
3853	Cp=	−3853	Cp−
3854	Cp>	−3854	Cp.
3855	Cp?	−3855	Cp/
3856	Cq0	−3856	Cq$^{S}_{P}$
3857	Cq1	−3857	Cq!
3858	Cq2	−3858	Cq"
3859	Cq3	−3859	Cq#
3860	Cq4	−3860	Cq$
3861	Cq5	−3861	Cq%
3862	Cq6	−3862	Cq&
3863	Cq7	−3863	Cq'
3864	Cq8	−3864	Cq(
3865	Cq9	−3865	Cq)
3866	Cq:	−3866	Cq*
3867	Cq;	−3867	Cq+
3868	Cq<	−3868	Cq,
3869	Cq=	−3869	Cq−
3870	Cq>	−3870	Cq.
3871	Cq?	−3871	Cq/
3872	Cr0	−3872	Cr$^{S}_{P}$
3873	Cr1	−3873	Cr!
3874	Cr2	−3874	Cr"
3875	Cr3	−3875	Cr#
3876	Cr4	−3876	Cr$
3877	Cr5	−3877	Cr%
3878	Cr6	−3878	Cr&
3879	Cr7	−3879	Cr'
3880	Cr8	−3880	Cr(
3881	Cr9	−3881	Cr)
3882	Cr:	−3882	Cr*
3883	Cr;	−3883	Cr+
3884	Cr<	−3884	Cr,
3885	Cr=	−3885	Cr−
3886	Cr>	−3886	Cr.
3887	Cr?	−3887	Cr/
3888	Cs0	−3888	Cs$^{S}_{P}$
3889	Cs1	−3889	Cs!
3890	Cs2	−3890	Cs"
3891	Cs3	−3891	Cs#
3892	Cs4	−3892	Cs$
3893	Cs5	−3893	Cs%
3894	Cs6	−3894	Cs&
3895	Cs7	−3895	Cs'
3896	Cs8	−3896	Cs(
3897	Cs9	−3897	Cs)
3898	Cs:	−3898	Cs*
3899	Cs;	−3899	Cs+

Table B-1 (cont)

REPRESENTING NUMBERS AS *INT* PARAMETERS

n	*int*: n	-n	*int*: -n
3900	Cs<	−3900	Cs,
3901	Cs=	−3901	Cs−
3902	Cs>	−3902	Cs.
3903	Cs?	−3903	Cs/
3904	Ct0	−3904	Ct^{SP}
3905	Ct1	−3905	Ct!
3906	Ct2	−3906	Ct"
3907	Ct3	−3907	Ct#
3908	Ct4	−3908	Ct$
3909	Ct5	−3909	Ct%
3910	Ct6	−3910	Ct&
3911	Ct7	−3911	Ct'
3912	Ct8	−3912	Ct(
3913	Ct9	−3913	Ct)
3914	Ct:	−3914	Ct*
3915	Ct;	−3915	Ct+
3916	Ct<	−3916	Ct,
3917	Ct=	−3917	Ct−
3918	Ct>	−3918	Ct.
3919	Ct?	−3919	Ct/
3920	Cu0	−3920	Cu^{SP}
3921	Cu1	−3921	Cu!
3922	Cu2	−3922	Cu"
3923	Cu3	−3923	Cu#
3924	Cu4	−3924	Cu$
3925	Cu5	−3925	Cu%
3926	Cu6	−3926	Cu&
3927	Cu7	−3927	Cu'
3928	Cu8	−3928	Cu(
3929	Cu9	−3929	Cu)
3930	Cu:	−3930	Cu*
3931	Cu;	−3931	Cu+
3932	Cu<	−3932	Cu,
3933	Cu=	−3933	Cu−
3934	Cu>	−3934	Cu.
3935	Cu?	−3935	Cu/
3936	Cv0	−3936	Cv^{SP}
3937	Cv1	−3937	Cv!
3938	Cv2	−3938	Cv"
3939	Cv3	−3939	Cv#
3940	Cv4	−3940	Cv$
3941	Cv5	−3941	Cv%
3942	Cv6	−3942	Cv&
3943	Cv7	−3943	Cv'
3944	Cv8	−3944	Cv(
3945	Cv9	−3945	Cv)
3946	Cv:	−3946	Cv*
3947	Cv;	−3947	Cv+
3948	Cv<	−3948	Cv,
3949	Cv=	−3949	Cv−
3950	Cv>	−3950	Cv.
3951	Cv?	−3951	Cv/
3952	Cw0	−3952	Cw^{SP}
3953	Cw1	−3953	Cw!
3954	Cw2	−3954	Cw"
3955	Cw3	−3955	Cw#
3956	Cw4	−3956	Cw$
3957	Cw5	−3957	Cw%
3958	Cw6	−3958	Cw&
3959	Cw7	−3959	Cw'
3960	Cw8	−3960	Cw(
3961	Cw9	−3961	Cw)
3962	Cw:	−3962	Cw*
3963	Cw;	−3963	Cw+
3964	Cw<	−3964	Cw,
3965	Cw=	−3965	Cw−
3966	Cw>	−3966	Cw.
3967	Cw?	−3967	Cw/
3968	Cx0	−3968	Cx^{SP}
3969	Cx1	−3969	Cx!
3970	Cx2	−3970	Cx"
3971	Cx3	−3971	Cx#
3972	Cx4	−3972	Cx$
3973	Cx5	−3973	Cx%
3974	Cx6	−3974	Cx&
3975	Cx7	−3975	Cx'
3976	Cx8	−3976	Cx(
3977	Cx9	−3977	Cx)
3978	Cx:	−3978	Cx*
3979	Cx;	−3979	Cx+
3980	Cx<	−3980	Cx,
3981	Cx=	−3981	Cx−
3982	Cx>	−3982	Cx.
3983	Cx?	−3983	Cx/
3984	Cy0	−3984	Cy^{SP}
3985	Cy1	−3985	Cy!
3986	Cy2	−3986	Cy"
3987	Cy3	−3987	Cy#
3988	Cy4	−3988	Cy$
3989	Cy5	−3989	Cy%
3990	Cy6	−3990	Cy&
3991	Cy7	−3991	Cy'
3992	Cy8	−3992	Cy(
3993	Cy9	−3993	Cy)
3994	Cy:	−3994	Cy*
3995	Cy;	−3995	Cy+
3996	Cy<	−3996	Cy,
3997	Cy=	−3997	Cy−
3998	Cy>	−3998	Cy.
3999	Cy?	−3999	Cy/
4000	Cz0	−4000	Cz^{SP}
4001	Cz1	−4001	Cz!
4002	Cz2	−4002	Cz"
4003	Cz3	−4003	Cz#
4004	Cz4	−4004	Cz$
4005	Cz5	−4005	Cz%
4006	Cz6	−4006	Cz&
4007	Cz7	−4007	Cz'
4008	Cz8	−4008	Cz(
4009	Cz9	−4009	Cz)
4010	Cz:	−4010	Cz*
4011	Cz;	−4011	Cz+
4012	Cz<	−4012	Cz,
4013	Cz=	−4013	Cz−
4014	Cz>	−4014	Cz.
4015	Cz?	−4015	Cz/
4016	C{0	−4016	C{^{SP}
4017	C{1	−4017	C{!
4018	C{2	−4018	C{"
4019	C{3	−4019	C{#
4020	C{4	−4020	C{$
4021	C{5	−4021	C{%
4022	C{6	−4022	C{&
4023	C{7	−4023	C{'
4024	C{8	−4024	C{(
4025	C{9	−4025	C{)
4026	C{:	−4026	C{*
4027	C{;	−4027	C{+
4028	C{<	−4028	C{,
4029	C{=	−4029	C{−
4030	C{>	−4030	C{.
4031	C{?	−4031	C{/
4032	C|0	−4032	C|^{SP}
4033	C|1	−4033	C|!
4034	C|2	−4034	C|"
4035	C|3	−4035	C|#
4036	C|4	−4036	C|$
4037	C|5	−4037	C|%
4038	C|6	−4038	C|&
4039	C|7	−4039	C|'
4040	C|8	−4040	C|(
4041	C|9	−4041	C|)
4042	C|:	−4042	C|*
4043	C|;	−4043	C|+
4044	C|<	−4044	C|,
4045	C|=	−4045	C|−
4046	C|>	−4046	C|.
4047	C|?	−4047	C|/
4048	C}0	−4048	C}^{SP}
4049	C}1	−4049	C}!

Appendix C

CODE EXAMPLES

This appendix contains a set of FORTRAN subroutines that were extracted from a system used to test the 4110 Series terminals. These routines are intended as an example of one way to send commands, parse reports, switch implicit modes, and communicate in block mode.

These routines are not software that is supported by Tektronix Inc. and anyone trying to implement them does so at his own risk.

While these routines are known to work in the environment where they were developed, they may not work in any other environment. For example, some common variables are set and used by routines not included in this appendix and several routines have been modified to not call other routines not included here.


```
C
C----------SUBROUTINE--ADEIN-----
C
      SUBROUTINE ADEIN (ILEN,IRAY)
C * THIS VERSION OF ADEIN CALLS A SINGLE CHAR INPUT MACRO-10 ROUTINE
      DIMENSION IRAY(1)
      ILEN=0
10    CALL ICHIN (JCHAR)
      IF (JCHAR.EQ.13) RETURN
      ILEN=ILEN+1
      IRAY(ILEN)=JCHAR
      GO TO 10
      END
C
C----------SUBROUTINE--ADEOUT----
C
      SUBROUTINE ADEOUT (ILEN,IRAY)
C * THIS VERSION OF ADEOUT CALLS A SINGLE CHAR OUTPUT MACRO-10 ROUTINE
      DIMENSION IRAY(1)
      IF (ILEN.LE.0) RETURN
      DO 10 J=1,ILEN
      CALL ICHOUT (IRAY(J))
10    CONTINUE
      RETURN
      END
C
```

```
C----------SUBROUTINE--ADERAY----
C
      SUBROUTINE ADERAY (ILEN,IRAY)
C * ADERAY SENDS A PACKED ADE ARRAY
C * ARGUMENTS:
C * ILEN    - LENGTH OF ARRAY
C * IRAY.   - ADE ARRAY TO BE SENT
      DIMENSION IRAY(1)
      CALL INTOUT (ILEN)
      CALL STOUT (ILEN,IRAY)
      RETURN
      END
C
C----------SUBROUTINE--ALFMOD----
C
      SUBROUTINE ALFMOD
C * ALFMOD SENDS A (US) TO THE TERMINAL
      COMMON /VALSYS/ KTERM,KMODE,KGFMAT,KRESLU,KCHARS(4),KLASTX,
     & KLASTY,KCOORD,KREPLN,KRELAB
      CALL CHOUT (31)
      KMODE=1
      RETURN
      END
C
C----------SUBROUTINE--BLKEND----
C
      SUBROUTINE BLKEND (IACK)
C * BLKEND ENDS THE BLOCK MODE PROTOCOL AND DISARMS BLOCK MODE
C * ARGUMENTS:
C * IACK    - ACKNOWLEDGE FLAG: 2 YES,  3 NO, DISARM BLOCK MODE
C *                             -2 YES, -3 NO, LEAVE BLOCK MODE ARMED
      COMMON /VALIO/ KBAUDH,KBAUDT,KMCDEF,KBUFSZ,KBFLIM,KOUTPT,
     & KOUTBF(512),KINEND,KINPT,KINBUF(512),KEOMC1,KEOMC2,KIGDEL,
     & KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),KBMODE,KBMSAV,KHLENH,
     & KHEADH(10),KHLENT,KHEADT(10),KCONTH,KCONTT,KENDH,KENDT,KNXNOH,
     & KNXMTH(20),KNXNOT,KNXMTT(20),KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,
     & KPACKT,KBLENH,KBLENT,KBLINE,KSTRIP,KBLOKH,KEOPH,KEOMH,KEOFH,
     & KEOMT,KEOFT
C * IF NOT IN BLOCK MODE, EXIT
      IF (KBMODE.EQ.0) RETURN
C * TURN OFF TERMINAL (IF IT SHOULD NOT BE ON NORMALLY)
      CALL CMDOUT (75,69)
      CALL INTOUT (0)
C * SET THE CONTROL BITS TO END THE BLOCK PROTOCOL
      KBLOKH=0
      KEOPH=MIN0(3,MAX0(2,IABS(IACK)))
C * DUMP THE BUFFER AND TURN OFF SOFTWARE BLOCK MODE
      CALL DUMP
      KBMODE=0
```

```
C * RESTORE OUTPUT BUFFER SIZE
      KBUFSZ=KBFLIM
C * DISARM BLOCK MODE
      IF (IACK.EQ.0) RETURN
      CALL CMDOUT (79,66)
      CALL INTOUT (0)
      RETURN
      END
C
C----------SUBROUTINE--BLKIN-----
C
      SUBROUTINE BLKIN (INACK)
C * BLKIN GETS A BLOCK FROM THE TERMINAL
C * ARGUMENTS:
C * INACK  - 0 (THE BLOCK WAS GOOD), 1 (THE BLOCK WAS BAD)
      COMMON /VALIO/ KBAUDH,KBAUDT,KMCDEF,KBUFSZ,KBFLIM,KOUTPT,
     & KOUTBF(512),KINEND,KINPT,KINBUF(512),KEOMC1,KEOMC2,KIGDEL,
     & KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),KBMODE,KBMSAV,KHLENH,
     & KHEADH(10),KHLENT,KHEADT(10),KCONTH,KCONTT,KENDH,KENDT,KNXNOH,
     & KNXMTH(20),KNXNOT,KNXMTT(20),KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,
     & KPACKT,KBLENH,KBLENT,KBLINE,KSTRIP,KBLOKH,KEOPH,KEOMH,KEOFH,
     & KEOMT,KEOFT
      COMMON /BLKLIN/ KLINE(256)
C * INITIALIZE MASTER CHARACTER FOUND FLAG
      JMASTR=0
C * INITIALIZE BLOCKING VARIABLES
      IF (KINPT.GE.KINEND) KINEND=0
      JINEND=KINEND
      JREGSR=0
      JRPOWR=1
      JOFSET=0
      IF (KPACKT.EQ.64) JOFSET=32
C * GET A LINE OF INPUT FROM THE TERMINAL
C * SIMULATE PROMPT MODE IF TURNED ON
10    IF (KPRMOD.GT.0) CALL ADEOUT (KPRLEN,KPSTRG)
      CALL ADEIN (JLEN,KLINE)
C * SCAN FOR HEADER
      JLINPT=0
20    JLINPT=JLINPT+1
C * IF THE HEADER IS INCORRECT, GET NEXT LINE
      IF (JLINPT.GE.JLEN) GO TO 10
      IF (KLINE(JLINPT).NE.KHEADT(JLINPT)) GO TO 10
      IF (JLINPT.LT.KHLENT) GO TO 20
C * UNPACK CHARACTERS; TRANSLATE MASTER CHAR PAIRS
30    JLINPT=JLINPT+1
      JCHAR=KLINE(JLINPT)
      IF (JCHAR.EQ.KENDT) GO TO 80
      IF (JCHAR.EQ.KCONTT) GO TO 10
      IF (JLINPT.GE.JLEN) GO TO 100
      IF (JMASTR.EQ.1) GO TO 40
      IF (JCHAR.NE.KMASTT) GO TO 50
      JMASTR=1
```

```
      GO TO 30
C * MASTER CHARACTER SUBSTITUTION
40    JCHAR=KNXMTT(JCHAR-64)
      JMASTR=0
C * CHARACTER UNPACKING
50    IF (KBYTET.EQ.KPACKT) GO TO 70
      JREGSR=JREGSR*KPACKT+JCHAR-JOFSET
      JRPOWR=JRPOWR*KPACKT
C * NEED MORE DATA
      IF (JRPOWR.LT.KBYTET) GO TO 30
60    JRPOWR=JRPOWR/KBYTET
      JINEND=JINEND+1
      KINBUF(JINEND)=JREGSR/JRPOWR
      JREGSR=MOD(JREGSR,JRPOWR)
      IF (JRPOWR.GT.KBYTET) GO TO 60
      GO TO 30
C * STRAIGHT TO THE BUFFER
70    JINEND=JINEND+1
      KINBUF(JINEND)=JCHAR
      GO TO 30
C * CHECK THE BLOCK COUNT BIT
80    JBYTE1=KINBUF(JINEND-3)
      IF (MOD(JBYTE1,4).NE.KBLOKH+KEOPH) GO TO 100
C * DO CHECKSUM TEST
      JMAXBT=KBYTET-1
      CALL CHKSUM (JINEND-KINEND,KINBUF(KINEND+1),JMAXBT,JCHK1,JCHK2)
      IF (JCHK1.NE.JMAXBT .OR. JCHK1.NE.JMAXBT) GO TO 100
C * STRIP CONTROL BYTES
      KINEND=JINEND-4
C * STRIP EOM CHARACTER IF APPROPRIATE
      IF (KINEND.LE.0) GO TO 90
      IF (KSTRIP.EQ.1 .AND. (KINBUF(KINEND).EQ.KEOMC1 .OR.
     & KINBUF(KINEND).EQ.KEOMC2)) KINEND=KINEND-1
C * POSITIVE ACKNOWLEDGE
90    INACK=0
      KEOMT=JBYTE1/64
      KEOFT=MOD(JBYTE1/32,2)
      RETURN
C * NEGATIVE ACKNOWLEDGE
100   INACK=1
      RETURN
      END
C
```

```
C----------SUBROUTINE--BLKINT----
C
      SUBROUTINE BLKINT (IPARM)
C * BLKINT INITIALIZES ALL BLOCK MODE PARAMETERS
      COMMON /VALIO/ KBAUDH,KBAUDT,KMCDEF,KBUFSZ,KBFLIM,KOUTPT,
     & KOUTBF(512),KINEND,KINPT,KINBUF(512),KEOMC1,KEOMC2,KIGDEL,
     & KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),KBMODE,KBMSAV,KHLENH,
     & KHEADH(10),KHLENT,KHEADT(10),KCONTH,KCONTT,KENDH,KENDT,KNXNOH,
     & KNXMTH(20),KNXNOT,KNXMTT(20),KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,
     & KPACKT,KBLENH,KBLENT,KBLINE,KSTRIP,KBLOKH,KEOPH,KEOMH,KEOFH,
     & KEOMT,KEOFT
C * HEADERS
      KHLENT=5
      KHEADT(1)=72
      KHEADT(2)=69
      KHEADT(3)=65
      KHEADT(4)=68
      KHEADT(5)=84
      KHLENH=5
      KHEADH(1)=72
      KHEADH(2)=69
      KHEADH(3)=65
      KHEADH(4)=68
      KHEADH(5)=72
C * MASTER CHARACTERS
      KMASTT=35
      KMASTH=35
C * END CHARACTERS
      KENDT=36
      KENDH=36
C * CONTINUE CHARACTERS
      KCONTT=38
      KCONTH=38
C * BLOCK LENGTH
      KBLENT=512
      KBLENH=512
C * LINE LENGTH
      KBLINE=256
C * NON-XMT-CHARS
      KNXNOT=3
      KNXMTT(1)=35
      KNXMTT(2)=36
      KNXMTT(3)=38
      KNXNOH=5
      KNXMTH(1)=17
      KNXMTH(2)=19
      KNXMTH(3)=35
      KNXMTH(4)=36
      KNXMTH(5)=38
```

```
C * PACKING
      KBYTET=7
      KPACKT=6
      KBYTEH=7
      KPACKH=7
C * STRIP EOM'S
      KSTRIP=1
      RETURN
      END
C
C----------SUBROUTINE--BLKOUT----
C
      SUBROUTINE BLKOUT
C * BLKOUT SENDS A BLOCK TO THE TERMINAL
      COMMON /VALIO/ KBAUDH,KBAUDT,KMCDEF,KBUFSZ,KBFLIM,KOUTPT,
     & KOUTBF(512),KINEND,KINPT,KINBUF(512),KEOMC1,KEOMC2,KIGDEL,
     & KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),KBMODE,KBMSAV,KHLENH,
     & KHEADH(10),KHLENT,KHEADT(10),KCONTH,KCONTT,KENDH,KENDT,KNXNOH,
     & KNXMTH(20),KNXNOT,KNXMTT(20),KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,
     & KPACKT,KBLENH,KBLENT,KBLINE,KSTRIP,KBLOKH,KEOPH,KEOMH,KEOFH,
     & KEOMT,KEOFT
      COMMON /BLKLIN/ KLINE(256)
C * ADD UP CONTROL BYTE BITS
      KOUTBF(KOUTPT+1)=KBLOKH+KEOPH+KEOFH*32+KEOMH*64
      KOUTBF(KOUTPT+2)=0
C * COMPUTE CHECKSUM ON KOUTBF AND THE FIRST TWO CONTROL BYTES
      JMAXBT=2**KBYTEH-1
      CALL CHKSUM (KOUTPT+2,KOUTBF,JMAXBT,JCHK1,JCHK2)
      JCHAR=JMAXBT-JCHK1-JCHK2
      IF (JCHAR.LE.0) JCHAR=JCHAR+JMAXBT
      KOUTBF(KOUTPT+3)=JCHAR
      KOUTBF(KOUTPT+4)=JCHK2
C * INITIALIZE VARIABLES
      JBUFLN=KOUTPT+4
      JRPOWR=1
      JREGSR=0
      JBUFPT=0
      JOFSET=0
      IF (KPACKH.EQ.6) JOFSET=32
C * PUT BLOCK HEADER IN OUTPUT LINE
      DO 10 I=1,KHLENH
      KLINE(I)=KHEADH(I)
10    CONTINUE
C * BUILD AND SEND LINES TO THE TERMINAL
20    JLINPT=KHLENH
C * GET ANOTHER CHARACTER FOR OUTPUT
30    IF (KBYTEH.EQ.KPACKH) GO TO 60
C * GET PACKED CHARACTER
C * SEE IF REGISTER ALREADY HAS ENOUGH BITS
      IF (JRPOWR.GE.2**KPACKH) GO TO 50
```

```
C * GET NEXT CHAR FROM BUFFER
      JBUFPT=JBUFPT+1
C * IF NONE LEFT, PAD REGISTER
      IF (JBUFPT.LE.JBUFLN) GO TO 40
      IF (JRPOWR.EQ.1) GO TO 120
      JCHAR=JREGSR*2**KPACKH/JRPOWR+JOFSET
      JRPOWR=1
      GO TO 70
C * PUT NEW CHAR INTO SHIFT REGISTER
40    JREGSR=JREGSR*2**KBYTEH+KOUTBF(JBUFPT)
      JRPOWR=JRPOWR*2**KBYTEH
      IF (JRPOWR.LT.2**KPACKH) GO TO 30
C * GET PACKED CHAR FROM SHIFT REGISTER
50    JRPOWR=JRPOWR/2**KPACKH
      JCHAR=JREGSR/JRPOWR
      JREGSR=JREGSR-JCHAR*JRPOWR
      JCHAR=JCHAR+JOFSET
      GO TO 70
C * GET NEXT CHARACTER STRAIGHT FROM BUFFER
60    JBUFPT=JBUFPT+1
      IF (JBUFPT.GT.JBUFLN) GO TO 120
      JCHAR=KOUTBF(JBUFPT)
C * SUBSTITUTE NON-TRANSMITTABLE CHARACTERS IF NEEDED
70    JCNTR=0
80    JCNTR=JCNTR+1
      IF (JCNTR.GT.KNXNOH) GO TO 90
      IF (JCHAR.NE.KNXMTH(JCNTR)) GO TO 80
      JLINPT=JLINPT+1
      KLINE(JLINPT)=KMASTH
      JLINPT=JLINPT+1
      KLINE(JLINPT)=64+JCNTR
      GO TO 100
90    JLINPT=JLINPT+1
      KLINE(JLINPT)=JCHAR
C * TEST FOR END OF LINE
100   IF (JLINPT.GE.KBLINE-2) GO TO 110
      GO TO 30
C * PUT IN CONTINUE CHAR AND SEND LINE TO TERMINAL
110   JLINPT=JLINPT+1
      KLINE(JLINPT)=KCONTH
      CALL ADEOUT (JLINPT,KLINE)
      GO TO 20
C * PUT IN END CHAR AND SEND LINE TO TERMINAL
120   JLINPT=JLINPT+1
      KLINE(JLINPT)=KENDH
      CALL ADEOUT (JLINPT,KLINE)
      RETURN
      END
C
```

```
C----------SUBROUTINE--BLOKGO----
C
      SUBROUTINE BLOKGO
C * BLOKGO STARTS BLOCK MODE IN OPERATION
      COMMON /VALIO/ KBAUDH,KBAUDT,KMCDEF,KBUFSZ,KBFLIM,KOUTPT,
     & KOUTBF(512),KINEND,KINPT,KINBUF(512),KEOMC1,KEOMC2,KIGDEL,
     & KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),KBMODE,KBMSAV,KHLENH,
     & KHEADH(10),KHLENT,KHEADT(10),KCONTH,KCONTT,KENDH,KENDT,KNXNOH,
     & KNXMTH(20),KNXNOT,KNXMTT(20),KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,
     & KPACKT,KBLENH,KBLENT,KBLINE,KSTRIP,KBLOKH,KEOPH,KEOMH,KEOFH,
     & KEOMT,KEOFT
C * IF ALREADY IN BLOCK MODE, EXIT
      IF (KBMODE.EQ.1) RETURN
C * ARM BLOCK MODE
      CALL CMDOUT (79,66)
      CALL INTOUT (1)
C * WAIT FOR ARMING TO BE DONE
      CALL CMDOUT (73,81)
      CALL CHOUT (79)
      CALL CHOUT (66)
      CALL CHIN (JCHAR)
      CALL CHIN (JCHAR)
      CALL INTIN (JBARM)
      IF (JBARM.EQ.0) RETURN
C * SET CONTROL BYTE BITS
      KBLOKH=1
      KEOPH=0
      KEOFH=0
      KEOMH=0
C * PUT SOFTWARE INTO BLOCK PROTOCOL
      KBMODE=1
C * CHANGE OUTPUT BUFFER SIZE TO BLOCK SIZE
      KBUFSZ=KBLENH-4
C * TURN TERMINAL ECHO ON
      CALL CMDOUT (75,69)
      CALL INTOUT (1)
      RETURN
      END
C
C----------SUBROUTINE--BLOKIO----
C
      SUBROUTINE BLOKIO
C * BLOKIO CALLS BLKOUT AND BLKIN TO PERFORM ONE BLOCK 'EXCHANGE'
      COMMON /VALIO/ KBAUDH,KBAUDT,KMCDEF,KBUFSZ,KBFLIM,KOUTPT,
     & KOUTBF(512),KINEND,KINPT,KINBUF(512),KEOMC1,KEOMC2,KIGDEL,
     & KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),KBMODE,KBMSAV,KHLENH,
     & KHEADH(10),KHLENT,KHEADT(10),KCONTH,KCONTT,KENDH,KENDT,KNXNOH,
     & KNXMTH(20),KNXNOT,KNXMTT(20),KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,
     & KPACKT,KBLENH,KBLENT,KBLINE,KSTRIP,KBLOKH,KEOPH,KEOMH,KEOFH,
     & KEOMT,KEOFT
```

```
C * INITIALIZE REPETITION COUNTER AND ACKNOWLEDGE FLAG
      JREPET=0
      JBLOKH=KBLOKH
      JNACK=0
C * OUTPUT BLOCK TO TERMINAL
10    CALL BLKOUT
      JREPET=JREPET+1
C * GET BLOCK FROM TERMINAL UNLESS NO-ACK END OF PROTOCOL
      IF (KEOPH.NE.3) CALL BLKIN (JNACK)
      IF (JNACK.EQ.0) GO TO 20
C * IF SECOND REPETITION, GIVE UP
      IF (JREPET.EQ.2) GO TO 30
      GO TO 10
C * FLIP BLOCK COUNT, ZERO OUTPUT BUFFER
20    KBLOKH=1-KBLOKH
      KOUTPT=0
      RETURN
C * TRY OTHER BLOCK NUMBER
30    KBLOKH=1-KBLOKH
      JREPET=0
      IF (KBLOKH.NE.JBLOKH) GO TO 10
C * DUMP UNBLOCKABLE BLOCK
      CALL ADEOUT (KOUTPT,KOUTBF)
      KOUTPT=0
      IF (KEOMH.GT.0) KINEND=0
      RETURN
      END
C
C----------SUBROUTINE--CHIN------
C
      SUBROUTINE CHIN (ICHAR)
C * CHIN CALLS STIN TO INPUT ONE ADE CHARACTER
      DIMENSION JRAY(1)
      CALL STIN (1,JREC,JRAY)
      ICHAR=JRAY(1)
      RETURN
      END
C
C----------SUBROUTINE--CHKSUM----
C
      SUBROUTINE CHKSUM (ILEN,IARRAY,IBYTE,ICHK1,ICHK2)
C * CHKSUM COMPUTES THE BLOCK CHECKSUM
C * ARGUMENTS:
C * ILEN   - LENGTH OF IARRAY
C * IARRAY - ARRAY CONTAINING THE BLOCK
C * IBYTE  - 2**BITS PER BYTE-1
C * ICHK1  - CHECKSUM 1
```

```
C * ICHK2  - CHECKSUM 2
      DIMENSION IARRAY(1)
      ICHK1=IBYTE
      ICHK2=IBYTE
      DO 10 I=1,ILEN
      ICHK1=ICHK1+IARRAY(I)
      ICHK2=ICHK2+ICHK1
10    CONTINUE
      ICHK1=ICHK1-(ICHK1-1)/IBYTE*IBYTE
      ICHK2=ICHK2-(ICHK2-1)/IBYTE*IBYTE
      RETURN
      END
C
C----------SUBROUTINE--CHOUT-----
C
      SUBROUTINE CHOUT (ICHAR)
C * CHOUT SENDS ONE ADE CHARACTER
      DIMENSION JRAY(1)
      JRAY(1)=ICHAR
      CALL STOUT (1,JRAY)
      RETURN
      END
C
C----------SUBROUTINE--CMDOUT----
C
      SUBROUTINE CMDOUT (ICHAR1,ICHAR2)
C * CMDOUT SENDS THE THREE CHARACTER TEK ESCAPE SEQUENCE
C * ARGUMENTS:
C * ICHAR1  - FIRST COMMAND CHARACTER
C * ICHAR2  - SECOND COMMAND CHARACTER
      DIMENSION JRAY(3)
      DATA JRAY(1)/27/
      JRAY(2)=ICHAR1
      JRAY(3)=ICHAR2
      CALL STOUT (3,JRAY)
      RETURN
      END
C
C----------SUBROUTINE--CORMOD----
C
      SUBROUTINE CORMOD (ICOORD,IREP)
C * CORMOD SENDS A SET-COORDINATE-MODE COMMAND
C * ARGUMENTS:
C * ICOORD - COORDINATE MODE
```

```
C * IREP   - REPORT-MODE
      COMMON /VALSYS/ KTERM,KMODE,KGFMAT,KRESLU,KCHARS(4),KLASTX,
     & KLASTY,KCOORD,KREPLN,KRELAB
      CALL CMDOUT (85,88)
      CALL INTOUT (ICOORD)
      CALL INTOUT (IREP)
      KCOORD=ICOORD
      IF (IREP.NE.0) KREPLN=IREP
      RETURN
      END
C
C----------SUBROUTINE--DRAW------
C
      SUBROUTINE DRAW (IX,IY)
C * DRAW SENDS A DRAW COMMAND TO THE TERMINAL
C * ARGUMENTS:
C * IX,IY  - POINT TO DRAW TO
      COMMON /VALSYS/ KTERM,KMODE,KGFMAT,KRESLU,KCHARS(4),KLASTX,
     & KLASTY,KCOORD,KREPLN,KRELAB
      IF (KGFMAT.EQ.1) GO TO 20
C * (GS) STYLE
      KRELAB=1
      IF (KMODE.EQ.2) GO TO 10
      CALL VECMOD
      CALL CHOUT (7)
10    CALL XYOUT (IX,IY)
      KRELAB=0
      KLASTX=IX
      KLASTY=IY
      RETURN
C * (ESC) STYLE
20    CALL CMDOUT (76,71)
      CALL XYOUT (IX,IY)
      RETURN
      END
C
C----------SUBROUTINE--DUMP------
C
      SUBROUTINE DUMP
C * DUMP DUMPS THE OUTPUT BUFFER
      DIMENSION IDUMMY(1)
      CALL STOUT (0,IDUMMY)
      RETURN
      END
C
```

```
C----------SUBROUTINE--EMPTIN----
C
      SUBROUTINE EMPTIN
C * EMPTIN ZEROES THE INPUT BUFFER
      COMMON /VALIO/ KBAUDH,KBAUDT,KMCDEF,KBUFSZ,KBFLIM,KOUTPT,
     & KOUTBF(512),KINEND,KINPT,KINBUF(512),KEOMC1,KEOMC2,KIGDEL,
     & KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),KBMODE,KBMSAV,KHLENH,
     & KHEADH(10),KHLENT,KHEADT(10),KCONTH,KCONTT,KENDH,KENDT,KNXNOH,
     & KNXMTH(20),KNXNOT,KNXMTT(20),KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,
     & KPACKT,KBLENH,KBLENT,KBLINE,KSTRIP,KBLOKH,KEOPH,KEOMH,KEOFH,
     & KEOMT,KEOFT
      KINEND=0
      KINPT=0
      RETURN
      END
C
C----------SUBROUTINE--FILIN-----
C
      SUBROUTINE FILIN (IREQST,IRECVD,ISTRNG,IEOF)
C * FILIN INPUTS LINES FROM A TERMINAL FILE, SCANNING FOR EOF
C * ARGUMENTS:
C * IREQST - NUMBER OF CHARS REQUESTED
C * IRECVD - NUMBER OF CHARS RECEIVED
C * ISTRNG - CALLERS INPUT ARRAY
C * IEOF   - 1 IF EOF DETECTED, 0 IF NOT
      COMMON /VALIO/ KBAUDH,KBAUDT,KMCDEF,KBUFSZ,KBFLIM,KOUTPT,
     & KOUTBF(512),KINEND,KINPT,KINBUF(512),KEOMC1,KEOMC2,KIGDEL,
     & KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),KBMODE,KBMSAV,KHLENH,
     & KHEADH(10),KHLENT,KHEADT(10),KCONTH,KCONTT,KENDH,KENDT,KNXNOH,
     & KNXMTH(20),KNXNOT,KNXMTT(20),KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,
     & KPACKT,KBLENH,KBLENT,KBLINE,KSTRIP,KBLOKH,KEOPH,KEOMH,KEOFH,
     & KEOMT,KEOFT
      DIMENSION ISTRNG(1)
C * GET INPUT FROM GENERAL INPUT ROUTINE
      CALL STIN (IREQST,IRECVD,ISTRNG)
C * SET END-OF-FILE ONLY WHEN NO MORE CHARS IN BUFFER
      IEOF=0
      IF (KEOFT.EQ.1 .AND. KINPT.GE.KINEND) IEOF=1
      RETURN
      END
C
```

```
C----------SUBROUTINE--INIT------
C
      SUBROUTINE INIT
C * INIT INITIALIZES THE COMMON AREAS
      COMMON /VALSYS/ KTERM,KMODE,KGFMAT,KRESLU,KCHARS(4),KLASTX,
     & KLASTY,KCOORD,KREPLN,KRELAB
      COMMON /VALIO/ KBAUDH,KBAUDT,KMCDEF,KBUFSZ,KBFLIM,KOUTPT,
     & KOUTBF(512),KINEND,KINPT,KINBUF(512),KEOMC1,KEOMC2,KIGDEL,
     & KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),KBMODE,KBMSAV,KHLENH,
     & KHEADH(10),KHLENT,KHEADT(10),KCONTH,KCONTT,KENDH,KENDT,KNXNOH,
     & KNXMTH(20),KNXNOT,KNXMTT(20),KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,
     & KPACKT,KBLENH,KBLENT,KBLINE,KSTRIP,KBLOKH,KEOPH,KEOMH,KEOFH,
     & KEOMT,KEOFT
      KCOORD=0
      KREPLN=3
      KRELAB=0
      KMODE=1
      KGFMAT=0
      KRESLU=12
      KIGDEL=0
      KMCDEF=0
      DO 30 J=1,4
      KCHARS(J)=0
30    CONTINUE
C * SET I/O BUFFER POINTERS AND SIZE
      KOUTPT=0
      KINPT=0
      KINEND=0
      KBFLIM=256
      KBUFSZ=KBFLIM
C * TURN PROMPT MODE ON
      CALL CMDOUT (78,83)
      CALL INTOUT (63)
      KPSTRG(1)=63
      CALL CMDOUT (78,77)
      CALL INTOUT (1)
      KPRMOD=1
C * GET TERMINAL TYPE
      CALL CMDOUT (73,81)
      CALL CMDOUT (63,84)
      CALL CHIN (JCHAR)
      CALL CHIN (JCHAR)
      CALL INTIN (KTERM)
      RETURN
      END
C
```

```
C----------SUBROUTINE--INTCIN----
C
      SUBROUTINE INTCIN (INT)
C * INTCIN INPUTS AND UNPACKS A 4100 FORMAT INTC-REPORT
C * ARGUMENTS:
C * INT   - RETURNED INTEGER
      COMMON /VALSYS/ KTERM,KMODE,KGFMAT,KRESLU,KCHARS(4),KLASTX,
     & KLASTY,KCOORD,KREPLN,KRELAB
      DIMENSION JRAY(6),JPOW2(5)
C * POWERS OF 2
      DATA JPOW2/16,1024,65536,4194304,268435456/
      CALL STIN (KREPLN,JREC,JRAY)
      INT=MOD(JRAY(KREPLN),16)
      JEND=KREPLN-1
      DO 10 J=1,JEND
      INT=INT+(JRAY(KREPLN-J)-32)*JPOW2(J)
10    CONTINUE
      IF (JRAY(KREPLN).LT.48) INT=-INT
      RETURN
      END
C
C----------SUBROUTINE--INTIN-----
C
      SUBROUTINE INTIN (INT)
C * INTIN INPUTS AND UNPACKS A 4100 FORMAT INT-REPORT
C * ARGUMENTS:
C * INT   - RETURNED INTEGER
      DIMENSION JRAY(3)
      CALL STIN (3,JREC,JRAY)
      INT=(JRAY(1)-32)*1024+(JRAY(2)-32)*16+MOD(JRAY(3),16)
      IF (JRAY(3).LT.48) INT=-INT
      RETURN
      END
C
C----------SUBROUTINE--INTOUT----
C
      SUBROUTINE INTOUT (INT)
C * INTOUT TRANSLATES AN INTEGER INTO 4100 FORMAT AND SENDS IT
C * ARGUMENTS:
C * INT    - INTEGER TO BE PACKED AND SENT
      COMMON /VALSYS/ KTERM,KMODE,KGFMAT,KRESLU,KCHARS(4),KLASTX,
     & KLASTY,KCOORD,KREPLN,KRELAB
      COMMON /VALIO/ KBAUDH,KBAUDT,KMCDEF,KBUFSZ,KBFLIM,KOUTPT,
     & KOUTBF(512),KINEND,KINPT,KINBUF(512),KEOMC1,KEOMC2,KIGDEL,
     & KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),KBMODE,KBMSAV,KHLENH,
     & KHEADH(10),KHLENT,KHEADT(10),KCONTH,KCONTT,KENDH,KENDT,KNXNOH,
     & KNXMTH(20),KNXNOT,KNXMTT(20),KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,
     & KPACKT,KBLENH,KBLENT,KBLINE,KSTRIP,KBLOKH,KEOPH,KEOMH,KEOFH,
     & KEOMT,KEOFT
```

```
      DIMENSION JCHARS(13)
C * SAVE LOI DATA
      JINT=IABS(INT)
      JCHARS(13)=MOD(JINT,16)+48
      IF (INT.LT.0) JCHARS(13)=JCHARS(13)-16
      JSTART=13
      JINT=JINT/16
C * COMPUTE HII'S
10    IF (JINT.EQ.0) GO TO 20
      JSTART=JSTART-1
      JCHARS(JSTART)=MOD(JINT,64)+64
      JINT=JINT/64
C * CHECK FOR (DEL) WHEN IGNORE-DEL IS ON
      IF (KIGDEL.EQ.0) GO TO 10
      IF (JCHARS(JSTART).NE.127) GO TO 10
      JCHARS(JSTART)=63
      JSTART=JSTART-1
      JCHARS(JSTART)=27
      GO TO 10
C * SEND PACKED STRING
20    CALL STOUT (14-JSTART,JCHARS(JSTART))
      RETURN
      END
C
C----------SUBROUTINE--INTRAY----
C
      SUBROUTINE INTRAY (ILEN,INTS)
C * INTRAY SENDS A PACKED INTEGER ARRAY
C * ARGUMENTS:
C * ILEN   - LENGTH OF ARRAY
C * INTS   - INTEGER ARRAY TO BE SENT
      DIMENSION INTS(1)
      CALL INTOUT (ILEN)
      JCOUNT=0
10    JCOUNT=JCOUNT+1
      IF (JCOUNT.GT.ILEN) RETURN
      CALL INTOUT (INTS(JCOUNT))
      GO TO 10
      RETURN
      END
C
C----------SUBROUTINE--IOEND-----
C
      SUBROUTINE IOEND
C * IOEND SHUTS DOWN THE I/O SYSTEM
      COMMON /VALIO/ KBAUDH,KBAUDT,KMCDEF,KBUFSZ,KBFLIM,KOUTPT,
     & KOUTBF(512),KINEND,KINPT,KINBUF(512),KEOMC1,KEOMC2,KIGDEL,
     & KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),KBMODE,KBMSAV,KHLENH,
     & KHEADH(10),KHLENT,KHEADT(10),KCONTH,KCONTT,KENDH,KENDT,KNXNOH,
     & KNXMTH(20),KNXNOT,KNXMTT(20),KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,
     & KPACKT,KBLENH,KBLENT,KBLINE,KSTRIP,KBLOKH,KEOPH,KEOMH,KEOFH,
     & KEOMT,KEOFT
```

```
C * TURN OFF BLOCK MODE
      CALL BLKEND (2)
C * TURN OFF PROMPT MODE IF IT WAS ON
      IF (KPRMOD.EQ.0) GO TO 10
      CALL CMDOUT (78,77)
      CALL INTOUT (0)
C * CLEAR THE OUTPUT BUFFER
10    CALL DUMP
      RETURN
      END
C
C----------SUBROUTINE--IRAYIN----
C
      SUBROUTINE IRAYIN (ILEN,INTRAY)
C * IRAYIN INPUTS AN INTEGER ARRAY IN 4100 REPORT FORMAT
C * ARGUMENTS:
C * ILEN   - RETURNED ARRAY LENGTH
C * INTRAY - RETURNED ARRAY
      DIMENSION INTRAY(1)
      CALL INTIN (ILEN)
      IF (ILEN.LE.0) RETURN
      DO 10 J=1,ILEN
      CALL INTIN (INTRAY(J))
10    CONTINUE
      RETURN
      END
C
C----------SUBROUTINE--KBCHIN----
C
      SUBROUTINE KBCHIN (ICHAR)
C * KBCHIN CALLS KYBDIN TO INPUT ONE ADE CHARACTER
      DIMENSION JRAY(1)
      CALL KYBDIN (1,JREC,JRAY)
      ICHAR=JRAY(1)
      RETURN
      END
C
C----------SUBROUTINE--KYBDIN----
C
      SUBROUTINE KYBDIN (IREQ,IREC,ISTRNG)
C * KYBDIN GETS KEYBOARD INPUT, EXITING AND REENTERING BLOCK MODE
C * ARGUMENTS:
C * IREQ   - NUMBER OF CHARACTERS REQUESTED
C * IREC   - NUMBER OF CHARACTERS RECEIVED <= IREQ
C * ISTRNG - STRING FOR ADE CHARACTERS
      COMMON /VALIO/ KBAUDH,KBAUDT,KMCDEF,KBUFSZ,KBFLIM,KOUTPT,
     & KOUTBF(512),KINEND,KINPT,KINBUF(512),KEOMC1,KEOMC2,KIGDEL,
     & KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),KBMODE,KBMSAV,KHLENH,
     & KHEADH(10),KHLENT,KHEADT(10),KCONTH,KCONTT,KENDH,KENDT,KNXNOH,
     & KNXMTH(20),KNXNOT,KNXMTT(20),KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,
```

```
     & KPACKT,KBLENH,KBLENT,KBLINE,KSTRIP,KBLOKH,KEOPH,KEOMH,KEOFH,
     & KEOMT,KEOFT
      DIMENSION ISTRNG(1)
      JBMODE=KBMODE
      CALL BLKEND (2)
      CALL STIN (IREQ,IREC,ISTRNG)
      IF (JBMODE.EQ.1) CALL BLOKGO
      RETURN
      END
C
C----------SUBROUTINE--MARKER----
C
      SUBROUTINE MARKER (IX,IY)
C * MARKER SENDS DRAW-MARKER COMMAND TO TERMINAL
C * ARGUMENTS:
C * IX,IY  - COORDINATES TO PLACE MARKER AT
      COMMON /VALSYS/ KTERM,KMODE,KGFMAT,KRESLU,KCHARS(4),KLASTX,
     & KLASTY,KCOORD,KREPLN,KRELAB
      IF (KGFMAT.EQ.1) GO TO 10
C * (FS) FORMAT
      KRELAB=1
      IF (KMODE.NE.0) CALL MRKMOD
      CALL XYOUT (IX,IY)
      KLASTX=IX
      KLASTY=IY
      KRELAB=0
      RETURN
C * (ESC) FORMAT
10    CALL CMDOUT (76,72)
      CALL XYOUT (IX,IY)
      RETURN
      END
C
C----------SUBROUTINE--MOVE------
C
      SUBROUTINE MOVE (IX,IY)
C * MOVE SENDS A MOVE COMMAND TO THE TERMINAL
C * ARGUMENTS:
C * IX,IY  - POINT TO MOVE TO
      COMMON /VALSYS/ KTERM,KMODE,KGFMAT,KRESLU,KCHARS(4),KLASTX,
     & KLASTY,KCOORD,KREPLN,KRELAB
      IF (KGFMAT.EQ.1) GO TO 10
C * (GS) STYLE
      CALL VECMOD
      CALL XYOUT (IX,IY)
      RETURN
C * (ESC) STYLE
10    CALL CMDOUT (76,70)
      CALL XYOUT (IX,IY)
      RETURN
      END
C
```

```
C----------SUBROUTINE--MRKMOD----
C
      SUBROUTINE MRKMOD
C * MRKMOD SENDS A (FS) TO THE TERMINAL
      COMMON /VALSYS/ KTERM,KMODE,KGFMAT,KRESLU,KCHARS(4),KLASTX,
     & KLASTY,KCOORD,KREPLN,KRELAB
      CALL CHOUT WITH (28)
      KMODE=0
      KRELAB=0
      RETURN
      END
C
C----------SUBROUTINE--REALIN----
C
      SUBROUTINE REALIN (RNUM)
C * REALIN INPUTS AND TRANSLATES A REAL-REPORT
      CALL INTIN (JMANT)
      CALL INTIN (JEXP)
      RNUM=FLOAT(JMANT)*2.**FLOAT(JEXP)
      RETURN
      END
C
C----------SUBROUTINE--RELOUT----
C
      SUBROUTINE RELOUT (RNUM)
C * RELPAK SENDS A PACKED REAL
C * ARGUMENTS:
C * RNUM   - REAL NUMBER TO BE OUTPUT
      JEXP=0
      SMANT=ABS(RNUM)
C * CHECK MANTISSA FOR SIZE, INTEGRALNESS
10    JMANT=IFIX(SMANT+.5)
      IF (JMANT.GT.16384) GO TO 20
      IF (ABS(SMANT-FLOAT(JMANT)).LT..0000305175) GO TO 20
      SMANT=SMANT*2.
      JEXP=JEXP-1
      GO TO 10
C * USE INTEGER ROUTINE TO SEND PACKED INTEGERS
20    IF (RNUM.LT.0) JMANT=-JMANT
      CALL INTOUT (JMANT)
      CALL INTOUT (JEXP)
      RETURN
      END
C
```

```
C----------SUBROUTINE--STIN------
C
      SUBROUTINE STIN (IREQST,IRECVD,ISTRNG)
C * STIN INPUTS CHARACTERS FROM A TERMINAL, SCANNING FOR EOF
C * ARGUMENTS:
C * IREQST - NUMBER OF CHARS REQUESTED
C * IRECVD - NUMBER OF CHARS RECEIVED
C * ISTRNG - CALLERS INPUT ARRAY
      COMMON /VALIO/ KBAUDH,KBAUDT,KMCDEF,KBUFSZ,KBFLIM,KOUTPT,
     & KOUTBF(512),KINEND,KINPT,KINBUF(512),KEOMC1,KEOMC2,KIGDEL,
     & KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),KBMODE,KBMSAV,KHLENH,
     & KHEADH(10),KHLENT,KHEADT(10),KCONTH,KCONTT,KENDH,KENDT,KNXNOH,
     & KNXMTH(20),KNXNOT,KNXMTT(20),KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,
     & KPACKT,KBLENH,KBLENT,KBLINE,KSTRIP,KBLOKH,KEOPH,KEOMH,KEOFH,
     & KEOMT,KEOFT
      DIMENSION ISTRNG(1)
C * ZERO THE RETURN LENGTH
      IRECVD=0
C * BRANCH IF BUFFER HAS CHARACTERS IN IT
      IF (KINEND.GT.KINPT) GO TO 40
C * BRANCH TO GET MORE INPUT VIA BLOCK MODE
      IF (KBMODE.EQ.1) GO TO 30
C * DUMP THE OUTPUT BUFFER TO BE SURE COMMANDS ARE SENT
      CALL DUMP
C * SIMULATE PROMPT FOR NON-PROMPTING SYSTEMS
C * REMOVE THIS LINE FOR PROMPTING SYSTEMS
      IF (KPRMOD.GT.0) CALL ADEOUT (KPRLEN,KPSTRG)
C * GET LINE OF INPUT FROM TERMINAL
      CALL ADEIN (KINEND,KINBUF)
      KINPT=0
      KEOFT=0
      IF (KINEND.EQ.0) GO TO 60
C * SCAN FOR EOF STRING
      IF (KEOFLN.NE.KINEND) GO TO 40
      JINPT=0
10    IF (JINPT.GE.KEOFLN) GO TO 20
      JINPT=JINPT+1
      IF (JINPT.GT.KINEND) GO TO 40
      IF (KINBUF(JINPT).NE.KEOFST(JINPT)) GO TO 40
      GO TO 10
20    KEOFT=1
      KINEND=0
      GO TO 60
C * GET INPUT VIA BLOCK MODE (MAY SET KEOFT)
30    KEOMH=1
      CALL BLOKIO
      KEOMH=0
      KINPT=0
```

```
C * MOVE CHARACTERS FROM INPUT BUFFER TO USER ARRAY
40    JTOMOV=MIN0(IREQST,KINEND-KINPT)
50    IF (IRECVD.GE.JTOMOV) GO TO 60
      IRECVD=IRECVD+1
      KINPT=KINPT+1
      ISTRNG(IRECVD)=KINBUF(KINPT)
      GO TO 50
C * PAD WITH BLANKS IF NEEDED
60    IF (KEOFT.EQ.1) RETURN
      JRECVD=IRECVD
70    IF (JRECVD.GE.IREQST) RETURN
      JRECVD=JRECVD+1
      ISTRNG(JRECVD)=32
      GO TO 70
      END
C
C----------SUBROUTINE--STOUT-----
C
      SUBROUTINE STOUT (ILEN,ISTRNG)
C * STOUT IS THE GENERAL OUTPUT ROUTINE
      COMMON /VALIO/ KBAUDH,KBAUDT,KMCDEF,KBUFSZ,KBFLIM,KOUTPT,
     & KOUTBF(512),KINEND,KINPT,KINBUF(512),KEOMC1,KEOMC2,KIGDEL,
     & KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),KBMODE,KBMSAV,KHLENH,
     & KHEADH(10),KHLENT,KHEADT(10),KCONTH,KCONTT,KENDH,KENDT,KNXNOH,
     & KNXMTH(20),KNXNOT,KNXMTT(20),KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,
     & KPACKT,KBLENH,KBLENT,KBLINE,KSTRIP,KBLOKH,KEOPH,KEOMH,KEOFH,
     & KEOMT,KEOFT
      DIMENSION ISTRNG(1)
C * SET NUMBER ALREADY SENT TO ZERO
      JSENT=0
C * DUMP BUFFER IF ILEN IS NOT POSITIVE
      IF (ILEN.LE.0) GO TO 20
C * ADD CHARS TO BUFFER UNTIL FULL OR DONE
10    IF (KOUTPT.GE.KBUFSZ) GO TO 20
      JSENT=JSENT+1
      KOUTPT=KOUTPT+1
      KOUTBF(KOUTPT)=ISTRNG(JSENT)
      IF (JSENT.GE.ILEN) RETURN
      GO TO 10
C * DUMP THE BUFFER
20    IF (KBMODE.EQ.1) GO TO 30
      CALL ADEOUT (KOUTPT,KOUTBF)
      GO TO 40
C * DO BLOCK EXCHANGE
30    CALL BLOKIO
C * BUFFER EMPTY NOW
40    KOUTPT=0
C * DO REST OF STRING IF THERE IS SOME LEFT
      IF (JSENT.LT.ILEN) GO TO 10
      RETURN
      END
C
```

```
C----------SUBROUTINE--VECMOD----
C
      SUBROUTINE VECMOD
C * VECMOD SENDS A (GS) TO THE TERMINAL
      COMMON /VALSYS/ KTERM,KMODE,KGFMAT,KRESLU,KCHARS(4),KLASTX,
     & KLASTY,KCOORD,KREPLN,KRELAB
      IF (KMODE.EQ.0) CALL CHOUT (31)
      CALL CHOUT (29)
      KMODE=2
      KRELAB=0
      RETURN
      END
C
C----------SUBROUTINE--XYIN------
C
      SUBROUTINE XYIN (IFORMT,IX,IY)
C * XYIN INPUTS AND UNPACKS TERMINAL-TO-HOST X-Y COORDINATE
C * ARGUMENTS:
C * IFORMT - INPUT FORMAT
C *          1 - 4100 OR 4953 12-BIT FORMAT
C *          2 - 4010 FORMAT
C *          3 - 4953 10-BIT FORMAT
C * IX,IY  - RETURNED COORDINATES
      COMMON /VALSYS/ KTERM,KMODE,KGFMAT,KRESLU,KCHARS(4),KLASTX,
     & KLASTY,KCOORD,KREPLN,KRELAB
      DIMENSION JRAY(5)
      IF (KCOORD.EQ.1) GO TO 20
      IF (IFORMT.GT.1) GO TO 10
C * 4100 AND 4953/12-BIT FORMAT
      CALL REPIN (5,JREC,JRAY)
      IX=(JRAY(4)-32)*128+(JRAY(5)-32)*4+JRAY(2)-JRAY(2)/4*4
      IY=(JRAY(1)-32)*128+(JRAY(3)-32)*4+(JRAY(2)-32)/4
      RETURN
C * 4010 FORMAT
10    CALL REPIN (4,JREC,JRAY)
      IX=((JRAY(1)-32)*32+JRAY(2)-32)*4
      IY=((JRAY(3)-32)*32+JRAY(4)-32)*4
      IF (IFORMT.EQ.2) RETURN
C * 4953/10-BIT FORMAT
      JTEMP=IX
      IX=IY
      IY=JTEMP
      RETURN
C * 32-BIT FORMAT
20    CALL INTCIN (IX)
      CALL INTCIN (IY)
      RETURN
      END
C
```

```
C----------SUBROUTINE--XYOUT-----
C
      SUBROUTINE XYOUT (IX,IY)
C * XYOUT SENDS OPTIMIZED X-Y COORDINATE STRING
C * ARGUMENTS:
C * IX,IY  - X,Y COORDINATES TO BE TRANSLATED
      COMMON /VALSYS/ KTERM,KMODE,KGFMAT,KRESLU,KCHARS(4),KLASTX,
     & KLASTY,KCOORD,KREPLN,KRELAB
      COMMON /VALIO/ KBAUDH,KBAUDT,KMCDEF,KBUFSZ,KBFLIM,KOUTPT,
     & KOUTBF(512),KINEND,KINPT,KINBUF(512),KEOMC1,KEOMC2,KIGDEL,
     & KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),KBMODE,KBMSAV,KHLENH,
     & KHEADH(10),KHLENT,KHEADT(10),KCONTH,KCONTT,KENDH,KENDT,KNXNOH,
     & KNXMTH(20),KNXNOT,KNXMTT(20),KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,
     & KPACKT,KBLENH,KBLENT,KBLINE,KSTRIP,KBLOKH,KEOPH,KEOMH,KEOFH,
     & KEOMT,KEOFT
      DIMENSION JCHARS(7)
C * BRANCH FOR 32 BIT COORDINATES
      IF (KCOORD.EQ.1) GO TO 60
C * FIRST BRING COORDINATES INTO VALID RANGE
      JX=MIN0(4095,MAX0(0,IX))
      JY=MIN0(4095,MAX0(0,IY))
C * CALCULATE 10-BIT RESOLUTION CHARACTERS
      JTEMP1=JY/128+32
      JTEMP3=JY/4-JY/128*32+96
      JTEMP4=JX/128+32
      JTEMP5=JX/4-JX/128*32+64
C * INITIALIZE ARRAY LENGTH
      JLEN=0
C * SEE IF HI-Y NEEDED
      IF (JTEMP1.EQ.KCHARS(1)) GO TO 10
C * INSERT HI-Y
      JLEN=1
      JCHARS(1)=JTEMP1
      KCHARS(1)=JTEMP1
C * SEE IF 12-BIT RESOLUTION
10    IF (KRESLU.NE.12) GO TO 20
C * COMPUTE EXTRA-LO-Y
      JTEMP2=(JY-JY/4*4)*4+(JX-JX/4*4)+112
C * SEE IF EXTRA-LO-Y NEEDED
      IF (JTEMP2.EQ.KCHARS(2)) GO TO 20
C * INSERT EXTRA-LO-Y
      JLEN=JLEN+1
      JCHARS(JLEN)=JTEMP2
      KCHARS(2)=JTEMP2
C * EXPAND EXTRA-LO-Y TO (ESC)(?) IF NECESSARY
      IF (JTEMP2.NE.127 .OR. KIGDEL.EQ.0) GO TO 30
      JCHARS(JLEN)=27
      JLEN=JLEN+1
      JCHARS(JLEN)=63
      GO TO 30
```

```
C * SEE IF LO-Y NEEDED
20    IF (JTEMP3.NE.KCHARS(3)) GO TO 30
      IF (JTEMP4.EQ.KCHARS(4)) GO TO 50
C * INSERT LO-Y
30    JLEN=JLEN+1
      JCHARS(JLEN)=JTEMP3
      KCHARS(3)=JTEMP3
C * EXPAND LO-Y TO (ESC)(?) IF NECESSARY
      IF (JTEMP3.NE.127 .OR. KIGDEL.EQ.0) GO TO 40
      JCHARS(JLEN)=27
      JLEN=JLEN+1
      JCHARS(JLEN)=63
C * SEE IF HI-X NEEDED
40    IF (JTEMP4.EQ.KCHARS(4)) GO TO 50
      JLEN=JLEN+1
      JCHARS(JLEN)=JTEMP4
      KCHARS(4)=JTEMP4
C * ALWAYS INCLUDE LO-X
50    JLEN=JLEN+1
      JCHARS(JLEN)=JTEMP5
C * SEND THE STRING
      CALL STOUT (JLEN,JCHARS)
      RETURN
C * 32-BIT COORDINATES
60    IF (KRELAB.EQ.1) GO TO 70
      CALL INTOUT (IX)
      CALL INTOUT (IY)
      RETURN
C * RELATIVE XY'S
70    CALL INTOUT (IX-KLASTX)
      CALL INTOUT (IY-KLASTY)
      RETURN
      END
C
```

```
C----------SUBROUTINE--XYRAY-----
C
      SUBROUTINE XYRAY (ILEN,IX,IY)
C * XYRAY SENDS AN ARRAY OF <XY> COORDINATES TO THE TERMINAL
C * ARGUMENTS:
C * ILEN    - LENGTH OF ARRAY TO BE SENT
C * IX      - X-ARRAY
C * IY      - Y-ARRAY
      COMMON /VALSYS/ KTERM,KMODE,KGFMAT,KRESLU,KCHARS(4),KLASTX,
     & KLASTY,KCOORD,KREPLN,KRELAB
      DIMENSION IX(1),IY(1)
      CALL INTOUT (ILEN)
      JLASTX=KLASTX
      JLASTY=KLASTY
      JCOUNT=0
10    JCOUNT=JCOUNT+1
      IF (JCOUNT.GT.ILEN) GO TO 20
      CALL XYOUT (IX(JCOUNT),IY(JCOUNT))
      KLASTX=IX(ICOUNT)
      KLASTY=IY(JCOUNT)
      KRELAB=1
      GO TO 10
20    KRELAB=0
      KLASTX=JLASTX
      KLASTY=JLASTY
      RETURN
      END
   ========
```

Appendix D

COLOR COORDINATE SYSTEMS

COLOR

The visual sensation of color is a complex response to many subtle visual stimuli. Color perception involves the physiological response of the eye to differing wavelengths of light and the psychological response of the human mind to the patterns that it perceives.

Color and its study have been a preoccupation of artists for many centuries. By combining colored pigments and seeing the changes in color, artists evolved an empirical color system that enabled them to consistently achieve predictable results. Artists found that they could produce a wide range of colors by mixing three "pure" colors: red, yellow, and blue. These colors, since they seemed to be the fundamental colors, came to be called *primarys*.

The discovery that white light from the sun, passed through a glass prism, contained many different colors, added new complications. Here were colors in their purest form, colors that were not made from mixtures of the primaries. These prismatic colors, colors of the rainbow, we now recognize as the response of the eye to different wavelengths of light.

Experimenters found that different mixtures of colored light would evoke sensations of color not present in either of the colors used to form the mixture. Thus another color system evolved. Three colors, red, green, and blue were found to be most useful and were called the *light primaries*.

Analysis of the light reflected from the artist's primaries revealed that each primary reflected light mostly within a narrow range of wavelengths. When these primaries were mixed, less light was reflected and the colors appeared darker. Thus, the artist's system of primary colors became known as the subtractive color system, while the physicist's system of adding colored light became known as the additive color system.

The synthesis of artificial pigments, with colors much more pure than the natural pigments, changed the subtractive color system slightly. The artist's blue became cyan, the artist's red became magenta, and yellow remained unchanged. The subtractive color primaries are now cyan, magenta, and yellow.

THE HLS COLOR CONE

One of the first attempts to summarize color knowledge was the *color wheel*. Artists found that by arranging the three primaries at 120° intervals around the periphery of a circle, intermediate shades fit between the primaries. A color can be specified as a rotation from a point on the color wheel. The *hue* of a color comes from the rotation of the color from the color wheel. Figure D-1 shows how colors are arranged around the color wheel.

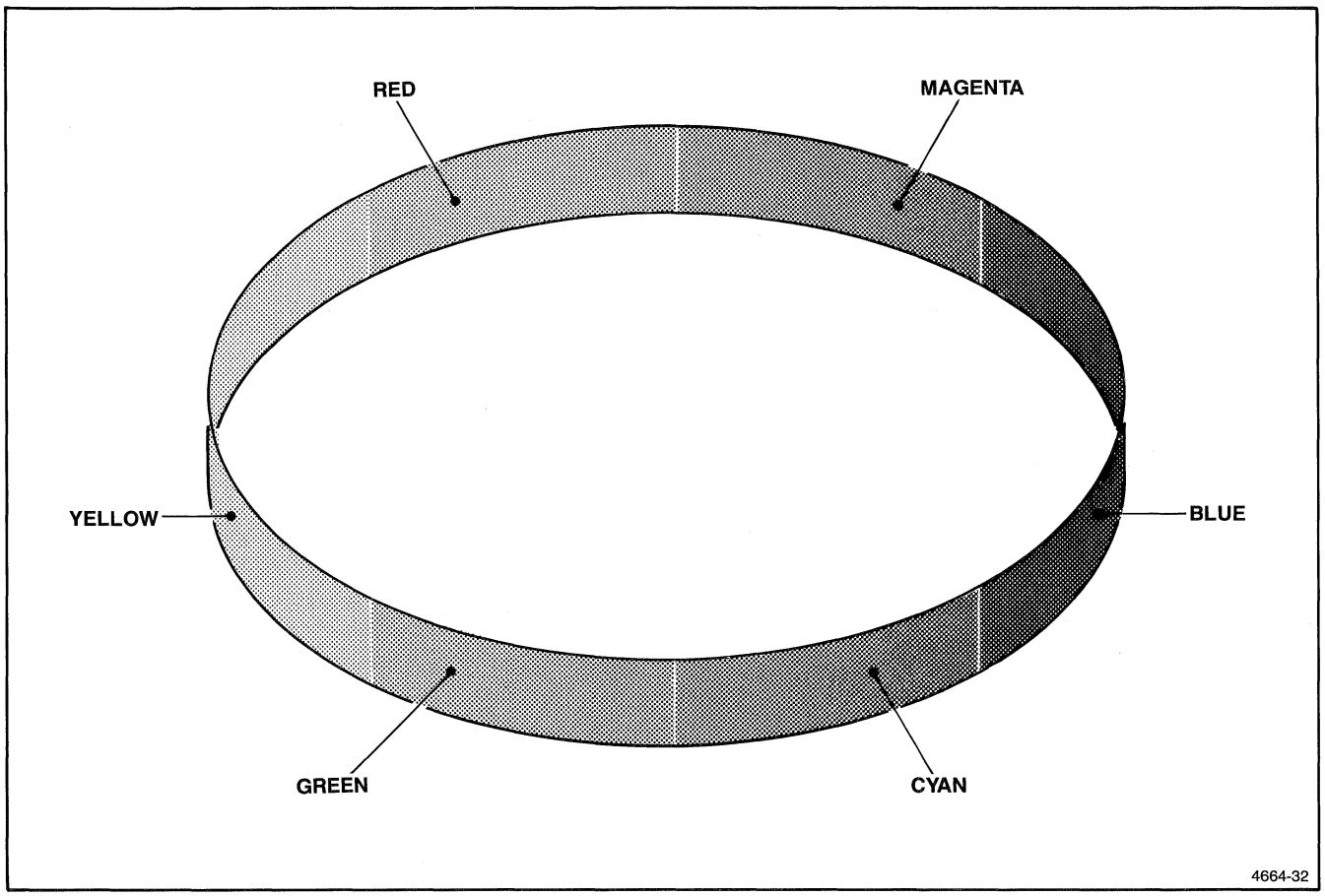


Figure D-1. Colors Arranged in a Wheel.

In addition to the basic color sensation, colors have *saturation*, or intensity. Two colors may have the same hue, but one may appear more vibrant and the other more muted. For example, two blues may have the same hue, but one can be very intense, while the other is nearly gray. By specifying saturation as the distance from the center of the color wheel, we obtain another specifier of a particular color. Figure D-2 shows a color wheel when we add saturation.

While hue and saturation specify a great deal about a color, two limiting cases are left out. Nowhere on the color circle is there a place for black and white. In addition, two colors may have the same hue and intensity, but one is darker than the other. The third specifier we need for color is *lightness*.

If we extend a line perpendicular to the plane of the color wheel, through the center of the circle, we can place white at one extreme and black at the other. Pairs of colors that share the same hue and saturation then go above and below the plane of the color wheel. This third dimension of color allows us to specify any color in a three-dimensional cone. Figure D-3 shows how the axis of this space becomes a gray scale.

A moment or two of reflection should convince you that the volume specified by all colors is a double ended cone. This color cone is the primary tool for visualizing the HLS color system. Figure D-4 shows the HLS color cone.

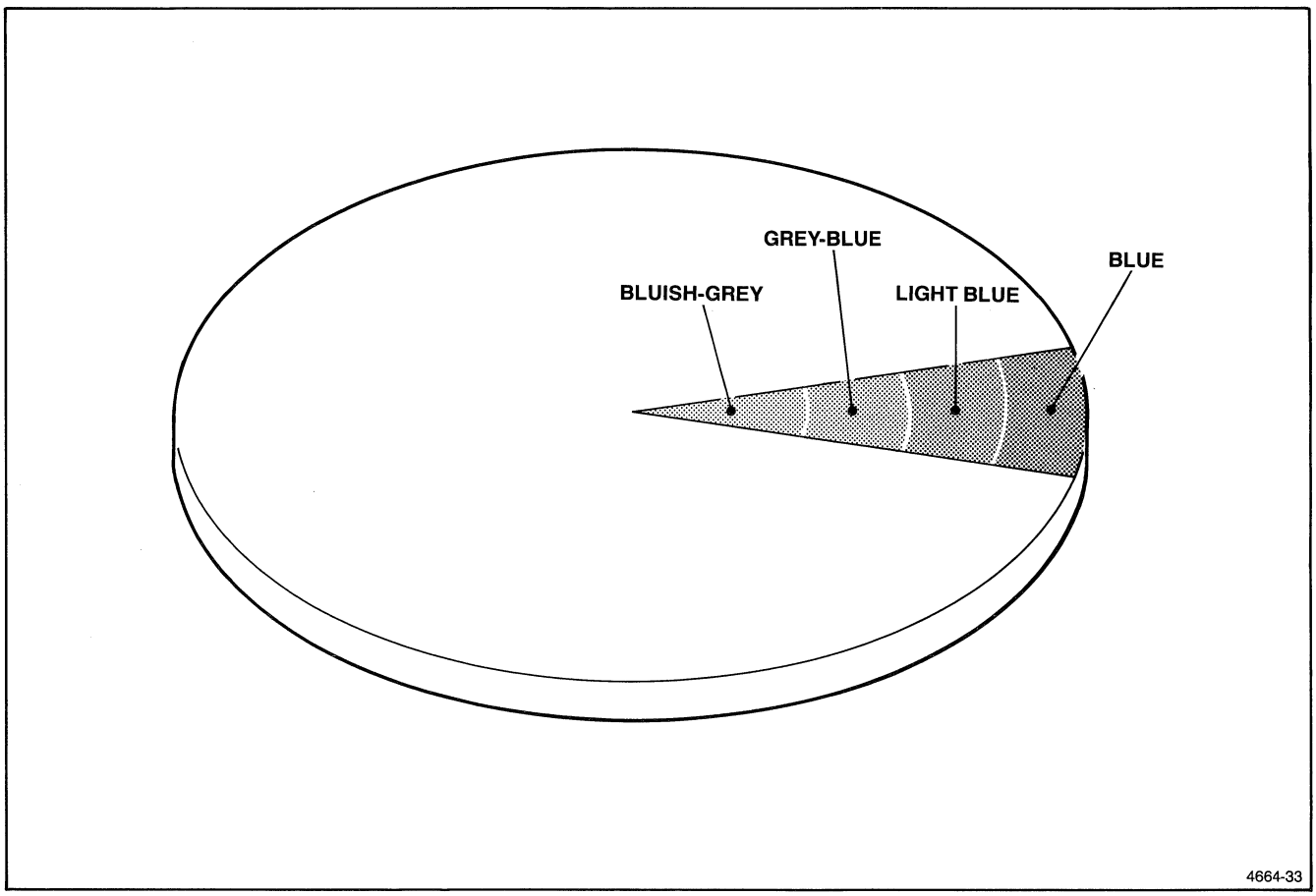


Figure D-2. A Color Wheel With Saturation.

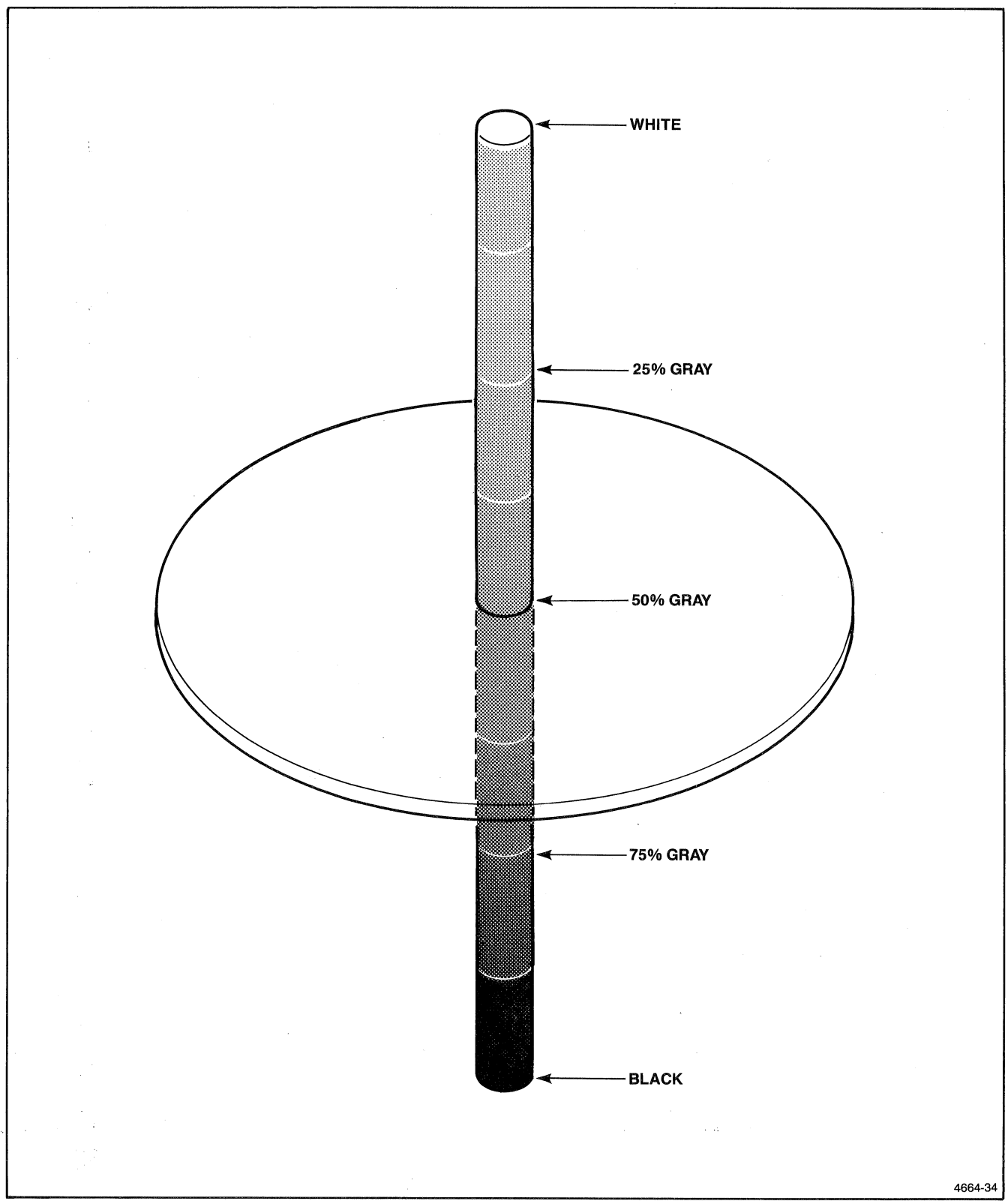


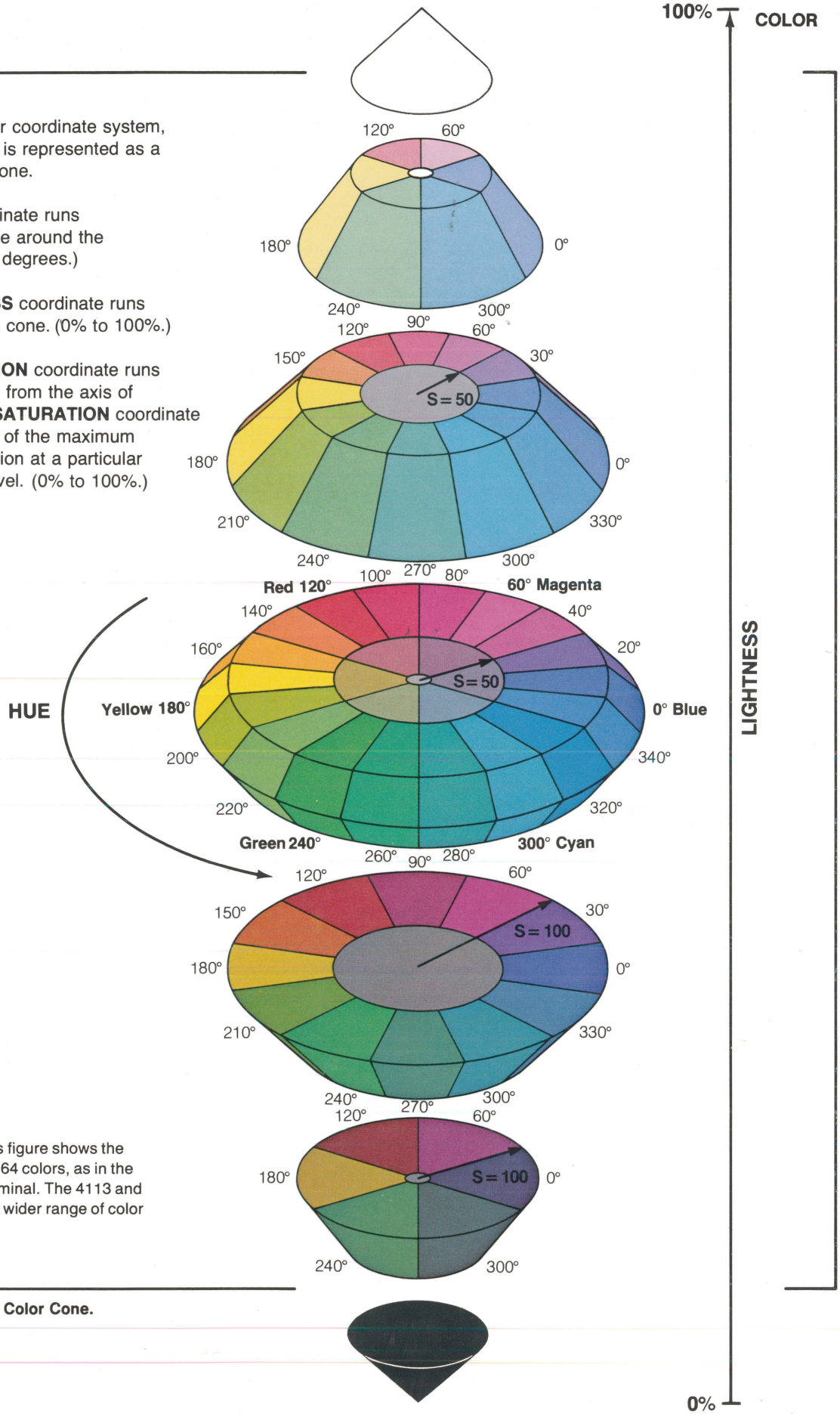
Figure D-3. Adding a Gray Scale to the Color Wheel.

In the **HLS** color coordinate system, the color space is represented as a double-ended cone.

The **HUE** coordinate runs counterclockwise around the cone. (0 to 360 degrees.)

The **LIGHTNESS** coordinate runs vertically up the cone. (0% to 100%.)

The **SATURATION** coordinate runs radially outward from the axis of the cone. The **SATURATION** coordinate is a percentage of the maximum possible saturation at a particular **LIGHTNESS** level. (0% to 100%.)



NOTE: For clarity, this figure shows the cone divided into only 64 colors, as in the TEKTRONIX 4027 terminal. The 4113 and 4115 terminals have a wider range of color mixtures.

Figure D-4. The HLS Color Cone.

TEKTRONIX COLOR STANDARD

Overview:

The world of color is filled with ambiguous terminology, i.e. intensity, purity, value, etc. Many color users feel that "color theory" is a prerequisite to operating color systems; T.V., Videotaping, Photography, Computer Graphics.

In order to end this confusion, Tektronix has developed a color language and function based on human engineering, rather than machine engineering. Below is a description of this system, which will provide a clear and concise means for understanding how color is defined and how our syntax was derived.

Color Concepts:

Color selection is specified by hue, lightness and saturation which is the HLS method. The definitions are as follows:

Hue: The characteristic associated with a color name such as red, yellow, green, blue, etc. Hue is a gradation of color advanced by degrees, thus represented as an angle from 0 to 360.

Lightness: The characteristic that allows the color to be ranked on a scale from dark to light. Lightness is expressed as a parameter ranging from 0 to 100% with black being 0 (bottom of cone) and white being 100% (top of cone).

Saturation: The characteristic which describes the extent to which a color differs from a gray of the same lightness. Saturation is expressed as percentage, ranging from 0% (maximum white content at that lightness level) to 100% (full saturated).

Geometrically, colors can be described in terms of a double cone.
Variations in lightness are represented along the axis, with white at the apex of the cone and black at the opposite apex. Variations in saturation are represented by radial distances from the lightness axis, in constant lightness planes. Hue is represented as an angular quantity from a known reference point.

In the HLS system, color is a vector quantity. Hue is the angle formed by rotating the vector around the axis of the double ended cone, while blue is the reference. A hue of 0° (or 360°) corresponds to blue, 120° to red and 240° to green, while intermediate shades correspond to intermediate rotations.

Lightness is the position of a vector along the axis of the cone. A lightness of 0% is black and a lightness of 100% is white. (At lightness 0% or 100%, saturation and hue are irrelevant.)

Saturation is the length of the vector from the cone axis. A saturation of 0% is a shade of gray, while a saturation of 100% gives the most intense possible color having that hue and brightness.

The HLS system is easy to use and gives a readily acessible feel for color. If your application demands color matching, it is easiest to change hue to approximate the color, adjust the saturation to approximate it, and move lightness close. This gives a good starting point from which to fine tune.

While the HLS system gives a good intuitive "feel" for a programmer or operator when attempting to specify colors, it suffers some limitations in specifying colors, particularly with colors of 50% lightness and 100% saturation. Although you might reasonably expect these colors (which lie on the equator of the color cone) to emit identical amounts of light when measured on the screen, their actual light emission will vary. For example, cyan (300,50,100) is twice as bright as green (240,50,100) or blue (0,50,100), although each is at 50% brightness and 100% saturation. There is no simple relationship between HLS and the colors available on the terminal. HLS allows you to specify approximately 3,600,000 colors, while the 4113 can display only 4096 and the 4115 can address over 16 million.

RGB AND CMY — THE COLOR CUBE

RGB and CMY stand for the additive color system and the subtractive color system respectively. RGB is formed from the initials of the three light primaries: red, green, and blue. CMY is formed from the initials of the three pigment primaries: cyan, magenta, and yellow.

The RGB and CMY systems treat color more as something derived from a recipe rather than a collection of qualities such as hue, lightness, and saturation. Both systems specify colors as a mixture of three primaries, with a particular color specified as a particular mix.

In either color system, if we assign each primary to a Cartesian axis, we define a volume of possible color mixes. If each axis includes all possible values of that primary, the solid formed by the three axes is a cube containing all possible mixes of these colors. Figure D-5 shows how the RGB and CMY systems form color cubes.

In the RGB system, the origin of the coordinate space, Point (0,0,0) is black. In the CMY system, the origin, Point (0,0,0) is white.

The two systems interlock in a very interesting manner. Each of the primaries in the CMY system is the result of adding two primaries in the RGB system and vice versa. Adding 100% of the three RGB primaries together produces white, which is equivalent to 0% each of the three CMY primaries. Adding 100% of the three CMY primaries together produces black, which is equivalent to 0% each of the RGB primaries.

The combined color cube for the RGB and CMY systems is shown in Figure D-6.

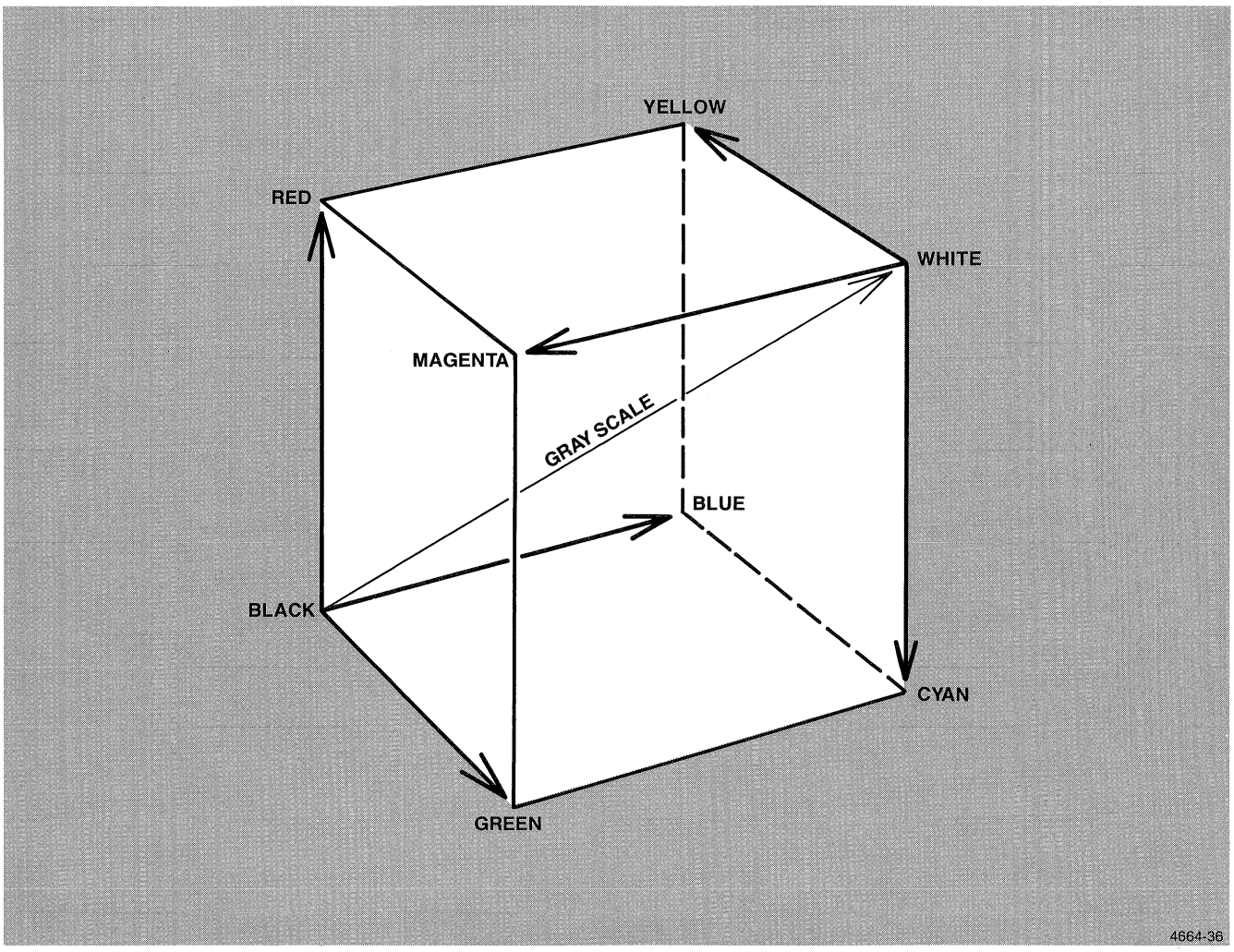


Figure D-6. The Combined RGB and CMY Color Cube.

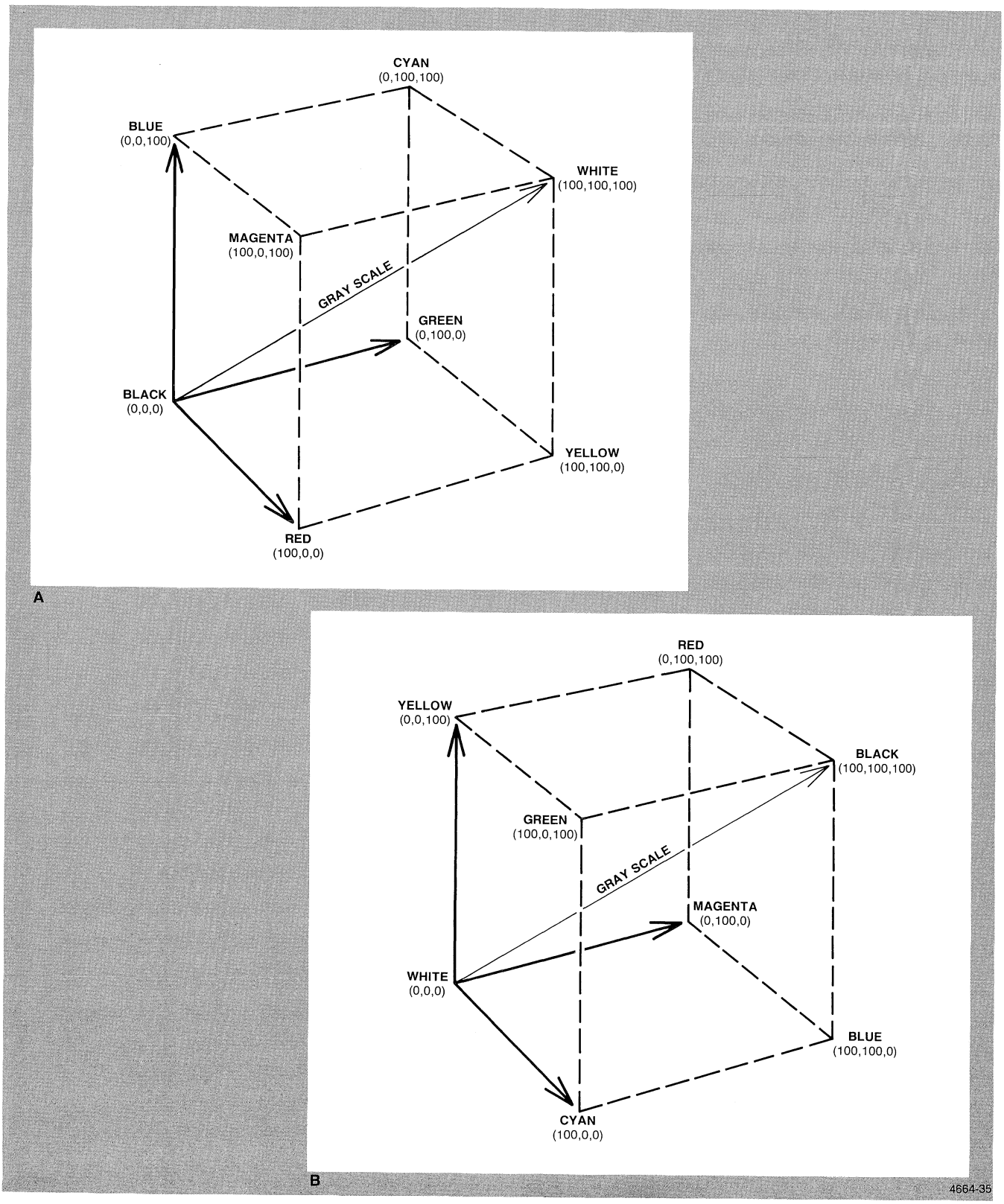


Figure D-5. The RGB and CMY Color Coordinates.

Every color a 4100 Series terminal is capable of producing is contained within the volume of the color cube. All 4100 Series terminals specify colors as percentages of from 0% through 100% of the RGB or CMY primaries. In addition, the 4115 Machine RGB mode allows you to specify RGB values from 0 through 255 where 255 is equivalent to 100%.

The gray scale for RGB and CMY runs through the center of the color cube from black through white. Each shade of gray is formed from equal portions of the three primaries of either the RGB or CMY system. Figure D-7 shows the gray scale in a cutaway color cube.

The CMY and RGB systems are corrected so that equal increments show approximately equal color changes. This means that changing an area colored red by 25% seems to give the same amount of color change as changing an area colored yellow by 25%.

Since the response of the human eye to color is extremely nonlinear, the changes of color level do not bear a linear relationship with the intesity of the color guns. The only color system where this is extremely significant is in machine RGB in the 4115. The machine RGB levels translate directly into gun levels, so you are working with uncorrected color data. Each gun is controlled by a single 8-bit byte, giving 256 possible gun levels. Three guns, each with 256 possible levels, address over 16 million points in color space. These points, however, in the less sensitive wavelengths, are indistiguishable to the eye. Although the terminal can address over 16 million colors, your eye can distinguish less.

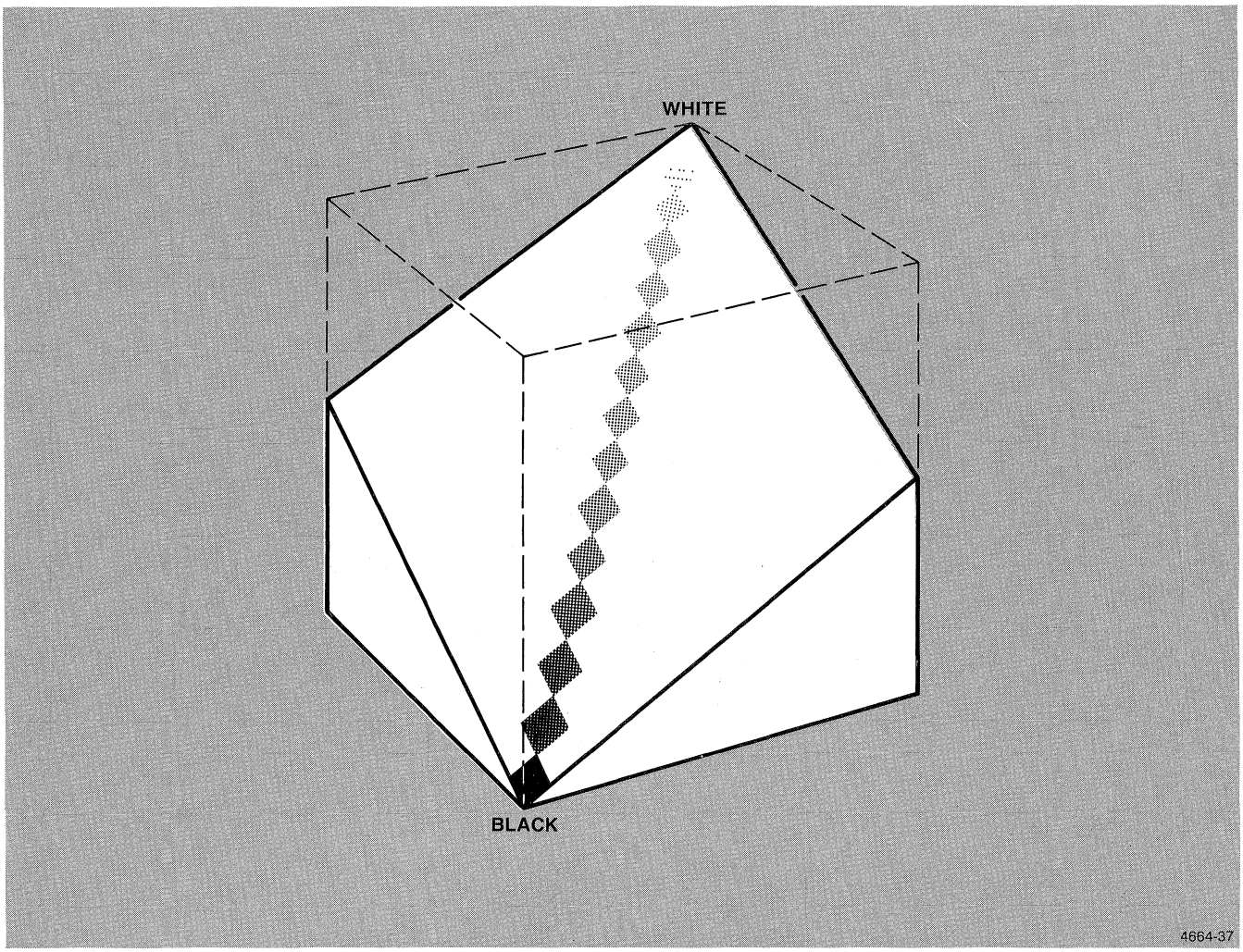


Figure D-7. Gray Scale in a Cutaway Color Cube.

Appendix E

CHARACTER INTERPRETATION BY TEK AND ANSI PARSERS

The following table summarizes how the TEK and ANSI parsers interpret incoming characters. Entries are abbreviated in many cases, and a key to abbreviations follows the table.

Table E-1
CHARACTER INTERPRETATION BY TEK AND ANSI PARSERS

CHAR	TEK Parser								ANSI Parser		
	Alpha	Vector	Marker	LCE-T	20C	*int*	*xy*	*char*	alpha	LCE-A	CSI
N_U				E_C							
S_H				no-op							
S_X				no-op							
E_X				no-op							
E_T				no-op							
E_Q				CMD-1							
A_K				no-op							
B_L	B_L	B_L [1]	B_L	B_L					B_L	B_L	B_L
B_L	B_L			B_L					B_L	B_L	B_L
H_T	H_T			H_T					H_T	H_T	H_T
L_F	L_F			E_C					L_F	L_F	L_F
V_T	V_T			V_T					V_F	V_T	V_T
F_F				PAGE					F_F	F_F	F_F
C_R	C_R	C_R [2]	C_R [2]	E_C				C_R	C_R	C_R	
S_O				CMD-2					INV-G1	INV-G1	INV-G1
S_I				CMD-3					INV-G0	INV-G0	INV-G0
D_L				no-op							
D_1				no-op							
D_2				no-op							
D_3				no-op							
D_4				no-op							
N_K				no-op							
S_Y				no-op							
E_B				CMD-4							
C_N				CMD-5						halt	halt
E_M				no-op							
S_B				CMD-6						halt	halt

Table E-1 (cont)
CHARACTER INTERPRETATION BY TEK AND ANSI PARSERS

CHAR	TEK Parser								ANSI Parser		
	Alpha	Vector	Marker	LCE-T	20C	*int*	*xy*	*char*	alpha	LCE-A	CSI
E_C	E_C	E_C	E_C	E_C	E_C	E_C 3	E_C 3	E_C 3	E_C 3	E_C	E_C
F_S	F_S	F_S	F_S	F_S	F_S	F_S 3	F_S 3	F_S 3	F_S 3		
G_S	G_S	G_S		G_S	G_S 3	G_S 3	G_S 3	G_S 3			
R_S				no-op							
U_S	U_S	U_S		U_S	U_S 3	U_S 3	U_S 3	U_S 3			
S_P	S_P	HiX/Y	HiX/Y	no-op	error	LoI–	HiX/Y	S_P	S_P	no-op	error
!	!	HiX/Y	HiX/Y	CMD-7	error4	LoI–	HiX/Y	!	!	no-op4	error
"	"	HiX/Y	HiX/Y	no-op	error	LoI–	HiX/Y	"	"	no-op	error
#	#	HiX/Y	HiX/Y	no-op	error	LoI–	HiX/Y	#	#	no-op	error
$	$	HiX/Y	HiX/Y	no-op	error	LoI–	HiX/Y	$	$	no-op	error
%	%	HiX/Y	HiX/Y	CMD-8	error	LoI–	HiX/Y	%	%	CMD-8	error
&	&	HiX/Y	HiX/Y	no-op	error	LoI–	HiX/Y	&	&	no-op	error
'	'	HiX/Y	HiX/Y	no-op	error	LoI–	HiX/Y	'	'	no-op	error
(	(	HiX/Y	HiX/Y	no-op	error	LoI–	HiX/Y	(	(	SCS	error
)	)	HiX/Y	HiX/Y	no-op	error	LoI–	HiX/Y	)	)	SCS	error
*	*	HiX/Y	HiX/Y	no-op	error	LoI–	HiX/Y	*	*	no-op	error
+	+	HiX/Y	HiX/Y	no-op	error	LoI–	HiX/Y	+	+	no-op	error
,	,	HiX/Y	HiX/Y	no-op	error	LoI–	HiX/Y	,	,	no-op	error
–	–	HiX/Y	HiX/Y	no-op	error	LoI–	HiX/Y	–	–	no-op	error
.	.	HiX/Y	HiX/Y	no-op	error	LoI–	HiX/Y	.	.	no-op	error
/	/	HiX/Y	HiX/Y	no-op	error	LoI–	HiX/Y	/	/	no-op	error
0	0	HiX/Y	HiX/Y	no-op	error	LoI+	HiX/Y	0	0	no-op	0
1	1	HiX/Y	HiX/Y	no-op	error	LoI+	HiX/Y	1	1	no-op	1
2	2	HiX/Y	HiX/Y	no-op	error	LoI+	HiX/Y	2	2	no-op	2
3	3	HiX/Y	HiX/Y	no-op	error	LoI+	HiX/Y	3	3	no-op	3
4	4	HiX/Y	HiX/Y	no-op	error	LoI+	HiX/Y	4	4	no-op	4
5	5	HiX/Y	HiX/Y	no-op	error	LoI+	HiX/Y	5	5	no-op	5
6	6	HiX/Y	HiX/Y	no-op	error	LoI+	HiX/Y	6	6	no-op	6
7	7	HiX/Y	HiX/Y	no-op	error	LoI+	HiX/Y	7	7	TEKSC	7
8	8	HiX/Y	HiX/Y	CMD-9	error	LoI+	HiX/Y	8	8	TEKRC	8
9	9	HiX/Y	HiX/Y	CMD-9	error	LoI+	HiX/Y	9	9	no-op	9
:	:	HiX/Y	HiX/Y	CMD-9	error	LoI+	HiX/Y	:	:	no-op	error
;	;	HiX/Y	HiX/Y	CMD-9	error	LoI+	HiX/Y	;	;	no-op	;

Table E-1 (cont)
CHARACTER INTERPRETATION BY TEK AND ANSI PARSERS

CHAR	TEK Parser								ANSI Parser		
	Alpha	Vector	Marker	LCE-T	20C	*int*	*xy*	*char*	alpha	LCE-A	CSI
<	<	HiX/Y	HiX/Y	no-op	error	LoI+	HiX/Y	<	<	no-op	<
=	=	HiX/Y	HiX/Y	no-op	error	LoI+	HiX/Y	=	=	no-op	=
>	>	HiX/Y	HiX/Y	no-op	error	LoI+	HiX/Y	>	>	no-op	>
?	?	HiX/Y	HiX/Y	$^{D}_{T}$	error	LoI+	HiX/Y	?	?	no-op	?
@	@	LoX	LoX	no-op	error	HiI	LoX	@	@	no-op	ICH
A	A	LoX	LoX	no-op	CMD-11	HiI	LoX	A	A	no-op	CUU
B	B	LoX	LoX	no-op	CMD-11	HiI	LoX	B	B	no-op	CUD
C	C	LoX	LoX	no-op	CMD-11	HiI	LoX	C	C	no-op	CUF
D	D	LoX	LoX	no-op	CMD-11	HiI	LoX	D	D	IND	CUB
E	E	LoX	LoX	no-op	CMD-11	HiI	LoX	E	E	NEL	error
F	F	LoX	LoX	no-op	CMD-11	HiI	LoX	F	F	no-op	
G	G	LoX	LoX	no-op	CMD-11	HiI	LoX	G	G	no-op	error
H	H	LoX	LoX	no-op	CMD-11	HiI	LoX	H	H	HTS	CUP
I	I	LoX	LoX	20C	CMD-11	HiI	LoX	I	I	no-op	CHT
J	J	LoX	LoX	20C	CMD-11	HiI	LoX	J	J	no-op	ED
K	K	LoX	LoX	20C	CMD-11	HiI	LoX	K	K	no-op	EL
L	L	LoX	LoX	20C	CMD-11	HiI	LoX	L	L	no-op	IL
M	M	LoX	LoX	20C	CMD-11	HiI	LoX	M	M	RI	DL
N	N	LoX	LoX	20C	CMD-11	HiI	LoX	N	N	no-op	error
O	O	LoX	LoX	20C	CMD-11	HiI	LoX	O	O	no-op	error
P	P	LoX	LoX	20C	CMD-11	HiI	LoX	P	P	no-op	DCH
Q	Q	LoX	LoX	20C	CMD-11	HiI	LoX	Q	Q	no-op	error
R	R	LoX	LoX	20C	CMD-11	HiI	LoX	R	R	no-op	error
S	S	LoX	LoX	20C	CMD-11	HiI	LoX	S	S	no-op	SU
T	T	LoX	LoX	20C	CMD-11	HiI	LoX	T	T	no-op	SD
U	U	LoX	LoX	20C	CMD-11	HiI	LoX	U	U	no-op	error
V	V	LoX	LoX	20C	CMD-11	HiI	LoX	V	V	no-op	error
W	W	LoX	LoX	20C	CMD-11	HiI	LoX	W	W	no-op	error
X	X	LoX	LoX	20C	CMD-11	HiI	LoX	X	X	no-op	ECH
Y	Y	LoX	LoX	20C	CMD-11	HiI	LoX	Y	Y	no-op	error
Z	Z	LoX	LoX	20C	CMD-11	HiI	LoX	Z	Z	no-op	CBT
[	[	LoX	LoX	no-op	error	HiI	LoX	[	[	CSI	error
		LoX	LoX	no-op	error	HiI	LoX			no-op	error

Table E-1 (cont)
CHARACTER INTERPRETATION BY TEK AND ANSI PARSERS

CHAR	TEK Parser								ANSI Parser		
	Alpha	Vector	Marker	LCE-T	20C	*int*	*xy*	*char*	alpha	LCE-A	CSI
]	]	LoX	LoX	no-op	error	Hil	LoX	]	]	no-op	error
^	^	LoX	LoX	no-op	error	Hil	LoX	^	^	no-op	error
_	_	LoX	LoX	no-op	error	Hil	LoX	_	_	no-op	error
`	`	LoY/Ex	LoY/Ex	CMD-10	error	Hil	LoY/Ex	`	`	DMI	error
a	a	LoY/Ex	LoY/Ex	CMD-10	error	Hil	LoY/Ex	a	a	no-op	error
b	b	LoY/Ex	LoY/Ex	CMD-10	error	Hil	LoY/Ex	b	b	EMI	error
c	c	LoY/Ex	LoY/Ex	CMD-10	error	Hil	LoY/Ex	c	c	RIS	error
d	d	LoY/Ex	LoY/Ex	CMD-10	error	Hil	LoY/Ex	d	d	no-op	error
e	e	LoY/Ex	LoY/Ex	CMD-10	error	Hil	LoY/Ex	e	e	no-op	error
f	f	LoY/Ex	LoY/Ex	CMD-10	error	Hil	LoY/Ex	f	f	no-op	HVP
g	g	LoY/Ex	LoY/Ex	CMD-10	error	Hil	LoY/Ex	g	g	no-op	TBC
h	h	LoY/Ex	LoY/Ex	CMD-10	error	Hil	LoY/Ex	h	h	no-op	SM
i	i	LoY/Ex	LoY/Ex	CMD-10	error	Hil	LoY/Ex	i	i	no-op	error
j	j	LoY/Ex	LoY/Ex	CMD-10	error	Hil	LoY/Ex	j	j	no-op	error
k	k	LoY/Ex	LoY/Ex	CMD-10	error	Hil	LoY/Ex	k	k	no-op	error
l	l	LoY/Ex	LoY/Ex	CMD-10	error	Hil	LoY/Ex	l	l	no-op	RM
m	m	LoY/Ex	LoY/Ex	CMD-10	error	Hil	LoY/Ex	m	m	no-op	SGR
n	n	LoY/Ex	LoY/Ex	CMD-10	error	Hil	LoY/Ex	n	n	no-op	DSR
o	o	LoY/Ex	LoY/Ex	CMD-10	error	Hil	LoY/Ex	o	o	no-op	error
p	p	LoY/Ex	LoY/Ex	no-op	error	Hil	LoY/Ex	p	p	no-op	error
q	q	LoY/Ex	LoY/Ex	no-op	error	Hil	LoY/Ex	q	q	no-op	error
r	r	LoY/Ex	LoY/Ex	no-op	error	Hil	LoY/Ex	r	r	no-op	error
s	s	LoY/Ex	LoY/Ex	no-op	error	Hil	LoY/Ex	s	s	no-op	error
t	t	LoY/Ex	LoY/Ex	no-op	error	Hil	LoY/Ex	t	t	no-op	error
u	u	LoY/Ex	LoY/Ex	no-op	error	Hil	LoY/Ex	u	u	no-op	error
v	v	LoY/Ex	LoY/Ex	no-op	error	Hil	LoY/Ex	v	v	no-op	error

Table E-1 (cont)
CHARACTER INTERPRETATION BY TEK AND ANSI PARSERS

CHAR	TEK Parser								ANSI Parser		
	Alpha	**Vector**	**Marker**	**LCE-T**	**20C**	*int*	*xy*	*char*	**alpha**	**LCE-A**	**CSI**
w	w	LoY/Ex	LoY/Ex	no-op	error	Hil	LoY/Ex	w	w	no-op	error
x	x	LoY/Ex	LoY/Ex	no-op	error	Hil	LoY/Ex	x	x	no-op	error
y	y	LoY/Ex	LoY/Ex	no-op	error	Hil	LoY/Ex	y	y	no-op	error
z	z	LoY/Ex	LoY/Ex	no-op	error	Hil	LoY/Ex	z	z	no-op	error
{	{	LoY/Ex	LoY/Ex	no-op	error	Hil	LoY/Ex	{	{	no-op	error
|	|	LoY/Ex	LoY/Ex	no-op	error	Hil	LoY/Ex	|	|	no-op	error
}	}	LoY/Ex	LoY/Ex	no-op	error	Hil	LoY/Ex	}	}	no-op	error
~	~	LoY/Ex	LoY/Ex	no-op	error	Hil	LoY/Ex			no-op	error
$^{D}_{T}>$		LoY/Ex	LoY/Ex	$^{E}_{C}$		Hil	LoY/Ex				

Comments

LCE-T means last-character-escape-TEK.

LCE-A means last-character-escape-ANSI.

CSI means Command Sequence Introduction mode; the terminal is waiting to complete a command.

No-op means a recognized command that does nothing.

Halt means stops parsing and ignore command up to here.

Notes

1. $^{B}_{L}$ also sets move/draw flag to move in Vector mode
2. $^{C}_{R}$ is ignored if dialog area enabled
3. These also terminate any command in process (except when TEK mode $^{E}_{C}$ is followed by (?))
4. Unless in SELECT-CODE command

Command Names Too Big to Fit in Table

1. REPORT-4010-STATUS
2. SET-ALPHA-FONT
3. SET-ALPHA-FONT
4. 4010-HARDCOPY
5. ENTER-BYPASS-MODE
6. ENABLE-4010-GIN
7. ENABLE-4953-GIN
8. SELECT-CODE
9. SET-4014-ALPHATEXT-SIZE
10. SET-4014-LINESTYLE
11. Potential 4100 command — may be not recognized

INDEX

12-bit coordinate mode, 4-12
12-bit space, 8-19, 8-20
2OC state, 4-11
32-Bit Coordinate mode, 4-12
32-bit space, 8-19, 8-20
32-bit *xy* parameters, 4-3, 4-4, 4-12
3PPI, 10-1, 10-5
4010-HARD-COPY, 4-1
4010-HARDCOPY, 5-5
4110 Series Command Reference Manual, 1-1,1-2,1-3,1-4
4110 Series Reference Guide, 1-2
4110 Series Terminal Architecture, 2-1
4112, 2-3
4113, 2-3
4114, 2-3
4115, 2-3
4116, 2-3
4662 or 4663 plotter, 9-1, 9-2
7-bit character set, 3-6, 3-9
8-bit character set, 3-6, 3-7, 3-9
addition array, 7-8, 7-9
additive color mixing, 8-10, 8-13, 8-15
address space, 2-3
all segments, 7-2, 7-10
Alpha mode, 4-1, 4-7, 4-10, 6-1, 6-3, 6-4
alphatext, 6-1, 6-2, 6-4, 6-5
alphatext attributes, 6-4
ALU modes, 8-16, 8-18
AND mode, 7-3
ANSI Alpha mode, 4-12
ANSI mode, 5-1, 5-2
ANSI Parser, 4-1, 4-7, 4-12
ARM-FOR-BLOCK-MODE, 3-7
ARRAY PARAMETERS, 1-4, 4-4
ASCII keyboard, 2-1
available surfaces, 8-6, 8-9
background color, 8-10, 8-16
BEGIN-FILL-PATTERN, 8-17, 8-18
BEGIN-GRAPHTEXT-CHARACTER, 6-6
BEGIN-HIGHER-SEGMENT, 7-5, 7-6
BEGIN-LOWER-SEGMENT, 7-5, 7-6
BEGIN-NEW-SEGMENT, 7-5, 7-6
BEGIN-PANEL-BOUNDARY, 6-7
BEGIN-PIXEL-OPERATIONS, 8-16, 8-17
BEGIN-SEGMENT, 7-5, 7-6
bit planes, 8-3, 8-6, 8-9, 8-10
bits-per-pixel value, 8-16, 8-17, 8-18
block count, 3-8
block data, 3-6, 3-7, 3-8
block format, 3-7
block header, 3-7
block length, 3-7
block lines, 3-6
Block mode algorithms, 3-13 — 3-15
Block mode parameters, 3-7
block packing, 3-7
block timeout, 3-7
block transactions, 3-6
block-continue character, 3-7, 3-8
block-control bytes, 3-7, 3-8, 3-9
block-control characters, 3-7, 3-8, 3-9
block-end character, 3-7,3-8
block-master characters, 3-7, 3-8
border index, 8-20
border visibility, 8-20
break signal duration, 3-3
Bypass mode, 3-3, 3-4
bypass-cancel character, 3-4
Byte macros, 4-6
char-array, 4-4
character flagging, 3-3, 3-5, 3-8
char parameters, 4-3
checksum, 3-6, 3-9
CLEAR-DIALOG-SCROLL, 5-1
CMY , 8-10, 8-14, 8-15, App. D
code examples, 1-1,1-3
color cone, 8-12, App. D
color coordinate systems, 8-10, App. D
Color Copier interface, 10-1, 10-6
Color Cube, 8-13, 8-14, App. D
color display, 8-1 ,8-2
color index, 8-5, 8-6, 8-9, 8-10, 8-17, 8-18, 8-20
color map, 8-3, 8-4, 8-10, 8-11, 8-15
command keys, 2-1
command names, 1-4
command parameters, 1-2,1-4
command parsing states, 4-10
command syntax, 1-2,1-4
communications parameters, 11-1
communications subsystems, 2-1
controlling report length, 3-4
COPY, 10-1, 10-2, 10-6

critical communications parameters, 3-3
crosshair cursor, 7-2
CSI state, 4-12
current matching class, 7-8
data packing, 1-4
DEFINE-MACRO, 4-6
defining graphtext, 6-6
DELETE-FILE, 10-4
DELETE-GRAPHTEXT-CHARACTER, 6-5
delimiters in setup mode, 1-4
detectability, 7-3, 7-5
device name parameters, 10-1
device names, 10-1, 10-2, 10-4
device-function-code, 9-1, 9-2, 9-5, 9-6
dialog area, 5-1, 5-2, 5-3, 11-1
dialog area parameters, 5-2
dialog scroll buffer, 4-5
differences in subsystems, 2-1
direct memory access, 2-1,10-6
DIRECTORY, 10-4
DISABLE-4953-TABLET-GIN, 9-6
DISABLE-GIN, 9-5, 9-6
disk commands, 10-4
disk interfaces, 2-1
disk storage, 10-4
display priority, 7-3, 7-4
display subsystems, 2-1
distance filter, 9-1, 9-2, 9-6
DMA interface, 10-1, 10-6
DRAW, 4-10, 6-1, 6-2, 6-3
DRAW-MARKER, 4-10, 6-3, 6-4
DRAW-RECTANGLE, 6-8
DTR/CTS, 3-3, 3-5, 3-8
DVST, 2-3
editing commands, 5-4
ENABLE-4010-GIN, 4-1
ENABLE-4010-GIN, 6-3
ENABLE-4010-GIN, 9-5
ENABLE-4953-GIN, 6-3
ENABLE-4953-TABLET-GIN, 9-5
ENABLE-DIALOG-AREA, 5-1
ENABLE-GIN, 9-2, 9-5, 9-6
enabling flagging, 3-5
END-FILL-PATTERN, 8-18, 8-19
END-GRAPHTEXT-CHARACTER, 6-6
END-PANEL, 6-7
END-SEGMENT, 7-5, 7-7
ENTER-ALPHA-MODE, 4-1, 4-10
ENTER-ALPHA-MODE, 6-6
ENTER-BYPASS-MODE, 3-4
ENTER-MARKER-MODE, 4-1, 4-10
ENTER-MARKER-MODE, 6-4
ENTER-VECTOR-MODE, 4-1, 4-10
ENTER-VECTOR-MODE, 6-3
EOF-string, 3-3, 4-5
EOL-string, 3-3
EOM indicator, 9-5
EOM-characters, 3-1, 3-4, 3-9
EOM-frequency, 4-5
erase index, 8-10, 8-16
error detection 6, 3-7
exclusion array, 7-8, 7-10
exclusion class register, 7-8, 7-9
EXPAND-MACRO, 4-6
explicit commands, 6-1
extended terminal space, 8-19, 8-20
file name parameters, 10-1
fill patterns, 8-10, 8-16, 8-18, 8-19
filters, 9-2, 9-5, 9-6
fixup level, 7-7, 8-19, 8-23
flagging, 3-3, 3-5, 3-8
FORMAT, 10-4
frame buffer, 8-5
Full- and Half-Duplex, 3-1, 3-3, 3-5, 3-7
Full-Duplex, 3-1, 3-3, 3-5, 3-7
function keys, 2-1
future segments, 7-2
future-segment attributes, 7-2
GIN areas, 9-4, 9-6
GIN cursor, 7-4, 9-1, 9-2, 9-5, 9-6
GIN event, 9-1, 9-2, 9-5
GIN function reports, 9-1, 9-5
GIN location, 9-1, 9-2, 9-5, 9-6
GIN Pick function, 7-3, 7-4
GIN space, 8-19, 8-20, 9-2, 9-3, 9-6
GIN windows, 9-3
GRAPHIC-TEXT, 6-5
Graphics Area Writing mode, 6-4
graphics beam position, 6-1, 7-6
graphics input cursor, 7-4
graphics primitive attributes, 6-1
graphics primitives, 7-1
graphics tablet, 9-1, 9-2, 9-6
Graphtext, 6-5
Graphtext character definition, 6-5
Graphtext fonts, 6-5
Graphtext precision, 6-5
Graphtext rotation, 6-5
Graphtext size, 6-5
Graphtext slant, 6-5
gridding, 9-1,9-6
Half-Duplex, 3-1, 3-3, 3-5, 3-7
handshaking, 3-3, 3-5, 3-8
hard copy units, 2-1

HARDCOPY, 5-5
highlighting, 7-3
HLS , 8-10, 8-12, 8-13, App. D
Home position, 8-23
Host Macros, 4-6
Hil , 4-3
Ignore Deletes mode, 4-12
implicit command modes, 4-7, 4-8, 4-10, 4-12, 6-1
implicit commands, 6-1
implicit mode flag, 4-7
INCLUDE-COPY-OF-SEGMENT, 7-6
inclusion array, 7-8, 7-9, 7-10
inclusion class register, 7-8
inking, 9-1,9-6
input queue, 2-1, 3-1, 3-3, 3-4, 3-5, 4-5, 4-6
inside of panels, 6-7
int-array, 4-4, 4-5
int parameters, 4-4
invisible surface, 8-6, 8-7
Key macros, 4-6
key-character, 9-5
keyboard thumbwheels, 9-1, 9-2, 9-6
LCE-A state, 4-12
LCE-T state, 4-10, 4-11
line attributes, 6-2
line index, 6-2
line style, 6-2
line width, 6-2
LOAD, 10-3
Local mode, 1-4
local peripherals, 2-1
LOCK-VIEWING-KEYS, 8-23
machine RGB, 8-10, 8-14, App. D
macro expansion, 4-6
macros, 4-1, 4-6
MAP-INDEX-TO-PEN, 10-5
Marker mode, 4-7, 4-8, 4-10, 6-3
marker types, 6-3
markers, 6-3
matching classes, 7-7
matching operation, 7-8
message flagging, 3-3, 3-5
MOVE, 4-10, 6-2, 6-3
non retained segments, 7-1
non-transmittable characters, 3-6, 3-7, 3-8
nonvolatile memory, 3-1, 3-3, 3-7
normalized screen space, 8-19, 8-20, 8-22, 8-23
one-op-code commands, 4-1
opaque, 8-10, 8-14, 8-15
Option 01, 3-1, 3-3, 3-5, 3-6
Option 02, 3-1
Option 09, 10-6
Option 10, 10-5
Option 42, 10-4
Option 43, 10-4
Option 45, 10-4
OR mode, 7-3
output gate, 3-4, 3-5
overview, 8-23
overview window, 8-23, 11-1
packed data, 3-6, 3-7, 3-8
packing parameters, 4-3, 4-4
packing *int* parameters, 4-3
packing *xy* parameters, 4-3
PAGE, 4-1, 4-10, 6-3
PAN , 8-20, 8-23
panel boundary, 6-7, 6-8
panel definition, 6-7
panels, 6-7
parameter parsing states, 4-11
parity, 3-1, 3-3
parsing an *intc* report, 4-5
parsing an *int* report, 4-5
parsing an *xy* report, 4-5
parsing array reports, 4-5
parsing GIN function reports, 9-5
partialview window, 8-23
physical devices, 10-2
Pick aperture, 9-2, 9-6
Pick ID, 9-2, 9-5
Picking and Dragging, 9-6
Pivot Point, 7-2
Pixel Beam Position, 8-16, 8-17
Pixel Operations, 2-3, 4-5, 8-16
pixel space, 8-6, 8-19, 8-20, 8-22
pixel viewport, 8-16
PIXEL-COPY, 8-18
PLOT, 10-5
PORT-ASSIGN, 10-5
PORT-COPY, 10-5
PORT-EOF-STRING, 10-5
PORT-PROTOCOL IDENTIFIERS, 10-5
position, 7-2
programming function keys, 5-5
prompt, 3-1, 3-3, 3-4, 3-5
Prompt mode, 3-1, 3-3, 3-4, 3-5, 3-7
prompt string, 3-1, 3-3, 3-4, 3-5
pseudo devices, 10-2
pseudocode, 1-3
pseudocode algorithms, 1-3
pseudocode syntax, 1-3
queue, 2-1
raster, 2-3
raster display, 2-3

raster memory, 8-3, 8-4, 8-6, 8-10, 8-16, 8-18
RASTER-WRITE, 8-16, 8-17, 8-18, 8-19
real parameters, 4-4, 4-11
RECTANGLE-FILL, 8-18
refresh graphics, 2-3
refreshed information, 2-3
removal array, 7-8, 7-10
RENEW-VIEW, 5-5, 8-23
REPORT-4010-STATUS, 3-5, 4-1
REPORT-COLORHARDCOPY-STATUS, 4-1
REPORT-DEVICE-STATUS, 4-1, 10-1, 10-3
REPORT-ERRORS, 4-1, 5-6
REPORT-GIN-POINT, 4-1, 9-1, 9-6
REPORT-PORT-STATUS, 4-1, 10-5
REPORT-SEGMENT-STATUS, 4-1
REPORT-TERMINAL-SETTINGS, 4-1
REPORT-TERMINAL-STATUS, 3-5
reports, 4-1, 4-4, 4-5
RESET, 6-3
RGB , 8-10, 8-13, 8-15, App D
rotation, 7-3
RS-232C, 3-1, 3-3, 3-5
rubberbanding, 9-1, 9-6
runcode, 8-17
RUNLENGTH-WRITE, 8-16, 8-17, 8-18
SAVE, 10-1, 10-3
scaling, 7-3
Segment –1, 7-2
Segment –2, 7-2
Segment –3, 7-2
Segment 0, 7-2
segment attributes, 7-2, 7-3
segment class, 7-7
segment class field, 7-7
segment class subfields, 7-9
segment classes, 7-7
segment definition, 7-6
segment detectability, 7-3, 7-5
segment highlighting, 7-3, 7-5
segment numbers, 7-2
segment origin, 7-2, 7-3
segment storage, 7-2
segment-writing mode, 4-5
segments, 2-3, 6-1
SELECT-CODE, 4-7, 5-2
SELECT-FILL-PATTERN, 6-6, 6-8
SELECT-VIEW, 8-23
Set mode, 7-3
SET-4014-ALPHATEXT-SIZE, 5-2, 6-4
SET-4014-LINE-STYLE, 4-1
SET-ALPHATEXT-FONT, 4-1, 6-4
SET-ALPHATEXT-SIZE, 5-2, 6-4
SET-ALPHATEXT-SIZE-GROUP, 6-4
SET-BACKGROUND-COLOR, 8-16
SET-BACKGROUND-GRAY-LEVEL, 8-16
SET-BACKGROUND-INDICES, 6-2, 8-16
SET-BAUD-RATES, 3-3
SET-BLOCK-CONTINUE-CHARS, 3-7, 3-8
SET-BLOCK-END-CHARS, 3-7, 3-8
SET-BLOCK-HEADERS, 3-7
SET-BLOCK-LENGTH, 3-7
SET-BLOCK-MASTER-CHARS, 3-7
SET-BLOCK-NON-XMT-CHARS, 3-7, 3-8
SET-BLOCK-PACKING, 3-7, 3-8
SET-BLOCK-TIMEOUT, 3-7
SET-BREAK-TIME, 3-3
SET-BYPASS-CANCEL-CHARACTER, 3-4
SET-COLOR-MODE, 8-10, 8-14, 8-16
SET-COORDINATE-MODE, 4-12
SET-CURRENT-MATCHING-CLASS, 7-8
SET-DIALOG-AREA-BUFFER-SIZE, 5-2
SET-DIALOG-AREA-CHARS, 5-2
SET-DIALOG-AREA-INDEX, 5-2
SET-DIALOG-AREA-LINES, 5-2
SET-DIALOG-AREA-POSITION, 5-2
SET-DIALOG-AREA-SURFACE, 5-2
SET-DIALOG-AREA-VISIBILITY, 5-1
SET-DIALOG-AREA-WRITING-MODE, 5-2
SET-DRAW-BOUNDARY-MODE, 6-8
SET-DUPLEX-MODE, 3-5
SET-ECHO, 5-6
SET-EOF-STRING, 3-3
SET-EOL-STRING, 3-3
SET-ERROR-THRESHOLD, 5-6
SET-FLAGGING-MODE, 3-5
SET-GIN-AREA, 9-4, 9-6
SET-GIN-CURSOR, 9-6
SET-GIN-DISPLAY-START-POINT, 9-6
SET-GIN-GRIDDING, 9-6
SET-GIN-INKING, 9-6
SET-GIN-RUBBERBANDING, 9-6
SET-GIN-STROKE-FILTERING, 9-6
SET-GIN-WINDOW, 9-4, 9-6
SET-GRAPHICS-AREA-WRITING-MODE, 6-4
SET-GRAPHTEXT-FONT, 6-5
SET-GRAPHTEXT-FONT-GRID, 6-5
SET-GRAPHTEXT-PRECISION, 6-5
SET-GRAPHTEXT-ROTATION, 6-5
SET-GRAPHTEXT-SIZE, 6-5
SET-GRAPHTEXT-SLANT, 6-5
SET-LINE-INDEX, 6-2
SET-LINE-STYLE, 6-2
SET-LINE-WIDTH, 6-2
SET-MARGINS, 5-5

SET-MARKER-TYPE, 6-3
SET-OVERVIEW-WINDOW, 8-23
SET-PAGE-FULL-ACTION, 5-6
SET-PANEL-FILLING-MODE, 6-8
SET-PARITY, 3-3
SET-PICK-APERTURE, 9-6
SET-PIVOT-POINT, 6-5, 7-5
SET-PIXEL-BEAM-POSITION, 8-16
SET-PIXEL-WRITING-FACTORS, 8-17
SET-PORT-BAUD-RATE, 10-5
SET-PORT-EOF-STRING, 10-5
SET-PORT-EOL-STRING, 10-5
SET-PORT-FLAGGING-MODE, 10-5
SET-PORT-PARITY, 10-5
SET-PORT-STOP-BITS, 10-5
SET-PROMPT-STRING, 3-5
SET-QUEUE-SIZE, 3-4
SET-REPORT-MAX-LINE-LENGTH, 3-4
SET-SEGMENT-CLASS, 7-5, 7-9
SET-SEGMENT-DETECTABILITY, 7-5
SET-SEGMENT-DISPLAY-PRIORITY, 7-3, 7-5
SET-SEGMENT-HIGHLIGHTING, 7-5
SET-SEGMENT-IMAGE-TRANSFORM, 7-3, 7-5
SET-SEGMENT-POSITION, 7-3, 7-5
SET-SEGMENT-VISIBILITY, 7-3, 7-5
SET-SEGMENT-WRITING-MODE, 7-3, 7-5
SET-SNOOPY-MODE, 4-7
SET-STOP-BITS, 3-3
SET-SURFACE-COLOR-MAP, 8-10, 8-16
SET-SURFACE-DEFINITIONS, 8-6
SET-SURFACE-GRAY-LEVELS, 8-10
SET-SURFACE-PRIORITY, 8-8
SET-SURFACE-VISIBILITY, 8-6
SET-TABLET-HEADER-CHARACTERS, 11-1
SET-TABLET-STATUS-STRAP, 11-1
SET-TEXT-INDEX, 6-5
SET-TRANSMIT-RATE-LIMIT, 3-3
SET-VIEW-ATTRIBUTES, 8-23
SET-VIEW-DISPLAY-CLUSTER, 8-23
SET-VIEWPORT, 8-22, 8-23
SET-WINDOW, 8-20, 8-23
Setup mode, 1-4
signal flagging, 3-3, 3-5, 3-8
signature characters, 9-1
single character commands, 4-1
Snoopy mode, 4-7
SPOOL, 10-1, 10-2
stop bits, 3-3
STOP-SPOOLING, 10-1, 10-3
stored information, 2-3
string precision graphtext, 6-5
string parameters, 4-4, 4-11
stroke precision graphtext, 6-5
subsystems, 2-1
subtractive color mixing, 8-10, 8-14, 8-15
super surface, 8-6, 8-9, 8-16
surface priority, 8-8
tabulation commands, 5-3
TEK parser, 4-7
terminal commands, 4-1
terminal file system, 2-1
terminal report commands, 4-4
terminal space, 8-1, 8-6, 8-19, 8-23
terminal viewing keys, 8-23
terminator characters, 4-10, 4-12
text in the graphics area, 6-4, 6-5
the key-execute character, 4-6
the keyboard, 5-5
thumbwheels, 2-1
transmit delay, 3-4, 3-5
transmit rate limit, 3-3
troubleshooting, 4-5
two-op-code commands, 4-1
user-defined fill pattern, 8-18, 8-19
Vector mode, 4-1, 4-7, 4-8, 4-9, 4-10, 6-2
view display cluster, 8-23
view number, 8-20
view surface, 8-20
viewport, 8-20, 8-22, 8-23
window, 8-20, 8-22, 8-23
window size, 11-1
window-viewport transform, 8-22
wipe index, 8-20
XOR mode, 7-3
xy-array, 4-4
xy parameters, 3-4, 4-4
ZOOM, 8-20, 8-23