OPERATOR'S MANUAL

CGC 7900 SERIES
COLOR GRAPHICS COMPUTERS

CHROMATICS

CGC 7900 COLOR GRAPHICS COMPUTER SYSTEM

Operator's Manual
TERMEM Version 1.4

2558 Mountain Industrial Boulevard
Tucker, Georgia 30084

Phone (404) 493-7000
TWX 810-766-8099

Part Number 070201 Rev. B
Printed 06 Sep 83

CHROMATICS

CGC 7900 SERIES COLOR GRAPHICS COMPUTERS

OPERATOR'S MANUAL

Document Number 070201B

Printed May, 1981

2558 Mountain Industrial Boulevard
Tucker, Georgia 30084

Telephone (404) 493-7000
TWX 810-766-8099

WARNING

This equipment generates, uses, and can radiate radio frequency energy and if not installed and used in accordance with the instructions manual, may cause interference to radio communications. It has been tested and found to comply with the limits for a Class A computing device pursuant to Subpart J of 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 user at his own expense will be required to take whatever measures may be required to correct the interference.

Conventions used in this Manual

1) Any keys which have labelled caps will be called by their full names, capitalized and underlined. For example, the carriage return key will be denoted by:

   RETURN

2) The modifier keys, CTRL, SHIFT, M1 and M2, must be held down while striking the key they are to modify. Note that these four keys do not generate any characters on their own, but simply modify the character which is struck simultaneously. This process of holding down a modifier key while striking another key will be denoted by the modifier AND the key being underlined together. For example,

   CTRL F

   would indicate that the CTRL key should be held down while striking the F key. If two or more modifiers are needed simultaneously, they will all be underlined together:

   CTRL SHIFT T

   would mean that BOTH modifiers, SHIFT and CTRL, should be held down while striking the T key.

   For keys marked with names for these modifiers (CTRL or SHIFT only), the modifier will not be specified. For example, the View Sub-buffer command is generated by holding down the SHIFT key and pressing the key marked REDRAW and VIEW; this is represented as:

   VIEW

   (This is a change from the previous manual.) Functions without names marked on the key are still represented by including the modifier. For example, the command to move the cursor to the lower left-hand corner of the window is represented as:

   CTRL HOME

   since this function is not marked on the HOME key.

3) Variable parameters will be enclosed in angle brackets, < >. Any items enclosed in these brackets will be explained in full in the text which immediately follows.

4) Optional parameters will be enclosed in square brackets [ ]. Any items which may be repeated will be followed by an ellipsis (three dots).

Example of (3) and (4):

<X>, [<Y1>,<Y2>,...]

The parameter <X> is required. The parameters <Y1>, <Y2>, and so on, are optional. Any number of these may be included. All three types of parameters would be explained immediately beneath the example which contained them.

5) Zeros will be slashed (Ø); alphabetic O will not be slashed.

Table of Contents

Preface

Chapter 1 -- Introduction

Chapter 2 -- The Overlay

Chapter 3 -- System Commands

Chapter 4 -- Terminal Emulator (TERMEM)

Table of Contents

Chapter 5 -- The Create Buffer

Chapter 6 -- Thaw and Memory Allocation

Chapter 7 -- Numerical Data and Window Variables

Table of Contents

Chapter 8 -- The Bitmap

Table of Contents

Chapter 9 -- Plot Submodes

Chapter 10 -- Raster Processor Graphics

Chapter 11 -- Expanded Image Memories

Table of Contents

Appendix A -- Special Codes

Appendix B -- Color Display Schemes

Appendix C -- The Monitor

Appendix D -- Traps

Table of Contents

Appendix E -- Custom Modules, Cursors and Character Sets

Appendix F -- ASCII Codes

Preface

How to Use this Manual

If you are not familiar with any Chromatics products, read **Chapter 1, Chapter 2, Chapter4** and **Chapter 7** through **Chapter 9. Appendix A** will be a good reference as you continue to learn the system.

Once you have become familiar with the CGC 7900, you will want to read **Chapter 3, Chapter 5, Chapter 6, Chapter 10** and **Chapter 11.** The Appendices will also be helpful.

Advanced users will want to use this manual mainly as a supplement to the CGC 7900 Reference Card. The Reference Card contains a complete list of CGC 7900 commands, and a very brief description of each one. This manual provides more detailed information.

Changes from the Previous Manual

The changes in this manual reflect in part the evolution of the 7900. The manual has been completely reorganized to make it easier to read and to accommodate newer features. If you have the old 7900 manual, you may wish to skim through this one to note the differences.

- A new table lists CGC 7900 commands in alphabetical order, rather than by Mode, Plot, etc. Another table lists the codes generated by each key.
- Plot submodes, Raster Processor commands, Thaw parameters and other topics are now under their own chapter headings.
- Operating systems are discussed to a small extent.
- Descriptions of commands in the text are alphabetized where possible to make them easier to find.
- A new Appendix, **Appendix E,** shows how to add user-defined commands to the 7900 and provides some "hooks" into the firmware.

Chapter 1 -- Introduction

This is the User's Manual for the Chromatics CGC 7900 Color Graphics Computer. It discusses how to operate the 7900, and gives examples of the system's capabilities. If you are just beginning to learn about the 7900, this is the first manual you should read.

This manual is divided into eleven chapters and six appendices:

Chapter 1 (this section) discusses the 7900 architecture, philosophy and installation.

Chapter 2 explains philosophy and commands for the Overlay.

Chapter 3 explains system commands which relate to the 7900 as a whole.

Chapter 4 discusses the Terminal Emulator (TERMEM).

Chapter 5 discusses commands for the Create Buffer.

Chapter 6 explains CMOS RAM and Thaw parameters.

Chapter 7 discusses numerical data entry and calculation.

Chapter 8 discusses Bitmap operations. These include color manipulation, Pan and Zoom, and cursor control.

Chapter 9 explains the various 7900 plot submodes.

Chapter 10 discusses Raster Processor commands.

Chapter 11 discusses commands for systems with sixteen image planes.

Appendix A lists commands alphabetically and by type.

Appendix B discusses color display schemes.

Appendix C discusses the CGC 7900 Monitor.

Appendix D lists Trap codes.

Appendix E is a "cookbook" for subjects such as adding new commands to the 7900 repertoire and designing custom cursors and character sets. Also included in this appendix is a list of "hooks" into the TERMEM firmware.

Appendix F is the mandatory ASCII code list.

CGC 7900 Overview

The CGC 7900 is the successor to Chromatics' CG series of color graphics computers. Many of the same philosophies have been retained in the development of the 7900, and a user who is familiar with the CG will quickly become accustomed to the 7900.

The 7900 contains at least three processors:

- Most system functions revolve around the MC68000 processor, selected for its high speed (8 megahertz) and large memory addressing range (16 megabytes). This powerful processor gives the 7900 outstanding capability for stand-alone computing applications. And when the 7900 is acting as a terminal (connected to a host system), the 7900's power relieves the host of many of the tasks normally required in a graphics environment.

- The 7900 also includes a processor in the keyboard, and a Raster Processor to provide high performance and speed in graphics operations.

- Optionally, the 7900 may contain a Serial Port Controller with its own on-board Z-80 processor. The optional Hardware Vector Generator can also be considered a processor.

Many graphics systems suffer from one drawback: they require a separate character-oriented terminal for command level interaction between the system and the operator. The 7900 addresses this need by providing a character-oriented "Overlay" display in addition to the high-resolution Bitmap graphics display. With eight standard colors and blink, the Overlay is a very effective tool for operator interaction. When not needed, the Overlay can become instantly "transparent" to reveal high-resolution graphics images in the Bitmap.

The concept of logical and physical devices, used in the Chromatics CG series of computers, has been expanded and applied to the 7900. All programs in the 7900 communicate only through logical devices, which are numbered 0 through 4. 7900 software associates each logical device to one or more physical devices. This association may be changed at any time, allowing total flexibility in programmed input/output. Any program can accept input from any physical device, and transmit output to any physical device. Some examples of physical devices are the keyboard, the serial ports, and the 7900 display screen.

The screen can be subdivided into several distinct areas known as windows. Each window is a separate physical device (emulated by software). Thus, each window can be used for a separate purpose, and independent simultaneous displays are possible. In some applications, one 7900 could replace up to 16 separate terminals.

All 7900 features discussed in this manual, and most currently available optional features, are contained in a single stand-alone package. Standard features include:

- 128K or 512K bytes of buffer memory
- One or four planes of Bitmap image memory
- A keyboard with 151 keys (21 lighted) and 24 user-definable function keys
- 8 programmable bezel keys
- Two serial ports
- 19-inch color display screen with 1024 x 768 resolution
- A PROM firmware package which allows easy control of all system functions.

Optional features currently include:

- Lightpen and joystick for interactive use
- A battery-backed Real-Time Clock and CMOS memory
- Disk Operating System (DOS) much like that of the CG series
- Floppy and hard disk drives
- Idris Multi-tasking Operating System
- A Hardware Vector Generator (HVG) which can draw up to 27,000 short vectors per second
- Extended graphics software features

- Parallel Input/Output and Direct Memory Access Interface
- Serial Port Controller (SPC) with 4 RS-232 ports
- Disk DMA (with Idris and DOS drivers available)
- Numerous peripheral interfaces
- Language support (FORTRAN, C, Pascal)

Installation

Unpack the CGC 7900 according to the instructions supplied in the shipping carton. Retain the packing material so that it may be used for shipping the 7900 in the future.

WARNING:

> If your 7900 system includes the optional Fixed Disk drive, REMOVE the locking screw and actuator lock which was used to secure the drive during shipment! See instructions on your unit for details. FAILURE TO REMOVE THESE LOCKS WILL DESTROY THE HARD DISK, AND VOID YOUR WARRANTY!

Connect the 7900 power cord to a power source of 110 volts AC, 60 hertz (220 VAC, 50 hertz optional), capable of supplying at least 10 amperes. For reliable operation, power should NOT come from a circuit with any heavy motors or industrial equipment connected which could create transients on the power lines. This includes equipment such as refrigerators and air conditioners.

The 7900 has no strict environmental requirements. But, like any precision instrument, the 7900 will perform best if it is not subjected to excessive heat or dust. 7900 ventilation removes heat from the unit through vents in the front and rear; these vents should not be obstructed. The rear door (and floppy drives, if installed) should be CLOSED during normal operation to provide proper air flow through the 7900 chassis.

7900 power is applied by pressing the square, lighted switch on the front of the unit (above the keyboard). The switch is lit whenever the system's five volt power supply is operating. When turning the 7900 on, observe the indicator light on the keyboard, just above the cursor keypad. It will glow green as the unit performs internal power-on diagnostics, and should extinguish after one or two seconds (the time this takes is proportional to the amount of user memory). By this time, the picture tube should have warmed up, and a blinking cursor should be visible on the screen.

Your 7900 is now running!

Booting Software

Currently, the CGC 7900 supports four operating environments:

- The **Chromatics Disk Operating System** (DOS), a simple operating system;
- **Idris**, a Unix-like multitasking operating system from Whitesmiths, Inc.;
- The Terminal Emulator (TERMEM);
- and the 7900 Monitor.

Four keys are dedicated to starting these operating environments. These are marked DOS, IDRIS, TERMINAL and MONITOR. Pressing one of these keys will boot the associated environment.

Typing SHIFT DOS will enter DOS without asking for a user password. SHIFT IDRIS enters the Idris Bootstrap Loader, which is normally used only to install the full Idris.

More information on DOS and Idris can be found in the DOS User's Manual and the Idris User's Manual, respectively. This manual describes all the TERMEM commands, and Monitor commands are discussed in Appendix C.

Keyboard Description

The 7900 system keyboard is divided into several areas. Each area is designed for a specific purpose, and the keys in each area are arranged and color-coded for ease of operation.

In general, keys on the keyboard are marked three ways:

- The marking on the top of the key is the primary function of that key. Pressing the key alone, without using any modifiers, will cause that code to be sent from the keyboard.
- The marking on the front of the key (if marked in white) is the code which is output when the SHIFT modifier is used in conjunction with the key (see the Conventions page in the front of this manual).
- The marking on the front of the key (if marked in blue) is the code which is output when the CTRL modifier is used in conjunction with the key (see the Conventions page in the front of this manual).

In the center of the keyboard, you will find a sculptured, typewriter-like set of keys. With few exceptions, these keys may be used just as if they were on a standard typewriter. On the right of this set are the control keys, RETURN (carriage return), LF (line feed), and BREAK (used to interrupt a running program). On the left are the modifiers, SHIFT, CTRL, M1, and M2. The high speed REPEAT key is also on the left.

Any key on the keyboard may be caused to repeat, at either low or high speed. To repeat a key at low speed, simply hold the key down. To cause high speed repeat, hold down both the desired key and the REPEAT key.

In the left of this area are the four "prefix" keys which the system understands: ESC, USER, MODE, and PLOT. Pressing any of these keys is a signal that one or more other keys will immediately follow, to complete a code sequence. To cause the keyboard to output a User code, hold down the modifier SHIFT and press the key marked ESC and USER. Similarly, for the code Plot, press the SHIFT key and the key marked MODE and PLOT.

To the far left are two special keys: QUIET LOCK and ALPHA LOCK. These two are alternate-action keys. Pressing one of these keys will lock it in the down (on) position; pressing it again will release it to the up (off) position.

QUIET LOCK will disconnect the built-in speaker when it is in the down (on) position. ALPHA LOCK will reverse the case of all letters typed from the typewriter section of the keyboard. When it is up (off), letters typed on the typewriter keyboard will be upper case when the shift key is not being used. They will come out in lower case when the shift key is used. (This is the opposite of a normal typewriter, but is useful when the unit is acting as a terminal.) To reverse this, and return to standard typewriter usage, press the ALPHA LOCK key into the down (on) position. Now characters typed on the typewriter keyboard will be in lower case, and SHIFT will change them to upper case. This is most useful for text editing applications.

To the right of the keyboard are two smaller, special purpose keypads. The cursor keypad is used to position the cursor, and for text editing functions such as inserting and deleting lines. The numeric keypad is a convenient way to input numeric data. The keys on the numeric keypad duplicate the functions of their counterparts on the typewriter keyboard.

The special function keys on the upper half of the keyboard are used to access most of the system's features. Most of this manual is dedicated to explaining, in detail, what each of these keys will do. In general, the following comments apply:

- The name on top of the key represents its primary function.
- The name on the front of the key represents its secondary function, and is accessed by holding the SHIFT (or CTRL) modifier and pressing the key.

Keys which have a built-in light are keys whose functions may be in one state or the opposite state. For example, the BLINK key has a light, and at any time, the "blink" attribute may either be on or off. The condition of the light will tell which state the system is in. (In some cases, it is possible for the keyboard lights to be "out of sync" with the rest of the system, in which case they will not present true information. See "Keyboard Sync" and "Assign" for the details.)

Lighted keys have an additional feature: Pressing a lighted key will turn a function on if it is off, and turn it off if it is currently on. Thus, repeatedly pressing a lighted key will toggle the light in that key on and off. The function produced by a key depends on whether the light in that key is currently on or off. (Using SHIFT with a lighted key will, however, always turn the function OFF.)

The character Control-Z acts as a "flush" command. It will clear out the keyboard buffer, so that any keystrokes that have not yet been executed will no longer wait in the queue.

To execute the "flush" command, hold down the CTRL modifier and press the Z key:

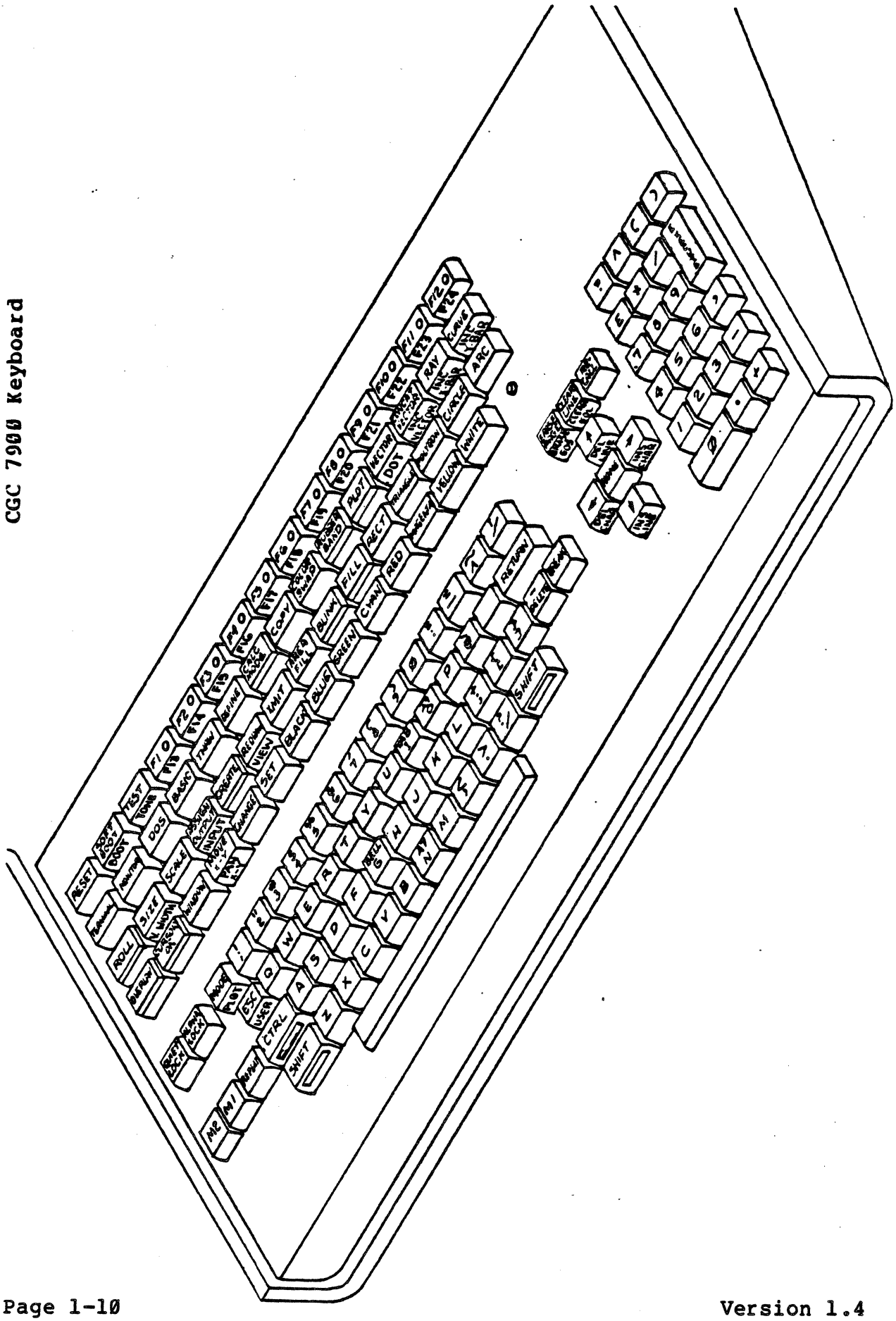
CTRL Z

The CALC MODE key is not currently defined.

A two-color light-emitting diode (LED) is located near the cursor keypad. This LED glows green during Reset and power-on diagnostics. It will glow red in the event of a system failure.

In addition to the special keys, all commands have a code sequence associated with them. If the 7900 is to run under the control of a host computer, these code sequences must be sent in lieu of a keystroke. Appendix A lists commands in alphabetical order, along with its code and associated key (if any).

CGC 7900 Keyboard



Codes and Code Sequences

The 7900 system is controlled through the keyboard, or through a communications port from a host system. The 7900 software allows all important system functions to be accessed through ASCII characters, which make up codes or code sequences. A **code** is a single ASCII character typed on the keyboard, or received from a host. A **code sequence** is a set of such characters.

Certain code sequences cause no immediate change in the visible state of the system. Commands such as "Set Color" have an effect on future displays, but do not alter anything currently being displayed. This can be disconcerting to an operator, since there is no feedback to indicate that the code sequence was accepted properly. It may be helpful to compare such code sequences to similar commands on a familiar office typewriter: the "set tab stop" function of a typewriter does not generate any feedback to indicate acceptance of this command. The command is silently accepted.

The 7900 recognizes four "prefix" codes: Escape, User, Plot and Mode. When a prefix code is entered, it signals the system that one or more additional codes will follow as arguments, and that the entire code sequence should be taken together to perform a function.

These prefix codes are NOT the same as the modifiers SHIFT, CTRL, M1 and M2. Modifiers do not generate codes; they merely alter the key which is pressed simultaneously with the modifier. The prefix keys DO generate codes on their own, and thus the prefix key must be pressed and released before the next key is struck.

Some of the named keys on the upper part of the keyboard cause actions which are equivalent to pressing more than one key on the lower keyboard. For example, pressing RECT to enter the "Plot Rectangle" mode, is equivalent to entering the sequence PLOT R. This is simply an alternate way of entering commands, and we will always use the simplest way to describe each command in this manual. See Appendix A for detailed information on which keys produce which codes.

If you enter a code sequence which is not defined in the 7900 software, a "bong" sound will be produced from the speaker, as a warning. The same sound will be heard if you attempt to access an optional software feature which is not installed in your unit.

Escape Code Sequences

Format:

ESC <char> [<arg1> <arg2> ...]

An Escape code sequence consists of the Escape character, CTRL [($1B), followed by a single character <char> which defines the type of Escape code sequence. This may be followed by one or more arguments, <arg>, depending on what the sequence requires. All arguments fall into one of two categories:

Numbers:	decimal or hexadecimal numbers which are delimited by a comma or semicolon.
Characters:	a single ASCII character.

In addition, a few Escape code sequences will accept an arbitrary number of arguments. Details are described in each command where applicable. In these cases, a special delimiter character (usually the semicolon) is used to signal the end of the argument list.

To produce an Escape code sequence, you would press and release the ESC key, then press whatever other keys are necessary to complete the sequence. The argument list is determined by the particular Escape code sequence you are executing, and examples will be found throughout this manual.

Escape code sequences affect the entire machine. They control aspects of the operation such as pan and zoom, Color Lookup Table assignments, etc.

User Code Sequences

Format:

USER <char> [<arg1> <arg2> ...]

The User character is produced by holding down the SHIFT key and pressing the key marked USER and ESC. The definitions of <char> and <arg> are identical to those for Escape code sequences.

User code sequences cause execution of a program, or affect the configuration of the 7900 in some manner. Some examples of User codes are I/O assignments, duplex selection, user-defined function keys, and Create Buffer operations.

Plot Code Sequences

Format:

PLOT <char>

Where <char> is a single character. A Plot code sequence will place the window in a Plot submode, such as Vector, Circle, Arc, etc. Plot code sequences affect only the currently assigned window.

The Plot character is produced by holding down the SHIFT modifier key and pressing the key marked PLOT and MODE.

NOTE:

The PLOT key used in this context is the key labelled Plot and Mode, located in the typewriter area of the keyboard. It is NOT the same as the lighted PLOT key in the upper keyboard area. The lighted PLOT key is used ONLY to move between plotting and text entry (Alpha) functions.

Mode Code Sequences

Format:

MODE <char> [<char>, <char>, ...]

Mode code sequences also affect only the currently assigned window. They are used in a wide variety of cases, from setting colors to scaling character size. Details on the available Mode code sequences are found throughout this manual.

Code Hierarchy

The code sequences described on the previous pages are arranged in a prioritized structure. It is possible, and often desirable, to interrupt one sequence, enter a higher priority sequence, then resume the previous code sequence. The priorities are arranged as follows:

Escape, User	Highest Priority
Mode	Intermediate
Plot	Lowest Priority

A common example would be: while entering coordinates to draw a rectangle, you decide to change the foreground color. Since coordinates belong to a Plot sequence, and colors are higher priority (Mode sequence), you may interrupt the coordinates at any time and set a color. Then, you may resume entering coordinate data with no lost information.

Escape and User codes have identical priority, and they take higher priority than any other code sequence. ANY Mode, Plot, or text entry function may be interrupted by an Escape or User code, and the code sequence will be processed. This means that important aspects of system operation, controlled by Escape and User codes, may be changed at any time, even in the middle of coordinate data or text.

System Error and Recovery

The 7900's structure of code sequences is not designed to be totally "user-proof." It is possible to enter a sequence which will force the system into an undefined state. However, the RESET key will usually allow recovery from errors without losing the work in progress (not true for Idris). This problem should not exist if important code sequences are being transmitted only from a host or from an applications program (assuming such programs do not contain errors).

The high priority of Escape and User codes can result in confusion under certain conditions. The key point is this: when an Escape or User code sequence is begun, it MUST be completed. For example, the Escape code sequence which changes entries in the Color Lookup Table requires four arguments: color, red component, green component, and blue component. If this sequence is begun by pressing the CHANGE key, the system will ignore all other commands until the four arguments required by the CHANGE command are satisfied. If the system appears to be suddenly unresponsive to commands, chances are good that an Escape or User code sequence has begun but has not been completed. Typing several commas, to satisfy any pending arguments, will usually regain control of the system.

Reset, Boot, Soft Boot

The 7900 recognizes several types of initialization procedures. At power-up time, a special type of initialization occurs, which erases all memory in the system (except CMOS). The image memory is erased, and all of user memory is zeroed.

Pressing the RESET key initializes all hardware to default states. All I/O ports are initialized, and all buffers are flushed. Any operations in progress are immediately halted. (In fact, Reset may be the only way out of some operations.) Following Reset, the Terminal Emulator program is executed by default (this may be altered with the THAW command). Reset leaves the contents of memory essentially unchanged. The Reset command may not be stored in the Create Buffer, since it is a hardware function and does not generate any code sequence. The most common use of the RESET key is to halt a process, such as a picture being drawn from the Create Buffer.

WARNING:

> Under Idris, do NOT press RESET unless you are shutting down the system or if it cannot respond to any other stimuli! If you are shutting Idris down, perform the "sync" command before hitting RESET. Failure to observe these warnings may result in a corrupted hard disk! After a reset, Idris will have to be rebooted.

After pressing the RESET key, it may be necessary to wait several seconds before the system will again respond to input. When the system acknowledges RESET, the green indicator near the cursor keypad will light briefly.

Boot is designed to cold-start the system. It is executed by holding down the CTRL key while pressing the key marked SOFT BOOT and BOOT. This key is found directly to the right of the RESET key. Boot initializes most of the 7900 system, based on default parameters (either in PROM or in the optional battery-powered CMOS memory). After Boot, the Terminal Emulator is executed unless CMOS contains orders to execute a different program. BOOT is the simplest way to reload most of the default parameters in the 7900 system. The Boot command may not be stored in the Create Buffer, and is ignored while running under Idris.

It is possible to simulate most of a power-up reset sequence, by pressing the three keys CTRL SHIFT RESET simultaneously (and releasing RESET before releasing the other keys). This sequence is equivalent to a Reset followed by a Boot, and also erases any image in memory. This

sequence takes longer to execute than the normal RESET, and is not usually needed. (It may be required if a user program writes into system memory and the RESET key will not recover control of the system.)

CMOS memory (optional) retains information concerning how the system is configured at Boot time. CMOS is set up using the Thaw command, to define buffer sizes and other defaults. As long as the data in CMOS remains intact, it continues to be used at Boot time. It is possible for a user to sufficiently scramble CMOS data so that the system cannot boot; in this case, the keys CTRL SHIFT M1 M2 RESET should be pressed simultaneously (releasing RESET first) to clear out CMOS and force the system to boot from PROM data instead. This sequence destroys any defaults entered by Thaw, so it is not recommended unless absolutely necessary. See "Thaw" for details.

NOTE:

> The preceding two sequences (M1 M2 CTRL SHIFT RESET and CTRL SHIFT RESET) should be used only when necessary. Do not get into the habit of using these sequences, since they will destroy any work in progress.

SOFT BOOT initializes only the window receiving the command. It reloads all default window parameters, such as color, character size, and other values associated with a window. It may be used to set a window to a known state at any time. Soft Boot is executed by pressing the SOFT BOOT key, and this command may be stored in the Create Buffer. It may be used at any time under Idris without disturbing the operating system.

Besides reloading the default window parameters, the SOFT BOOT key also erases the Overlay. To reload the window parameters without erasing the Overlay, enter the sequence:

MODE =

This performs the same functions as the SOFT BOOT key, but without erasing anything.

When beginning a new process on the 7900 system, it is usually sufficient to press the SOFT BOOT key. To interrupt a process, pressing RESET followed by SOFT BOOT will usually suffice. Neither of these key sequences will affect the Bitmap image.

Convergence and Degaussing

The 7900 analog circuitry requires periodic adjustment for best performance. Convergence adjusts the red, blue and green portions of the picture so that they join properly without "fringes" around the edges of the display. Degaussing demagnetizes the screen, removing residual magnetic fields which can affect color purity.

Controls for convergence and degaussing are located behind the door on the right side of the CRT (picture tube). These controls are located so that these operations may be conveniently performed whenever necessary.

The "DE-GAUSS" switch is located at the top of the recessed area behind the door. To determine whether the 7900 needs degaussing, perform the following steps:

1) Hold down the SHIFT key and press the key marked SET. The SET key is located above the "5" key.

2) Release SHIFT and press the blank red key. This key is to the right of the SET key.

3) Now press the grey key marked ERASE PAGE. This key is the top left of the cluster of grey keys to the right of the typewriter keyboard. The screen will be filled with red.

If the screen is not uniformly red (the red may tend toward green or purple in some areas), the 7900 should be degaussed.

To degauss the screen, press the DE-GAUSS button. After pressing the button, degaussing proceeds automatically for several seconds. Colors on the screen will shimmer for a moment as the degaussing is performed.

NOTE:

> You must wait at least 15 minutes after turning on the 7900 before degaussing will function. Once degaussing is complete, it may not be restarted for 15 minutes. If you press the switch again before 15 minutes have elapsed, nothing happens and you must wait another 15 minutes before trying again.

It is generally necessary to degauss the screen after moving the 7900, even if it has been moved across a room. A weekly degaussing is also beneficial.

Before converging the 7900, perform degaussing first. Then proceed with the steps below.

Below the DE-GAUSS button, there are nine groups of three small potentiometers (variable resistors; they look like set screws). These are the convergence adjustments. Each "pot" adjusts either red, green or blue, and each group adjusts one of nine sectors of the screen. For example, the upper left group adjusts convergence for the upper left corner of the screen.

NOTE:

> Use a small screwdriver (a jeweler's screwdriver is ideal) and BE GENTLE when adjusting convergence. The "pots" are fragile.
>
> Wait at least 15 minutes after turning on the 7900 to perform convergence, since it may drift until the 7900 is completely warmed up.

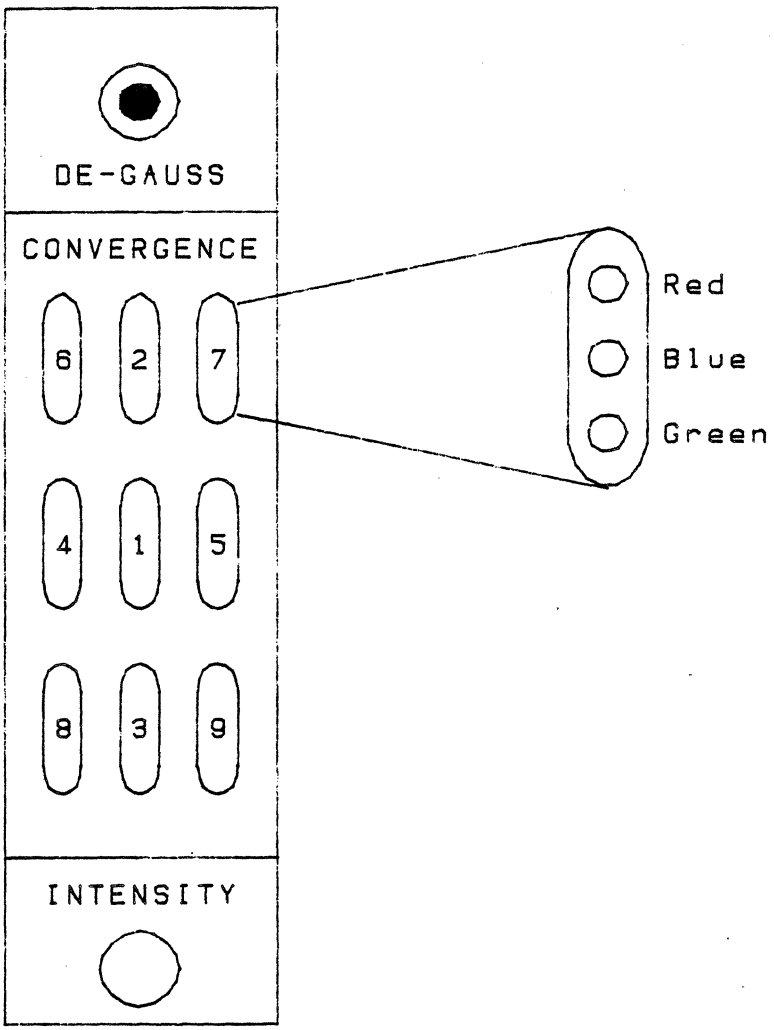
To set up a test pattern for convergence, perform the following steps:

1) Press the key marked SOFT BOOT. This key is next to the red RESET key at the upper left corner of the keyboard.

2) Press the key marked TEST. This key is adjacent to the SOFT BOOT key. Now press the period (.) key.

This produces a pattern of white dots on a black background.

Beginning with the center group, turn the screws until the dots in the center of the screen are solid white (with no colored fringes). Proceed to the other eight groups according to the numbers in the following diagram.



Convergence and Degaussing Controls

Chapter 2 -- The Overlay

The Overlay screen is a powerful feature of the 7900. It was designed to relieve the high-resolution Bitmap screen from time-consuming operations associated with text manipulation. The Overlay provides an area where messages to and from the operator may be displayed. It is compatible with several menu selection techniques, including the Light Pen and the Bezel Keys.

The Overlay also provides an environment which is isolated from the Bitmap screen, so drawings in the Bitmap will not be obliterated by prompts or error messages in the Overlay.

As an illustration, imagine the Overlay as being a screen in FRONT of the high-resolution Bitmap screen. The Overlay is organized as a set of character cells, arranged 85 horizontally and 48 vertically.

Each cell of the Overlay has the following attributes:

- Foreground Color (Foreground and Background color
- Background Color may be selected by any color key)
- Foreground Blink (on or off)
- Foreground Transparency
- Background Transparency
- Hardware cursor (available at each cell)

All 96 standard ASCII characters are available in the Overlay. The 32 control characters defined in ASCII may also be displayed, in "Visible Control-Character" mode. An additional set of 128 characters is available, and may be selected with the "Select Character Set" command, described in section 1. This set contains Greek characters and other special symbols.

All characters in the Overlay are formed from a 6 by 8 dot matrix. The ASCII character set falls into a 5 x 7 grid within this matrix.

A low resolution plotting mode is available in the Overlay, which allows any cell to display a group of eight pixels, two horizontally and four vertically. This provides a plotting capability which, in the Overlay alone, exceeds the resolution available on many other computers! The resolution of the Overlay plot mode is 170 (horizontal) by 192 (vertical). Some of the plotting features of the Bitmap screen are also available in the Overlay, such as drawing vectors.

Overlay Defaults

When the system is powered up, or when the BOOT or SOFT BOOT keys are struck, the Overlay is initialized with certain attributes as follows:

- Background Black
- Foreground White
- Cursor visible in upper left corner ("home" position)
- Visible (not transparent) Foreground and Background

SOFT BOOT and BOOT do not affect the high-resolution Bitmap screen. However, since the Overlay is not transparent at this time, any picture on the Bitmap screen will not be visible (see "Modify Overlay Visible Attributes"). If the system has just been powered up, no picture will be present on the Bitmap screen anyway.

Overlay Operations

When the system is initially powered up, or booted, you will be addressing the Overlay. This section discusses the types of commands accepted by the Overlay.

The Overlay is designed for easy communication with a host computer or other remote device. Its main purpose is to display alphanumerics; it can do this much faster than the Bitmap. Most messages to and from the operator will be displayed in the Overlay. For this reason, the Overlay is the default mode of operation when BOOT is executed.

When you are addressing the Overlay, the OVERLAY key will be lighted. Note that this light is on when the system is booted. When this key is extinguished, you are addressing the Bitmap, which is discussed in later sections.

Entering and Exiting the Overlay

If you are addressing the Overlay, and want to address the Bitmap, press:

SHIFT OVERLAY

Hold the SHIFT key and press the lighted key marked OVERLAY. The light will go out.

When you want to talk to the Overlay again, press

OVERLAY

The OVERLAY key produces the following code:

MODE O 1

MODE V 3

SHIFT OVERLAY produces the following code:

MODE O Ø

MODE V Ø

The MODE O command tells the 7900 whether the Overlay or Bitmap is to be addressed. The MODE V command is discussed later in this chapter.

Overlay Cursor Control

The Overlay cursor is a flashing white overscore and underscore. It is initially located in the upper left corner of the window, which is termed its "home" position. Pressing the HOME key on the cursor keypad will always return the cursor to this position.

Two keys on the typewriter section of the keyboard have an effect on the cursor. RETURN (carriage return) moves the cursor to the beginning of the current line. LF (line feed) moves the cursor down one line.

Tab stops are located every four character cells in the Overlay. Pressing the CTRL modifier with the I key will generate a Tab character, and will move the cursor to the next tab stop.

CTRL I

The other keys on the cursor keypad are used to position the cursor in the Overlay, and to perform text editing functions:

- The four arrow keys move the cursor in their respective directions. Simultaneously pressing two adjacent arrow keys moves the cursor diagonally.
- HOME moves the cursor to the upper left corner of the window. CTRL HOME moves the cursor to the LOWER left corner of the window ("home lower").
- ERASE PAGE clears the Overlay window to whatever background color is in effect (initially black).
- CLEAR LINE erases the line at which the cursor is currently positioned.
- The RECALL key is only active when you are entering input to the Disk Operating System, the Monitor, or some other controlling program. See the description of INLINE (the Inline Editor), in Appendix C.

The functions labelled in blue on the front of the cursor control keys are also text editing features. They are accessed by holding down the CTRL modifier and pressing the required key.

- ERASE EOS erases from the current cursor position to the end of the screen (the current window).
- CLEAR EOL clears from the current cursor position to the end of the same line.
- DEL LINE deletes the line at which the cursor is positioned. All lines below this one move up to fill in the gap.
- DEL CHAR deletes one character at the current cursor position. All characters to the right of the deleted character move left one position.
- INS CHAR moves all characters to the right of the cursor, one position to the right. One character may now be inserted in the gap.
- INS LINE moves all lines below the cursor, down one line. One line may now be inserted in the gap.

Several other commands are involved with the cursor:

CURSOR ON turns the cursor on, while using the SHIFT modifier with CURSOR ON turns the cursor off.

MOVE X-Y positions the cursor absolutely.

Format:

MOVE X-Y <X>,<Y>,

Where:

<X> is a decimal number between Ø and 84

<Y> is between Ø and 47

The cursor is moved immediately to the required coordinates. The coordinate system is arranged with Ø, Ø in the upper left corner. X increases from left to right. Y increases going down.

It is possible, using MOVE X-Y, to move the cursor outside the limits of the current window. Pressing HOME will bring it back inside the window.

The following command moves the cursor relative to its current screen location.

Format:

MODE m <dX>, <dY>,

Where:

<dX> and <dY> are each decimal numbers, followed by a comma.

The parameters <dX> and <dY> are added to the current cursor position. Note that a lower case **m** must be entered, and this may require use of the SHIFT key (depending on the state of ALPHA LOCK).

Overlay Cursor Blink ON/OFF

Format:

ESC q <Ø or 1>

NOTE:

This command requires entering a lower case letter, **q**. If the ALPHA LOCK key is in its normal (up) position, it is necessary to use the SHIFT key with the "Q" key to produce a lower case **q**.

The character Ø disables Overlay cursor blink, and the character 1 enables it (the default condition). If more than one Overlay cursor is in use, all are affected by this command: all Overlay cursors will blink, or none of them will.

Overlay Roll and Page

The Overlay defaults to Page operation. In Page, no lines of the window are ever moved. When the last line of the window is completed, the cursor moves to the first line of the window. If additional text is entered, it will overwrite the first lines of the window. Page is primarily useful for plotting applications, where it is imperative that output be fixed at a particular screen location.

In Roll, as the last line in the window is completed, the cursor remains on that line. The first line in the window is "rolled" off the top of the window, and is lost. All other lines in the window move up one line. Roll is standard for computer terminal applications.

To enter Roll, press the ROLL key:

ROLL

The light in the ROLL key will illuminate. The window is in Roll.

To leave Roll and return to Page, press

SHIFT ROLL

The ROLL key will extinguish. The window is back in Page.

Remember that each window has an Overlay part and a Bitmap part, and that the Roll command affects a window. This means that if a window is in Roll operation, both its Overlay and Bitmap parts are in Roll. If your application requires Overlay to be in Roll, and simultaneously requires that Bitmap be in Page, then you have two options:

1. Always transmit the Roll On/Off code whenever you move between Overlay and Bitmap
2. Use two windows, and leave one in Roll and one in Page.

Overlay Color and Blink

Each cell of the Overlay has a foreground and a background color. The available colors are the eight color keys on the keyboard, black through white. Additionally, each cell may be specified to blink. If blink is on, the character in the cell will blink from its normal foreground color to the background color of the same cell.

Each cell may also be transparent, in foreground or background or both. If part of a cell is transparent, the image in Bitmap may be seen behind the Overlay image. See "Modify Overlay Attributes" for details.

All of the commands in this section affect the future contents of the Overlay. The present color and blink attributes of the Overlay are unaffected by SET or BLINK commands. You must remember to specify color and blink before entering information into the Overlay.

NOTE:

Only the foreground color is allowed to blink in the Overlay. The foreground color will toggle between its normal color and the current background color. The background color of the character cell will never blink.

Set Foreground Color

Format:

SET <color>

<color> may be any one of the eight color keys. The Overlay foreground color is now set to the same color as the key.

Alternatively, a color number may be specified. Color numbers 0 through 7 in the Overlay correspond to the eight color keys, black through white. This form would most often be used from a host computer.

Alternate format:

SET <n>,

<n> is the color number, and must be terminated by a comma. The eight color keys correspond to the following color numbers:

Color Key	Color Number
Black	0
Blue	1
Green	2
Cyan	3
Red	4
Magenta	5
Yellow	6
White	7

Example:

SET 7, Set foreground to white (default)

These eight colors are the only colors available in the Overlay. These colors remain available in the Overlay, regardless of the color configuration of the Bitmap. (The Bitmap allows a selection from over 16 million colors. It is discussed in Chapter 8.)

Set Background Color

Format:

SHIFT SET <color>

Hold down the SHIFT modifier while striking the SET key to set background color. Foreground color is not affected.

As above, a color number may be used instead of a color key:

SHIFT SET <n>,

Overlay Blink ON

Format:

BLINK

The BLINK key will light up. The blink attribute is now on. Any characters now entered into the Overlay will have a blinking foreground. (Overlay background color cannot blink.)

Overlay Blink OFF

Format:

SHIFT BLINK

Holding down the SHIFT modifier while pressing the BLINK key will turn off the blink attribute. The light in the BLINK key will go out. Future characters entered into the Overlay will have a non-blinking foreground color. This is the default condition.

Modify Overlay Visible Attributes

Format:

MODE V <n> Modify Present Attributes

MODE v <n> Modify Future Attributes

Where <n> is interpreted as follows:

Ø Foreground and Background Transparent
1 Foreground Transparent (Visible Background)
2 Background Transparent (Visible Foreground)
3 Foreground and Background Visible

Modify Present Attributes will immediately cause the entire window to assume the attributes set by <n>.

- If <n>=Ø, the window will immediately become totally transparent (except for the Overlay cursor, if it is currently visible). The Bitmap image will be fully visible.

- If <n>=1, the entire window will assume a visible background, transparent foreground condition. The Bitmap screen will only be visible through the characters in the Overlay window.

- If <n>=2, the entire window will assume a visible foreground, transparent background condition. The Bitmap screen will be visible everywhere except where characters exist in the Overlay. This condition is useful for superimposing titles (in the Overlay) onto pictures (in Bitmap).

- If <n>=3, the entire window will be visible (foreground and background) and no Bitmap image will be seen through the window. Only Overlay material will be visible. This is the default condition at power-up.

Example:

MODE V 2 will cause lettering in the window to be visible over a picture in Bitmap.

MODE V 3 will cause the window to be entirely visible (non-transparent). This is the default condition when the system is booted or reset.

Modify Future Attributes will cause no immediate change in the Overlay window. However, any further characters sent to that window will be affected by the command. The interpretation of <n> is basically the same as above:

- If <n>=0, future characters sent to the window will not be visible at all.
- If <n>=1, future characters sent to the window will have a visible background and transparent foreground.
- If <n>=2, future characters sent to the window will have a transparent background and visible foreground, and will be superimposed over the image in Bitmap.
- If <n>=3, future characters sent to the window will have a visible foreground and background, and will completely block out the Bitmap image where they appear.

Example:

MODE v 0 will inhibit all characters sent into the Overlay window from being seen.

MODE v 2 will cause characters sent into the Overlay window to be superimposed over the image in Bitmap.

If a character in the Overlay has been specified to blink, and has visible foreground and transparent background, that character will blink from visible to transparent. For example, an important part of a drawing (in the Bitmap) could be labelled by lettering (in the Overlay) and if the lettering was specified to blink, the drawing could be seen in full whenever the characters blinked to transparent. (All 7900 blink functions occur at a rate of 1.9 Hertz, with a 50% duty cycle. This means a letter will be visible for 0.26 seconds and transparent for the next 0.26 seconds.)

Overlay Plotting Functions

Overlay plotting functions are extensions of plotting features available in the Bitmap. Each of the Plot Submodes is discussed in detail in Section 9. If you are not familiar with Bitmap plotting features, please read Sections 8 and 9 before attempting to use the Overlay for plotting.

The Overlay can produce many of the same kinds of figures available in the high-resolution Bitmap. All Plot Submodes operate in the Overlay (see Chapter 9). The Overlay resolution is 170 by 192 (as opposed to 1024 by 768 in the Bitmap).

Plotting in the Overlay works much like Bitmap plotting. Here are the differences:

- Scale factors are not allowed. All coordinate data should be entered as numbers between 0 and 169 for X, and 0 to 191 for Y.
- The cursor position may not be used to enter coordinates using the decimal point.
- The Overlay handles primitive filled figures. The optional Complex Area Fill and Edge Fill routines will not work, however.
- Pixels in the Overlay are 3 x 2, compared to 1 x 1 square pixels in the Bitmap. This causes circles to come out as ovals.

All currently defined foreground attributes are applied to the plotted figure, including transparency. You can plot a "transparent" figure and view the Bitmap through an irregularly shaped area.

Plotting in the Overlay is slow, and it is easy to accidentally switch to the Overlay from the Bitmap and end up drawing a large figure, say, a circle with a 500-pixel radius. This circle would wrap around, enveloping the Overlay; while the figure is being drawn, the system would appear to hang.

NOTE:

If the MODE V command is used to change the background visibility, a filled area may change. This is intentional and is required to allow multiple colored lines to be drawn at full resolution. Filled areas are filled using background character cells.

Chapter 3 -- System Commands

This section describes commands which affect the entire system. Unlike Mode or Plot commands (described in following chapters), these commands by themselves do not immediately change anything on the screen. Commands covered in this chapter include:

- Assign Devices
- Select Character Set
- Delay
- Function Keys
- Light Pen
- Real-Time Clock
- Test Mode
- Tone Generator
- Visible Control Characters
- Warmstart
- Windows
- Scaling
- Hardcopy

Assigning Physical Devices

Assign Output Device

Format:

ASSIGN OUTPUT <n> <dev> <dev> <dev> <dev>

Where:

<n> is the Logical Device number (Ø through 3)

<dev> is the letter corresponding to a physical device (see table 3-1)

Each Logical Output Device may access up to four physical devices. This allows transmitting commands to more than one window, or sending to a window and a peripheral device simultaneously.

All four physical devices must be specified. If less than four are required, you must enter dummy devices using the character Z.

Examples:

ASSIGN OUTPUT 1 A B Z Z

Logical Output Device number 1 has been connected to physical devices A and B (the first two windows). Two dummy devices are used.

ASSIGN OUTPUT Ø A K Z Z

Logical Output Device Ø has been connected to physical device A (the first window), and the keyboard lights (see below). No other physical devices are connected to Logical Output Device Ø. This is the default condition.

Physical Output Device K is the keyboard lights. The keyboard handler interprets all codes it receives, and decides which lights should be on or off. For example, when the FILL key is pressed, the "Fill" attribute is turned on. The keyboard recognizes the code sequence produced by the FILL key and turns on the light in that key.

It is possible to assign other Logical Output Devices to the keyboard lights, or to arrange the assignments so that the keyboard lights receive no information at all. If this is done, the keyboard lights may not reflect the current state of the system.

Example:

```
ASSIGN OUTPUT Ø A K T Z
```

Logical Output Device Ø has been connected to window A, to the keyboard lights (K), and to the tone generator (T). Physical output device T is a special configuration of the tone generator. When assigned in this manner, it will sound a "click" every time a key is pressed. Some users appreciate this feedback, especially if they are not used to typing on a computer keyboard. We have maintained a connection to the keyboard lights in this assignment.

The ASSIGN OUTPUT command can be used to change both the current window and the current Logical Device. For example,

```
ASSIGN OUTPUT 1 B K Z Z
```

makes window B the current window and changes the output to Logical Device 1. To return to Logical Device Ø, send another Assign Output command.

Under Idris, change to a different Logical Device by typing:

```
M1 M2 <num>
```

Where <num> is Ø through 3, corresponding to a Logical Device number. This avoids having to remember which Physical Devices are associated with each Logical Device.

Example:

```
ASSIGN OUTPUT Ø A K Z Z    (Logical Device Ø -> Window A)
WINDOW Ø,Ø, 511,383,       (Window A is upper left corner)
ASSIGN OUTPUT 1 B K Z Z    (Logical Device 1 -> Window B)
WINDOW 511,Ø, 1Ø23,383,    (Window B is upper right corner)
ASSIGN OUTPUT 2 C K Z Z    (Logical Device 2 -> Window C)
WINDOW Ø,383, 1Ø23,767,    (Window C is bottom half of screen)
M1 M2 Ø                    (Send output to Window A)
   ...                     (Commands to Window A)
M1 M2 2                    (Send output to Window C)
```

Assigning Input Devices

Format:

ASSIGN INPUT <n> <dev>

Input devices are assigned using the SHIFT modifier with the ASSIGN key. Only one physical input device is assigned to a Logical Input Device.

WARNING:

Logical Input Device 0 is normally assigned to the keyboard. Most programs will read from Logical Input Device 0 to receive their input. If you connect Logical Input Device 0 to another physical device, or to a dummy device, the system could hang. Recovery will only be possible via RESET.

This can be useful for disabling the keyboard while under control of a host system. RESET, however, will still function.

Table 3-1. Physical Device Assignment List

OUTPUT DEVICES

A window A
B window B
C window C
D window D
E window E
F window F
G window G
H window H

K keyboard lights

L RS-232 serial port
M RS-449 serial port

P parallel port

S Serial Port Controller port #1
T Serial Port Controller port #2
  Tone Generator (if SPC is not installed)
U Serial Port Controller port #3
V Serial Port Controller port #4

Z dummy

INPUT DEVICES

K keyboard

L RS-232 serial port
M RS-449 serial port

P parallel port

S Serial Port Controller port #1
T Serial Port Controller port #2
U Serial Port Controller port #3
V Serial Port Controller port #4

Z dummy

Select Character Set

Two character sets are supplied in the 7900. The standard character set contains 96 printable ASCII characters, plus 32 visible control characters. The standard character set is selected by default when the system is booted.

Format:

CTRL O ("A7 off" - standard character set)

CTRL N ("A7 on" - alternate character set)

The alternate character set is called the "A7" set. It contains graphics figures, Greek characters, and other special symbols. It may be selected at any time by pressing CTRL N (hold down the CTRL key and press N). Only the window which receives the command will switch to the alternate character set.

To return to the standard character set, hold down the CTRL key and press O.

The alternate character set is available in both Overlay and Bitmap. See Appendix F for the available character fonts. Note that the Overlay also has a Plot Dot capability, and this is not related to the character fonts. Plot dots are used for Overlay plotting features.

Delay

Format:

ESC D <n>,

Where:

<n> is the number of 60ths of a second to delay;
$-1 \leq n \leq 32767$.

Delay is often used in conjunction with pictures drawn in the Create Buffer (see Chapter 5). Delay allows time for a user to examine a drawing or read text. Delays may be specified to the nearest 60th of a second.

Example:

ESC D 120, Delay two seconds

If <n> is -1, the system will wait until a key is pressed on the keyboard:

ESC D -1, Delay until a key is pressed

ESC D 0, Wait for next vertical retrace

NOTE:

The Delay command polls the vertical retrace bit. If the processor is interrupting on vertical retrace (e.g., while running Idris), the delay time may be inaccurate. Also, the keyboard wait function assumes that the 7900 firmware has control of the keyboard interrupt vector (not true while running Idris). The system may hang if the vector was used earlier by another program. Therefore, do not use this command with Idris.

Function Keys

Format:

DEFINE <Fn> <command sequence> <Fn>

Where:

<Fn>	is a function key, F1 through F24
<command sequence>	is any set of characters and/or commands

The twenty-four function keys, F1 through F24, are user-definable. You may program these keys to perform any sequence of commands, and once programmed, the keys will remember their definitions until the definitions are changed. If your system is equipped with battery-powered CMOS memory (optional), then the user-defined keys will remain defined even while system power is off.

The function keys F13 through F24 are accessed by holding down the SHIFT key while pressing the appropriate function key. Thus,

SHIFT F1

is the same as

F13

To define a function key, press DEFINE, followed by the function key you wish to define. To program the F1 key, type

DEFINE F1

The light on top of the function key will come on to remind you that a function key is being defined. Then type any sequence of commands or characters. The keys you press will not have any effect on the display, but they will each be acknowledged by a "hissing" sound from the speaker.

NOTE:

The function key lights will NOT come on while any of F13-F24 are being defined; however, the speaker will still hiss during the definition.

When you have typed all the keystrokes you wish to enter into this function key definition, hit the function key again (the same key you started with). This will be acknowledged by a "gong" sound from the speaker, and the function key definition is now complete. Pressing the same function key at any future time will produce the stream of characters (or commands) which you entered as the definition.

Example to define function key F1:

DEFINE F1 T H I S SPACE I S SPACE A SPACE T E S T F1

Example to define function key F24:

DEFINE F24 1 2 3 4 5 6 7 8 9 0 F24

Note that to CLOSE the definition of F24, you must be sure to hold down the SHIFT key at the end of the definition. This is because a function key is allowed to "call" other function keys:

DEFINE F2 F1 F1 F1 F1 F2

Pressing F2 is now equivalent to pressing F1 four times.

Storage for function key definitions is dynamically allocated. A key will use only as much space as it needs. This means the available storage may be used all by one key, spread evenly among many keys, or distributed in any other way. The default function key buffer allocation is 768 bytes, in CMOS (if present) or in static RAM. The "Thaw" command may be used to alter this allocation.

If function key storage overflows while defining a key, the normal "gong" sound for closing the definition is produced, and no more characters are accepted. It is then necessary to "un-define" some keys (define them to be null) to provide space for future definitions. Since definitions are stored in battery-powered CMOS memory (if installed), it is possible for function key definitions to remain in the system long after their usefulness has ended.

Example to delete the definition for F1:

DEFINE F1 F1

Bezel Keys

Eight bezel keys are located on the bottom of the frame surrounding the 7900 screen. They are defined and used in the same way as the function keys described above.

Format:

DEFINE <bezel> <command sequence> <bezel>

Where:

<bezel>	is any one of the eight bezel keys
<command sequence>	is any stream of characters or commands

See the previous section for more information on user-defined keys. The SHIFT key may not be used with bezel keys to generate unique codes. If SHIFT is depressed while pressing a bezel key, it is ignored.

Overflows are handled the same way for bezel keys as for the other function keys.

The bezel keys have an additional use when debugging assembly language programs. See Appendix C for a description.

Defining Function Keys from a Host

Obviously, a host computer or a controlling program cannot push buttons to define function keys. There are three ways, however, that the function keys can be modified without touching the keyboard:

- Directly load the entire Function Key buffer. The function keys normally reside in CMOS RAM from hex address $E40900 to $E40BFF. These boundaries can be changed with Thaw (see Chapter 6). Note that this method loads all 32 function keys, wiping out any old definitions.
- From a program running under Idris, individual function keys can be modified using drivers **/dev/fkey1** through **/dev/fkey32**. The driver **/dev/fkeys** accesses the entire buffer.
- Use USER J and USER j commands to define individual function keys from a host or internal program. This is the easiest method to use from a host, and is discussed below.

To define a function key from a host or internal program, the proper input port must be assigned as the primary input device rather than the keyboard. Assuming your host is connected to the RS-232 port, the proper command would be:

ASSIGN INPUT ØL

- OR -

USER IØL

This code can be sent from the host or the keyboard.

The function keys can now be defined. Assuming that F1 is the desired key, the following sequence will define it:

USER j	(Code for the DEFINE key)
USER JA	(Code for function key F1)
<command sequence>	
USER JA	(Close the definition)

If you do this from the keyboard (get out of Idris or DOS first), you will see F1 light up after you type USER JA the first time. The system will "bong" as usual and turn off the function key light after the second USER JA is sent.

After defining the function keys, reconnect the keyboard with the following code:

USER IØK

Function keys can be "nested" in this manner, also:

USER j	(Code for DEFINE)
USER JB	(Code for function key F2)
USER JA USER JA	(Enter F1 twice)
USER JB	(Close F2)

Now, pressing F2 will have the same effect as pressing F1 twice.

To execute function key F1, simply send USER JA. Codes for the other keys are shown below.

Key	Char	Key	Char
F1	A	F19	S
F2	B	F2Ø	T
F3	C	F21	U
F4	D	F22	V
F5	E	F23	W
F6	F	F24	X
F7	G		
F8	H	Bezel 1	_ (underline)
F9	I	Bezel 2	` (accent grave)
F1Ø	J	Bezel 3	]
F11	K	Bezel 4	^
F12	L	Bezel 5	[
F13	M	Bezel 6	\
F14	N	Bezel 7	Y
F15	O	Bezel 8	Z
F16	P		
F17	Q		
F18	R		

For example, USER JI corresponds to function key F9.

Light Pen

The (optional) light pen is housed on the right side of the CGC 7900 display behind the tall, thin door. It is held in place by a clip behind the door. While the light pen is in use, the clip can be used to hold the pen in a convenient position.

To use the pen, hold its barrel in your right hand, and touch the pen's tip with your index finger. The finger makes electrical contact across the black insulating ring, and causes an interrupt in the 7900 system. Aim the pen at the screen, holding it at a right angle to the screen and about one inch away.

To enable the pen, use the following command:

USER | <f1> <f2> <n>

Where:

<f1> and <f2> are single characters, A through Z

<n> is a single digit, 0 through 4 (see below)

The character | is located on the same key with the backslash, \.

<f1> and <f2> are specifiers which indicate function keys to execute, before and after the light pen hit. The characters A through X are used to designate function keys F1 through F24 (see page 3-12). Enter a character outside this range to disable either or both of the function keys.

<n> is a flag which determines the state of the light pen:

<n>	Function
0	Disable light pen.
1	Enable light pen, using Bitmap coordinates, with blue flood.
2	Enable light pen, using Bitmap coordinates, without blue flood.
3	Enable light pen, using Overlay coordinates, with blue flood.
4	Enable light pen, using Overlay coordinates, without blue flood.

Blue flood occurs only in areas of the Overlay which are defined to be "foreground visible." See "Modify Overlay Visible Attributes" (in Chapter 2). If blue flood is

disabled, only bright blue or white areas of the screen will be recognized by the light pen.

Examples:

DEFINE F1 INC VECTOR F1	Define key F1 to be a Incremental Vector command. The INC VECTOR key is directly below function key F10; you have to hold down the SHIFT key while pressing INC VECTOR.
DEFINE F2 @CC@@cc@ F2	Define F2 to hold arguments to Incremental Vector.
USER | A B 1	Activate light pen with function keys A (F1) and B (F2), use Bitmap coordinates, and enable blue flood.
SHIFT OVERLAY	Send the results to the Bitmap.

Now, each time the light pen is detected on the screen, a small rectangle will be drawn.

Note that the light pen produces coordinates from 0 to 1023, if <n> is 1 or 2; it produces coordinates from 0 to 84 if <n> is 3 or 4. In either case, the Overlay foreground must be visible if blue flood is desired. The command MODE V 2 will make the Overlay window have a visible foreground.

Other examples:

USER | 000	Disable the pen.
USER | 001	Enable the pen to simply output coordinates each time a hit is detected. This is useful for "freehand" drawings (using Concatenated Vectors).

Real-Time Clock

The Real-Time Clock (optional) keeps track of months, days, the day of the week, hours, minutes, seconds, and fractions of seconds. Several commands are provided to set and display the time and date information.

Set Clock

Format:

USER q MM DD N hh mm ,

Where:

MM is the two-digit month code (01 through 12)

DD is the two-digit day code (01 through 31)

N is the day-of-week code (1=Sunday through 7=Saturday)

hh is the two-digit hours code (00 through 23)

mm is the two-digit minutes code (00 through 59)

NOTE:

This command requires entry of a lower case letter, **q**. If the ALPHA LOCK key is in its normal (up) position, you must use the SHIFT modifier with the "Q" key to produce a lower case **q**.

All of the above parameters are set simultaneously, by a single entry to the Real-Time Clock. It is not possible to set the parameters individually.

Either a comma or semicolon may be used to terminate this command.

Examples:

USER q 0101102235, (January 1, Sunday, 2:35 AM)

USER q 112141745, (November 21, Wednesday, 5:45 PM)

When setting the clock to an exact time, use the following procedure: Enter the date and time information as above. Enter the minutes as being one or two minutes ahead of exact time. Wait until the time you entered equals the exact time, THEN enter the comma.

NOTE:

Entering too many characters without the delimiter can cause a stack overflow.

Display Time

Format:

USER Q <0 or 1>

Use the character 1 to enable the time display, or 0 to disable it.

This command causes time and date information to be continuously displayed in the upper right corner of the Overlay. The information is updated once per second.

Entering a 0 prevents the displayed time from being updated, but does not remove the last displayed time from the Overlay screen. It is necessary to erase the Overlay, or make it transparent, to completely remove the clock display.

NOTE:

If your system is not equipped with the Real-Time Clock option, this command will cause the system to hang. Press RESET to recover.

This function will work under Idris, but the last digit will sometimes be obscured by the disk access indicator.

Test

Format:

TEST <character>

Where <character> is any displayable character.

The TEST command will fill the window in use with the specified character. TEST is available in both Overlay and Bitmap. Whichever window is currently being addressed will be filled with the character. (Either the Overlay window OR the Bitmap window -- not both -- will be filled with the character.)

In the Overlay, all currently defined attributes will be applied to the test character. This includes foreground and background color, foreground and background transparency, and blink.

In the Bitmap, the currently defined attributes will also be applied. Remember that these attributes may not be the same as the currently defined attributes of the Overlay. The currently defined foreground and background colors, blink, and character size, will be applied to the test character.

Following completion of the TEST function, the cursor will be placed in the "home" position of the window.

Tone Generator

Format:

TONE <A>,<B>,<C>,<L>,

Where:

<A>,<B>,<C>	are the frequencies (in hertz) of each of the three voices from the tone generator.
<L>	is the tone duration (in milliseconds).

The TONE command is located on the same key as TEST. Typing SHIFT TEST begins the Tone command.

The tone generator is programmable from the keyboard. You may program the frequency of each of the three voices and the duration of the tone. When programming from the keyboard, the envelope is fixed and causes the amplitude to decay steadily over the duration of the tone.

NOTE:

The QUIET LOCK key on the keyboard must be in its normal (up) position, or the speaker is disabled and no tones can come out!

The tone generator can produce tones covering nearly the entire audible range. The range of useful values for <A>, <B> and <C> is from 30 to 15000 (hertz).

NOTE:

Selecting a frequency of 0 on any channel turns that channel off.

The duration of the tone may be set to any value up to approximately 9300 (milliseconds). This would produce a tone 9.3 seconds long.

Example: (to select voice A to be "middle A" on the piano, or 440 Hz)

<A> = 440

If we want voice B to be one octave higher, then let <B> = 880. Then, we could let voice C could be one octave above that, which would make <C> = 1760.

Finally, select a duration of 1 second. Then <L> = 1000. Our finished command then becomes:

TONE 440,880,1760,1000,

The following list shows approximate frequencies for different notes. Some of them may be slightly "out of tune" because of roundoff error.

Note	Freq.	Note	Freq.	Note	Freq.	Note	Freq.
C 1	33	C 3	131	C 5	523	C 7	2093
C# 1	35	C# 3	139	C# 5	554	C# 7	2218
D 1	37	D 3	147	D 5	587	D 7	2349
D# 1	39	D# 3	156	D# 5	622	D# 7	2489
E 1	41	E 3	169	E 5	659	E 7	2637
F 1	44	F 3	175	F 5	699	F 7	2794
F# 1	46	F# 3	185	F# 5	740	F# 7	2960
G 1	49	G 3	196	G 5	784	G 7	3136
G# 1	52	G# 3	208	G# 5	831	G# 7	3322
A 1	55	A 3	220	A 5	880	A 7	3520
A# 1	58	A# 3	233	A# 5	932	A# 7	3729
B 1	62	B 3	247	B 5	988	B 7	3951
C 2	65	C 4	262	C 6	1047	C 8	4186
C# 2	69	C# 4	277	C# 6	1109	C# 8	4435
D 2	73	D 4	294	D 6	1175	D 8	4698
D# 2	78	D# 4	311	D# 6	1245	D# 8	4978
E 2	82	E 4	330	E 6	1319	E 8	5274
F 2	87	F 4	349	F 6	1397	F 8	5588
F# 2	93	F# 4	370	F# 6	1480	F# 8	5920
G 2	98	G 4	392	G 6	1568	G 8	6272
G# 2	104	G# 4	415	G# 6	1661	G# 8	6645
A 2	110	A 4	440	A 6	1760	A 8	7040
A# 2	117	A# 4	466	A# 6	1867	A# 8	7459
B 2	124	B 4	494	B 6	1976	B 8	7902

Visible Control Characters

The 7900 has the capacity to make control characters visible. Control characters are normally non-printing symbols which have an effect on the state of the system, such as Break or Return. For debugging purposes, it is sometimes useful to observe the characters themselves without letting them actually function.

Format:

ESC V <window> <0 or 1>

Where:

<window> is the name of the window where visible control characters are desired. This will usually be the default window, named A.

<0 or 1> is the character 0 or 1. 0 turns visible control characters off (the default condition) and 1 turns them on.

After visible control characters are turned on, all ASCII control characters sent to the window will be displayed in abbreviated form.

Example:

ESC V A 1

Visible control characters are now ON in window A. Now, if we strike the RETURN key, we see the symbol:

C
R

which is the abbreviation for Carriage Return. Other control characters may be displayed in the same way.

NOTE:

Escape and User codes will be executed (and not displayed) whether or not visible control characters are active.

Alternatively, a visible control character can be displayed by holding down the M1 and M2 keys and striking a control character. This is useful for quickly examining the symbol produced by a particular control character. M1 and M2 will always display a visible control character, regardless of whether the ESC V command was given.

Example:

M1 M2 ERASE PAGE

displays the abbreviation:

```
F
 F
```

which stands for Form Feed, the standard ASCII erase-page code.

To examine a control character which does not have a separate key assigned to it, you must hold the M1 and M2 keys and the CTRL modifier, and press the desired key.

Example:

M1 M2 CTRL D

displays

```
E
 T
```

which is the abbreviation for EOT, or End Of Transmission.

Note that some keys (especially the labelled keys in the upper part of the keyboard) produce several codes for each keystroke. This is normal. Appendix A provides a complete list of code sequences, and the advanced user will want to memorize many of these sequences.

Warm Start

Programs which interact with the user and take over control of the 7900 system may be re-entered after an interruption, if set up properly. To re-enter an interrupted program, execute the Warm Start sequence:

USER W

The "warmstart vector" is located at $0C8E (hex), and is four bytes long. Programs which use this feature must load the warmstart vector with some "warmstart address" so that typing USER W will cause the proper action to occur.

If a program is in a "runaway" condition, such as a very long listing that will not terminate, it is generally possible to stop the program by pressing RESET, followed by USER W. In a properly designed program, this will interrupt the current process but not destroy any data in memory.

The details of warm-starting a program will depend on the program itself. Consult the instructions for the program in question.

Windows

A window is one of the system's primary output devices. It may occupy the full screen, or any rectangular subset of the screen. Its limits may be defined in the Overlay, or in Bitmap. It may have different dimensions in each.

A window is handled in the same manner as other physical devices. When characters or commands are sent to a window, it will take some action based on the data it receives (and on previous data). For example, if a window receives a "set color" command, all future text sent to that window will be affected by that command, until another "set color" command arrives.

The system will handle up to eight windows. When the system is initialized, only window "A" is assigned, so the existence of the other windows is not immediately apparent. Other windows are assigned with the "Assign Input" command (see page 3-4). Unless otherwise assigned, all primary output from the system is directed to window A.

The 7900 defaults to four active windows, named A, B, C, and D. The number of active windows may be set by Thaw, up to eight maximum.

Window A is termed the master window. In some cases it is necessary for the system to reference a known window, in order to properly execute a command. (For example, Pan and Zoom make reference to the current cursor position. Since each window may have its own cursor, the system must choose a particular window. The master window is always chosen in these cases.)

Set Window Limits

The dimensions of the window you are currently addressing may be set with the WINDOW command.

Format:

WINDOW <X1>, <Y1>, <X2>, <Y2>,

Where <X1> and <Y1> are the coordinates of one corner of the window, and <X2> and <Y2> are the coordinates of the diagonally opposite corner.

Following the WINDOW command, the cursor will be moved to the "home" position (upper left corner) of the newly defined window.

An alternate form of the WINDOW command may be used when the new dimensions are located inside the previous dimensions:

WINDOW . .

To use this form of the command, first press the WINDOW key. Move the cursor to where one of the corners of the window will be. Press the "period" (decimal point) key. Move the cursor diagonally opposite to the corner of the desired window and press the period key again. The cursor will move to its "home" position and the limits of the window have now been defined. The cursor control keys will not be able to move the cursor outside the window limits. Any text written into the window will remain inside the established limits.

Windows may be defined and redefined at any time during the execution of a user program. Text or figures may be created inside a window; then, the limits of the window may be redefined so as to protect that text from being altered. Text editing commands, such as ERASE PAGE, will not affect areas outside the window to which they are sent. In fact, nearly all Mode and Plot code sequences affect only the window to which they are sent. Thus, such attributes as color, transparency, plot submodes, blink, fill, character size, and many others, are specific to a window. Only the window receiving such commands will act upon them.

Window dimensions are defined differently in Overlay and in the Bitmap. A WINDOW command given while addressing the Bitmap will define limits only for Bitmap operation. A WINDOW command given while addressing the Overlay will define limits for the window only in Overlay operation. Thus a single window can have different limits in Overlay and in Bitmap.

The possible ranges for window limits are as follows:

Overlay: Ø <= X <= 84
Ø <= Y <= 47

Bitmap: Ø <= X <= 1Ø23
Ø <= Y <= 1Ø23

Where X is either <X1> or <X2>, and Y is either <Y1> or <Y2>, in the WINDOW command format shown above.

One alternate form of the WINDOW command is available to reset a window's limits to the maximum size possible. This relieves the operator of remembering what the maximum limits are for each type of window.

WINDOW Ø,Ø,-1,-1,

or, SHIFT WINDOW

This will reset an Overlay window to Ø, Ø, 84, 47, or a Bitmap window to Ø, Ø, 1Ø23, 1Ø23, depending on whether you are addressing the Overlay or Bitmap. The SHIFT WINDOW command may only be used from the keyboard; the other version may be used from a host or a program as well.

Example: (in the Overlay)

WINDOW Ø,Ø,84,23,

The window size has been set to the upper half of the overlay screen. The cursor control keys will not move the cursor outside this area.

Example: (in the Bitmap)

WINDOW Ø,Ø,511,383,

The window size has been set to the upper left-hand corner of the visible Bitmap screen.

You can easily see the limits of the current window by changing the background color and erasing the window to that color:

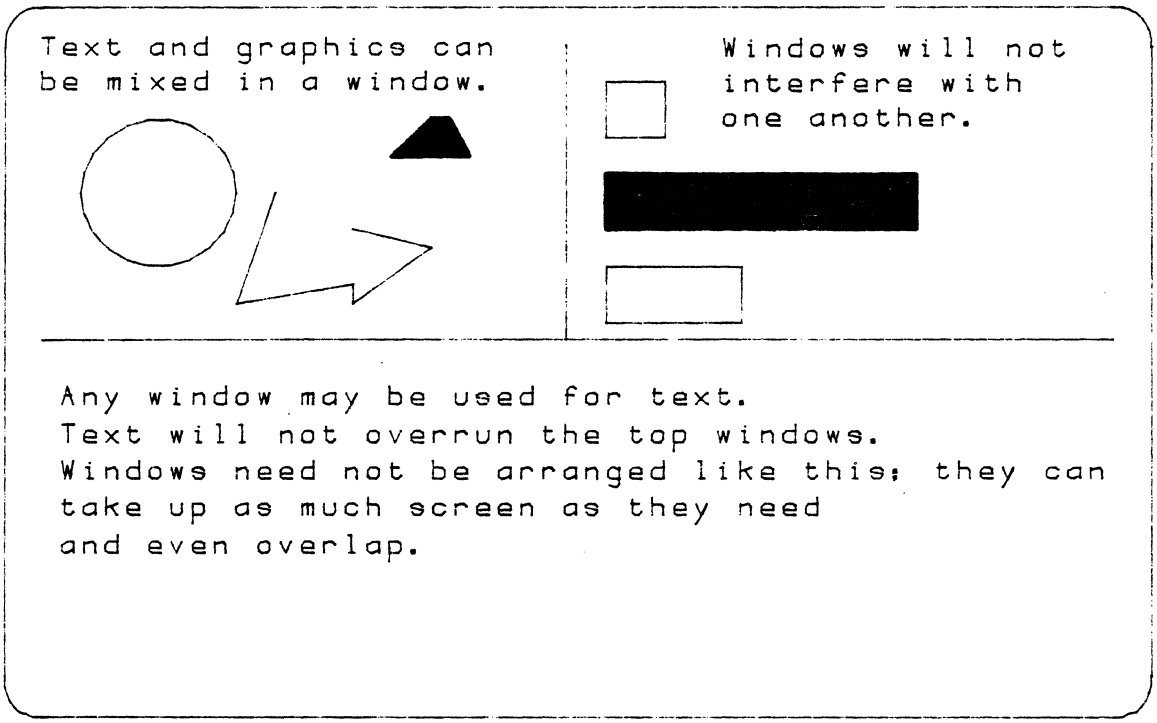
SHIFT SET GREEN (set background color to green)

ERASE PAGE (clear the window to green)

NOTE:

Window restrictions only control the coordinate location of cursor movements when used with character operations. Character cells will not clip on window edges if forced across an edge with a Move Cursor operation.

On all commands that reference a coordinate, coordinates are expected to lie within the currently scaled limits. If clipping is active, the values still should lie within this area but may be clipped to a smaller window.



Example of CGC 7900 Window Usage

Scale Factors (Virtual Coordinates)

Format:

SCALE <X1>,<Y1>, <X2>,<Y2>,

Where:

<X1>,<Y1>, are the desired coordinates of the upper left corner of the window

<X2>,<Y2>, are the desired coordinates of the lower right corner

All numbers must be between -16384 and +16383.

The SCALE command forces a window to assume a set of coordinates. After the command, any coordinates sent to the window are scaled according to the coordinates given. SCALE is an easy way to adjust the size or aspect ratio of a plot, or to plot data which was formatted for a different device.

NOTE:

Scaling out of range, or meaningless scaling (such as scaling down to one pixel) may produce unexpected results.

Press BOOT before each example in this section to avoid interaction between the examples.

Example:

```
BOOT

SHIFT OVERLAY

WINDOW 0,0, 1023,767,

SCALE -1000,1000, 1000,-1000,
```

The upper left corner of the window now has coordinates -1000, 1000. The lower right corner of the window now has coordinates 1000, -1000. The center of the window now has coordinates 0, 0. This would be handy for plotting data points which are symmetric about the origin. You can easily see the effect of this command now by performing some MOVE X-Y instructions. MOVE X-Y to 0, 0 will place the cursor in the center of the window, rather than the upper left corner.

NOTE:

In this example we have caused the entire width of the screen (1024 pixels) to have a coordinate distance of 2000. The entire height of the screen (768 pixels) also has a coordinate distance of 2000. Our coordinate system now does not have a square aspect ratio. One unit of travel in the X direction is 4/3 as many pixels as one unit of travel in the Y direction.

To overcome this problem, we should make our scaling data have the same aspect ratio as the CRT, which is 4 horizontal to 3 vertical.

Example:

```
BOOT

SHIFT OVERLAY

WINDOW 0,0, 1023,767,

SCALE -2000,1500, 2000,-1500,
```

Now the upper left corner has coordinates -2000, 1500, and the lower right corner has coordinates 2000, -1500. The coordinate system encompasses 4000 units of X and 3000 units of Y, so our aspect ratio is preserved at 4 to 3 (assuming the window is the full screen -- see below.)

```
CIRCLE 0,0,1000,
```

After giving the SCALE command above, we can draw a circle with center 0, 0, and radius 1000, and it fits neatly onto the center of the window.

SCALE can also be used to reflect or mirror-image a plot.

Example:

```
BOOT

SHIFT OVERLAY

WINDOW 0,0, 1023,767,

SCALE 1023,767, 0,0,
```

Since the normal coordinates for the upper left corner of the screen are 0, 0, and the normal coordinates for the lower right corner for the screen are 1023, 767, the above example reverses them. Any figure drawn now will be reflected about a line from the upper left to lower right. It would also be possible to flip only the X coordinate, or only the Y coordinate.

It is important to remember that SCALE causes coordinate data to be scaled before it is processed by the window. It is as if your program performed scaling and translation before sending coordinates to the window. ONLY coordinate data are scaled; relative data (such as cursor key movements) are unaffected.

It is possible for different scale factors to be used in the X and Y directions.

Example:

```
BOOT

SHIFT OVERLAY

WINDOW 0,0, 1023,767,

SCALE 0,0, 10230,767,
```

In this case, only one change has been made: the X axis is now labelled 0 to 10230 instead of 0 to 1023. We have introduced a scale factor of 10 in the X direction. The Y scale factor is unchanged. This kind of scaling can produce a problem. Using the scale factors above, suppose we try to draw a circle. We supply the center X and Y coordinates and the radius, as usual:

```
CIRCLE 5110,383, 200,
```

The given X coordinate (5110) is adjusted by the current scale factor (10) to decide that the X coordinate of the circle's center is at pixel 511. The Y coordinate is adjusted similarly, but since its scale factor is 1, the Y coordinate remains 383. In "absolute" coordinates then, the circle is centered at 511, 383, which is the center of the screen.

But the radius of the circle must be adjusted by one of these scale factors as well. In this situation, where it is ambiguous which factor should be used, the X scale factor is always chosen. The given radius of 200 is adjusted by the X scale factor of 10, and a circle is drawn with a radius of 20 pixels.

NOTE:

Most of the examples in this manual assume that no scale factors are in use. You may get quite interesting (and unexpected) results if you use the SCALE command prior to executing the examples in other sections of this manual.

Extreme scale changes can cause inaccurate scaling.

Scaling ON/OFF

Format:

MODE S <Ø or 1>

Use the character Ø to disable scaling, or 1 to re-enable it.

After establishing scale factors with the SCALE command, you can disable scaling with MODE S Ø. Later, if you wish, you can enable scaling again with MODE S 1.

MODE S 1 will re-enable scaling, using the same factors you set up when you gave the SCALE command. Note that SCALE automatically turns scaling on when it executes.

From the keyboard, you may also disable scaling by the command

SHIFT SCALE

which is equivalent to MODE S Ø.

You may want to disable scaling when defining window limits, or when performing other actions that need absolute coordinate references. See the example below, concerning the possible conflicts between WINDOW and SCALE.

Window and Scale

A potential problem arises when using both the WINDOW and SCALE commands. The results depend on which command is used first.

Example:

WINDOW 0,0, 511,383,

SCALE -1000,1000, 1000,-1000,

The window limits have been set to the upper left quarter of the screen. Then, the coordinate space of the window has been defined to be from -1000 to +1000 in both X and Y directions. The window limits are still confined to the upper left quadrant of the screen, however. You can see this by the command sequence

SHIFT SET BLUE (Set background color to Blue)

ERASE PAGE (Clear the window to background color)

If we do the same steps in a different order, we get an entirely different result.

Example:

SHIFT SCALE (Turn off scaling)

SHIFT WINDOW (Set window size to full screen to undo the above example)

SCALE -1000,1000, 1000,-1000,

WINDOW 0,0, 511,383,

First, the SCALE command causes the screen corners to assume coordinates of -1000, 1000, and 1000, -1000. Then the WINDOW command, acting within these new coordinates, defines the window limits to be 0, 0 (the center of the screen) to 511, 383 (a point not far from the center). Now try the same sequence:

SHIFT SET BLUE

ERASE PAGE

The window limits of 0, 0 to 511, 383 are now acting within a coordinate space of plus and minus 1000. Instead of covering the upper left quarter of the screen, the blue background now covers only a small area near the center.

Defining a window, and then scaling it, can be very useful. A window can be defined and then scaled to the full screen coordinates.

Example:

```
WINDOW 0,0, 511,511,
SCALE 0,0, 1023,1023,

    <full-scale drawing>

SHIFT SCALE
WINDOW 512,0,1023,511,
SCALE 0,0, 1023,1023,

    <another full-scale drawing>
```

This is an easy way to put two (normally) full-sized drawings on the screen at once. Up to four drawings can be drawn at this size; scaling smaller windows will allow more simultaneous displays.

NOTE:

> The easiest way to set both window size and scale factors to default is to enter SHIFT SCALE and SHIFT WINDOW (in that order).

Hardcopy

The CGC 7900 can interface to several hardcopy devices, including the ACT1 printer and the Versatec V-80 printer/plotter.

Software for these drivers are shipped either on DOS disks or in PROM. Depending on the driver, the Bitmap (or the visible image) can be copied to the device by pressing:

SHIFT COPY

- OR -

USER h

Complete instructions are shipped with the interface.

Chapter 4 -- Terminal Emulator (TERMEM)

The Terminal Emulator (TERMEM) is a program which executes automatically when the 7900 is booted. It configures the system to act like a communications terminal. The primary function of the Terminal Emulator is to read characters from input devices and transmit these characters to output devices.

The default input devices are the keyboard (device K), and RS232 serial port (device L). The default output devices are Window A (device A), and the RS232 serial port (device L). As is the case with most of the 7900's defaults, these assignments may easily be changed.

TERMEM may run in local, half duplex or full duplex modes, with a wide variety of baud rates to suit many peripheral or host devices. This section describes each configuration, and options available in TERMEM.

NOTE:

> Local, half duplex, and full duplex modes have no meaning except within a program such as the Terminal Emulator. Other programs may also communicate with external devices, but they do so through the logical device assignment system. See "Assigning Physical Devices" in this chapter.

If another program is running, the Terminal Emulator may be re-entered by pressing the key marked TERMINAL. (This key is disabled while running Idris.)

The code sequence structure of the CGC 7900 is affected by the configuration (half duplex, full duplex or local) of TERMEM. Escape and User code sequences are processed on the input stream, so typing an Escape or User code sequence will always result in immediate processing, regardless of the system configuration.

On the other hand, Mode and Plot codes are not processed until they reach an output device which recognizes them (the windows). If the 7900 is operating as a full duplex terminal, the host system must echo Mode and Plot codes or they will not be executed by the 7900.

If it is necessary to transmit an Escape or User code to the host system, and NOT have this code trapped by the 7900 code processor, you must hit Escape or User twice in a row:

ESC ESC transmit ONE Esc to the host

Restricted Terminal Emulator

A restricted version of TERMEM is available by typing the command:

USER `

The restricted Terminal Emulator is similar to the normal TERMEM, except that control codes from a host system are trapped. This prevents the host from putting the system in an unknown state.

The following control codes are accepted by the restricted TERMEM:

- Return
- Linefeed
- Bell
- Backspace
- Tab
- Erase Page

Other control codes sent from the host are displayed as visible control-characters for diagnosis.

All Escape and User sequences will be executed normally from the 7900 keyboard. Mode and Plot codes will execute from the 7900 keyboard normally ONLY if the system is not in Full Duplex mode. (In full duplex, Mode and Plot codes are not executed until they are echoed from the host, and the restricted TERMEM traps them).

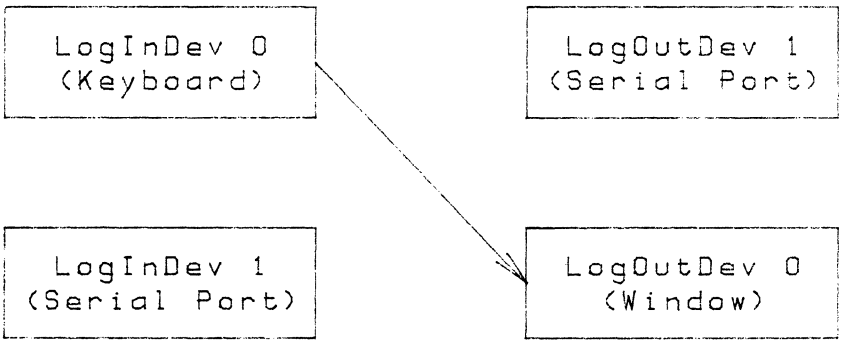
Local Mode

Local operation is one of the three communications arrangements which are provided. In Local operation, Logical Input Device 0 (normally the keyboard) is connected to Logical Output Device 0 (normally a window). The external device is ignored. This is the default arrangement, and must be used whenever a host or peripheral device is not connected.

Format:

USER L (Local operation)

Hold down the SHIFT modifier and press the USER key. Then press the L key.



(The device assignments in parentheses are default assignments. These may be altered with the ASSIGN key.)

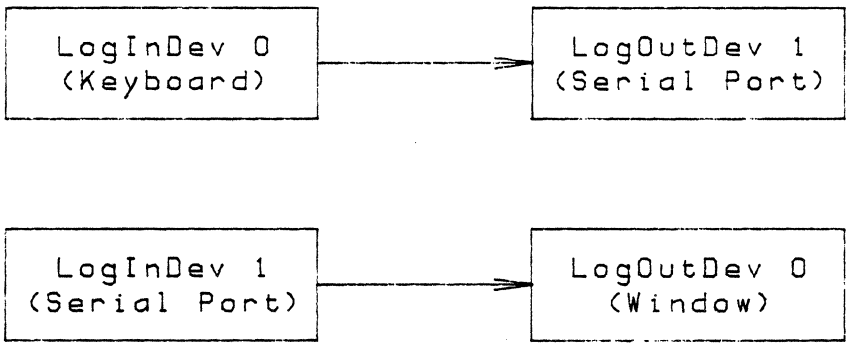
Half Duplex Mode

In half duplex mode, Logical Input Device Ø is connected to Logical Output Devices Ø and 1. This is the only case where one Logical Input Device is connected to more than one Logical Output Device (see "Assigning Physical Devices"). Also, in half duplex, Logical Input Device 1 is connected to Logical Output Device Ø.

Format:

USER H (Half duplex operation)

The following figure illustrates the connections defined in Half duplex.



(The device assignments in parentheses are default assignments. These may be altered with the ASSIGN key.)

Half duplex is used when it is necessary to transmit keyboard data to an external device and simultaneously display it on the screen.

Full Duplex Mode

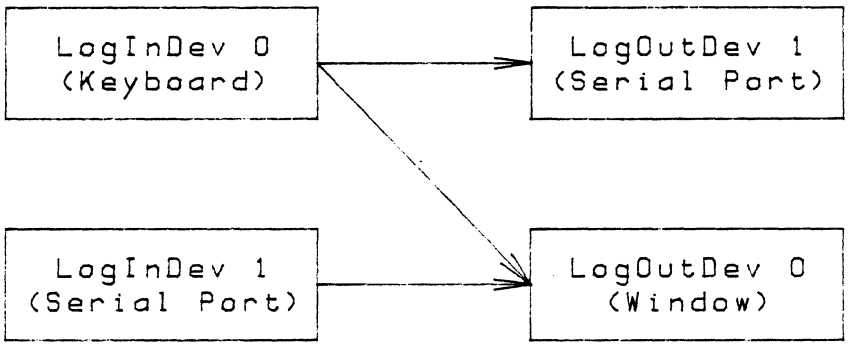
If the external device is a host computer, or other intelligent interface, full duplex mode may be required. Full duplex is similar to half duplex, in that it defines a path to an external device. However, in full duplex mode, the external device is required to echo back the characters it receives. Thus, anything typed on the keyboard will be received back from the external device as an echo, and the double assignment shown above is not required.

All data received from the host will be displayed, whether it originated as data from the 7900 or was originated by the host.

Format:

USER F (Full duplex operation)

The following figure illustrates the connections defined in full duplex.



(The device assignments in parentheses are default assignments. These may be altered using the ASSIGN key.)

Keyboard Sync

The keyboard sync command forces the lighted keys on the keyboard to accurately reflect the status of the current window (the window receiving the Keyboard Sync command).

Format:

MODE ?

Remember that most Mode code sequences alter only the state of the window which receives them, and consider the following set of events:

1) The system is addressing window A (the default window), and the user presses one of the lighted keys.

2) The key lights up to indicate the window has changed states. For example, assume the user has turned on the "Blink" attribute, and the BLINK key is now lit. (Appendix A tells us that the "Blink" command is a Mode code sequence, and will only affect the window which receives it.)

Now, suppose the user assigns the output to window B (using the "Assign" command above). Window B has not received a "Blink On" command, so it does not have its "Blink" attribute turned on. But the keyboard lights still reflect the status of window A! This can result in considerable confusion. To avoid these problems, always perform two steps whenever assigning output to a window:

1) Remember to assign the keyboard lights (device K) whenever assigning output to a new window.

2) Immediately after assigning output to a new window, perform a Keyboard Sync (MODE ?) command.

Serial Communications

The 7900 is equipped with two serial ports, for communication to a host computer or other device. One of these ports is RS-232, the other is RS-449. (RS-232 has been the "industry standard" interface for data communications for many years, and RS-449 is a new standard now coming into use.) A Case Table provides translation of characters to and from the host system.

Each port uses an interrupt-driven input and output buffer. The default size of each buffer is 2048 bytes. Each port defaults to 7 bits, even parity, one stop bit, 9600 baud. All of these defaults may be altered using the "Thaw" command.

Often, it will not be possible to transmit data continuously into or out of the 7900, due to delays in processing data. The interrupt-driven buffers allow the system to handle a certain amount of delay, but if the buffers become full, some form of handshaking is required so that the receiving system may tell the transmitting system to pause. The 7900 recognizes two types of handshake: software handshake (also known as X-On / X-Off protocol), and hardware handshake.

When connecting the serial ports for software handshake (or if no handshake will be required), the ports should be connected according to the diagram on the following page. Since many systems use non-standard wiring on serial ports, you should study this diagram carefully and connect ONLY those pins required for your application. RS-232 communication generally requires only 3 wires.

If hardware handshake is required, you must connect other signals:

DTR Data Terminal Ready, and the equivalent RS-449 signal, indicates that the port is able to receive data.

DSR Data Set Ready, and the equivalent RS-449 signal, determines whether the host is prepared to accept data.

Use these two signals only if hardware handshaking is specified. For software handshaking, the 7900 asserts DTR always true, and ignores DSR. Note that RS-449 has two signals which correspond to Data Set Ready; these are DM (Data Mode) and TM (Test Mode). If either of these signals are true, they indicate a "ready" condition.

The diagrams also show how an RS-449 port may be used to communicate with an RS-232 device. This arrangement will often work, but success will be determined by the exact nature of the equipment at each end. It is important to realize that an RS-449 receiver will withstand signals up to plus or minus 15 volts, but some RS-232 drivers will produce up to 25 volts and could damage the RS-449 receiver. (There is no danger if you are interfacing between Chromatics CG Series and/or 7900 Series computers, since the maximum voltage used in these systems is 12 volts.)

RS-232 communications cables, or cables between RS-232 and RS-449 ports, should be kept under 50 feet in length. A cable between RS-449 devices may be up to 2000 feet in length.

The 7900 asserts RTS (Request To Send) always TRUE.

Table 4-1. Serial Port Pinouts

RS-232 (25-pin connector)

Pin #	Signal	Description
2	TxD	Transmitted Data (output)
3	RxD	Received Data (input)
4	RTS	Request To Send (output)
5	CTS	Clear To Send (input)
6	DSR	Data Set Ready (input)
7	Gnd	Signal Ground
20	DTR	Data Terminal Ready (output)

Other pins are not connected in the 7900.

RS-449 (37-pin connector)

Pin #	Signal	Description
4	SD-A	Send Data (output: equiv. to TxD)
22	SD-B	
6	RD-A	Receive Data (input: equiv. to RxD)
24	RD-B	
7	RS-A	Request to Send (output: equiv. to RTS)
25	RS-B	
9	CS-A	Clear to Send (input: equiv. to CTS)
27	CS-B	
11	DM-A	Data Mode (input: equiv. to DSR)
29	DM-B	
13	RR-A	Receiver Ready (output: equiv. to DTR)
31	RR-B	
18	TM	Test Mode (input: equiv. to DSR)
20	RC	Receiver Common (used for TM only)
19	Gnd	Signal Ground

Other pins are not connected in the 7900.

Serial Port Connectors

The two serial port connectors are located on the rear edge of the CPU card, on the right side of the chassis (see the figure below). After connecting your cables, the rear door of the 7900 should be closed to maintain proper ventilation. Sufficient clearance exists under the door to allow a serial port cable to exit.

Recommended Serial Port Wiring

The connections below are recommended for applications where software handshaking (X-On / X-Off) will be used. If a modem is used (instead of a terminal), the definitions of "transmit" and "receive" must be reversed at the modem connector.

Table 4-2. Serial Port Connections (for software handshake)

RS-232 Terminal			RS-232 Terminal	
TxD	2	----------	3	RxD
RxD	3	----------	2	TxD
Gnd	7	----------	7	Gnd

RS-232 Terminal			RS-449 Terminal	
TxD	2	----------	24	RD-B
Gnd	7	----------	6	RD-A
RxD	3	----------	22	SD-B

RS-449 Terminal			RS-449 Terminal	
SD-A	4	----------	6	RD-A
SD-B	22	----------	24	RD-B
RD-A	6	----------	4	SD-A
RD-B	24	----------	22	SD-B
Gnd	19	----------	19	Gnd (see text)

If hardware handshaking is required, the following wires will also need to be connected (in addition to the wires listed above):

Table 4-3. Serial Port Connections (for hardware handshake)

RS-232 Terminal			RS-232 Terminal	
DTR	20	----------	6	DSR
DSR	6	----------	20	DTR

RS-232 Terminal			RS-449 Terminal	
DTR	20	----------	29	DM-B
Gnd	7	----------	11	DM-A
DSR	6	----------	31	RR-B

RS-449 Terminal			RS-449 Terminal	
RR-A	13	----------	11	DM-A
RR-B	31	----------	29	DM-B
DM-A	11	----------	13	RR-A
DM-B	29	----------	31	RR-B

Set Serial Baud Rate

Format:

USER S <port #>, <baud rate>,

Where:

<port #> is the identifying number of the serial port

<baud rate> is the baud rate to which the port will be set

Two serial ports are installed in the standard CGC 7900 system. They are identified by number as follows:

RS-232 serial port: 0

RS-449 serial port: 1

Sixteen standard baud rates are provided:

50
75
110
134.5
150
300
600
1200
1800
2000
2400
3600
4800
7200
9600
19200

NOTE:

To set the baud rate to 134.5, you must enter the number 134 as <baud rate>. Decimal fractions are not accepted.

If an unrecognized baud rate is entered, no action will be taken.

Examples:

USER S 0,300, (set RS-232 port to 300 baud)

USER S 1,1200, (set RS-449 port to 1200 baud)

The system defaults to 9600 baud for both serial ports. This may be changed using the "Thaw" command.

Set Serial Parity, Word Length, Stop Bits

Format:

USER s <port #>, <bits> <parity> <stop>

Where:

<port #> is the identifying number of the serial port

<bits> specifies the number of bits per character (5, 6, 7, or 8)

<parity> specifies EVEN, ODD, or NO parity (E, O, or N)

<stop> specifies the number of stop bits per character (see below)

NOTE:

Each of the three parameters <bits>, <parity>, and <stop> is entered as a single character. No commas are used to separate them.

Two serial ports are installed in the standard CGC 7900 system. They are identified by number as follows:

RS-232 serial port: 0

RS-449 serial port: 1

<bits> is a single character. It may be either the character **5, 6, 7,** or **8.** <bits> determines the number of data bits transmitted for each character sent out the serial port (excluding parity bits).

<parity> is the single character **E, O** or **N.** It determines even or odd parity, or no parity.

<stop> is the single character **1, 2** or **3.** If <stop> = 1, one stop bit is transmitted after each character. If <stop> = 2, two stop bits are transmitted. <stop> = 3 is a special case and produces 1.5 stop bits per character.

Examples:

USER s 0, 7N1 7 bits, no parity, one stop bit.

USER s 1, 5E2 5 bits, even parity, two stop bits.

Both serial ports default to 7 bits, even parity, one stop bit. This corresponds to standard ASCII transmission. (This can be changed using the "Thaw" command.)

The three parameters may be entered in any order; thus, 7E1 is the same as 17E or E17.

NOTE:

Entering an invalid sequence of characters for <bits>, <parity>, or <stop> may hang the serial port.

Reconfigure SPC Ports

The Chromatics Serial Port Controller (SPC) is an optional plug-in board. It adds four RS-232 serial ports to the 7900. The ports are numbered **S, T, U** and **V** (see Table 3-1). All SPC ports have the same pinouts as shown in previous diagrams.

All four ports default to software handshaking (X-ON/X-OFF), 7 bits, even parity and one stop bit. The default baud rates are 9600 baud for devices S and T (ports 0 and 1), 1200 for device U (port 2) and 300 for device V (port 3).

A "reconfigure" command allows you to alter these default settings.

Format:

USER r <p>,<h>,<b>, <bits><parity><stop>;

Where:

<p> is the port number on the SPC board (0 to 3)

<h> is the handshaking code for the port (0 to 3)

<b> is the baud rate. The available baud rates are the same as those for the CPU RS-232 port.

<bits> is the number of bits per character (5 to 8)

<parity> is the parity type (E, O or N)

<stop> is the number of stop bits (1 to 3 -- 3 produces 1.5 stop bits per character)

Handshaking codes are as follows:

0 No handshaking
1 Software handshake (X-ON/X-OFF)
2 Hardware handshake (using DTR and DSR)
3 Both software and hardware handshaking)

Example:

USER r 0,2,4800,8N2;

This example would set device S (port 0) to hardware handshake, 4800 baud, 8 bits per character, no parity and 2 stop bits.

Set Host EOL Sequence

The Set Host EOL (End-Of-Line) Sequence command is used in conjunction with other commands, to allow a host system to read data from the 7900 system. The commands which transmit the values of Window Variables also transmit the EOL Sequence.

Format:

USER z <n>,<n>,<n>,<n>,<n>,<n>,<n>,<n>,

Where:

<n> is the decimal equivalent of an ASCII character.

NOTE:

This command requires entering a lower case character, **z**. If the ALPHA LOCK key is in its normal (up) position, press the SHIFT key with the "Z" key to produce a lower case **z**.

The Host EOL Sequence may contain up to eight characters. Each character is entered as a decimal number. Refer to Appendix F for the decimal equivalents of ASCII characters. If your required EOL Sequence includes less than eight characters, zeros must be entered to bring the total number of characters to eight.

Examples:

USER z 4,0,0,0,0,0,0,0,

The Host EOL Sequence is now a Control-D, required by some systems as an End-Of-Text marker.

USER z 13,10,0,0,0,0,0,0,

This sets the Host EOL Sequence to a Carriage Return/Line Feed. This is the default condition.

DMA Access

This command requires the PIO/DMA card, an optional plug-in card available from Chromatics. The PIO/DMA card must have its base address set to $FF8400.

Format:

USER D <addr>, <words>, <0 or 1>

Where:

addr is the hexadecimal base address of memory to be affected. This value MUST be even.

words is the number of words to be transferred.

0 or 1 determines the direction of transfer:

0: DMA from host to 7900
1: DMA from 7900 to host

Examples:

USER D A00000, 32768, 1

Send 32768 words starting at address A00000 (hex) to the host from the 7900.

USER D 10000, 4700, 0

Send 4700 words from the host to address 10000 (hex) in the 7900.

Case Table

The 7900 is a highly flexible terminal for any host computer system, in addition to its stand-alone computing ability. Since several characters (Mode, Plot, Escape and User) are reserved by the 7900 software for control sequences, these characters would not normally be available for use by a host system. If the host system assigns a special meaning to one of these characters, a conflict could occur. The Case Table translates characters transmitted or received through a serial port into whatever characters the host system requires.

The Case Table is a list of 256 eight-bit values. Whenever a character is transmitted or received through EITHER of the serial ports on the CPU, it is first translated through the Case Table. A separate Case Table is not provided for each port.

The Case Table is loaded with the values 0 through 255, so that no translation occurs. For example, an ASCII 'A' indexes into location 65 in the table and finds the value 65. The value from the Case Table would be transmitted, which causes the Case Table to appear transparent.

NOTE:

> DEL (rubout) is not filtered by the Case Table by the Terminal Emulator in versions 1.2 and later. This was changed to allow easier use of Binary Mode in coordinate data (Binary Mode uses DEL characters). The default Case Table is now transparent to all characters.

To return to the arrangement provided in TERMEM 1.1, use the following commands:

USER v 127,0, (translate DEL to null)

USER v 255,0, (8-bit DEL likewise)

Swap Case Table Entry

The following command allows swapping any two characters in the Case Table:

USER w <N1>, <N2>,

Where <N1> and <N2> are decimal numbers between 0 and 255, and are the decimal equivalents of ASCII characters to be translated.

NOTE:

> This command requires entering a lower case letter, **w**. If the ALPHA LOCK key is in its normal (up) position, press the SHIFT key with the "W" key to produce a lower case **w**.

This command provides a way to resolve the "host conflict" problem described above. For example, if the host system used Control-A (the 7900 Mode character) as a system interrupt code, it would not be possible to use this character in programs. The command

USER w 1, 5,

would interchange CTRL A with CTRL E every time either of these characters went through the 7900 serial ports. (We are assuming here that CTRL E would be an unused character in the vocabulary of the host system.) This translation would allow the host system to use CTRL E as a Mode character in transmission to the 7900.

Set Case Table Entry

This routine sets a single entry in the Case Table instead of swapping two entries.

Format:

USER v <entry>,<value>,

Where <entry> is the number of the entry in the Case Table and <value> is the value to be placed in that entry.

Example:

USER v 13,10,

will change the Carriage Return character to a Linefeed (NewLine).

Chapter 5 -- The Create Buffer

The Create Buffer is an area of system memory used to store commands and characters. When the Create Buffer is active, any characters or commands typed will be executed normally. They will also be stored in the Create Buffer. After a set of instructions has been stored in the Create Buffer, it may be repeated using the REDRAW key. Commands also allow editing the contents of the Create Buffer.

If the Create Buffer fills up, characters are no longer stored in the Create Buffer. The system says "Bong," and the light in the CREATE key goes out. (In practice, the Create Buffer seldom fills up. The Create Buffer is normally allocated a larger portion of system memory than any other buffer in the 7900.)

Create Buffer processing occurs at the same point as Escape and User code processing. All input from Logical Input Device 0 is trapped for the Create Buffer, and if the system is in Half or Full duplex, all input from Logical Input Device 1 is also trapped. This results in all characters typed on the keyboard or received through the serial port being inserted into the Create Buffer when it is active.

Create Buffer ON

Format:

CREATE

Pressing the CREATE key causes that key to light up, indicating that the Create Buffer is ON. This action initializes the buffer, so if there was any information in it previously, it is now gone. All characters or commands typed at this point will enter the Create Buffer.

Create Buffer OFF

Format:

SHIFT CREATE

Holding down the SHIFT modifier and pressing CREATE will turn off the Create Buffer. Anything typed after this command will not be stored in the Create Buffer. The lamp in the CREATE key will go out. This command is also used to terminate the Append and Insert functions described on the following pages.

If Create is on, simply pressing CREATE will also turn off the Create Buffer.

Append to Create Buffer

Format:

USER A

The Create Buffer is turned back on, but is not initialized. Anything typed will be appended to the end of the Create Buffer. The previous contents of the Create Buffer are still intact.

NOTE:

This command should NOT be used if there is nothing in the Create Buffer (as would be the case after power-up). Append will not work properly if the Create Buffer has not been initialized using Create On.

The Append function is terminated using SHIFT CREATE.

Redraw the Create Buffer

Format:

REDRAW

The contents of the Create Buffer are transmitted to Logical Output Device 0. Under default conditions, this will cause the Create Buffer contents to be transmitted to the screen. Only RESET can interrupt a REDRAW. If the Create Buffer is not already off when REDRAW is executed, REDRAW turns it off.

Transmit the Create Buffer

Format:

XMIT

The contents of the Create Buffer are transmitted to Logical Output Device 1. Under default conditions, this is the RS-232 serial port.

Define a Sub-Buffer

In many cases, you will want to set up a very complex sequence of commands in the Create Buffer. It is useful to be able to define Sub-Buffers, which can be manipulated individually. A Sub-Buffer is established by inserting a CTRL X into the Create Buffer. Up to 32767 sub-buffers may exist.

Format:

CTRL X

(Hold down the CTRL modifier and type an X.) Sub-Buffers may be accessed individually with the following commands. (REDRAW and XMIT will always operate on the entire Create Buffer, regardless of any Sub-Buffers which may exist.)

NOTE:

CTRL X will be transmitted to a host system like a normal ASCII character. Make sure that this will not interfere with the host.

Insert into Sub-Buffer

Format:

USER ^ <n>,

Where:

<n> is a decimal number, telling which sub-buffer to insert into.

A search is performed for the start of Sub-Buffer <n>. At the completion of the search, the Create Buffer is turned on. Any characters or commands typed from that point on will be stored, beginning at the front of Sub-Buffer <n>.

If Sub-Buffer <n> is not found, the insertion takes place at the end of the Create Buffer. This is the same as the "Append to Sub-Buffer" function.

Examples:

USER ^ Ø,

Characters typed in will now be stored at the front of Sub-Buffer Ø, which means they will be at the front of the Create Buffer. Even if no Sub-Buffer markers are being used, this is a handy way to insert something at the beginning of the Create Buffer.

USER ^ 2,

Characters typed in will now be stored at the front of Sub-Buffer 2. When a REDRAW is performed, they will occur after the contents of Sub-Buffer 1, but before the present contents of Sub-Buffer 2.

NOTE:

If you are inserting near the front of the Create Buffer, and the Create Buffer contains a large amount of data, there may be a noticeable delay while the insertion is performed.

This command should not be used if there is nothing in the Create Buffer (as would be the case after power-up). See the warning under "Append to Create Buffer." The Insert function is terminated by SHIFT CREATE.

Kill a Sub-Buffer

Format:

USER K <n>,

Where:

<n> is a decimal number corresponding to the Sub-Buffer you wish to eliminate.

The Sub-Buffer numbered <n>, if it exists, will be eliminated. All Sub-Buffers with numbers greater than <n> will move down one number. All Sub-Buffers with numbers less than <n> will be untouched.

Examples:

USER K Ø,

Sub-Buffer Ø (the first one) will be killed. Sub-Buffer 1 will become Sub-Buffer Ø, 2 becomes 1, and so on. If no Sub-Buffer markers exist, the entire Create Buffer is killed. (This can also be accomplished by typing CREATE, followed by SHIFT CREATE.)

USER K 4,

Sub-Buffer 4 will be killed, if it exists. (If fewer than four Sub-Buffers exist, no action is taken.) Sub-Buffers Ø through 3 are unchanged. Sub-Buffer 5 will become 4, 6 will be 5, and so on.

View a Sub-Buffer

Format:

VIEW <n>,

Where:

<n> is a decimal number, corresponding to the Sub-Buffer you wish to view.

Sub-Buffers are numbered consecutively, beginning with the number Ø. To view any Sub-Buffer, hold down the SHIFT modifier and press the key marked REDRAW and VIEW. Then enter the number of the Sub-Buffer you want to see. The contents of that Sub-Buffer will be redrawn to Logical Output Device Ø (normally the screen).

Examples:

VIEW Ø,

The first Sub-Buffer will be redrawn. If no Sub-Buffer markers exist, this will perform the same function as REDRAW.

VIEW 3,

The fourth Sub-Buffer will be redrawn. If the system cannot find three Sub-Buffer markers, it will assume a fourth Sub-Buffer does not exist, and no action will occur.

Literal Create

Format:

USER [

Literal Create is used to allow inserting Escape code sequences into the Create Buffer. Escape code sequences are normally not allowed in the Create Buffer, since they can cause undesired changes in the system configuration. If you wish to insert Escape codes into the Create Buffer, you should go through the following steps:

1) Turn on Create with one of "Create On," "Append to Create Buffer" or "Insert Into Sub-Buffer"

2) Turn on Literal Create with USER [

Now, any Escape code sequences entered will be executed AND stored in the Create Buffer for future redrawing.

Literal Create remains in effect until the Create Off command is given. Normal codes will still go into the Create Buffer while Literal Create is turned on.

NOTE:

Many of the keys on the upper half of the keyboard produce Escape code sequences. It is not always obvious which ones will, or will not, be put into the Create Buffer. If Literal Create is NOT on, commands such as Zoom and Pan, or Change Color Lookup Table Entry, will be executed but will NOT be stored in the Create Buffer. Consult Appendix A to see which keys produce Escape code sequences.

Under no circumstances are User code sequences allowed in the Create Buffer.

Chapter 6 -- Thaw and Memory Allocation

Thaw Parameters

The THAW command allows you to configure various aspects of the 7900 system, such as buffer sizes and default baud rates. The information supplied to Thaw is stored in optional CMOS RAM (with battery backup while power is down), or in static RAM memory. If CMOS is installed, parameters set up by Thaw are remembered during power outages, and used the next time the system is powered up. If the CMOS option is not installed, information given to Thaw is only remembered until the system is shut off.

Format:

THAW

Thaw also displays some important system variables, which are not stored in CMOS, but may be of interest to the systems programmer. It is possible (but not recommended) to alter these variables as well; however, the values of items not in CMOS will not be remembered when the system is turned off or Booted.

Using Thaw, it is possible to allocate system RAM so that it serves your purposes efficiently. It is also possible to allocate more RAM than your system contains. In this case, the system will attempt to recover by reverting to PROM default values.

NOTE:

> To erase CMOS completely, and return to the defaults your system was shipped with, press M1 M2 CTRL SHIFT RESET. Release RESET before releasing the other keys.

Thaw takes you through a list of system parameters. For each parameter, it displays a brief description, followed by the currently assigned value for that parameter. It then displays a colon (:) to ask whether you want to change that parameter. You may hit RETURN to leave the value unchanged, or enter a new value, followed by RETURN. You MUST hit RETURN after entering a value, or the value you enter will not be accepted. If you press DELETE (instead of RETURN), the system will ignore your input even if you did enter a value.

All numerical values supplied to Thaw are in hexadecimal. DO NOT enter the dollar sign to indicate a hex number. Thaw displays an 8-bit, 16-bit, or 32-bit number, depending on the legal range of values for the entry.

After stepping through all entries, Thaw begins again from the start. Once you have changed all the entries you need to change, press RESET. (RESET is the ONLY way to exit from Thaw. After pressing RESET, you will note the green light on the keyboard turns on, then goes off.) If you had changed any of the "buffer size" parameters, you should also press BOOT. This will re-configure the system according to the new defaults you have entered.

CAUTION:

> If you alter any of the "buffer size" parameters using Thaw, and then exit Thaw WITHOUT executing Boot, the system may become confused and fail!

The following abbreviations are used by Thaw: ("Z" generally indicates a buffer size description.)

DOSTranZ	Size of DOS transient program area
DOSBuffZ	DOS buffer size
#Windows	Number of windows
KeyBuffZ	Keyboard buffer size
Fnk Nest	Function key stack size (for nesting)
232 InZ	RS-232 Input buffer size
232 OutZ	RS-232 Output buffer size
449 InZ	RS-449 Input buffer size
449 OutZ	RS-449 Output buffer size
Esc ArgZ	Escape code argument stack size
StackZ	System stack size
UpperRAM	System upper memory limit
FnkStart	Starting address of function key buffer
FnkEnd	Ending address of function key buffer
CaseTble	Case table address
DefltPrg	Default program to execute at Reset
RAM MDLE	Address of RAM modules to be linked
232 Mode	RS-232 USART mode command
449 Mode	RS-449 USART mode command
232HandS	RS-232 Handshake flags
449HandS	RS-449 Handshake flags
232 Baud	RS-232 Baud rate flag
449 Baud	RS-449 Baud rate flag
Planes	Image planes in system
WinTable	Base address of Window Tables

StackTop Top of system stack

StackBtm Bottom of system stack

CreaStrt Starting address of Create Buffer

CreatEnd Ending address of Create Buffer

Boot$ String to execute at Boot time

Reset$ String to execute at Reset time

IDnbufs Number of block buffers used by Idris to cache disk I/O

IDclists Number of lists used by Idris to buffer character I/O

IDchar Controls disk I/O indicator (at top right corner for Idris)

IDheap Stack+heap size used by Idris

IDbase Base address for Idris execution

IDrswap Space used by Idris swap cache

IDtop Top of Idris user memory

IDroodev Idris root device flags

IDpipdev Idris pipe device flags

IDswpdev Idris swap device flags

IDswapa Beginning block of Idris swapping area

IDnswap Size (in blocks) of Idris swapping area

IDcsw Idris "console switch register"

These parameters are discussed on the following pages.

DOSTranZ: default $4000 bytes (16K bytes)
DOSBuffZ: default $1A00 bytes

Range: 0 to $7FFE bytes (even numbers only)

The DOS transient area and buffer are allocated consecutively in memory. Generally, programs run in the transient area, and use the buffer area for any large data storage needs. These two areas may be combined and used as a single DOS area in some applications.

The size of the DOS buffer determines how much memory is available for disk operations such as COPY or COMPRESS. The larger this area is, the faster such operations will run. Making this area very small will result in excessive wear to the disk media during these operations.

#Windows: default 3

Range: 0 to 7

The system may have from one to eight windows active. The "#Windows" parameter contains a number ONE LESS than the number of active windows.

KeyBuffZ: default $40 bytes (64 bytes)

Range: 0 to $7FFE bytes (even numbers only)

This buffer provides "type-ahead" capability. Characters typed on the keyboard are stored here until read by a program.

Fnk Nest: default $80 bytes (128 bytes)

Range: $10 to $7FFE bytes (even numbers only)

This area is a stack, used to provide "nesting" when one Function Key is used to execute another Function Key. Each level of nesting requires 8 bytes. A minimum of $10 bytes (16 bytes) is recommended.

232 InZ: default $800 bytes (2K bytes)
232 OutZ: default $800 bytes (2K bytes)
449 InZ: default $800 bytes (2K bytes)
449 OutZ: default $800 bytes (2K bytes)

Range: $10 to $7FFE bytes (even numbers only)

Each buffer is used by the serial port routines to handle incoming and outgoing characters. Do not set buffer sizes below $10 bytes (16 bytes).

Esc ArgZ: default $400 bytes (1K bytes)

Range: $10 to $7FFE bytes (even numbers only)

Escape and User codes accept arguments, and these arguments are stored temporarily in this area. Do not set the buffer size below $10 bytes (16 bytes).

StackZ: default $800 bytes (2K bytes)

Range: $800 to $7FFE bytes (even numbers only)

The system stack is used for subroutine calls and for temporary storage in many system programs. It is used especially during the Complex Fill routines to store screen coordinate data. The system stack size should not be reduced below $800 bytes (2K bytes).

UpperRAM: default $1F000

Range: depends on system hardware

The system allocates RAM among all of the buffers listed in this section. Whatever RAM is left over, up to the limit set by UpperRAM, is allocated to the Create Buffer. In a standard 7900 system, only one Buffer Memory card is installed, and the physical end of Buffer Memory is at address $1FFFF. If you have purchased additional Buffer Memory cards, you may wish to move UpperRAM to a higher address so that the Create Buffer can use more memory.

The system defaults to $1F000 for UpperRAM so that the area between $1F000 and $1FFFF may be used for small user programs. UpperRAM may be moved down to provide more room for user programs.

See also RAM MDLE on page 6-8.

FnkStart: default $E40900
FnkEnd: default $E40BFF

Range: may be moved to Buffer Memory (see below)

Function Key definitions, including Bezel Key definitions, are normally stored in CMOS memory between addresses $E40900 and $E40BFF. This provides 768 bytes of storage. If your application requires more room for Function Key definitions, you may choose to move these addresses into Buffer Memory. For example, you could set FnkStart to $1F000, and FnkEnd to $1FFFF, to provide 4K of storage for Function Keys. Note that this removes Function Keys from CMOS, so their definitions would be lost when the system is turned off.

Memory addresses $E40C00 to $E40CFF are reserved for the Inline Editor (used in Thaw, DOS, and the Monitor). The Function Key buffer may not be expanded into this area.

CaseTble: default $E40800

Range: may be moved to Buffer Memory (see below)

This is the address of the Case Table (see "Serial Ports.") It may be moved to Buffer Memory if desired. However, the information contained in the Case Table would then be lost when system power is turned off. The Case Table is 256 bytes long.

DefltPrg: default $802008

Range: may be any address in PROM or RAM

DefltPrg contains the address to which the system will jump after Reset or Boot. The default address is that of the Terminal Emulator program. Other useful addresses are:

```
Monitor: $80A008
DOS:     $80C008
Idris:   $80E008
```

NOTE:

These addresses are valid for TERMEM 1.4, but may change in future firmware releases.

If this location is set to an address where a valid program does not exist, the system will crash.

RAM MDLE: default $1F000

Range: may be any RAM address

When the 7900 Boots, it "links" all system routines together and enters them into "dispatch tables" for processing via code sequences. User routines may be linked as well, if they conform to the requirements of the 7900. User routines to be linked must be loaded at address RAM MDLE. See Appendix E for details on adding commands to the 7900.

232 Mode: default $7A
449 Mode: default $7A

Range: any value from the table below

The RS-232 ports and RS-449 ports are initialized with a "Mode Select" byte, which determines the number of bits per word, number of stop bits, and parity. The Mode Select byte is fed to the 8251 USART on the CPU card. You can determine the proper Mode Select for your application from the following table:

Upper 4 Bits		Lower 4 Bits	
1 stop bit, no parity:	4	5 bits:	2
1 stop bit, odd parity:	5	6 bits:	6
1 stop bit, even parity:	7	7 bits:	A
		8 bits:	E
1.5 stop bits, no parity:	8		
1.5 stop bits, odd parity:	9		
1.5 stop bits, even parity:	B		
2 stop bits, no parity:	C		
2 stop bits, odd parity:	D		
2 stop bits, even parity:	F		

For example, the default Mode Select is $7A. This corresponds to 1 stop bit and even parity (7), and 7 bits (A).

Further details are available in Intel literature describing the 8251 USART.

232HandS: default $01
449HandS: default $01

Range: $00 to $03

These flags set the handshaking characteristics of the serial ports, according to the following table:

00: No handshaking

01: Software handshaking (X-on, X-off)

02: Hardware handshaking (DTR and DSR signals)

03: Both software and hardware handshaking

232 Baud: default $0E
449 Baud: default $0E

Range: $00 to $0F

These values set the default baud rates of the serial ports, according to the following table:

00: 50 baud	08: 1800
01: 75	09: 2000
02: 110	0A: 2400
03: 134.5	0B: 3600
04: 150	0C: 4800
05: 300	0D: 7200
06: 600	0E: 9600
07: 1200	0F: 19200

Planes: default $0001, $0087 or $00FF (typical systems)

NOT CMOS!

The number of Image Memory planes installed in your 7900 determines the value of this entry. This is a 16-bit number, and each bit which is SET corresponds to an existing plane. Bits which are ZERO correspond to planes which are not installed. A four-plane system, containing planes numbered 0, 1, 2 and 7, would display the value $0087 for Planes.

This item is displayed for information only. It should not be altered with Thaw. The value of Planes is determined by testing each Image Memory plane when the system is booted.

WinTable: default $763C

NOT CMOS!

WinTable is the base address of the Window Tables, the system memory areas used to define the attributes of each window.

This item is displayed for information only. It should not be altered using Thaw. WinTable will vary depending on buffer allocations.

StackTop: default $AAFC
StackBtm: default $A2FC

NOT CMOS!

The top and bottom of the area reserved for the system stack are displayed here. The system loads the Stack Pointer (SP) with StackTop at boot time.

These items are displayed for information only. They should not be altered using Thaw. StackTop and StackBtm will vary depending on buffer allocations.

CreaStrt: default $AAFC
CreatEnd: default $1EFF8

NOT CMOS!

The starting and ending addresses of the Create Buffer are displayed here. The Create Buffer is allocated most of the system's memory, after all other buffers have been allocated. The Create Buffer ends slightly below UpperRAM (a few bytes of margin for EOF are provided).

These items are displayed for information only. CreaStrt and CreatEnd will vary depending on other system parameters.

Boot $: default USER I 1 L USER O 1 L Z Z Z

Range: any 75 characters

The "Boot String" is executed at power-up and when Boot is executed. It acts just as if you had typed in these characters from the keyboard. The Boot String and Reset String (described below) provide a very flexible way to establish system defaults. Virtually any system function may be executed by default, simply by including it in the Boot String or Reset String.

The Boot String is stored in CMOS. The default Boot String, listed above, assigns Logical Input Device 1 to the serial port, and assigns Logical Output Device 1 to the Serial Port and three dummy devices.

User and Escape code sequences may be entered into the Boot String. The "User" character is abbreviated with a "UR," and the "Escape" character is abbreviated with an "EC."

It is not useful to enter alphabetic characters, or Mode and Plot codes, into the Boot String. This is because the window for Logical Output Device Ø has not yet been assigned: this occurs during execution of the Reset String (see below). The Boot String is primarily useful for executing commands such as "Clock On," "Half Duplex," or setting Logical Device Assignments as shown above.

All of the character-oriented text editing features labelled on the cursor keypad may be used to edit the Boot String. This includes Insert/Delete Character, Clear EOL, and the arrow keys. See Appendix C for a complete discussion of the Inline Editor.

Reset $: default USER I Ø K USER O Ø A K Z Z MODE J 1

Range: any 75 characters

The "Reset String" is executed after a Reset, and also after a Boot. (At Boot or power-up time, the Boot String executes first, followed by the Reset String). The Reset String is stored in CMOS.

The default Reset String, listed above, assigns the keyboard and window A as the default Logical Devices. It also turns on the cursor with a Mode code sequence. Mode and Plot characters may be included in this string, and they will be displayed in abbreviated form.

See the discussion of Boot String above for more details.

Thaw Parameters for Idris

IDnbufs This is the number of 512-byte block buffers used by Idris to cache disk I/O. Idris will run, albeit slowly, with only 2 buffers. An incredibly huge number will also slow Idris down due to the search overhead. By the mean value theorem, therefore, there must be happy medium. For normal applications, the default value of 15 (0x0f), consuming 8.5K, works well.

IDclists Clists are data structures used by Idris to buffer tty-type character I/O. Each clist can hold 12 characters, and the clists are shared among all tty-type devices. Fully burdened, a tty-type character device can consume up to 512 characters worth of clists. A safe position would be to provide 512 bytes worth of clists for each actively used tty-type device. As it turns out, the default value of 55 provides only 660 bytes worth of clists. This is adequate for one tty, but not so good for more than one. A better default might be 512*2/12 = 85 to support two fully loaded ttys.

IDchar This parameter controls the funny little character that winks on and off during disk I/O. You can completely eliminate it by setting IDchar to zero. The upper byte sets the foreground and background transparency attributes, while the lower byte is the actual ASCII character.

IDheap This is the stack+heap size used by Idris. The default is 5000 bytes which is a good guess.

IDbase This is the address to which the boot prom relocates Idris for execution. The default is 0x20000, which is the second 128K bank of memory. At the expense of the create buffer, you can move Idris downward and recover at least 64K of often unused memory. To do this, first set UpperRAM to 0xF000, then set IDbase to 0x10000.

IDrswap This is the amount of space allocated to the Idris swap cache. The swap cache is another one of those avoid-going-to-disk techniques which is extremely effective. A value of 64K is usually sufficient. If IDrswap is zero, Idris goes looking for a bank of memory following the first 128K gap. This implies changing around the address select switches on the memory boards in order to make them non-contiguous. We recommend using IDrswap instead.

IDtop If non-zero, IDtop places a maximum memory restriction upon Idris. This can be used if a section of upper RAM is to be devoted to some other purpose.

IDroodev
IDpipdev
IDswpdev These parameters alter Idris's idea of the device containing the root file system, the device on which to perform piping, and the device on which to perform swapping. They only become valid, however, if the 0x10000 bit is set in IDcsw (below). Otherwise, the compile-time values within Idris are used. The default Thaw values are coincidentally the same as the compiled-in values. The defaults are 0x0402 (major device 4, minor device 2, alias /dev/hard00) for the root device and the piping device, and 0x0406 (alias /dev/hard01) for swapping.

IDswapa
IDnswap These two values instruct Idris as to the starting block address and size (in blocks) of the swapping area on the swap device. IDswapa must not be zero, due to the swap space allocation scheme. As with the device parameters above, these parameters are not effective unless the 0x10000 bit is set in IDcsw.

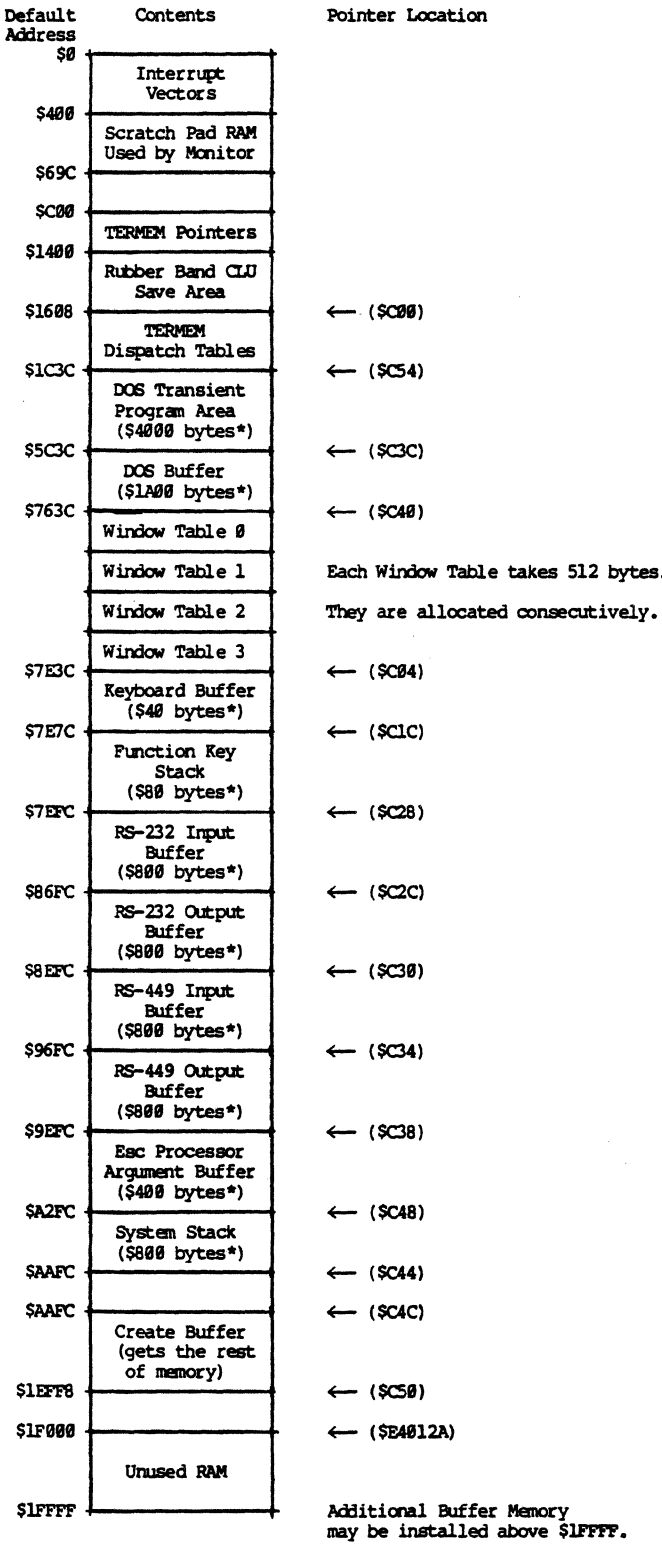
IDcsw This is the console switch register (what ever happened to the good ole days of switches and lights?). Programs can read its value using the getcsw function, which is erroneously documented as csw in the Idris Programmers Manual. The upper 16 bits are reserved for Chromatics use, while the lower 16 are all yours. The only bit currently dedicated is the 0x10000 bit as described above.

Default RAM Allocation

The following chart describes default RAM allocation in version 1.4 of 7900 system software. The sizes of buffers marked with asterisks (*) may be altered with Thaw, which may alter the starting addresses of other buffers.

To the right of the table are the addresses of pointers. These pointers contain long-word addresses which will always indicate where buffers start, regardless of Thaw. For example, $C28 always holds the address of the RS-232 Input Buffer.

All addresses between $0 and $1C3C are fixed in this version of firmware, and cannot be changed with Thaw.


Default Address
Contents
Pointer Location
$0
Interrupt Vectors
$400
Scratch Pad RAM Used by Monitor
$69C
$C00
TERMEM Pointers
$1400
Rubber Band CLU Save Area
$1608
← ($C00)
TERMEM Dispatch Tables
$1C3C
← ($C54)
DOS Transient Program Area ($4000 bytes*)
$5C3C
← ($C3C)
DOS Buffer ($1A00 bytes*)
$763C
← ($C40)
Window Table 0
Window Table 1
Each Window Table takes 512 bytes.
Window Table 2
They are allocated consecutively.
Window Table 3
$7E3C
← ($C04)
Keyboard Buffer ($40 bytes*)
$7E7C
← ($C1C)
Function Key Stack ($80 bytes*)
$7EFC
← ($C28)
RS-232 Input Buffer ($800 bytes*)
$86FC
← ($C2C)
RS-232 Output Buffer ($800 bytes*)
$8EFC
← ($C30)
RS-449 Input Buffer ($800 bytes*)
$96FC
← ($C34)
RS-449 Output Buffer ($800 bytes*)
$9EFC
← ($C38)
Esc Processor Argument Buffer ($400 bytes*)
$A2FC
← ($C48)
System Stack ($800 bytes*)
$AAFC
← ($C44)
$AAFC
← ($C4C)
Create Buffer (gets the rest of memory)
$1EFF8
← ($C50)
$1F000
← ($E4012A)
Unused RAM
$1FFFF
Additional Buffer Memory may be installed above $1FFFF.

Chapter 7 -- Numerical Data and Window Variables

Numerical Data

In an earlier chapter, we mentioned briefly that the 7900 expects decimal numbers to be delimited by a comma or semicolon. This chapter discusses the various forms in which numerical data may be accepted by the 7900. Some advanced features of the 7900 are mentioned in this chapter, but are not described in detail until later chapters.

Numerical data is used to enter coordinates for plotting, for window limit definition, for color numbers, and to set up many other system features. The numerical data will usually have a value between -32768 and +32767, or between 0 and +65535. Numerical data may be accepted in one of several forms, depending on the current state of the system:

1) A decimal number. Decimal numbers are entered as you would normally type a number on the keyboard. A decimal number ALWAYS ends with a comma or semicolon. Failure to provide the comma or semicolon will usually cause the system to continue accepting characters until a comma or a semicolon IS found. This will almost certainly enter erroneous data into the system.

   A decimal number may have a leading + or - sign. If, while entering a decimal number, you discover that you have entered a wrong digit (BEFORE striking the terminating comma), you may press the + or - key to cancel that entry and then re-enter the correct data. Pressing the + or - key resets the "argument collector" so that it begins parsing that entry again.

   Examples of decimal numbers:

```
17,    (Each of these numbers
+17,    evaluates to seventeen.)
13+17,
```

2) If Binary mode is on, each number must be entered in a special binary form. Binary mode is discussed later in this chapter.

3) If Binary mode is NOT active, the system will accept a Window Variable anywhere a decimal number is allowed. Window Variables are discussed next.

Window Variables

Each window has a number of variables which may be used to store temporary data. These variables are referenced by alphabetic names, **A** through **z**, and the names of variables may be used anywhere a decimal number may be used in any system command. Each variable is allocated 16 bits (1 word) of storage, so a variable can store numbers from -32768 to 32767.

The Window Variables named **A** through **W**, and **[**, ****, **]**, **^**, **_**, **`**, are completely available for the user. Variables **X**, **Y** and **Z** are also available if the optional Digitizer Pad is not being used. Variables named **a** through **z** provide access to certain items in the "window table" (the set of data defining the current status of the window). These variables should be modified with caution.

The values of most Window Variables are lost when the "Soft Boot" command is executed. Only the current cursor positions are preserved.

Since the names of Window Variables are alphabetic characters, and entering a Window Variable name is equivalent to entering a decimal number, you are cautioned against entering text while the system is in Plot mode. The text would be interpreted as Window Variable names, causing unpredictable numbers to be plotted.

Table 7-1. Window Variable Assignments

Variable	Usage
A	User-defined
B	"
.	.
.	.
W	"
X	Used by optional Digitizer Pad
Y	Used by optional Digitizer Pad
Z	Used by optional Digitizer Pad
[	User-defined
\	"
]	"
^	"
_	"
`	"
a	Overlay cursor X position
b	Overlay cursor Y position
c	Bitmap cursor X position
d	Bitmap cursor Y position
e	Overlay window X1 position
f	Overlay window Y1 position
g	Overlay window X2 position
h	Overlay window Y2 position
i	Bitmap window X1 position
j	Bitmap window Y1 position
k	Bitmap window X2 position
l	Bitmap window Y2 position
m	Bitmap character raster size (X)
n	Bitmap character raster size (Y)
o	Bitmap intercharacter spacing (X)
p	Bitmap intercharacter spacing (Y)
q	Bitmap character X multiplier
r	Bitmap character Y multiplier
s	Virtual coord X minimum
t	Virtual coord Y minimum
u	Tab stop spacing
v	Vector width
w	Background color
x	Foreground color
y	Plane select value
z	(Not used, reserved)

Operate on Window Variable

Format:

MODE] <var> <op> <N>

Where:

<var> is one of the window variables, A through z

<op> is an arithmetic, logical, or assignment operator in the set = + - * / & !

<N> is a decimal number followed by a comma, or another Window Variable

This command assigns a value to a Window Variable, or modifies it in some way.

The = operator assigns the value <N> to the variable <var>. The arithmetic operators +, -, * and / perform addition, subtraction, multiplication and division. The logical operators & and ! perform a 16-bit logical AND and OR, respectively.

Examples:

MODE] A = 1, set variable A equal to 1

MODE] A + 1, increment the variable A

MODE] A + B let A = A + B

MODE] A & 7, let A = A AND 7

NOTE:

Do NOT put spaces in any of these command sequences. Spaces are shown here for readability ONLY.

Display and Transmit Window Variables

Commands are provided to display the value of a Window Variable or transmit this value to a host system. They operate identically except that the Display command sends the value to Logical Output Device Ø (normally a window), and the Transmit command sends the value to Logical Output Device 1 (normally the RS-232 serial port).

Each command also sends the "Host EOL sequence" after sending the variable value. The default EOL sequence is a carriage return and line feed.

These commands always send values as decimal, -32768 to +32767.

Format:

USER x <win> <var> (Transmit variable)

USER y <win> <var> (Display variable)

Where:

<win> is the desired window, **A** through **H**

<var> is the name of a Window Variable, **A** through **z**

NOTE:

Each of these commands requires entering a lower case character, **x** or **y**. If the ALPHA LOCK key is in its normal (up) position, hold down the SHIFT key and press the "X" or "Y" key to produce a lower case **x** or **y**.

These commands point out that the value of a Window Variable is specific to a window. You must specify which window you are interested in, when requesting the value of a Window Variable. Windows are named **A** through **H**. Often window **A** (the "Master Window") is the only window in use; however, its name must still be specified.

Example:

USER y A v display the currently specified vector width for window A.

Example of variable operations and display:

DEFINE F1	Define Function Key F1 to be:
USER y A A	display variable A (window A),
MODE] A + 1.	and increment variable A.
F1	End of F1 definition.

Now, pressing F1 repeatedly displays the integers, 1. 2. 3. and so on.

Window Variable Uses

Window Variables give the user direct access to many of the attributes which define a window. They are especially useful to alter some window attributes which are not otherwise accessible.

Example:

MODE] u = 8,

This example sets the tab stops in a window to 8, producing a stop every eight columns. There is no Mode code sequence assigned to perform this function, so the Window Variable operator is the only means available for setting tab stops.

Example:

SHIFT OVERLAY

RECT

i j k l

The four Window Variables i, j, k and l define the coordinates of a window in the Bitmap. This example draws a rectangle with those coordinates, producing a rectangle surrounding the current window.

Binary Mode

Binary mode is provided so that a host system, or user program, can transmit coordinate data more efficiently. When Binary mode is on, all numbers which are normally entered in decimal must be entered in a special binary fashion. This includes all arguments to MODE and PLOT codes, except for flags (<0 or 1> type arguments).

Format:

MODE B <0 or 1>

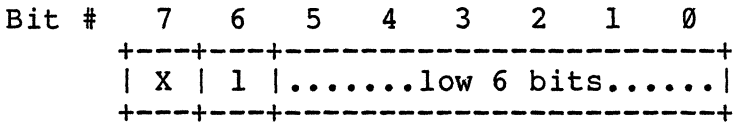
Using the character 0 will turn off Binary mode (the default state), the character 1 will turn it on.

NOTE:

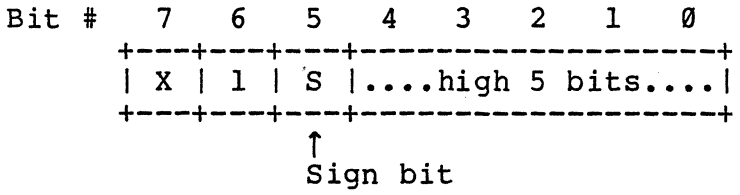
> Binary mode is NOT recommended for use from the keyboard! In Binary mode, color keys and many other keys on the upper half of the keyboard will not operate normally.

In Binary mode, each number entering the system must be encoded into two 8-bit bytes. The bits are arranged as follows:

First Byte:



Second Byte:



In each byte, bit 7 is ignored and bit 6 must be a 1. This insures that only printing ASCII characters will be used. Bits 5 through 0 of the first byte make up the low-order 6 bits of the resulting value, and bits 5 through 0 of the second byte make up the high 6 bits. Binary mode thus limits arguments to being signed 11-bit numbers.

If bit 5 of the second byte is a 1, the number will be negative.

Example:

```
MODE B 1            (Turn on Binary mode)

MOVE X-Y PC PC      (Move cursor to location 200.200 --
                     Binary mode value "PC")
```

Binary Mode (Escape Code Processor)

Binary Mode can be confusing since it can be ON or OFF for each window. The Escape Code Processor can also work in Binary Mode; it operates on all ESC and USER codes without regard to windows.

Format:

```
ESC B <0 or 1>
```

Usually, you will want to set Binary Mode for the Escape Code Processor AND the windows you will work in.

Chapter 8 -- The Bitmap

The 7900 Bitmap screen is designed for the creation of very high resolution images. It is arranged as a square set of square pixels, 1024 horizontally by 1024 vertically. Because of the dimensions of the display CRT, an area of 1024 horizontally by 768 vertically is the maximum viewable at any time (see Figure 8-1 on the next page). The remaining pixels are normally held offscreen below the viewable CRT area.

The Bitmap can display images in up to 256 colors, depending on the hardware installed in your unit. The basic model 7900 contains enough Bitmap memory to allow simultaneous display of 16 colors. As each additional Bitmap memory plane is added, the number of displayable colors doubles.

Regardless of the number of Bitmap memory planes in your system, the 7900 always has a "palette" of 16 million colors (actually 16,777,216). Your selection of colors may always be chosen from the full palette.

A second set of Bitmap planes can be added to the 7900. This second set does not increase the number of available colors, but instead allows you to create two independent images. This second image memory must contain the same number of Bitmap planes as the first image, allowing the two separate images to each display the same colors. A maximum of sixteen Bitmap memory planes (eight in each of two image memories) may be installed.

The term "Bitmap" refers to the way in which pixels are addressed in the 7900 hardware memory circuits. In a conventional memory arrangement, the units or "bits" of memory are not individually addressable. It is not possible to alter a single bit of memory without reading a group, or "word," of such bits, altering one of them, and writing the entire word back into memory. Since each bit of memory is associated with a single pixel, it is much faster to write to a single pixel without disturbing its neighbors.

The 7900 does just that. The Bitmap hardware and software can write or read the value of any individual pixel on the Bitmap screen. If only one Bitmap memory plane was installed, it would be necessary to read only a single bit for each pixel. If several planes are installed, the Bitmap operates on one bit OF EACH PLANE every time a pixel is altered.

If your system contains four Bitmap memory planes, four bits are responsible for the color of each pixel. These four bits are added in a binary fashion to produce a number. The number refers to an entry in the Color Lookup Table, and this entry determines the color of that pixel. (The Color Lookup Table is explained later in this section.)

Bitmap Defaults

When the 7900 is powered up, or booted with the BOOT key, the Bitmap is initialized as follows:

- Alpha (not Plot)
- Page (not Roll)
- Cursor at upper left corner of screen (0,0)
- Color Lookup Table loaded with default values (see page 8-15)
- Bitmap image is NOT erased (except at power-up)

Remember that when the system is booted, the Overlay is not transparent. You cannot see the Bitmap image after a Boot until you press SHIFT OVERLAY (see "Bitmap Operations") or until you make the Overlay transparent (see "Modify Overlay Visible Attributes").

The Bitmap image is not erased when the system is booted. If an error occurs while you are creating an image on the Bitmap screen, in most cases you will be able to recover without losing the image. (The Color Lookup Table may have to be reloaded if you have altered it.)

After a Boot, a new cursor is installed at the "home" location of the screen. If a cursor had been on the screen when Boot was executed, the old cursor will still be at its old location after the Boot and you will have to remove it.

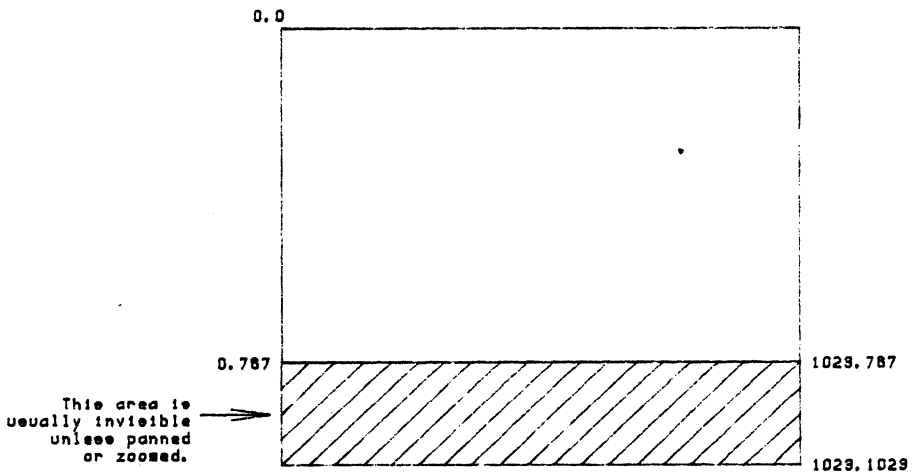


Figure 8-1. Screen Layout (Bitmap)

Bitmap Operations

This section discusses the types of commands accepted by the Bitmap. Before you can talk to the Bitmap, you have to get its attention. This is accomplished by the important command:

SHIFT OVERLAY

Hold the SHIFT key and press the lighted key marked OVERLAY. The light will go out. The Overlay window will become immediately transparent. foreground and background, to allow you to see the Bitmap. Any commands or characters entered will now be processed (or, if illegal, ignored) by the Bitmap.

When you want to talk to the Overlay again, press

OVERLAY

The light in the key will light up, and you are again addressing the Overlay. This action causes the Overlay window to become non-transparent, so that any messages in the Overlay will now be visible.

NOTE:

> When switching back and forth between Overlay and Bitmap, be aware of the light in the PLOT key. If you are plotting in the Bitmap and then return to the Overlay to enter text, you will still be in Plot (as indicated by the lighted PLOT key). To return to Alpha for entering text, press SHIFT PLOT. See "Exiting Plot Submodes" on page 9-28 for details.

If you want to address the Overlay or the Bitmap, without disturbing the Overlay transparency attributes, use the following commands:

MODE O Ø (Overlay "off" - addresses Bitmap)

MODE O 1 (Overlay "on" - addresses Overlay)

These commands will not affect the present attributes of the Overlay, but they will determine whether the Overlay or the Bitmap will process future commands.

As an illustration, remember that the 7900 uses "windows" as its primary output devices. A window can display characters, perform graphics functions, and handle other actions. Each window is allowed to perform functions in the Overlay and in the Bitmap. For example, when a window receives a character, it may place this character into an Overlay character cell, or it may generate a series of Bitmap pixels to form the character.

The command MODE O <Ø or 1> determines which part of the window software will handle an incoming character, and whether the resulting display will occur in the Overlay or the Bitmap. Then, the Overlay transparency attributes will determine whether this character will be visible or not.

The OVERLAY key produces the following code:

```
MODE O 1

MODE V 3
```

SHIFT OVERLAY produces the following code:

```
MODE O Ø

MODE V Ø
```

NOTE:

Each example in this section is independent. Every example assumes that it will not be affected by any prior example (unless noted). To insure that each example executes independently, it is advisable to execute the command sequence:

```
RESET BOOT SHIFT OVERLAY
```

before beginning each example. In addition, to insure a "clean" screen for each example, you may wish to erase the screen.

Bitmap Blink

In the Bitmap, the Blink attribute occupies one Refresh Plane of image memory. The highest numbered plane is normally used for blink, so the "blink plane" will normally be plane 7. (Regardless of how many planes your system has, one of the planes may be configured to be number 7.) If a bit is ON in the "blink plane," it means the pixel corresponding to that bit will blink. If the bit is OFF, the pixel will not blink.

Refer to the Color Lookup Table (page 8-15). The upper half of the Table, beginning at entry number 128, will be accessed by a pixel whose seventh bit is ON. If bit 7 is OFF, the lower half of the Table is accessed.

Now, if we allow that seventh bit to blink on and off, we are alternating between the upper and lower halves of the Color Lookup Table. The Table is default-loaded with full intensity color in the first eight entries, and half-intensity of the same colors in the first eight entries of the upper half. So any pixel which has the blink bit set will blink between full- and half-intensity color. (Black is the exception: since half-intensity black would be black, the Table defaults to grey in its place. White and black will both blink to half-intensity white, which is grey.)

Since blink occupies one Refresh Plane, it cuts the number of available colors in half. In a four-plane system, you have a choice:

1) eight colors and blink

2) sixteen colors without blink

Actually, you always have 16 colors even with blink: the second eight colors are only seen if the color is blinking. For example, under default conditions, an object might be specified to be "blinking red." It would be blinking between full-intensity and half-intensity red, entries 4 and 132 in the Color Lookup Table. If we alter entry 132 to be brown, then the "blinking red" object will blink between red and brown. This technique allows any object on the screen to blink between any two colors.

Blink ON

Format:

BLINK

The BLINK key will light up. Future SET commands will reference the upper half of the Color Lookup Table, and will cause foreground or background color to blink.

If you want the foreground color of characters or figures to blink, turning on the blink attribute is sufficient. Any figures drawn while the BLINK key is lit will have a blinking foreground.

If you want the background to blink, the procedure is a bit more involved. First, turn on blink. Set the background color using SHIFT SET <color>, and turn off blink. The background, not the foreground, of a character entered into the Bitmap will now blink. (This procedure is only relevant to characters, since graphics figures do not use background color.)

NOTE:

When Blink is ON, the 7th bit of any color number entered will be set. Color numbers will not be modified when Blink is OFF.

Blink Off

Format:

SHIFT BLINK

The BLINK key will go out. Future SET commands will reference the lower half of the Color Lookup Table, and will not cause blink.

NOTE:

Use of the BLINK key requires that the default blink plane be in effect (plane 7). If the "Select Blink Plane" command has been used to alter the blink plane number, the BLINK key may not produce normal results.

Select Blink Plane(s)

Format:

ESC b <n>,

Where <n> is a decimal number, Ø to 255. The plane or planes specified by bits in <n> will blink. Other planes will not.

Blink means to alternate between "data" and "Ø" for that plane's contribution to the CLU indexing.

NOTE:

This command requires entry of a lower case letter, **b**. When the ALPHA LOCK key is in its normal (up) position, use the SHIFT modifier with the "B" key to produce a lower case b.

Examples:

ESC b Ø,

No bits are set "on" in the number Ø, so no planes will blink. This is necessary for applications requiring all available colors (16 in a 4-plane system). Blink in the Bitmap is now completely inhibited.

ESC b 128,

The number 128 is examined as a binary number. Only the highest bit (bit 7) is found to be "on," so only plane 7 will blink. This is the default condition.

ESC b 255.

The number 255 has all eight bits set. All planes will blink. This is not a terribly useful condition.

Set Bitmap Character Size

Format:

SIZE <X>,<Y>,

Where:

<X> is the character multiplication factor in the horizontal direction

<Y> is the vertical factor

The limit for <X> is 170, and the limit for <Y> is 96. This size (170 by 96) corresponds to a size where one character fills the entire Bitmap screen.

Example:

SIZE 1,1,

This sets the smallest available character size on the 7900. You can now fit 170 characters on a line, and 96 lines on the CRT. With this character size you have the ability to put 16320 characters on the Bitmap screen.

SIZE 2,2,

The Bitmap character size is now exactly equal to the size of Overlay characters. This is the default condition.

SIZE 10.30.

Note that the <X> and <Y> factors need not be equal.

Set Bitmap Intercharacter Spacing

Format:

MODE I <X>, <Y>,

Where:

<X> is the pixel spacing in the horizontal direction.

<Y> is the vertical spacing.

Each time a character is entered into the Bitmap, the cursor is automatically updated to prepare for a new character. When a line is complete, or when the LINE FEED key is pressed, the cursor is moved down to prepare for the next line.

The spacing between characters on a line, and between lines, is set by this command. The default values are 6 for <X>, and 8 for <Y>, since the standard character font is 6 by 8 pixels.

The spacings entered for <X> and <Y> are always multiplied by the current character size before use. This allows proper spacing regardless of the current Bitmap character size.

Example:

MODE I 6, 8, This is default spacing.

For bold characters, use the following command:

MODE I 8, 12.

NOTE:

This command only affects the cursor movement and has no bearing on the size of the background field of a character cell.

Load User-defined Character Set

Format:

MODE i <x>, <y>, <addr>,

Where:

<x> is the horizontal character raster size

<y> is the vertical character raster size

<addr> is the memory address of the character set

<x> and <y> are decimal numbers. <x> is a positive integer from 1 to 16, and <y> ranges from 1 to 256.

<addr> is a hexadecimal number. It must be a legal memory address, or the system will Trap. <addr> must also be an even number. If <addr> is zero, the normal character set is loaded.

Example:

MODE i 8, 10, 10000,

This command would install a user-defined character set (8 by 10 pixel font) which has been loaded into RAM at address 10000 (hex).

To return to the normal character set, use SOFT BOOT or enter the command:

MODE i 6, 8, 0,

This loads the normal character set and sets the raster sizes to normal (6 by 8).

There should be <y> words of memory for each of 128 character definitions. Characters should be in ASCII format. See Appendix E for an example of the character set format.

Overstrike

Format:

MODE \ <Ø or 1>

Using the character 1 will turn Overstrike on, and Ø will turn it off.

NOTE:

> This command requires entering a backwards slash, the character \. The backslash is located in the upper right corner of the typewriter area of the keyboard, near the RETURN key. It should not be confused with the standard slash, /, which is located on the question-mark key.

With Overstrike off (the default condition), writing a character to the Bitmap causes a rectangular block of pixels to be written. Some of these pixels will form the character, and will be written in the current foreground color. The other pixels in the rectangular block are written in the current background color.

When Overstrike is on, only the pixels which form the foreground of the character are written. The background is not altered. This allows multiple characters to be written to the same screen location without having each character obliterate the previous one. Overstrike can be used to create custom characters or symbols, or for underlines, accent marks, or other punctuation.

Example:

MODE \ 1

RETURN T E S T RETURN _ _ _ _

Press the RETURN key to move the cursor to the left edge of the window, if it is not already there. Then type the word "TEST." Press RETURN again, and while holding the SHIFT key down, press the underline key four times. (The underline key is on the right side of the typewriter area of the keyboard, and is marked with a horizontal line and the name "DEL.") In this example, the word "TEST" was underlined without disturbing the characters in the word. Now type the sequence MODE \ Ø (Overstrike off) and try the same example again.

Bitmap Color

In the Bitmap, pressing a color key generates a number. This number is a reference to a value in the Color Lookup Table. The BLACK key, for example, generates the number 0, which refers to entry 0 in the Lookup Table. Normally, entry 0 is loaded with components of zero red, zero green, and zero blue, which gives black.

Set Foreground Color

Format:

SET <color>

Where:

<color> may be one of the eight color keys, or a number, 0 to 255.

The current foreground color will be set to the specified color number in the Color Lookup Table. Any figures or text which are entered into the Bitmap will have this foreground color.

If a color key is used, and the BLINK attribute is off, then the number generated by the color key will be in the range 0 - 7. BLACK generates 0, BLUE generates 1, and so on. If the BLINK attribute is currently on, then the color numbers generated will be in the upper half of the Lookup Table, and will be in the range 128 to 135. This arrangement assumes that plane 7 is the Blink Plane, and allows you to specify blinking colors without concern for plane numbers.

Example:

SET RED

The foreground color has been set to red, which is color number 4 in the Lookup Table. If BLINK is turned on, the foreground color would be set to blinking red, which is color number 132.

Example:

SET 4,

The foreground color has been set to color number 4 in the Lookup Table. This approach bypasses the question of whether blink is on or off. This format would most often be used from a host computer.

Set Background Color

Format:

SHIFT SET <color>

Where:

<color> is a color key, or a number between 0 and 255.

All of the comments concerning foreground color apply to background color as well. The background color will be set to the specified number in the Color Lookup Table. If BLINK is on, the upper half of the Table will be referenced by the color keys.

Color Lookup Table

The 7900 Bitmap has a color palette of 16,777,216 colors. At any time, a subset of these colors is available. The number of available colors is determined by the number of Refresh Planes (Bitmap memory boards) installed in your system.

In the standard system, with four Refresh Planes, sixteen colors are available at any time. Each additional board doubles the number of available colors. In a fully expanded system with eight Refresh Planes, 256 colors are available at any time.

With sixteen million colors to choose from, which ones do you use?

Color selection in Bitmap memory is performed through an indirect technique called the Color Lookup Table. Each pixel on the Bitmap screen is associated with a bit in each plane of Bitmap memory. The sum of these bits is a number, from 0 to 255, which references a particular entry in the Color Lookup Table. Thus, each pixel "points to" some entry in the Color Lookup Table, and the information in that entry will determine the color of that pixel. The Color Lookup Table has 256 entries, enough to accommodate a configuration having eight planes (per image memory). The colors are numbered from 0 to 255.

Each entry in the Color Lookup table contains three 8-bit numbers, one for R (the Red component of the color), one for G (the Green component), and one for B (the Blue component). The intensity of each of these components may be altered from 0 (totally off) to 255 (full intensity in the color). For example, pure red is 255 R, zero G, and zero B. Black is zero, zero, zero. White is 255, 255, 255. Colors with less than full saturation (such as greys) will contain intermediate numbers between 0 and 255. Neutral grey is 128, 128, 128 (half intensity of each component).

The following chart shows default assignments in the Color Lookup Table.

Color #	R	G	B
255			
254			
.			
.			
.			
.			
135	128	128	128
134	128	128	Ø
133	128	Ø	128
132	128	Ø	Ø
131	Ø	128	128
130	Ø	128	Ø
129	Ø	Ø	128
128	128	128	128
.			
.			
.			
.			
7	255	255	255
6	255	255	Ø
5	255	Ø	255
4	255	Ø	Ø
3	Ø	255	255
2	Ø	255	Ø
1	Ø	Ø	255
Ø	Ø	Ø	Ø

The lowest eight entries in the table (0 through 7) correspond to the eight color keys on the keyboard: 0 is Black, 1 is Blue, etc. Halfway up the chart, beginning at entry 128, another eight entries are defined to be half-intensity of the same eight colors. Only sixteen colors are assigned in the default Color Lookup Table because only sixteen colors are available in the basic 7900 system.

Pixels on the Bitmap screen do not have a color, but they have a color value which points to some location in the Color Lookup Table. The Color Lookup Table is the mechanism by which each pixel is given a color.

Modifying the Color Lookup Table

Change Color Lookup Table Entry (RGB Units)

CHANGE is the most immediate way to change colors on the Bitmap screen. No actual change to the contents of Bitmap memory takes place. Each pixel in the Bitmap references some entry in the Color Lookup Table, so changing an entry in the Table will immediately change the color of each pixel referencing that entry.

Format:

CHANGE <color>,<R>,<G>,<B>.

Where:

<color> is an entry in the Color Lookup Table, 0 to 255. or a color key. (If a color key is used, the comma after <color> must not be entered.)

<R>,<G>,<B> are components of Red, Green, and Blue, respectively, to be entered into the Color Lookup Table. Each must be between 0 and 255.

The entry in the Color Lookup Table corresponding to <color> will be changed to have the components <R>, <G>, and <B>. The former components of that entry of the Lookup Table are gone.

Example:

CHANGE 0,255,255,255.

Color 0 (the first entry in the Color Lookup Table) has been changed to white. Since the background color defaults to color 0, pressing ERASE PAGE will now erase the screen to white. (This is not the normal way to change background color: see page 8-13.)

After this change is made, both the BLACK key (which normally references color 0) and the WHITE key (which normally references color 7) will cause white to be put on the screen. This is because our example above has eliminated black from the Color Lookup Table. Unless other changes are made, there is no entry of the CLU which will produce the color black.

This points out an important consideration when using the Change command: since the Change command alters entries in the Color Lookup Table, it is possible to create a situation where the color keys have no meaning. The example above will change the BLACK key to mean white. If you change entries in the Color Lookup Table, it is your responsibility to keep track of those entries and what the color keys now mean.

NOTE:

> The Overlay colors are unaffected by the CLU. Thus, the color keys will always have their proper meanings in the Overlay.

The relationship between the color keys and the Color Lookup Table is now apparent. In the Bitmap, the color keys reference numbers in the Color Lookup Table. Under default conditions, these numbers are set up to produce the colors designated on the color key. For example, pressing BLUE generates the number 1. Entry number 1 in the Color Lookup Table is default-loaded with zero red, zero green, and 255 blue, which produces full intensity blue. If, however, you change entry number 1 in the Lookup Table to be some color other than blue, the BLUE key will still reference that color. Pressing a color key is equivalent to entering a number from 0 to 7; the shifted versions produce numbers from 8 to 15. These numbers are followed by a comma.

Example:

CHANGE 0,0,0,0,

Color number 0 in the Color Lookup Table has been changed to black. This is the default condition.

If you try to change an entry in the Color Lookup Table which is not accessible in your system, no visible changes will take place on the screen.

If you change an entry which is accessible in your system, but which is not currently referenced by any pixel on the screen, you will not be able to detect a change until you use that color again.

Example:

SET WHITE	(set foreground color to white - color 7)
SHIFT SET BLACK	(set background to black - color 0)
ERASE PAGE	(clear the screen to background color)
CHANGE 7.255.0.0.	(change entry 7 - which used to be white - to red)

The screen is still black, since entry 7 is not used anywhere on the screen at the present time. But anything typed will now appear in red, even though the current foreground color is number 7, which should be white.

The primary use for the Change command is to produce new colors, which are not normally present in the Color Lookup Table. In most cases, minor changes in color will be desired which will not grossly affect the meaning of the color keys. Here are a few possibilities:

CHANGE BLUE 0.140.240.

The BLUE key now produces a "sky blue" composed of no red, 140 green, and 240 blue.

CHANGE YELLOW 255.170.120.

The YELLOW key now produces a shade of brown.

As mentioned, when entering a color NUMBER to be changed (as in the earlier examples), the number was terminated by a comma. When using CHANGE with a color key, no comma is entered after the color key. Color keys simply generate the digits 0-7 followed by a comma. Shifted color keys generate the digits 8-15 followed by a comma.

In each case, the Color Lookup Table is changed. This changes the color of every pixel in Bitmap memory which references that color. If no images had been drawn in the color, no change will be apparent.

Change Color Lookup Table Entry (HVS Units)

Format:

SHIFT CHANGE <color>, <hue>,<value>,<saturation>,

Where:

<color>	is an entry in the Color Lookup Table, 0 to 255, or a color key. (If a color key is used, the comma after <color> must not be entered.)
<hue>	are the HVS units which describe
<value>	the desired color (see below).
<saturation>	Each must be between 0 and 255.

The entry in the Color Lookup Table corresponding to <color> will be changed. The parameters <hue>, <value>, and <saturation> will be converted to RGB units and placed in the Color Lookup Table.

The HVS color model is described in Appendix B. Except for the HVS-to-RGB conversion, this command acts exactly like the Change command on the previous page.

Examples:

SHIFT CHANGE WHITE 0,128,0,

The WHITE key now produces 50% grey (hue is undefined, value is 128 or 50%, saturation is zero).

SHIFT CHANGE BLUE 171,255,128,

The BLUE key now produces a pastel blue (hue is 171, blue; value is 255, full brightness; saturation is 128, or 50%, which adds some white to the color).

Default Color Lookup Table

Format:

ESC 1 <c>

Where:

<c> is a single character.

NOTE:

This command requires entering a lower case letter, l. If the ALPHA LOCK key is in its normal (up) position, hold down the SHIFT key and press the "L" key to produce a lower case l.

This command will load the Color Lookup Table, based on a default arrangement of colors. If <c> is the character "Ø," the Color Lookup Table is loaded with the values it had when the system was Booted. Pressing the BOOT key also reloads the Color Lookup Table.

Example:

ESC 1 Ø

Other default arrangements of the Color Lookup Table, for imaging and other applications, are planned for future expansion.

Colorswap

Format:

COLORSWAP <color 1>, <color 2>,

Where:

<color 1> and <color 2> are each a color key or a decimal number (terminated by a comma).

Colorswap exchanges pixel colors in the Bitmap. Any pixels which were <color 1> are changed to <color 2>, and any pixels which were <color 2> are changed to <color 1>. Other colors are not altered.

Colorswap affects all pixels in the window receiving the command.

Contrast this command with the CHANGE command: CHANGE alters data in the Color Lookup Table to produce a color change, but does not alter Bitmap pixel data. Colorswap alters only pixel data. This takes considerably longer if the window is large.

Colorswap is useful for altering Bitmap data to suit the requirements of a hardcopy device, or other peripheral, which may require Bitmap data to be configured in a certain way.

Example:

COLORSWAP RED BLACK

Any pixels which were red are changed to black, and vice versa.

Colorset

Format:

SHIFT COLORSWAP <color 1>, <color 2>,

This command acts exactly like Colorswap, except that pixel data is altered from <color 1> to <color 2> only. Pixels which are <color 2> are NOT changed to <color 1>.

Bitmap Cursor Control

The Bitmap has two cursors: an overscore/underscore for Alphanumeric use (similar to the Overlay cursor), and a cross hair for plotting. Only one of the cursors is present in a window at any time. If more than one window is in use, more than one cursor may be visible. Unless otherwise noted, the following information applies to both Alpha and Plot cursors.

The cursor control keys:

RETURN	LF	HOME
CTRL HOME	ERASE PAGE	CLEAR LINE
RECALL	ERASE EOS	CLEAR EOL

the four arrow keys

have the same meanings they had in the Overlay. Pressing two adjacent arrow keys simultaneously will move the cursor diagonally. Since the Bitmap allows variable size characters (see "Set Bitmap Character Size"), some functions may depend on the currently specified character size. For example, the right arrow will always move the cursor one character width to the right, regardless of how wide the characters are currently defined to be.

WARNING:

> ERASE PAGE will clear out the image in the current window in Bitmap. There is NO WAY to recover this image unless it can be reconstructed from the Create Buffer or some other source.

The four text editing functions are also available in the Bitmap. These functions are Insert Line, Delete Line, Insert Character, and Delete Character. They operate in the same manner as their counterparts in the Overlay.

The arrow keys have an additional meaning in the Bitmap: using the SHIFT modifier with an arrow key moves the cursor one pixel in the specified direction. Holding down SHIFT while pressing two adjacent arrow keys will move the cursor diagonally, one pixel at a time.

SHIFT left-arrow	Dot Left
SHIFT right-arrow	Dot Right
SHIFT up-arrow	Dot Up
SHIFT down-arrow	Dot Down

The CURSOR ON key works the same way as it did in the Overlay: pressing the key alone turns the cursor on, and using SHIFT with the key turns the cursor off.

NOTE:

If you turn the cursor off while in the Overlay, then switch to Bitmap, the cursor is still off. Both cursors are handled by the same command.

When you switch from Overlay to Bitmap, the Overlay cursor disappears and the Bitmap cursor appears. Only one of the two cursors will ever be visible in a window, depending on whether you are currently talking to the Overlay or the Bitmap.

MOVE X-Y positions the cursor absolutely, as it did in the Overlay.

Format:

MOVE X-Y <X>,<Y>,

Where:

<X> is a decimal number between 0 and 1023,
<Y> is between 0 and 767.

This is identical to the form used in the Overlay except that the limits for <X> and <Y> are adjusted to the Bitmap. The 0, 0 position is in the upper left corner. X increases from left to right, and Y increases going down. The Relative Move command is also present in Bitmap:

MODE m <dX>, <dY>,

This command moves the cursor relative to its current position, by <dX> pixels horizontally, and <dY> vertically. <dX> and <dY> are each decimal numbers terminated by a comma.

Tab stops are located every four character cells. Pressing the CTRL modifier and the character I moves the cursor to the next tab stop.

CTRL I

NOTE:

Tab stops can be changed using the Operate on Window Variable command, MODE]. See page 7-7 for details.

Set Cursor Color

Format:

MODE Q <color key>

The Bitmap cursor will assume the color specified by the color key. If the Blink attribute is currently on, the cursor will blink.

Alternate format:

MODE Q <n>,

Where <n> is the number of an entry in the Color Lookup Table, 0 through 255. This format would most often be used from a host computer.

The default cursor color is blinking white, which may be established by either of the commands

MODE Q 135,

MODE Q -1,

NOTE:

The bitmap cursor is software generated and is subject to being written over and destroyed. Internal precautions are taken to avoid this, but using multiple windows makes it possible to damage the cursor. The MODE Q command can be used to repair the cursor if this happens.

Fill Mode

Many of the "primitive" figures supported by the plotting software are "fillable," that is, they can be filled as they are drawn. The fillable primitives are: arcs, circles, rectangles, triangles, incremental X-bar and Y-bar, and the optional polygon (tiler) algorithm.

To specify that the figure is to be filled as it is drawn, the "Fill" attribute must be turned on.

Format:

FILL

The light in the FILL key will illuminate, and the "Fill" attribute will be turned on. Whenever the "Fill" attribute is on (signified by the lighted key), a figure which is fillable will be drawn and filled in the currently specified foreground color.

To turn off the "Fill" attribute, use the command:

SHIFT FILL

Complex Fill Algorithms

The software described in this section is optional, and may or may not be installed in your system.

Complex Fill algorithms are used to fill, or "paint," an area with a color. In some cases this area must be enclosed by a boundary of pixels, as described below. Complex Fill may be used on ANY types of figures, not only the figures supported by the plotting routines.

There are, however, some limitations:

- Complex Fill is a routine which demands a great deal of resources from the 7900 processor. An area is filled by drawing horizontal vectors, and whenever the algorithm encounters a "problem" in the area (such as a corner), it must remember the location of the "problem" and return to that location later to "solve" it. Each location that must be remembered requires a certain amount of space in the system's memory stack, and trying to fill an extremely complex area can overflow the available stack space. If this occurs, the fill routine will stop. If the fill routine stops before completing a fill of a very complex area, an overflow has probably occurred. The **StackZ** parameter in Thaw can be enlarged to remedy this problem; see page 6-6.

- Another point to remember when using Complex Fill is this: the area to be filled MUST be bounded. The boundary color must be the same as the color being used to fill (Edge Fill) or different from the color being used to fill (Area Fill). If a proper boundary is not present, the fill algorithm may "leak" and the color being used to fill may fill the entire screen. Or, if no border exists between the filled area and the top of the screen, the fill routine may stop filling at some apparently random location.

Area Fill

Format:

AREA FILL <color> <X>,<Y>,

Where:

<color> is one of the eight color keys, or a number corresponding to an entry in the Color Lookup Table (0 to 255). If a number is used, it must be delimited by a comma.

<X>,<Y>, are the coordinates of a point INSIDE the area which is to be filled.

The area containing the point <X>, <Y> will be filled with the color specified. The fill will stop at any edge containing a color other than the color specified.

Example:

```
CIRCLE 511,383,100,          (draw a circle)

AREA FILL RED 511,383,       (fill it with red)

AREA FILL BLUE 511.383,      (now fill it with blue)
```

In this example, we drew a white circle, and filled it with red. Note that the border of the circle is still white. The fill routine terminated when it reached the white edge of the circle. Note also that we can re-fill the area with another color (in this case, blue).

Alternate Format:

AREA FILL <color> .

The cursor position may also be used to specify coordinates. In this case, the cursor should be moved inside the area to be filled, and then the AREA FILL command may be executed using the cursor position to specify the <X> and <Y> coordinates.

Edge Fill

Format:

SHIFT AREA FILL <color> <X>,<Y>,

Where:

<color> is one of the eight color keys, or a number corresponding to an entry in the Color Lookup Table (0 to 255). If a number is used, it must be delimited by a comma.

<X>,<Y>, are the coordinates of a point INSIDE the area which is to be filled.

Edge Fill is very similar to Area Fill, except that Edge Fill will fill an area until it reaches a boundary of the SAME color it is using to fill. Area Fill would stop at any dissimilar boundary.

Example of Area Fill and Edge Fill:

RECT 0,0, 511,383,	(rectangle around 1/4 of screen)
SET RED	(set foreground color to red)
FILL	(turn on the "fill" attribute)
CIRCLE 256.192.70	(circle inside the rectangle)
MOVE X-Y 20.20.	(move cursor inside rectangle)
AREA FILL BLUE .	(fill the area with blue)

At this point (see illustration), the upper left quarter of the screen contains a white rectangle, filled with blue, and a filled red circle. The Area Fill command did not cover the red circle, nor did it cover the rectangle's white border.

SHIFT AREA FILL WHITE . (edge fill with white)

This time, the Edge Fill command filled until it reached the rectangle's white border. It covered the red circle, since it does not stop at anything except the color it is filling with (white).

Joystick

The CGC 7900 joystick (optional) is a three-axis input device. It generates an interrupt to the system whenever it is moved off its rest position, in any one of the three axes.

Format:

USER \ <0 or 1>

Use the character 0 to disable the joystick, or 1 to enable it.

When the joystick is enabled, it controls cursor movement and zoom. The cursor may be moved in the X or Y direction by moving the joystick in the appropriate direction. Holding the SHIFT key while moving the joystick will move the cursor one pixel at a time, for exact positioning. The Z-axis of the joystick is used to zoom in and out: twist the top knob clockwise to zoom in, counterclockwise to zoom out.

In addition, the joystick allows selection of color. Once the cursor is positioned over an area whose color is to be changed, hold down the M1 key and use the joystick to modify the color of the area. This "Joy-Color" mode allows you to change the red component of a color by moving the joystick in the X-direction; Y corresponds to green, and Z (twist) corresponds to blue. Once the color is satisfactory, release the joystick and the chosen color will remain. The SHIFT key may be used for slow color changes.

At the moment the joystick is enabled, the system reads values from all three axes, to establish the "zero" position. It is important that the joystick be at rest when the "enable" command is given.

NOTE:

Since the joystick operates at the interrupt level, nothing done with the joystick may be stored in the Create Buffer.

Pan and Zoom

The CGC 7900 includes hardware facilities to zoom or pan a high-resolution image in the Bitmap. These functions do not affect the contents of Bitmap memory; they alter the way in which the information in the Bitmap is displayed on the CRT screen.

Zoom is performed by magnifying each pixel in memory. The system will zoom in any integer multiple from 1 to 16 times. Commands described below allow setting the X and Y zoom levels simultaneously, or independent of one another.

Pan is performed by altering the address of the pixel being displayed in the upper left corner or the screen. When the zoom level is higher than 1 (default), pan allows viewing different areas of the image at high magnification. Pan is also used to allow viewing the bottom 256 lines of Bitmap memory, which are normally hidden offscreen below the viewable area.

Note that all pan and zoom functions apply to Bitmap memory only. The Overlay cannot be panned or zoomed. This allows menus, or other information in the Overlay to remain stationary while the Bitmap image may be examined with pan or zoom.

Pan and zoom are manipulated using the cursor control keys (the four arrows and HOME), and the ERASE PAGE, CLEAR LINE, and RECALL keys, all used in conjunction with the M2 modifier key.

Pan Left	M2 left-arrow
Pan Right	M2 right-arrow
Pan Up	M2 up-arrow
Pan Down	M2 down-arrow
Reset Pan	M2 HOME

You can use the REPEAT key to speed up any of the panning operations.

The two following commands always zoom towards (or away from) the Bitmap cursor in window A (the "Master Window"):

Zoom Up	M2 ERASE PAGE
Zoom Down	M2 RECALL

The following command adjusts the Pan registers so that the cursor is visible. The cursor is not moved.

Pan to Cursor Location M2 CLEAR LINE

See the Joystick description for other ways of controlling zoom and pan.

NOTE:

Joystick zoom and pan always tries to keep the cursor centered on the screen without causing the image to wrap. This is an aesthetic judgement and can only be overridden by using Absolute Zoom and Pan.

Absolute Pan

Format:

PAN X-Y <X>,<Y>,

Where <X>, <Y> are the desired coordinates of the upper left corner of the screen. The pan registers will be aligned to place the point <X>, <Y> in the upper left corner of the viewable screen. (Only absolute coordinates are allowed in this command: any scale factors in use will not be recognized.)

NOTE:

PAN X-Y is a single key. This label is on the front of the key marked MOVE X-Y.

Examples:

PAN X-Y 511,383,

The point 511, 383, which is normally the center of the screen, will be moved to the upper left corner of the screen.

PAN X-Y 0,0.

This command will reset pan to its default state, and is equivalent to M2 HOME.

Absolute Zoom

Format:

ESC Z <ZX> <ZY>

Where <ZX> and <ZY> are each single characters. Note that no commas are used to separate <ZX> and <ZY>.

The binary composition of the ASCII characters <ZX> and <ZY> will set the zoom registers. Since the zoom factor can vary from 1 to 16, the contents of the zoom registers may also take on 16 different values. Each of the two zoom registers may be set to any value from Ø to 15 (Ø corresponds to a zoom factor of 1, or default conditions). The X zoom and Y zoom registers may be set to different values, if desired.

The lowest 4 bits of the ASCII characters <ZX> and <ZY> are used to set the zoom registers. Any printing ASCII characters can be used. The following table provides a sample list of characters which will produce good results.

Zoom Factor	ASCII Character
1 (default)	@
2	A
3	B
4	C
5	D
6	E
7	F
8	G
9	H
1Ø	I
11	J
12	K
13	L
14	M
15	N
16 (maximum)	O

Examples:

ESC Z B B

Horizontal and vertical zoom factors are now set to 3.

ESC Z @ @

Horizontal and vertical zoom factors are now set to 1. This is the default condition.

ESC Z @ D

Horizontal zoom is now 1, and vertical zoom is now set to 5. The image on the CRT is "stretched" vertically.

Bitmap Roll and Page

The Bitmap defaults to Page. This is most suitable for drawing images, since no part of the window is ever moved as new items are added to the image.

If the Bitmap is to be used for extensive alphanumerics, Roll may be desirable. Roll is most suited for computer terminal applications. In Roll, as the last line of the window is filled with text, the top line "rolls" off the top of the window and is lost. To enter Roll, press the ROLL key:

ROLL

The key will illuminate, and the window is in Roll.

To leave Roll and return to Page, use the SHIFT modifier with the ROLL key:

SHIFT ROLL

The lighted key will extinguish and the window is in Page mode again.

The Bitmap is considerably slower than the Overlay when rolling data, since there are over a million pixels in Bitmap memory; a roll may move most of these pixels. Only the pixels within a given window are affected by a roll, however, so the smaller the window is defined to be, the faster the roll will occur (usually).

The exception to this rule is when the window is defined to be the full Bitmap memory, 0, 0 to 1023, 1023. (This is the default window size at power-up.) When the window is this size, a fast hardware roll circuit is used to roll the Bitmap. However, this hardware roll takes place when text moves past the BOTTOM of Bitmap memory, and this area is normally invisible (unless Pan is being used). See "Overlay Roll and Page" for more comments on this subject.

Rubber Band

Format:

RUBBER BAND

When the RUBBER BAND key is pressed, several things happen. The current cursor position, and the current foreground color, are memorized. The upper half of the Color Lookup Table (which contains all blinking color information) is saved, and reloaded with the current foreground color. Blink is disabled. The RUBBER BAND key lights up.

While Rubber Band is on, moving the cursor will cause a vector to be drawn from a previous cursor position to the current position. One end of this vector appears to follow the cursor, and "stretches" to meet the cursor; thus the name, "Rubber Band." The "rubber vector" is always drawn in whatever foreground color was in effect when Rubber Band was entered. (You may set a new foreground color while in Rubber Band, but this will affect the color of the figures you draw, NOT the color of the "rubber vector.")

While Rubber Band is on, you may be in any Plot Submode, or in Alpha. Rubber Band will simply continue to draw vectors wherever the cursor goes. Every time a coordinate argument is entered, Rubber Band memorizes the cursor position and begins drawing the "rubber vector" from this new location.

To leave Rubber Band, press:

SHIFT RUBBER BAND

The key will extinguish. The upper half of the Color Lookup Table is restored, and Blink is re-enabled.

NOTE:

Rubber Band requires that your image memory contain a plane number 7. If plane 7 does not exist in your system, Rubber Band cannot operate.

Plane 7 is also the default Blink plane. It is not possible to use Rubber Band and Blink at the same time, unless Blink is assigned to another plane. Rubber Band may destroy any parts of a picture that were drawn with blinking colors.

Chapter 9 -- Plot Submodes

This section describes the different Plot Submodes available for generating primitive figures.

Each Submode is entered by pressing a key, such as CIRCLE, or entering a Plot code sequence. After entering a Submode, you may draw as many figures of that type as you wish, without retyping the original Submode key.

Each figure will require that you enter a certain number of coordinates or parameters. For example, each circle requires a set of X, Y coordinates for the center and a radius. After the first three parameters have been entered, the system has sufficient information to draw one circle, and it will proceed to do so. If three more numbers are entered, they will be interpreted as instructions to draw another circle. This process continues until the Submode is exited, by entering another Submode or by leaving Plot altogether (and returning to Alpha).

As you will see in the examples, each argument given to a Plot Submode must be delimited. The delimiter may be a comma or a semicolon; either character is recognized as a valid delimiter. (The exception is for Polygons; you must use commas to separate each argument and terminate the argument list with a semicolon.) We suggest that a comma be used to separate numbers in a group (such as the three arguments which determine a circle), but that the semicolon be used to end the group. When organizing large volumes of numeric data, the group separators will be very useful.

For the sake of simplicity, in our examples, we have used a comma everywhere as a delimiter. The semicolon will be most useful when entering large amounts of data from a host device.

Figure 9-1. Plot Submodes (Primitives)

Figure 9-2. Other Plot Submodes

Arc

Format:

ARC <X>,<Y>, <R>, <start>, <delta>,

Where:

<X>,<Y>, are the coordinates of the center of the arc

<R> is the radius of the arc

<start> is the starting angle of the arc (degrees)

<delta> is the number of degrees of arc to plot

This software is optional, and may or may not be installed in your system.

Arc submode will draw a part of a circle. The circle (of which the arc would be a part) is centered at <X>, <Y>. The arc is drawn from the angle specified by <start>, and will proceed for <delta> degrees. (The zero degree position for <start> is toward the right.)

If the "Fill" attribute is currently on, the arc will be filled. This provides a quick way to draw "pie-chart" displays.

Example:

```
SET YELLOW
FILL
ARC 511,383, 30, 30, 300,
```

An arc will be drawn at the center of the screen, with radius 100. It will start at the zero-degree angle and proceed for 90 degrees (ending directly above the center). This example draws a quarter of a circle.

<start> and <delta> are always entered as integer degree values, and are unaffected by any scale factors in use.

NOTE:

This algorithm sacrifices accuracy for speed. Arc is accurate to one degree.

Circle

Format:

CIRCLE <X>, <Y>, <R>,

Where:

<X>,<Y>, are the coordinates of the center of the circle

<R> is the radius

A circle will be drawn at the specified center coordinate <X>, <Y>, with radius <R>. The circle will be drawn in the currently specified foreground color. If the "Fill" attribute is currently on, the circle will be filled with the foreground color.

Example:

CIRCLE 511,383,100,

would draw a circle centered at 511, 383, which is the center of the visible Bitmap area. The radius will be 100.

Alternate Format:

CIRCLE . <R>,

A circle will be drawn at the current cursor position ("Circle at this Point") with radius <R>.

Two-point Circle

Format:

PLOT O <X>,<Y>, <X1>,<Y1>,

Where:

<X>,<Y>, are the coordinates of the circle's center

<X1>,<Y1>, are used to set the radius of the circle (see below)

NOTE:

The PLOT key used in this command is the key labelled MODE and PLOT, in the typewriter area of the keyboard. It is NOT the illuminated PLOT key in the upper keyboard area.

The character "O" used in this function is the alphabetic, upper-case letter O and not the character zero.

Two-Point Circle operates exactly like Circle, except that the radius is determined by the absolute value of either |X-X1| or |Y-Y1|, whichever is greater. The smaller number is ignored.

The Two-Point Circle is most useful with certain input devices, like the Light Pen, which always send pairs of numbers. The Light Pen does not work with Circle since Circle requires three arguments. Two-Point Circle requires **four** arguments for each figure it draws, like most other Plot Submodes.

If the Fill attribute is on, the Two-Point Circle will be filled.

Curve

Format:

CURVE <X1>,<Y1>, <X2>,<Y2>, <X3>,<Y3>, <X4>,<Y4>,

This software is optional, and may or may not be installed in your system.

The Curve algorithm draws a figure called a Four-Point Bezier Curve. Each curve requires that four coordinates be entered. The four coordinates describe the curve as follows:

<X1>, <Y1> (point 1) is the starting point for the curve, and the curve will pass through this point.

<X4>, <Y4> (point 4) is the ending point for the curve, and the curve will pass through this point.

<X2>, <Y2>, (point 2) and <X3>, <Y3> (point 3) are points which exert an "influence" on the curve. The curve will tend to be "pulled" toward these points, but will not usually pass through either of them. The curve is well-behaved, however, since the curve will always lie entirely within the polygon formed by drawing a line between the four points.

At point 1, the curve will be tangent to the line from point 1 to point 2.

At point 4, the curve will be tangent to the line from point 3 to point 4.

Example:

CURVE 0,0, 0,767, 1023,0, 1023,767,

This example draws a curve which begins at the upper left corner of the screen and ends at the lower right corner. The other two corners "pull" the curve into an S-shape.

Alternate Format:

CURVE . . . .

The cursor position may be used to specify coordinates. For each coordinate pair, move the cursor to the desired location and strike the period (decimal point) key.

The curve function is well suited to human interaction. Very complex figures can be designed by linking together some simple curves. Since the curve is always well-defined at its end points, it is possible to join two or more curves perfectly. This is done by arranging point 4 of one curve to be the same as point 1 of the next, and making points 3 and 4 of one curve collinear with point 2 of the next.

NOTE:

Since Curve uses a recursive algorithm, points may not always join perfectly due to roundoff error. This could be disastrous when using a Complex Fill mode.

Curve points MUST always lie within the scaled screen boundaries.

Dot

Format:

DOT <X>,<Y>,

To initiate the Dot submode, hold down the SHIFT key and press the key labelled DOT and VECTOR. After each pair of coordinates is entered, a single dot will be placed on the screen, in the currently specified foreground color.

Example:

DOT 511,383,

This will place a dot in the center of the visible Bitmap screen. A single dot is very small, and may be difficult to see.

As with all Plot Submodes, now that we have entered the Submode we can enter more information and additional figures will be drawn. In DOT Submode, a single dot will be placed on the screen every time we enter another pair of <X>, <Y>, coordinates.

Alternate Format:

DOT .

A single dot will be placed on the screen under the cursor. It will be necessary to move the cursor in order to see the dot.

Incremental Vector

Format:

INC VECTOR <X>,<Y>, [<dX><dY>...]

Where:

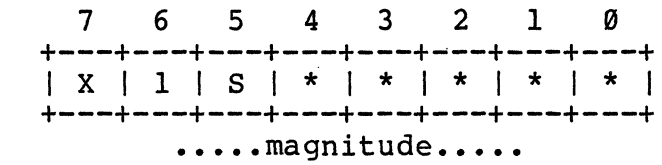
<X>,<Y>, are the starting coordinates for the set of short vectors,

<dX><dY> are single characters (NOT numbers) which determine the displacement in the X and Y directions. Note that no commas are used to separate <dX> and <dY>.

Incremental vector draws a short vector from the point <X>, <Y>. The length and direction are determined by the characters <dX> and <dY>. The bits of <dX> determine a displacement in the X direction, up to plus or minus 31 pixels. <dY> does the same in the Y direction.

If more <dX> and <dY> characters are entered, more short vectors will be drawn and will be concatenated with previous short vectors.

The bits in <dX> and <dY> are interpreted as follows:



The state of bit 7 is ignored. Bit 6 must be a 1. Bit 5 is a sign bit, 0 for a positive displacement and 1 for a negative displacement. Bits 4 through 0 are used as a 5-bit magnitude, capable of specifying numbers up to 31. Negative numbers are NOT interpreted in two's complement form.

The following table illustrates the displacement corresponding to each ASCII character. Note that since bit 6 is always set to 1, the characters used for <dX> and <dY> will always be in the printable ASCII set. No control-characters are used (except DEL).

Binary	ASCII	displacement	Binary	ASCII	displacement
1000000	@	0	1100000	`	0
1000001	A	1	1100001	a	-1
1000010	B	2	1100010	b	-2
1000011	C	3	1100011	c	-3
1000100	D	4	1100100	d	-4
1000101	E	5	1100101	e	-5
1000110	F	6	1100110	f	-6
1000111	G	7	1100111	g	-7
1001000	H	8	1101000	h	-8
1001001	I	9	1101001	i	-9
1001010	J	10	1101010	j	-10
1001011	K	11	1101011	k	-11
1001100	L	12	1101100	l	-12
1001101	M	13	1101101	m	-13
1001110	N	14	1101110	n	-14
1001111	O	15	1101111	o	-15
1010000	P	16	1110000	p	-16
1010001	Q	17	1110001	q	-17
1010010	R	18	1110010	r	-18
1010011	S	19	1110011	s	-19
1010100	T	20	1110100	t	-20
1010101	U	21	1110101	u	-21
1010110	V	22	1114110	v	-22
1010111	W	23	1110111	w	-23
1011000	X	24	1111000	x	-24
1011001	Y	25	1111001	y	-25
1011010	Z	26	1111010	z	-26
1011011	[	27	1111011	{	-27
1011100	\	28	1111100	|	-28
1011101	]	29	1111101	}	-29
1011110	^	30	1111110	~	-30
1011111	_	31	1111111	DEL	-31

Examples:

INC VECTOR 511,383,D E

The displacements for ASCII characters D and E are +4 and +5, respectively. A vector will be drawn from the point 511, 383, to a point 4 pixels to the right and 5 pixels below the original point. Note that a positive Y-displacement moves DOWN the screen since the Y=0 line is at the top.

INC VECTOR 600,400,D E D e

This set of short vectors begins at the point 600, 400. The first vector will be drawn from this point, 4 pixels to the right, and 5 down. A second vector will follow, 4 to the right and 5 up (the displacement for the e character is -5). So the end point of the set has the same Y coordinate it had at the beginning, but its X coordinate is shifted 8 to the right.

INC VECTOR . J A

The cursor position may be used to enter the initial coordinate of the set. To use this format, position the cursor to the desired location and strike the decimal point key to enter the cursor position as a coordinate pair. Then enter the ASCII character displacements as before.

Incremental X-Bar

Format:

INC X-BAR <XØ>,<YØ>,<X1>, [<X2>,<X3>,...]

Where:

<XØ>,<YØ>	are the coordinates of one endpoint of the first bar
<X1>	is the X-coordinate of the other endpoint of the first bar (the previous Y-coordinate is used as a reference -- see below)
<X2>,<X3>,...	are X-coordinates of the endpoints of subsequent bars (see below)

A horizontal bar (actually a rectangle) is drawn, with corners located at <XØ>, <YØ>, and <X1>, <YØ>+<w>. The default value of <w> is 3. The value of <w> may be altered by the "Set Vector Width" command. If the "Fill" attribute is currently on, the bar will be filled.

After the first bar is drawn, more X-coordinates may be entered. Additional bars will be drawn adjacent to the existing ones. The effective coordinates for the corners of each bar are:

<XØ>, <YØ> + (n-1)<w> for one corner,
<Xn>, <YØ> + (n)<w> for the diagonally opposite corner,

where n is the bar number (n=1 for the first bar).

Example:

INC X-BAR 511,100, 611, 711, 811,

Three bars are drawn. Each will be drawn from the vertical line X=511 to the required X-coordinate. The first bar will begin at the horizontal line Y=100, and will extend to the line Y=104. The next bar will be drawn from Y=105 to Y=109. As many additional X-coordinates may be entered as you require.

INC X-BAR 511,400, 611, 411, 711, 311,

This time four bars are drawn. Notice that the bars may be drawn either to the right or left of the original Y=<Y0> line.

If desired, the decimal point may be used to enter the cursor position for each point. The first time the decimal point is pressed, it will enter the initial <X0>, <Y0> position. Each subsequent time the decimal point is pressed, the X-coordinate will be taken from the current cursor position (the Y-coordinate will be discarded).

Incremental Y-Bar

Format:

INC Y-BAR <X0>,<Y0>,<Y1>, [<Y2>,<Y3>,...]

Where:

<X0>,<Y0>	are the coordinates of one endpoint of the first vector
<Y1>	is the Y-coordinate of the other endpoint of the first vector (the previous X-coordinate is used as a reference -- see below)
<Y2>,<Y3>,...	are Y-coordinates of the endpoints of subsequent vectors (see below)

Operation is identical to Incremental X-Bar. Vertical bars are drawn with width <w>, set by the "Set Vector Width" command. The default value of <w> is 3. If the "Fill" attribute is currently on, the bar will be filled.

The diagonal coordinates of the first bar are <X0>, <Y0> and <X0>+<w>, <Y1>. If additional Y-coordinates are entered, more bars will be drawn. The effective coordinates for each bar are:

<X0> + (n-1)<w>, <Y0> for one corner,
<X0> + (n)<w>, <Yn> for the diagonally opposite corner,

where n is the bar number (n=1 for the first bar).

Example:

INC Y-BAR 0,383, 300, 200, 100, 200, 300,

Five vertical bars will be drawn, in a histogram-type format.

If desired, the decimal point may be used to enter the cursor position for each point. The first time the decimal point is pressed, it will enter the initial <X0>, <Y0> position. Each subsequent time the decimal point is pressed, the Y-coordinate will be taken from the current cursor position (the X-coordinate will be discarded).

Polygon

Format:

POLYGON <X1>,<Y1>,<X2>,<Y2>, [<Xn>,<Yn>,...] ;

This software is optional, and may or may not be installed in your 7900 system. If Polygon is not installed, a polygon may be drawn with concatenated vectors and filled with AREA FILL, if needed.

Polygon draws a set of concatenated vectors, defined by the coordinate pairs <X1>, <Y1>, through <Xn>, <Yn>. The endpoint of the last vector is joined to the first vector, to form a closed polygon. Up to 30 coordinate pairs may be entered. (If you need more than 30 points, use "Large Polygon," described on the next page.)

The semicolon MUST be entered after the final coordinate pair, to signal that all the coordinates have been entered and that drawing should now begin. NOTHING IS DRAWN until the semicolon is entered.

When entering coordinates in standard decimal form, as in the format above, the last coordinate pair must be terminated by a comma and then by the required semicolon.

Example:

POLYGON 0,0, 511,383, 1023,0, 511,767, ;

The polygon is drawn from the origin (upper left corner of the screen) to the center of the screen, to the upper right corner, to the center of the bottom. The final vector connects the center bottom to the origin.

Note that the final product is very similar to a set of concatenated vectors. But since all the coordinates are entered before drawing begins, the object appears to be drawn much faster.

Alternate Format:

POLYGON . . . . ;

The cursor position may also be used to specify coordinates. The cursor should be moved to each point where a vertex of the polygon is desired. The decimal point key should then be struck at each point. After all points have been entered in this manner, the final semicolon must be entered.

The polygon will be filled if the Fill attribute is active. Filled polygons are drawn somewhat differently than unfilled figures, but the result is the same.

Large Polygons

Format:

ESC P <X1>,<Y1>, <X2>,<Y2>, ... ;

The Large Polygon command must be used for figures with more than 30 points. It is independent of windows and assignable devices. It can be used the same way as the normal POLYGON command, except that the dot (.) cannot be used to enter coordinates.

The default size limit for Large Polygon is 250 coordinates. The Thaw parameter **EscArgZ** controls this limit and may be altered using THAW. Allow 4 bytes per coordinate plus room for 1 extra coordinate.

NOTE:

The fill attribute of window 0 is used with Large Polygon.

Ray

Format:

RAY <X>,<Y>, <R>, <angle>,

Where:

<X>,<Y>, are the coordinates of one end of the ray

<R> is the length of the ray

<angle> is the angle from <X>,<Y> at which the ray is drawn.

This software is optional, and may or may not be installed in your system.

Ray submode draws a vector in polar form. The origin of the vector is specified as an <X>, <Y> coordinate pair. The length and direction are then specified, which determine the other end point of the vector.

Example:

RAY 511,383, 100, 45,

A ray will be drawn from the center of the screen, toward the upper right quadrant, at an angle of 45 degrees. If this example is executed after the Arc example in the previous section, it will divide the 90 degree arc into two 45 degree segments.

One application for Ray is the development of "pie-chart" graphs. Ray allows the programmer to subdivide a circle into any desired segments, simply by entering the angle in degrees. (The <angle> is always entered in integer degrees, and is not affected by any scale factors in use.)

Rectangle

Format:

RECT <X1>,<Y1>, <X2>,<Y2>,

A rectangle will be drawn whose opposite corners are located at <X1>, <Y1>, and <X2>, <Y2>, in the currently specified foreground color. If the "Fill" attribute is currently on, the rectangle will be filled with the foreground color.

Example:

RECT 0,0, 511,383,

would draw a rectangle around the upper left quarter of the screen.

Alternate Format:

RECT . .

The cursor position is used to specify the opposite corners of the rectangle. The cursor should be moved to the desired locations and the decimal point struck at each location ("Rectangle from this Point to this Point").

Triangle

Format:

TRIANGLE <X1>,<Y1>, <X2>,<Y2>, <X3>,<Y3>,

A triangle will be drawn between the three specified points, in the currently specified foreground color. If the "Fill" attribute is currently on, the triangle will be filled with the foreground color.

Example:

TRIANGLE 0,0, 0,383, 511,383,

will draw a triangle from the upper left corner, vertically halfway down the screen, to the center of the screen, and return to the starting point.

Alternate Format:

TRIANGLE . . .

The cursor position may be used to specify coordinates. The cursor should be moved to each location where a vertex of the triangle is desired, and the decimal point struck at each location ("Triangle from this Point to this Point to this Point").

NOTE:

For filled triangles, use POLYGON if your system has it. It is faster and more accurate.

Vector

Format:

VECTOR <X1>,<Y1>, <X2>,<Y2>,

Where:

<X1>,<Y1>, are the coordinates of one end of the vector
<X2>,<Y2>, are the coordinates of the other end

A vector will be drawn from point <X1>, <Y1>, to point <X2>, <Y2>, in the currently specified foreground color.

If additional pairs of points are entered, a new vector will be drawn between them. The new vector will not be connected to the first one (see Concatenated Vector).

Example:

VECTOR 0,0,511,383,

A vector will be drawn from the 0, 0, position (the upper left corner) to the center of the visible screen.

Alternate Format:

VECTOR . .

The decimal point may be used to enter coordinates. The cursor control keys should be used to move the cursor to the desired end points of the vector, and the decimal point should be struck once at each point ("Vector from this Point to this Point").

Concatenated Vector

Format:

CONCAT VECTOR <X1>,<Y1>, <X2>,<Y2>, [<X3>,<Y3>,...]

Where:

<X1>,<Y1>, are the starting coordinates of the set of concatenated vectors

<X2>,<Y2>, are the coordinates of the endpoint of the first vector in the set

<X3>,<Y3>, and all other coordinates, are endpoints of subsequent vectors in the set

A vector will be drawn from <X1>, <Y1>, to <X2>, <Y2>, and another connected vector from <X2>, <Y2>, to <X3>, <Y3>, and so on. Each additional coordinate entered will cause a new vector to be drawn from the endpoint of the previous vector.

Example:

CONCAT VECTOR 0,0, 511,383, 1023,0, 1023,767,

A vector will be drawn from the upper left corner, to the center, to the upper right corner, to the lower right corner. Any number of additional points could be added.

Alternate Format:

CONCAT VECTOR . . . .

The decimal point may be used to enter coordinates. The cursor should be moved to each location where a vector endpoint should lie, and the decimal point struck at each location. ("Concatenated Vector from this Point to this Point to this Point...")

NOTE:

If you wish to terminate one set of concatenated vectors and begin another set, you must hit the CONCAT VECTOR key again.

Bold Vector

Format:

PLOT v <X1>,<Y1>, <X2>,<Y2>,

This software is optional, and may or may not be installed in your 7900 system.

NOTE:

The PLOT key used in this command is the key labelled MODE and PLOT in the typewriter area of the keyboard. It is NOT the lighted PLOT key in the upper keyboard area.

This command requires entering a lower case letter, **v**. If the ALPHA LOCK key is in its normal (up) position, hold down the SHIFT key while pressing the "V" key to produce a lower case **v**.

Operation is identical to Vector, except that a bold vector is drawn whose width is determined by the "Set Vector Width" command. The bold vector has rounded ends, produced by drawing small circles at each end of the vector. This makes connected bold vectors smoother (see "Concatenated Bold Vector" on the next page).

The value of <w> given by the "Set Vector Width" command is used as a radius for the circle at each end of the bold vector. Thus, the actual width of the bold vector is twice <w> plus one. If <w> is zero, a normal (not bold) vector is drawn.

Concatenated Bold Vector

Format:

PLOT w <X1>,<Y1>,<X2>,<Y2>,[<Xn>,<Yn>,...]

This software is optional, and may or may not be installed in your 7900 system.

NOTE:

The PLOT key used in this command is the key labelled MODE and PLOT, in the typewriter area of the keyboard. It is NOT the illuminated PLOT key in the upper keyboard area.

This command requires entering a lower case letter, w. If the ALPHA LOCK key is in its normal (up) position, hold down the SHIFT key while pressing the "W" key to produce a lower case w.

Operation is identical to Concatenated Vector, except that a series of bold vectors is drawn. See the description of Bold Vector on the previous page.

Set Vector Width

Format:

V. WIDTH <w>,

Where:

<w> is a decimal number, 0 to 65535

NOTE:

V. WIDTH is a single key. The name "V. WIDTH" is printed on the front of the SIZE key.

This command affects the Incremental X-Bar and Incremental Y-Bar Plot Submodes. It also sets the vector width for the optional "Bold Plotting" software routines.

When a wide vector is required, the most recently specified value of <w> is used. The default value of <w> is 3. If <w> is 0, normal (thin) lines will be drawn.

The value of <w> is specific to a window; each window may have its own vector width defined. If <w> has not been defined in a window, the default value will be used.

Vector-Drawn Characters

Format:

PLOT E <angle>, <flag>

Where:

<angle> is a decimal angle, in degrees

<flag> is the character 1, or Ø, to indicate whether proportional spacing should be used

NOTE:

The PLOT key used in this command is the key marked with MODE and PLOT labels, on the typewriter area of the keyboard. It is NOT the lighted PLOT key in the upper keyboard area.

This software is optional, and may or may not be installed in your system.

Vector-drawn characters may be drawn at any angle. The characters are drawn using bold vectors. After entering the command to begin this Plot Submode, you may enter characters normally and they will be plotted on the Bitmap screen.

The "Set Character Size" and "Set Vector Width" commands may be used to set up the characters for your application. In addition, since the vector-drawn characters use an 8 by 12 matrix (instead of the normal 6 by 8), the "Set Intercharacter Spacing" command may be used to adjust the system for a larger character font. An appropriate setting is MODE I 8, 12, when proportional spacing is NOT in use.

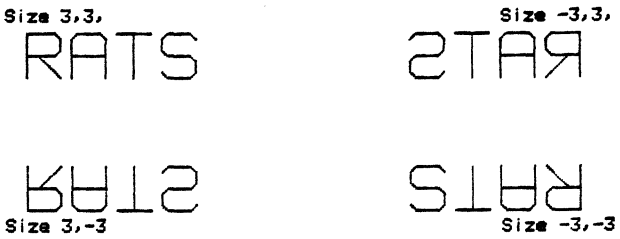
The vector-drawn characters are always in "overstrike" mode. There is no way to backspace, erase, or otherwise perform text editing functions on these characters. It is recommended that this character set be used primarily from a program or a host system, so that backspacing will not be required.

Example:

PLOT E 0, 1	Plotted characters, angle of zero degrees (horizontal), proportional spacing ON
SIZE 4, 4,	Character size 4 X
MOVE X-Y 10, 10,	Move cursor out in the open
This is a test	Type in some text.

If you wish the text to come out "mirrored" upside-down or backwards, you can specify a negative number in the SIZE command. The arrow keys will move backwards too, depending on whether X or Y size is negative. See the example below.

Example of Vector-drawn characters using funny numbers for Size. Each string was typed as "RATS"



Exiting Plot Submodes

Any time you enter a Plot Submode, the light on the PLOT key will come on. This light indicates you are in a Plot Submode. Pressing a different Plot Submode key will change the Submode you are in, but the lighted PLOT key will remain lit whenever you are in a Plot Submode.

To return to Alpha (text entry), type

SHIFT PLOT

Hold down the key marked SHIFT, and press the lighted key marked PLOT. The light will go out, and you are out of whatever Plot Submode you were in. You are back to Alpha.

To resume plotting, you may simply press any Plot Submode key, as you originally did to begin plotting. You can also re-enter the last Plot Submode you were using, by pressing the lighted PLOT key (without the SHIFT modifier).

Chapter 10 -- Raster Processor Graphics

The software described in this section is optional, and may or may not be installed in your 7900 system.

The Raster Processor is a special circuit in the 7900 hardware. Its main function is to move pixels from one area of the Bitmap screen to another. The functions described in this section make use of some of the capabilities of the Raster Processor.

Besides the "block move" capability of the Raster Processor, you can use the text editing commands on the four arrow keys (INS LINE, DEL LINE, INS CHAR, DEL CHAR) to move parts of the screen around. For example, if you did not leave enough room at the top of the screen to draw something, you could move the cursor into this area and use the INS LINE key to push everything below the cursor down.

For the purposes of this chapter, we define a "raster" to be a block of Bitmap pixels.

Define Source Raster

Format:

MODE [<X1>, <Y1>, <X2>, <Y2>,

Where:

<X1>,<Y1>, are the coordinates of one corner of the desired source raster

<X2>,<Y2>, are the coordinates of the diagonally opposite corner

A "raster" is a rectangular area of the Bitmap screen. The Raster Processor operates only on these rectangular areas. A "source raster" is defined exactly as you would enter the corner points of a rectangle, and all the pixels within this area are used in the commands which are discussed later in this section.

The source raster definition is specific to a window. Each window may have its own source raster defined to be a different area of the screen.

Note that the offscreen Bitmap area (the pixels not normally visible unless Zoom and Pan are in effect) may be used to store source rasters.

The definition of a source raster contains only the screen location of the raster, not the pixels within it. If a source raster is defined, and the pixels within that area are altered, the source raster is also altered.

Alternate Format:

MODE [. .

The current cursor position may be used to enter the X-Y coordinates of the source raster, by moving the cursor to each corner and striking the period (decimal point) key.

Copy Raster

Format:

COPY <X>, <Y>,

Where:

<X>, <Y> are the coordinates to which the raster should be copied.

A copy of the source raster is made at screen position <X>, <Y>. This may be thought of as a "rubber stamp" operation. The cursor position may be used instead of <X> and <Y>:

COPY .

This command copies the source raster to the current cursor position.

Example:

SHIFT OVERLAY	(look at Bitmap)
HOME A B C D RETURN LINE FEED	
E F G H	(put some characters in upper left corner of screen)
MODE [0, 0, 48, 32,	(define the source raster to be these characters)
COPY 511, 383,	(copy raster to the center of the screen)

Copy Raster with Overstrike

Format:

MODE u <X>, <Y>,

Where:

<X>, <Y> are the coordinates to which the raster should be copied.

NOTE:

This command requires entering a lower case character, **u**. If the ALPHA LOCK key is in its normal (up) position, press the SHIFT key with the "U" key to produce a lower case **u**.

This command is identical to the "Copy Raster" command, except that any pixels in the source raster matching the current background color are not written.

The cursor position may be used instead of <X> and <Y>:

MODE u .

This command copies the source raster to the current cursor position, using overstrike. It is slower than copying without overstrike, since the CPU gets involved.

Set Raster Direction

Format:

MODE { <n>,

Where:

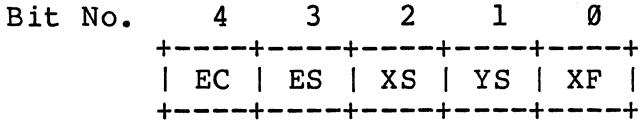
<n> is a decimal number, 27 through 31

NOTE:

This command is only applicable to the Copy Raster commands.

The Raster Processor uses a "control byte" to set the attributes of the source and destination rasters. This command allows you to modify the attributes of the source raster definition. Internally, the Raster Processor defines a raster by using a coordinate pair (called the "operating point"), a value for delta-X, and a value for delta-Y. Whenever the Raster Processor copies a raster, it begins at the operating point and performs operations determined by delta-X, delta-Y, and the control byte.

The bits in the control byte are defined as follows:



All other bits are unused.

XS is X Sign. If SET, the raster is scanned from left to right.

YS is Y Sign. If SET, the raster is scanned from top to bottom.

XF is X Fine. If SET, X is incremented first as the raster is scanned. When X overflows (becomes greater than delta-X), Y is incremented and X is set to its original value from the operating point. If this bit is CLEAR, Y is fine and is incremented first.

EC and **ES** are Enable Carry and Enable Step, respectively. If these bits are CLEAR, the Raster Processor does not increment certain registers. This prevents the entire raster from being scanned. This is not a useful condition, since the same effect can be achieved by defining the source raster to be one line (or one pixel). When using the Set Raster Direction command, you should leave these two bits SET.

The default value for <n> is 31. This corresponds to all bits SET, resulting in normal upright raster copies.

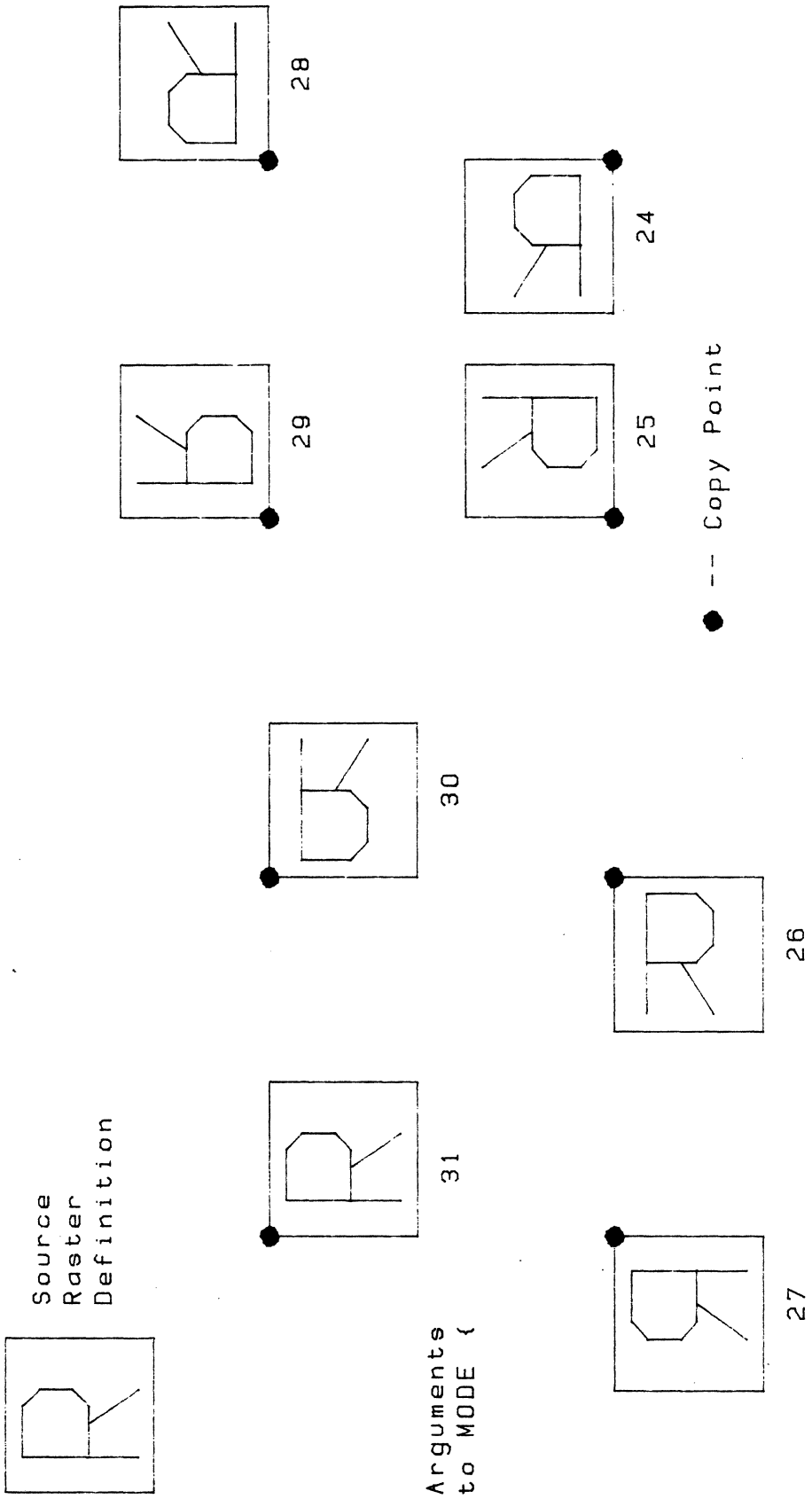
The illustration on the following page should help to clarify use of the Set Raster Direction command. Assume we have created a test pattern, consisting of a square containing a letter, and defined it to be the source raster. We chose the letter R because it is not symmetric about any axis, and rotation can be easily observed.)

Now if a Copy Raster command is executed, the square will be copied to the specified location. The control byte is now 31 (default), so an upright copy is generated.

If the X Fine bit is now cleared by performing a MODE { 30, the copy will be reflected about the 45 degree axis. (For this to work properly, the source raster MUST be square!)

If the Y Sign bit is now cleared by performing a MODE { 29, (this also SETS the X Fine bit), the raster will be copied upside-down and backwards.

Other combinations of these three bits will produce the other raster copies shown.


Raster Rotation and Reflection
-- Copy Point
Source Raster Definition
Arguments to MODE (
24
25
26
27
28
29
30
31

Patterned Vectors

Format:

MODE T <pattern> <clip>

This command enables patterned vectors with or without clipping. Both <pattern> and <clip> can be either 1 or 0.

For <pattern>, if a **1** is entered, patterned vectors are used for ALL Bitmap graphics functions, including vectors, circles, arcs, etc. The current source raster is used as a source for the pattern. While patterned vectors are active, the pattern is used INSTEAD of the current foreground color for all graphics operations. Entering a 0 disables patterned vectors.

When <clip> is 1, all lines and points drawn will be clipped to lie within the boundaries of the current window.

This command can be cancelled by the command MODE O 1 (Overlay ON) or by hitting the OVERLAY key.

Example:

SHIFT OVERLAY	(look at Bitmap)
HOME A B C D RETURN LINE FEED	
E F G H	(put some characters in upper left corner of screen)
MODE [0, 0, 48, 32,	(define the source raster to be these characters)
MODE T 1 0	(patterns ON, clipping OFF)
CIRCLE FILL 511, 383, 300,	(draw patterned circle)

Chapter 11 -- Expanded Image Memories

The commands in this section apply to CGC 7900 systems containing two banks of image memory planes.

As mentioned in the introduction to Section 3, the 7900 may be equipped with up to 16 planes of image memory. Only 8 planes at one time may feed the Color Lookup Table to form an image, but two independent images may be kept in memory simultaneously. The commands in this section provide access to the second set of image planes.

Select Image

Format:

ESC s <0 or 1>

Use the character 0 to select planes 0 through 7 for display (the default condition), or the character 1 to select the second image, planes 8 through 15.

NOTE:

This command requires entry of a lower case letter, **s**. If the ALPHA LOCK key is up, use the SHIFT modifier with the "S" key to produce a lower case **s**.

If planes 8 through 15 are not installed in your system, selecting planes 8 through 15 will cause the the Bitmap screen to go blank.

Plane/Video/Blink Select

These commands are described later in this chapter, but take on new meanings when more than 8 image planes are installed. They are: Plane Select, Plane Video Switch, and Blink Select.

Each of these commands requires entering a decimal number to enable selected image planes in a particular function. With only 8 image planes, the useful range of numbers is 0 to 255; this allows writing to any or all of the 8 planes. (255 is the highest decimal number possible from an 8 bit, or 8 plane, system.)

With up to 16 planes installed, the useful range of numbers for these commands increases to a 16-bit number. This allows values from 0 to 65535.

Examples:

MODE : 255,	Planes 0 through 7 are selected (write-enabled)
MODE : 65280,	Planes 8 through 15 are selected (65535 - 255 = 65280)
ESC b 32896,	Planes 7 and 15 are specified to blink (2^15 + 2^7 = 32896)
ESC S 771,	Video from planes 0, 1, 8 and 9 is enabled for display (2^0 + 2^1 + 2^8 + 2^9 = 771)

Obviously, the numbers can get out of hand quickly. Calculation of plane numbers in a 16-plane system is more suited to an applications program.

The decimal number -1 may still be entered, and it will always enable all planes in the system.

Writing to the Second Image

The following example demonstrates how to create two independent images.

ESC s 0	View the first image (planes 0-7)
MODE : 255,	Write-enable the first image
CIRCLE FILL SET BLUE 511,383,300,	Draw a blue circle in the first image
ESC s 1	View the second image (planes 8-15)
MODE : 65280,	Write-enable the second image
RECT SET RED 100,100,700,300,	Draw a red rectangle in the second image

Now, the command ESC s 0 displays a blue circle, and ESC s 1 displays a red rectangle.

Note that the default cursor color is color 135 (blinking white), so the Bitmap cursor will not be visible when planes 8 through 15 are being displayed. To make the cursor appear in all planes, and thus in both images, use

MODE Q -1,

NOTE:

When using a second image, be sure to set the Plane/Video/Blink select latches to the appropriate configuration before use.

Plane Select

Format:

MODE : <n>,

Where <n> is a number, 0 to 255, or -1.

<n> is examined as a binary number. Any bits in <n> which are set to one represent planes of Bitmap memory which will be write-enabled. Any bits in <n> which are zero represent planes which are not write-enabled, and cannot be altered until the next Plane Select command is given. This command only affects the window which receives it.

Examples:

MODE : 0,

Since all bits of the number 0 are zero, all planes of Bitmap memory are now write-disabled. They cannot be altered.

MODE : 6,

The number 6 is converted to a binary number, and examined as follows:

Plane #	7	6	5	4	3	2	1	0
	0	0	0	0	0	1	1	0

Since bits 1 and 2 are ones, only planes 1 and 2 of Bitmap memory can now be altered. The other planes will be unaffected by any commands or characters sent to the window.

Example:

MODE : -1,

Negative numbers are treated in "two's complement" form. Negative one is treated as a binary word of all ones, so all planes are now write-enabled. This is the default condition when the system is booted.

Plane Video Switch

Format:

ESC S <n>,

Where <n> is a number, 0 to 65535, or -1.

<n> is examined as a binary number. Any bits in <n> which are set to one represent planes of Bitmap memory which will be allowed to feed the Color Lookup Table. Any bits in <n> which are set to zero will prevent their corresponding planes of Bitmap from feeding the CLU.

NOTE:

This command affects ALL images from Bitmap memory. It is NOT specific to a window.

Example:

ESC S 0,

All bits in the number 0 are zero. All planes of Bitmap memory will not be fed to the Color Lookup Table (their bits will be masked to zero). Thus, the only color of the CLU which will be accessed is color zero (normally black). We have just shut off all image from Bitmap memory.

Example:

ESC S 5,

The number 5 is examined as follows:

Plane #	7	6	5	4	3	2	1	0
	0	0	0	0	0	1	0	1

Since bits 0 and 2 are set to one, only planes 0 and 2 will be allowed to feed the Color Lookup Table. All other planes will be masked to zero before being used to select colors. This configuration allows the following color choices:

Color 0 (all bits off)

Color 1 (bit 0 on)

Color 4 (bit 2 on)

Color 5 (bits 0 and 2 on)

Under default conditions, these four colors would correspond to Black, Blue, Red, and Magenta, respectively.

It can be seen from this example that the Plane Video Switch command limits the available color choices. The total number of choices which are available is 2 raised to the power <b>, where <b> is the number of bits set to one in the number <n>. In the example above, <n> is 5, which contains two bits (<b>) set to one. Two to the power two is four, the total number of colors we have made available.

Example:

ESC S -1,

Negative numbers are interpreted in "two's complement" form. Negative one is a binary word of all bits set to one, so this command will set all planes to feed the Color Lookup Table. This is the default condition when the system is booted.

A primary use for the Plane Video Switch command would be animation. Different pictures could be written to the various planes (using Plane Select, previously described), and then the system could quickly switch from one picture to another with Plane Video Select. The Color Lookup Table might have to be modified for best results.

Another use would be the layout of multiple-layer printed circuit boards. Each layer could be written into a single plane of Bitmap memory. Areas which overlapped would reference different areas of the Color Lookup Table, and would be displayed as different colors. The Plane Video Switch command would allow viewing each layer of the board separately.

Appendix A -- Special Codes

For the sake of clarity, the names of modifier keys are not underlined in this section. Remember, however, that control-codes are always generated by holding the CTRL key and simultaneously pressing the indicated key.

Control Codes

CTRL @	Null
CTRL A	Mode
CTRL B	Plot
CTRL C	
CTRL D	
CTRL E	Dot Up
CTRL F	
CTRL G	Bell
CTRL H	Backspace
CTRL I	Tab
CTRL J	Linefeed
CTRL K	Cursor Up (Line)
CTRL L	Erase Page
CTRL M	Carriage Return
CTRL N	A7 On (Alternate Character Set)
CTRL O	A7 Off (Standard Character Set)
CTRL P	
CTRL Q	X-On
CTRL R	
CTRL S	X-Off
CTRL T	
CTRL U	User
CTRL V	Dot Down
CTRL W	
CTRL X	End Of Record (Sub-Buffer marker)
CTRL Y	Dot Left
CTRL Z	Cancel (Flush input buffer)
CTRL [	Escape
CTRL \	Home
CTRL]	Cursor Right (Char)
CTRL ^	End Of File
CTRL _	Dot Right

Mode, Escape and User code sequences may have one or more arguments. The type of argument required is indicated as follows:

a decimal number terminated by a comma, or a binary number (if in binary mode)

XY an X-Y coordinate pair. It may be either the period (.) or two decimal numbers ("##" as described above)

F the character 1, or Ø, used as an on/off flag

n a single digit (Ø through 9)

C a single character (details depend on the command)

Colors are described as **#**, since they may be entered as numbers.

Mode Codes

Mode codes always begin with the MODE character, CTRL A, Ø1 hex.

MODE : #	Plane Select
MODE ;	
MODE < 1	Delete Character
MODE < 2	Delete Line
MODE =	Soft Boot
MODE > 1	Insert Character
MODE > 2	Insert Line
MODE ?	Keyboard Sync
MODE @	Clear line
MODE A	
MODE B F	Binary Mode (MODE and PLOT codes)
MODE C #	Set Foreground color
MODE D	
MODE E F	Rubber Band on/off
MODE F F	Fill on/off
MODE G	
MODE H	
MODE I ##	Set Bitmap intercharacter spacing
MODE J F	Cursor on/off
MODE K F	Blink on/off
MODE L	
MODE M XY	Move cursor to X-Y
MODE N	
MODE O F	Overlay on/off
MODE P F	Plot on/off
MODE Q #	Set Cursor color
MODE R F	Roll on/off
MODE S F	Scale on/off
MODE T nn	Set vector type
MODE U XY	Copy raster to X-Y
MODE V n	Set Overlay present visibility
MODE W XY XY	Define Window limits
MODE X ##	Set character size (X, Y factor)
MODE Y ##	Colorswap X and Y
MODE Z	
MODE [XY XY	Define source raster
MODE \ F	Overstrike on/off
MODE] CC#	Operate on variable
MODE ^ # XY	Area Fill (color, X, Y)
MODE `	Clear to end of line
MODE a	
MODE b	
MODE c #	Set Background color
MODE d	
MODE e	
MODE f	

```
MODE g
MODE h
MODE i ###      Load User Character Set
MODE j
MODE k
MODE l          Erase to end of screen
MODE m ##       Move Cursor relative
MODE n
MODE o
MODE p
MODE q
MODE r
MODE s XY XY    Set Scale Factors
MODE t C        Test window
MODE u XY       Copy raster to X-Y with overstrike
MODE v n        Set Overlay future visibility
MODE w
MODE x #        Set vector width
MODE y ##       Set colors (X to Y)
MODE z
MODE { #        Set Source Raster direction
MODE |          Home lower (lower left corner)
MODE }
MODE ~ # XY     Edge Fill (color, X, Y)
MODE DEL C      Ignore argument (undefined keys)
```

Plot Codes

Plot codes will accept repeated arguments. They may also require an initial set of arguments, as in the case of Incremental plot submodes. In this list, the repeated arguments are shown in braces {}, and initial arguments (if any) precede the braces.

Plot codes always begin with the PLOT character, CTRL B, 02 hex.

```
PLOT @
PLOT A        Arc: {X, Y, radius, start, delta,}
PLOT B
PLOT C        Circle: {X, Y, radius,}
PLOT D        Dot: {X, Y,}
PLOT E        Vector-drawn characters: angle, F, {char}
PLOT F
PLOT G
PLOT H
PLOT I        Incremental Vector: X, Y, {dXdY}
PLOT J
PLOT K
PLOT L
PLOT M
PLOT N
PLOT O        Two-point Circle: {X, Y, X1, Y1,}
PLOT P        Polygon: {X, Y,} ;
PLOT Q
PLOT R        Rectangle: {X1, Y1, X2, Y2,}
PLOT S
PLOT T        Triangle: {X1, Y1, X2, Y2, X3, Y3,}
PLOT U        Curve: {X1,Y1, X2,Y2, X3,Y3, X4,Y4,}
PLOT V        Vector: {X1, Y1, X2, Y2,}
PLOT W        Concatenated Vector: X, Y, {X, Y,}
PLOT X        Incremental X-bar: X0,Y0, {X,}
PLOT Y        Incremental Y-bar: X0,Y0, {Y,}
PLOT Z
PLOT {
PLOT \        Ray: {X, Y, radius, angle,}
PLOT }
PLOT ^

PLOT `
PLOT a
PLOT b
PLOT c
PLOT d
PLOT e
PLOT f
PLOT g
PLOT h
PLOT i
PLOT j
PLOT k
```

```
PLOT l
PLOT m
PLOT n
PLOT o
PLOT p
PLOT q
PLOT r
PLOT s
PLOT t
PLOT u
PLOT v          Bold Vector: {X1, Y1, X2. Y2,}
PLOT w          Bold Concat. Vector: X, Y, {X, Y,}
PLOT x
PLOT y
PLOT z
PLOT {
PLOT |
PLOT }
PLOT ~
PLOT DEL
```

Escape Codes

Escape codes always begin with the ESC character, CTRL [, 1B hex.

```
ESC @
ESC A           Align Pan to Cursor
ESC B F         Binary Mode (ESC and USER codes)
ESC C ####      Change color (color, R, G, B)
ESC D #         Delay <#> retraces (or till keypress)
ESC E
ESC F
ESC G
ESC H
ESC I
ESC J           Zoom up
ESC K           Zoom down
ESC L
ESC M XY        Pan to X-Y
ESC N
ESC O
ESC P XY...XY;  Polygon ( >30 points)
ESC Q
ESC R
ESC S #         Plane Video Switch
ESC T ####      Tone (Hz, Hz, Hz, milliseconds)
ESC U
ESC V C F       Visible control-chars in window on/off
ESC W
ESC X
ESC Y
ESC Z CC        Absolute Zoom
ESC [
ESC \
ESC ]
ESC ^

ESC `
ESC a
ESC b #         Blink Select
ESC c ####      Change color (color, H, V, S)
ESC d
ESC e
ESC f
ESC g
ESC h           Pan left
ESC i
ESC j           Pan down
ESC k           Pan up
ESC l n         Load default color table
ESC m
ESC n
ESC o
ESC p
ESC q F         Overlay cursor blink on/off
```

ESC r	
ESC s F	Select image
ESC t	
ESC u	
ESC v #	View sub-buffer
ESC w	
ESC x	
ESC y	
ESC z	
ESC {	
ESC |	
ESC }	Pan right
ESC ~	
ESC DEL	

User Codes

User codes always begin with the USER character, CTRL U, 15 hex.

USER @	Execute Terminal Emulator
USER A	Append to Create Buffer
USER B	
USER C	Create On
USER D addr,#n	DMA Access
USER E	
USER F	Full Duplex
USER G	
USER H	Half Duplex
USER I nC	Assign Input Device
USER J C	Execute Function Key
USER K #	Kill Sub-Buffer
USER L	Local Duplex
USER M	Execute Monitor
USER N	
USER O nCCCC	Assign Output
USER P	
USER Q F	Clock display on/off
USER R	Redraw the Create Buffer
USER S ##	Set Serial baud rate: port, baud
USER T	Execute THAW routine
USER U	
USER V	
USER W	Warm-start (re-enter program)
USER X	Transmit the Create Buffer
USER Y	
USER Z	
USER [	Literal Create On
USER \ F	Enable Joystick
USER]	
USER ^ #	Insert into Sub-Buffer
USER `	Restricted Terminal Emulator
USER a	Boot Idris (normal)
USER b	Boot the system
USER c	Create Off
USER d	Execute Disk Operating System (DOS)
USER e	
USER f	
USER g	
USER h	Hardcopy
USER i	Idris Boot Routine
USER j C	Define Function Key
USER k	
USER l	
USER m	
USER n	
USER o	
USER p	Boot DOS (no password entry)
USER q nnnnnnnnn;	Set time clock

```
USER r ###nnn;          Modify SPC port


USER s #CCC             Set Serial stop bits, parity, word length
USER t
USER u
USER v ##               Change Case Table entry
USER w ##               Swap Case Table entry
USER x CC               Transmit window variable
USER y CC               Display window variable
USER z ########         Set EOL sequence
USER {
USER | CCn              Enable light pen
USER }
USER ~
USER DEL
```

Command Reference List

This section is a brief guide to the operation of the CGC 7900. Commands are listed here by name. For details on any command, consult the detailed descriptions in the main body of this manual.

Function	Code Sequence	Key	Page #
Arc	PLOT A XY #,#,#,	ARC	9-4
Area Fill	MODE ^ #, XY	AREA FILL	8-28
Assign Input Device	USER I nC	ASSIGN INPUT	3-4
Assign Output Device	USER O nCCCC	ASSIGN OUTPUT	3-2
Background Color	MODE c #,	SHIFT SET	2-10, 8-13
Boot	USER b	BOOT	1-16
Binary Mode:			
ESC and USER	ESC B F		7-9
MODE and PLOT	MODE B F		7-8
Blink	MODE K F	BLINK	2-10, 8-6
Bold Vector	PLOT v XY XY		9-23
Case Table:			
Change entry	USER v #,#,		4-20
Swap entries	USER w #,#,		4-20
Character Set:			8-10
Default	MODE i 6,8,0,		
Defined	MODE i #,#,addr,		
Character Size	MODE X #,#,	SIZE	8-8
Circle:			
Normal	PLOT C XY #,	CIRCLE	9-5
Two-Point	PLOT O XY XY		9-6
Colorset	MODE y #,#,	SHIFT COLORSWAP	8-22
Colorswap	MODE Y #,#,	COLORSWAP	8-22
Color Lookup Table:			
Change Entry (HVS)	ESC c #,#,#,#,	SHIFT CHANGE	8-20
Change Entry (RGB)	ESC C #,#,#,#,	CHANGE	8-17
Default	ESC 1 #,		8-21
Communications:			
Full Duplex	USER F		4-5
Half Duplex	USER H		4-4
Host EOL Sequence	USER z #,#,#,#,#,#,#,#,		4-17
Local	USER L		4-3
Serial Baud Rate	USER S #,#,	(port,baud,)	4-12
Serial Parity, Word Length, Stop Bits	USER s #,C C C		4-14

Function	Code Sequence	Key	Page #
Concatenated Bold Vector	PLOT w XY XY ...		9-24
Concatenated Vector	PLOT W XY XY ...	CONCAT VECTOR	9-22
Create Buffer:	USER C (ON)	CREATE	5-2
	USER c (OFF)	SHIFT CREATE	5-2
Append	USER A		5-3
Define Sub-Buffer	CTRL X		5-5
Insert in Sub-Buffer	USER ^ #,		5-6
Kill Sub-Buffer	USER K #,		5-7
Literal Create	USER [		5-9
Redraw	USER R		5-4
Transmit	USER X		5-4
View Sub-Buffer	ESC v #,	SHIFT VIEW	5-8
Cursor Color	MODE Q #,		8-25
Cursor ON/OFF	MODE J F	CURSOR ON	2-5
Curve	PLOT U XY XY XY XY	CURVE	9-7
Delay	ESC D #,		3-7
DMA Access	USER D addr,#,n		4-18
DOS:			1-6
Normal	USER d	DOS	
No Password Entry	USER p	SHIFT DOS	
Dot	PLOT D XY	DOT	9-9
Edge Fill	MODE ~ #, XY	SHIFT AREA FILL	8-29
Editing Commands:			
Clear to End of Line	MODE `	CLEAR EOL	2-5
Clear Line	MODE @	CLEAR LINE	2-4
Delete Character	MODE < 1	DEL CHAR	2-5
Delete Line	MODE < 2	DEL LINE	2-5
Erase to End of Screen	MODE 1	ERASE EOS	2-5
Erase Page	CTRL L	ERASE PAGE	2-4
Insert Character	MODE > 1	INS CHAR	2-5
Insert Line	MODE > 2	INS LINE	2-5
FILL Mode	MODE F F	FILL	8-26
Foreground Color	MODE C #,	SET	2-9, 8-12
Function Keys:			3-8
Define	USER j	DEFINE	
Execute	USER J C		
Hardcopy	USER h	SHIFT COPY	3-34

Function	Code Sequence	Key	Page #
Idris:			1-6
Normal	USER a	IDRIS	
Special Monitor	USER i	SHIFT IDRIS	
Incremental Vector	PLOT I XY #,#,...	INC VECTOR	9-10
Incremental X-Bar	PLOT X XY #,...#,	INC X-BAR	9-13
Incremental Y-Bar	PLOT Y XY #,...#,	INC Y-BAR	9-15
Intercharacter Spacing	MODE I #,#,		8-9
Joystick	USER \ F		8-30
Keyboard Sync	MODE ?		4-16
Light Pen	USER | CCn		3-13
Monitor	USER M	MONITOR	1-6, C-1
Move Cursor:			
Absolute	MODE M XY	MOVE X-Y	2-5, 8-24
Relative	MODE m XY		2-6, 8-24
Cursor Up	CTRL K	up-arrow	2-4
Cursor Down	CTRL J	down-arrow	2-4
Cursor Left	CTRL H	left-arrow	2-4
Cursor Right	CTRL]	right-arrow	2-4
Dot Up	CTRL E		8-23
Dot Down	CTRL V		8-23
Dot Left	CTRL Y		8-23
Dot Right	CTRL _		8-23
Home	CTRL \	HOME	2-4
Home Lower	MODE |	CTRL HOME	2-4
Tab	CTRL I		2-4
Overlay:	MODE O F	OVERLAY	2-3, 8-3
Cursor Blink	ESC q F		2-6
Visible Attributes	MODE V n (present)		2-11
	MODE v n (future)		2-11
Overstrike	MODE \ F		8-11
Pan:			
Absolute	ESC M XY	PAN X-Y	8-33
Pan Left	ESC h	M2 left-arrow	8-31
Pan Right	ESC }	M2 right-arrow	8-31
Pan Up	ESC k	M2 up-arrow	8-31
Pan Down	ESC j	M2 down-arrow	8-31
Reset Pan	ESC M 0,0,	M2 HOME	8-31
Align Pan to Cursor	ESC A	M2 CLEAR LINE	8-32

Function	Code Sequence	Key	Page #
Patterned Vectors, Clip	MODE T nn		10-8
Plane Select	MODE : #,		11-4
Plane Video Switch	ESC S #,		11-5
Plot Mode ON/OFF	MODE P F	PLOT	9-28
Polygon:			
Large (>30 points)	ESC P XY...XY;		9-17
Normal	PLOT P XY...XY;	POLYGON	9-16
Raster Processor:			
Define Source Raster	MODE [XY XY		10-2
Copy Raster	MODE U XY	COPY	10-3
Copy with Overstrike	MODE u XY		10-4
Set Raster Direction	MODE { #,		10-5
Ray	PLOT \ XY #,#,	RAY	9-18
Recall:			
Forward	MODE DEL 1	RECALL	C-3
Backward	MODE DEL 2	SHIFT RECALL	C-3
Reset	RESET CTRL SHIFT RESET CTRL SHIFT M1 M2 RESET		1-17
Real-Time Clock:			
Display Time	USER Q F		3-16
Set Time	USER q nnnnnnnnn;		3-15
Rectangle	PLOT R XY XY	RECT	9-19
Roll	MODE R F	ROLL	2-7, 8-36
Rubber Band	MODE E F	RUBBER BAND	8-37
Scaling ON/OFF	MODE S F		3-31
Scale Factors	MODE s #,#,#,#,	SCALE	3-27
Serial Port Controller	USER r #,#,#,nnn;		4-16
Soft Boot	MODE =	SOFT BOOT	1-17
Select Blink Plane(s)	ESC b #,		8-7
Select Character Set:			
Alternate	CTRL N		3-6
Standard	CTRL O		3-6

Function	Code Sequence	Key	Page #
Select Image	ESC s F		11-1
Termem:			
Normal	USER @	TERMINAL	4-1
Restricted	USER `		4-2
Test	MODE t C	TEST	3-17
Thaw Parameters	USER T	THAW	6-1
Tone Generator	ESC T #,#,#,#,	TONE	3-18
Triangle	PLOT T XY XY XY	TRIANGLE	9-20
Vector	PLOT V XY XY	VECTOR	9-21
Vector-Drawn Characters	PLOT E #, F chars		9-26
Vector Width	MODE x #,	V. WIDTH	9-25
Visible Control Characters	ESC V C F		3-20
Warm Start	USER W		3-22
Window Limits	MODE W XY XY	WINDOW	3-24
Window Variables:			
Display	USER y C C		7-5
Operate	MODE] C C #,		7-4
Transmit	USER x C C		7-5
Zoom:			
Absolute	ESC Z C C		8-34
Zoom Down	ESC K	M2 RECALL	8-31
Zoom Up	ESC J	M2 ERASE PAGE	8-31

Key Sequences

All the keys in the top four rows of the 7900 keyboard generate multiple codes when struck (except RESET and CALC MODE). The keys on the cursor keypad (between the alphanumeric keyboard and the numeric keypad) also generate one or more codes when struck. The following list shows which keystrokes produce sequences of characters. Function key sequences are also described on page 3-12.

Keystroke	Sequence
ARC	PLOT A
AREA FILL	MODE ^
ASSIGN INPUT	USER I
ASSIGN OUTPUT	USER O
BLINK	MODE K 1
BOOT	USER b
CHANGE	ESC C
CIRCLE	PLOT C
COLORSWAP	MODE Y
CONCAT VECTOR	PLOT W
COPY	MODE U
CREATE	USER C
CURSOR ON	MODE J 1
CURVE	PLOT U
DEFINE	USER j
DOS	USER d
DOT	PLOT D
FILL	MODE F 1
IDRIS	USER a
INC VECTOR	PLOT I
INC X-BAR	PLOT X
INC Y-BAR	PLOT Y
MONITOR	USER M
MOVE X-Y	MODE M
OVERLAY	MODE O 1 MODE V 3
PAN X-Y	ESC M
PLOT	MODE P 1
POLYGON	PLOT P
RAY	PLOT \
RECT	PLOT R
ROLL	MODE R 1
RUBBER BAND	MODE E 1

Keystroke	Sequence
SCALE	MODE s
SET	MODE C
SHIFT AREA FILL	MODE ~
SHIFT BLINK	MODE K Ø
SHIFT CHANGE	ESC c
SHIFT COLORSWAP	MODE y
SHIFT COPY	USER h
SHIFT CREATE	USER c
SHIFT CURSOR ON	MODE J Ø
SHIFT DOS	USER p
SHIFT FILL	MODE F Ø
SHIFT IDRIS	USER i
SHIFT OVERLAY	MODE O Ø MODE V Ø
SHIFT PLOT	MODE P Ø
SHIFT ROLL	MODE R Ø
SHIFT RUBBER BAND	MODE E Ø
SHIFT SCALE	MODE S Ø
SHIFT SET	MODE c
SHIFT WINDOW	MODE W,,-1.-1.
SIZE	MODE X
SOFT BOOT	MODE = CTRL L
TERMINAL	USER @
TEST	MODE t
THAW	USER T
TONE	ESC T
TRIANGLE	PLOT T
V. WIDTH	MODE x
VECTOR	PLOT V
VIEW	ESC v
WINDOW	MODE W

Cursor Keypad codes

down-arrow	CTRL J
left-arrow	CTRL H
right-arrow	CTRL]
up-arrow	CTRL K
SHIFT down-arrow	CTRL V
SHIFT left-arrow	CTRL Y
SHIFT right-arrow	CTRL _
SHIFT up-arrow	CTRL E
CLEAR EOL	MODE `
CLEAR LINE	MODE @
CTRL HOME	MODE |
DEL CHAR	MODE < 1
DEL LINE	MODE < 2
ERASE EOS	MODE 1
ERASE PAGE	CTRL L
HOME	CTRL \
INS CHAR	MODE > 1
INS LINE	MODE > 2
RECALL	MODE DEL 1
SHIFT RECALL	MODE DEL 2
M2 CLEAR LINE	ESC A
M2 ERASE PAGE	ESC J
M2 HOME	ESC M Ø,Ø,
M2 RECALL	ESC K
M2 down-arrow	ESC j
M2 left-arrow	ESC h
M2 right-arrow	ESC }
M2 up-arrow	ESC k

Function Key codes

F1	USER J A	F13	USER J M
F2	USER J B	F13	USER J N
F3	USER J C	F13	USER J O
F4	USER J D	F13	USER J P
F5	USER J E	F13	USER J Q
F6	USER J F	F13	USER J R
F7	USER J G	F13	USER J S
F8	USER J H	F13	USER J T
F9	USER J I	F13	USER J U
F1Ø	USER J J	F13	USER J V
F11	USER J K	F13	USER J W
F12	USER J L	F13	USER J X
Bezel 1	USER J _	Bezel 5	USER J [
Bezel 2	USER J `	Bezel 6	USER J \
Bezel 3	USER J]	Bezel 7	USER J Y
Bezel 4	USER J ^	Bezel 8	USER J Z

Color Key codes

BLACK	0,	SHIFT BLACK	8,
BLUE	1,	SHIFT BLUE	9,
GREEN	2,	SHIFT GREEN	10,
CYAN	3,	SHIFT CYAN	11,
RED	4,	SHIFT RED	12,
MAGENTA	5,	SHIFT MAGENTA	13,
YELLOW	6,	SHIFT YELLOW	14,
WHITE	7,	SHIFT WHITE	15,

Inline Editor Commands

To date, the Inline Editor is used in the following Chromatics software:

- DOS (including Editor)
- CGC 7900 Monitor
- Idris Editor "ne"

Command	Key Sequence
Clear Line	CLEAR LINE
Clear to Beginning of Line	CTRL SHIFT CLEAR LINE
Clear to End of Line	CTRL CLEAR EOL
Delete Character	CTRL DEL CHAR
Delete Word	CTRL SHIFT ERASE PAGE
Insert Characters	CTRL INS CHAR
Literal Mode	CTRL G
Move Cursor	<left arrow> <right arrow>
Move to Beginning of Line	HOME
Recall Back	SHIFT RECALL
Recall Forward	RECALL

Appendix B -- Color Display Schemes

The human eye is a remarkable instrument for distinguishing color. It can adjust to an enormous variety of lighting conditions and intensities, and interpret extremely subtle shades of color. No present method of generating a color graphics display can produce the entire range, or "gamut," of colors the eye is able to interpret. All present methods must employ some sort of model, representing a certain subset of the eye's color gamut.

The way in which most color graphics systems (including the CGC 7900) produce color is to provide three color "guns" in a Cathode-Ray Tube, or CRT. The three guns each produce a primary color. One gun is assigned to red, one to green, and one to blue. If the intensities of each gun can be independently varied, a good color gamut can be represented.

The 7900 allows each gun to vary from "full off" to "full on" in 256 discrete steps. Studies have shown that this 8-bit resolution in the intensity scale is necessary to produce the appearance of continuous shading. A change in intensity of one part in 256 is not detectable by the eye.

The Color Cube

Since we use three guns in this system, one for each of three primary colors, we can represent this system by a set of three-dimensional axes. Assume the X-axis represents red, Y represents green, and Z represents blue. If we allow the intensity of each gun to vary between zero (completely dark) to 255 (maximum brightness), then we must label each axis with numbers from 0 to 255. (The limit 255 is, for now, an arbitrary limit. We could just as well have defined limits to be from zero to one, or any other pair of numbers.)

The origin of this three-dimensional space is at the point 0, 0, 0, which is black (no intensity in any gun). The point of maximum intensity in all three guns is labelled 255, 255, 255, and is white. A point along any of the three axes will represent a color with only one component: for example, any point along the "red" axis will have a component of red, but no green or blue. Greys have equal intensity of all three guns, so all grey levels fall along a diagonal line between 0, 0, 0 and 255, 255, 255.

We restrict each axis of this system to numbers between 0 and 255. In doing so, we define a "box" or "cube"-shaped area. Any possible color in the gamut of our CRT system may be represented by some point within this cube. Any points outside the cube represent colors which might exist in the eye's color gamut, but cannot be represented on the CRT.

What we have just described is known as the **Color Cube** model, or RGB model. A set of three-dimensional axes may be used to represent R, G, and B, and any color displayable on the CRT may be described as a point in the space defined by these axes. The upper limit of intensity must be defined, of course; in the 7900, the upper limit is 255 units of intensity along each axis.

The RGB model is useful. It accurately describes the gamut of colors we have available on the 7900 (or any other CRT device). Its drawback is that it is not easily related to the colors we see in the real world.

The HVS Hexcone

A second model has been developed that accurately describes the CRT color gamut. The Hexcone is defined in units which are more appealing for humans to work with: **Hue**, **Value** and **Saturation.**

Hue is the quality we most often refer to when we talk about "color." Hue defines the dominant wavelength of a color: red, blue, green, or some combination. A spectrum seen emerging from a prism displays a stream of colors, changing in Hue only.

Value is the quality we refer to as "brightness" or "energy" in a color. When Value approaches zero, the color becomes black and Hue no longer has meaning. As Value increases, the subjective brightness of the color increases. A decrease in Value may be thought of as an increase in the amount of "black" in a color.

Saturation is the quality that distinguishes a weak, or pastel color, from a strong, vibrant one. (For example, pink is the name we give a color whose main component is red, but which has low Saturation.) If Saturation approaches zero, the color becomes grey. (How bright a grey? Holding Saturation at zero, if Value changes from zero to maximum, the color changes from black to white.) When Saturation is zero, Hue has no meaning. Saturation is always maximum at the primary colors, red, green, and blue.

The Hexcone model shows many of the features we have described above. Every point on the Hexcone, or inside it, can be found in the gamut of a color CRT. Thus, the Hexcone is simply an alternate mapping of the same information contained in the Color Cube. And, in fact, there are mathematical functions which can translate RGB coordinates to HVS and vice versa.

The top of the Hexcone has six corners: the three primary colors, red, green, and blue, and the three secondary colors formed by adding pairs of primaries: magenta, yellow, and cyan. White is located at the top of the center plane. Black is at the bottom vertex of the Hexcone. It follows that all shades of grey lie on the vertical axis which passes through the center of the Hexcone.

Hue is seen to be an angle; it is always measured by drawing a vector from the center axis of the Hexcone. Hue equals zero when this vector points to the red vertex, increases toward green, and increases still more toward blue. Hue reaches its maximum when it returns to its starting point, red.

Value is vertical height on the Hexcone. Value is zero at the black vertex, and reaches its maximum at any point on the top plane of the Hexcone.

Saturation is measured by drawing a vector from the center axis of the Hexcone, through the point at which Saturation is to measured, to the edge of the Hexcone. (Note that the total length of this vector will vary, depending on the cross-sectional width of the Hexcone at the point of interest.) The total length of this vector always represents a Saturation of 100%, regardless of its actual length; and the Saturation at any point P along the vector is defined to be the distance from P to the center axis, divided by the total length of the vector to the Hexcone edge. Saturation is thus a relative measure, which interacts with Hue and Value.

Until now, we have carefully avoided using any absolute units for Hue, Value and Saturation. Units of measure for color have not been fully standardized; some systems define all RGB and HVS units to fall between zero and one. In the CGC 7900 we prefer to manipulate only integer numbers, so we use a range of zero to 255 for all numbers in the Hue, Value, Saturation system.

Using this range (0 to 255), here are some examples of colors defined in both RGB and HVS units:

Color	R	G	B	H	V	S
Red	255	0	0	0	255	255
Green	0	255	0	85	255	255
Blue	0	0	255	171	255	255
Yellow	255	255	0	43	255	255
Cyan	0	255	255	128	255	255
Magenta	255	0	255	213	255	255
White	255	255	255	*	255	0
50% grey	128	128	128	*	128	0
Black	0	0	0	*	0	*

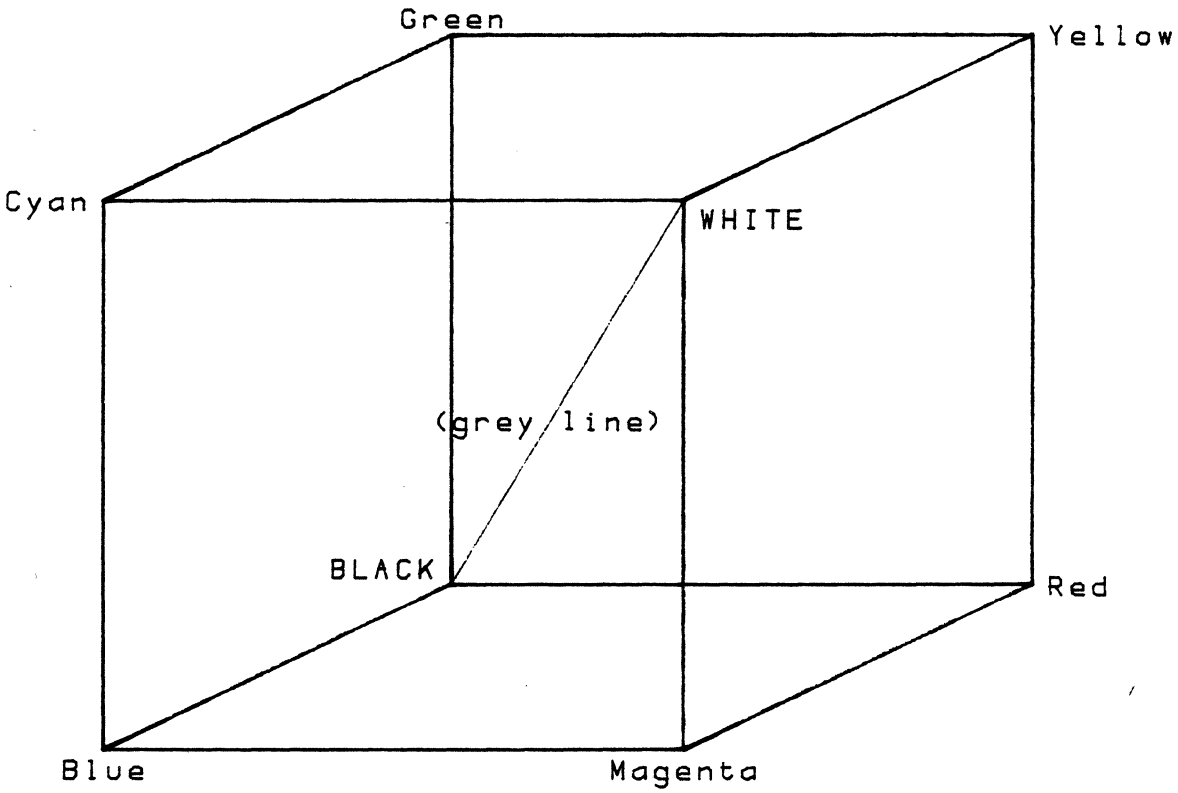
* = undefined

Red	255	0	0	0	255	255
Pink	255	128	128	0	255	128
Light Orange	255	190	128	21	255	128
Brown	144	95	64	21	128	128

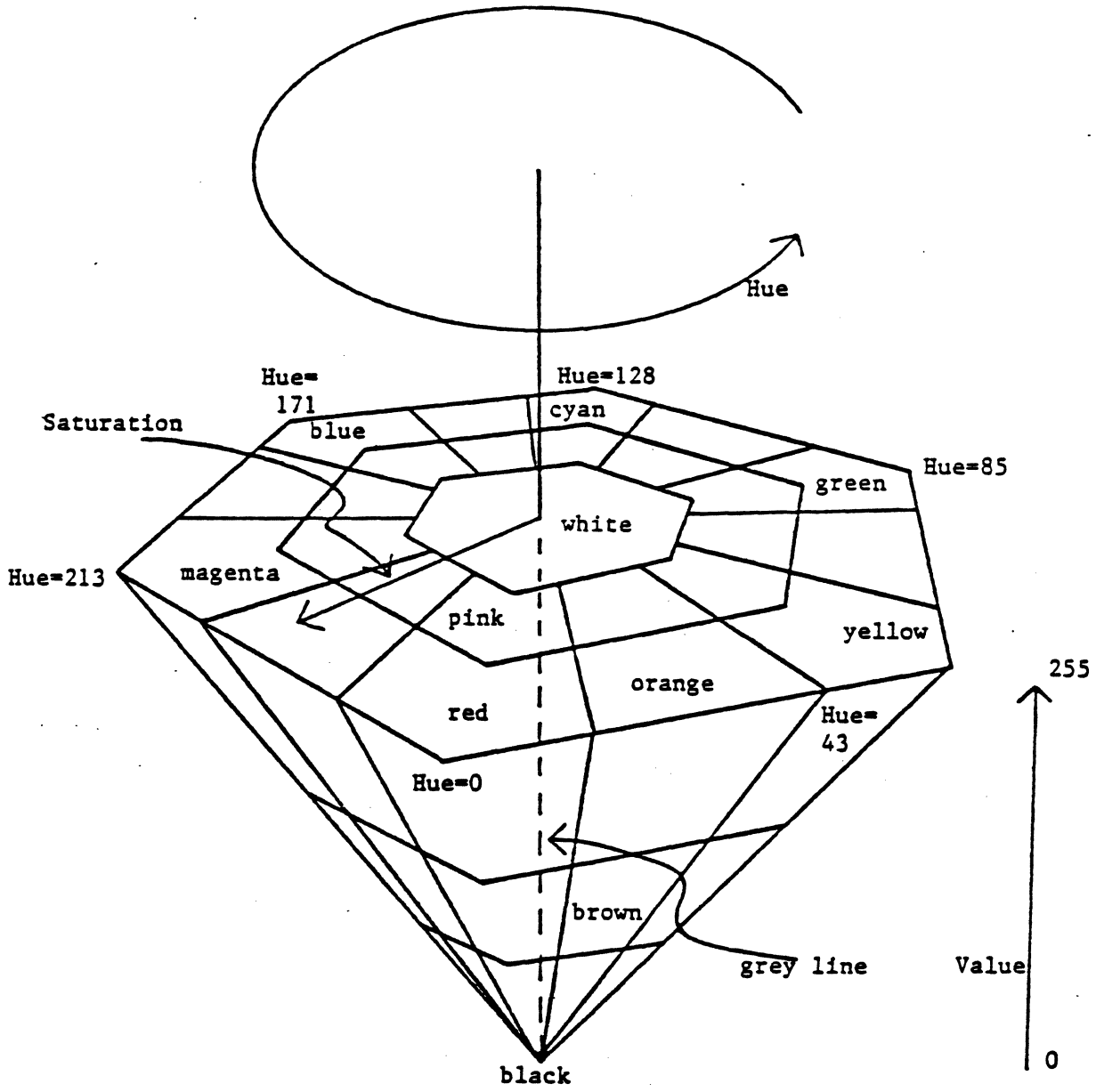
Changing from red to pink requires a change in both the green and blue components, but only changes Saturation in the HVS system. Similarly, the change from light orange to brown involves altering all three of the three RGB components, but only changes one of the HVS components (Value). These types of color changes would be relatively difficult to calculate if we were forced to use RGB, but they are easy to comprehend in HVS.

Despite the lengthy definitions above, HVS units are generally very well-behaved and easy for an operator to work with. The CGC 7900 allows the user to vary a color by adjusting Hue, Value and Saturation. Users quickly become accustomed to using this method of color selection, much as they would mix paints by eye: select a Hue by mixing primary colors, then add white (reduce Saturation) to lighten or mute the color, or add black (reduce Value) to darken it.

The Color Cube



The HVS Hexcone



Appendix C -- The Monitor

The Monitor is a program which provides primitive facilities for manipulating memory, registers, and input/output devices within the 7900. Its main use is debugging machine language programs.

It is not necessary to understand this section before learning to use the rest of the CGC 7900 system.

The Monitor is not designed to be used by beginners! Properly used, it can be a great aid for programmers working in 68000 code, whether they are experienced with this particular processor or not. But the user who is totally unfamiliar with assembly level programming may find the Monitor quite baffling.

This section describes the utilities available in the Monitor. To enter the Monitor, press the labelled key:

MONITOR

Monitor Operations

Some general comments are presented here concerning the commands accepted by the Monitor.

Most commands consist of one or two characters, followed by the arguments to the command. Each command is terminated by the carriage return character (RETURN). The arguments to the command are separated by delimiters.

The characters space, comma, and colon are all recognized as delimiters. They may be used interchangeably in the commands except where noted. In the format for each command we will use the comma for convenience.

When the Monitor is displaying information, such as a list of the contents of memory, the display may be stopped by entering CTRL S (hold down the CTRL modifier and type an S). Pressing another key will continue the display. The display may be interrupted at any time with the DELETE key, which will return you immediately to the Monitor prompt.

The Inline Editor

When entering a command line to the Monitor, the text editing functions labelled on the cursor keypad may be used to edit an input line. These functions include Insert and Delete Character, Cursor Left and Right, Clear Line, Clear EOL, and Recall Last Line. (The functions labelled in blue are accessed by holding the CTRL modifier while pressing the indicated key.)

The left and right arrows move the cursor around on the input line. The HOME key moves the cursor to the first character of the input line.

CLEAR LINE deletes the entire input line. CLEAR EOL (Clear to End Of Line) deletes all characters from the cursor position to the end of the input line. DEL CHAR (Delete Character) deletes one character at the current cursor position.

INS CHAR (Insert Character) begins the Insert Character input mode. After entering Insert Character, the character under the cursor begins flashing. Any characters typed in this mode will be inserted into the input line at the current cursor position. To leave this mode, use the left or right arrow key to move the cursor. Now, any characters entered will overwrite the character underneath the cursor. (This is the default input mode.)

CTRL SHIFT CLEAR LINE deletes all characters from the cursor to the beginning of the line. CTRL SHIFT ERASE PAGE deletes a word from the input line. These two functions are not labelled on the cursor keys.

The Recall function brings back a copy of a line previously entered. This is useful for repeating a command several times, without having to retype the entire command. Press RECALL to bring back a copy of the last line entered. Press the key several times to bring back earlier lines, moving backward in the Recall buffer. If you go too far back, press SHIFT RECALL to move forward in the buffer. The number of lines in the buffer is limited by the size of the buffer and the length of the lines.

When the RETURN key is pressed, the entire visible input line is entered as a command to the Monitor, regardless of the current cursor position within that line. This means that it is not sufficient to "backspace" over characters to delete them: the unwanted characters must be physically deleted from the line with the Delete Character, Clear Line, or Clear EOL functions.

If you need to execute special code sequences while in the Monitor, simply enter the desired sequence. Escape and User code sequences will be immediately executed. Mode and

Plot codes will be displayed on the input line (using special characters), and will be executed when the RETURN key is pressed. This allows you to enter a Mode code sequence, edit it using Inline, then execute it. Similarly, if you wish to execute an Erase Page or other special control function, type it in and it will appear using special characters. It may be edited in the standard manner, and when RETURN is pressed, it will be executed.

Pressing CTRL G (the "bell" code) causes Inline to enter Literal mode. In Literal mode, ALL control-characters may be entered into the Inline editor, including the carriage return code. Thus, once Literal mode is in effect, you are unable to press RETURN and have the system execute your input line. Literal mode is designed for text editors and other advanced programs, and it is not normally used in the Monitor. If you do enter Literal mode, typing a second CTRL G will exit this mode.

Monitor Commands

Abort

If a machine language program is executed using the Monitor, it may be aborted at any time by pressing any Bezel Key. All registers are displayed, and control returns to the Monitor.

NOTE:

After entering the Monitor, the Bezel Keys will ONLY perform this Abort function. Re-entering the Terminal Emulator and trying to define a Bezel Key will result in aborting the Terminal Emulator program! If the Bezel Keys are to be used as user-defined keys after leaving the Monitor, the system must be Booted.

Change Memory

Format:

CB <addr> RETURN

CW <addr> RETURN

CL <addr> RETURN

The Change command allows you to modify memory, starting at address <addr>. You may modify bytes, words, or long words, using the CB, CW or CL form of the command.

Change displays an address, and the contents of memory at that address, then asks for input. You have four options:

- Enter a hex number and press RETURN, to enter a value into that address and examine the next address.
- Press RETURN, to skip that address without modification.
- Enter a carat (^) and press RETURN, to examine the previous address.
- Press DELETE, to terminate the Change command and return to the Monitor prompt.

Checksum Memory

Format:

+ <addr1>, <addr2> RETURN

Where:

<addr1> and <addr2> are hex numbers delimiting the memory range to be summed.

The memory locations between <addr1> and <addr2>, inclusive, are summed, and the result displayed. The sum is a sixteen-bit number, obtained by summing either all of the EVEN bytes, or all of the ODD bytes, in the memory range specified. If <addr1> is even, the even bytes are summed. If not, the odd bytes are summed.

Example:

+4000,5FFE RETURN sum all EVEN bytes 4000 - 5FFE

Compare Memory

Format:

K <addr1>, <addr2>, <addr3> RETURN

Where:

<addr1>, <addr2>, <addr3> are hex numbers specifying the memory ranges to be compared.

The contents of memory between locations <addr1> and <addr2>, inclusive, are compared to consecutive locations beginning at <addr3>. Any locations which do not match are displayed. During this display, you may press CTRL S to pause, or DELETE to stop.

Example:

K4000,4FFF,5000 RETURN

The contents of memory locations 4000 through 4FFF (hex) are compared to locations 5000 through 5FFF.

Disk Read

Format:

R <disk#>, <block>, <#blks> <addr> RETURN

Where:

<disk#> is the disk you want to read from. Values of 0 and 1 refer to the left and right floppy drives, respectively. Values of 2 and 3 refer to the internal and remote hard disks.

<block> is the first block to read from. Legal values depend on particulars of each disk.

<#blks> is the number of blocks to read. Each block is 256 bytes long, except blocks on track 0 of the floppy disks. These blocks are only 128 bytes long.

<addr> is the starting address of the block of memory receiving the disk data.

The contents of the specified disk blocks are read into memory starting at **<addr>**.

Example:

R0,100,2,10000 RETURN

Blocks 100 and 101 of the left-hand floppy disk are read into memory starting at location $10000.

Disk Write

Format:

W <disk#>, <block>, <#blks> <addr> RETURN

The parameters for Disk Write are the same as those for Disk Read, above. The contents of memory starting at **<addr>** are written to the specified disk blocks.

Example:

W0,100,2,10000 RETURN

The memory block starting at location $10000 is written to blocks 100 and 101 of the left-hand floppy disk.

Dump Memory

Format:

D <addr1> [,<addr2>] RETURN

Where:

<addr1> and <addr2> are hexadecimal numbers specifying the range of memory to be dumped.

The Dump command outputs the contents of memory to the screen. The information is displayed in hexadecimal and ASCII. Sixteen bytes are displayed on each line of the screen.

<addr2> is optional. If included, it must be separated from <addr1> by a delimiter, such as the comma shown above. As mentioned earlier, a space or a colon are also valid delimiters.

If <addr2> is greater than <addr1>, then the range of memory will end at location <addr2>. Since sixteen bytes are always displayed on each line, a few bytes past <addr2> may also be displayed.

If <addr2> is less than <addr1>, then <addr2> is taken as the number of bytes to display, rather than the last address to display.

Examples:

D4000 RETURN

dumps sixteen bytes, beginning at address 4000 (hex).

D4000.4FFF RETURN

dumps all bytes between addresses 4000 and 4FFF, inclusive.

D4000.20 RETURN

dumps 32 (20 hex) bytes, beginning at address 4000 hex.

Evaluate Math Expression

Format:

? <integer expression> RETURN

This command parses an integer math expression and returns the result in decimal and hexadecimal. All numbers in the expression are assumed to be in decimal, unless they are preceded by the dollar-sign character ($), in which case they are interpreted as hex.

Examples:

?10 * (147/3 -5) + 19 RETURN

?1900 - $FE06 RETURN

?$14A60 & $1FFF RETURN

?(32+756) ! 1024 RETURN

?'HI' + 'ABC' RETURN

NOTE:

In the examples above, spaces have been included for clarity. Spaces must NOT be entered when typing expressions into the Monitor, since a space is a delimiter, and evaluation will terminate when the space is reached.

In the third example, the Logical And operator (&) is used. This operator performs a 32-bit logical AND. The fourth example contains the 32-bit logical OR operator (!).

The last example shows that strings may be used as operands. Up to four characters may be used in a string constant, since the maximum number size in this system is 32 bits.

NOTE:

All decimal numbers, as well as any multiplies and divides, must be limited to 16-bit accuracy.

Examine Registers

Format:

X [<Reg>] [,<value>] RETURN

Where:

<Reg> is the name of a register, or a question mark

<value> is a hex number, up to 32 bits

This command allows you to examine or modify the contents of the 68000 processor registers. These are the names of the registers:

D0 Data register 0
D1 Data register 1
.
.
D7 Data register 7

A0 Address register 0
A1 Address register 1
.
.
A7 Address register 7

PC Program counter
US User stack pointer
SR Status Register

If the command X is used with no arguments, the values of all registers are displayed.

If the name of a register is given, only the value of that register is displayed.

If the name of a register is given, and a value is specified, the register is loaded with that value.

If the question mark is used instead of a register name, a list is printed of the registers designated for Trace (see Trace Display).

NOTE:

These registers are "pseudo-registers" used by the Monitor during trace and breakpoint operation. The values in these registers are not initialized unless a breakpoint is reached, or unless specifically loaded using the X command.

Examples:

X RETURN	displays all registers
XD1 RETURN	displays the value of D1
XD1,3E4F RETURN	loads D1 with hex value 3E4F
X? RETURN	displays registers designated for trace

Fill Memory

Format:

F <addr1>, <addr2>, <value> RETURN

Where:

<addr1> and <addr2> are hex numbers specifying the memory range to fill;

<value> is the hex number (8 bits or less) to fill with.

Memory locations between addresses <addr1> and <addr2>, inclusive, are filled with <value>.

Example:

F4000,4FFF,A5 RETURN

This example fills locations 4000 through 4FFF (hex) with the value A5 (hex).

Go (with Breakpoints)

Format:

G [<addr>] [,<bkpl>] [,<bkp2>] RETURN

Where:

<addr> are each 32-bit hex addresses
<bkpl>
<bkp2>

Go causes execution to begin at address <addr>. If <addr> is omitted, execution begins at the current value of PC.

Up to two breakpoint addresses may be specified, and they must be separated by delimiters (commas, spaces, or colons). Execution will continue until any one of the breakpoints is hit. At that time, the registers designated by Trace Display will be listed, and control is returned to the Monitor.

When either breakpoint is encountered, BOTH breakpoints are removed from memory.

If no breakpoint addresses are specified, the program will not return to the Monitor (unconditional Go).

Examples:

G45Ø2 RETURN	Go at address 45Ø2 hex.
GE6A2,E6BØ RETURN	Go at address E6A2, with a breakpoint at E6BØ.
G,16FFØ RETURN	Go at current PC, with a breakpoint at 16FFØ.
G RETURN	Unconditional Go.

Load

Format:

L [<optional offset>] RETURN

The Load command accepts input from Logical Input Device 1, normally the RS-232 serial port. The input is expected to be Motorola 68000 object code in the standard format. See the 68000 processor literature for format details.

The code is loaded at its normal address in memory, unless an offset address is specified in the load command. If specified, this offset is added to the memory addresses.

Examples:

L RETURN

L1000 RETURN

Move Memory

Format:

M <addr1>, <addr2>, <addr3> RETURN

Where:

<addr1>, <addr2>, <addr3> are hex numbers specifying memory addresses.

The contents of memory between locations <addr1> and <addr2>, inclusive, are copied to consecutive locations beginning at <addr3>.

Example:

M4000,4FFF,5000 RETURN

The contents of memory locations 4000 through 4FFF (hex) are copied to locations 5000 through 5FFF.

Punch

Format:

P <addr1>, <addr2> RETURN

The Punch command transmits output to Logical Output Device 1, normally the RS-232 serial port. The output is in Motorola 68000 object code format. It contains all bytes between addresses <addr1> and <addr2>, inclusive.

Example:

P1000,2FFF RETURN

NOTE:

The Punch command does NOT produce the standard end-of-record marker on its output. See the End Punch command below.

End Punch

Format:

E RETURN

The Punch command produces output records in standard Motorola 68000 object code format. The Punch command does not produce the required end-of-record mark. To properly close out the object code, use the End Punch command.

Example:

E RETURN

This command is provided separately so that you may Punch several different portions of memory to your output device, keeping them all as part of a single file, then terminate the entire set with one end-of-record using End Punch. The resulting object code can all be read in with one Load command, even though parts of it may be loaded into non-contiguous areas of memory. The Load command continues to read until it reads the end-of-record provided by End Punch.

Set Memory

Format:

SB <addr>, <string of values> RETURN

SW <addr>, <string of values> RETURN

SL <addr>, <string of values> RETURN

Where:

<addr> is the first location to set

<string of values> is a list of hex values separated by commas, or an ASCII string in single quotes (SB command only)

Use the SB command to set bytes, SW for words (16 bits), and SL for long words (32 bits). You can set consecutive locations by entering several values. The only limit is that the entire command must fit on one line of the screen (85 characters).

The string of values may be an ASCII character string, when using the SB command. The SW and SL commands will only accept hex arguments, separated by commas.

Examples:

SB4000,1E RETURN

SW4010,12AE,1B67,FF,9C00 RETURN

SL6F00,E34F01,0,2E,A5A5A5 RETURN

SB5021,'This string goes into memory.' RETURN

The apostrophe character (single quote) is used to delimit an ASCII string. It is possible to insert a single quote into memory by entering two of them:

SB4900,'It''s easy to do that!' RETURN

A single apostrophe will be inserted between the t and s.

Trace

Format:

T [<addr>] RETURN

The Trace command executes one instruction in the program. The current instruction is pointed to by the register PC (program counter). After executing one instruction, the designated registers are displayed.

Before using Trace, it is necessary to set PC using the "X" command (unless PC already contains a value, from a previous command).

If <addr> is specified, tracing continues until that address is reached. The registers are printed after each instruction executes. The process may be interrupted at any time by pressing DELETE or by pressing any Bezel Key. Pressing CTRL S will pause the display, and pressing another key will resume tracing.

Examples:

T RETURN

T41E2 RETURN

After using the Trace command, the next Monitor prompt you receive will be "TM" to indicate that you are still in trace mode. When you receive the "TM" prompt, you may press the RETURN key to execute a single instruction. Entering any other command will remove you from trace mode, and the "TM" prompt will no longer be presented.

See also Trace Display.

Trace Display

Format:

[<list of registers> RETURN

[ ] RETURN

[Ø RETURN

Trace Display defines which registers will be displayed during Trace or Breakpoint execution.

Each register in the <list of registers> must be in the following form:

Reg . Length

Where Reg is the name of a register, and Length is the type of display desired: B for byte, W for word, L for long word.

The second format given above causes a listing of all registers to be printed at each break in execution. The third format turns off all register listing.

Examples:

[DØ.B,A1.L,A2.L RETURN

[PC.L,D1.W RETURN

[] RETURN (display all registers)

[Ø RETURN (display none)

A list of the registers currently specified for tracing may be obtained with the command:

X? RETURN

Virtual Search

Format:

V <addr1>, <addr2>, <string of values> RETURN

Where:

<addr1> is the start of the memory range to search
<addr2> is the end of the range
<string of values> is what to look for (hex or ASCII)

The range of memory between <addr1> and <addr2>, inclusive, is searched for the string. If a match is found, the locations containing the match are displayed.

Examples:

V0,1000,FF RETURN

V4000,8000,0,0,0,0 RETURN

V0,FFFF,'Find me' RETURN

In the first example, a search is performed for all occurrences of the value FF hex between addresses 0 and 1000 hex. In the second example, the range 4000 hex to 8000 hex is searched for four consecutive zero bytes. In the third example, the command searches the entire first 64K of memory (addresses 0 to FFFF) for the ASCII string "Find me." As with the S (Set Memory) command, an apostrophe may be entered in the ASCII string by typing two apostrophes.

The Virtual Search command may be used to determine the revision level of the PROM software in your 7900 system. Execute the Monitor, and type the following command:

V800000 80FFFF 'VER#'

This command searches addresses $800000 through $80FFFF (all PROMs) for the character string 'VER#'. PROM software versions 1.1' and later contain this character string, followed by the version number, for each major program in the system. In version 1.4, the following would be displayed:

VER# TERMEM 1.4
VER# Monitor 1.3

This indicates that the system contains version 1.4 of TERMEM (the Terminal Emulator) and version 1.3 of the Monitor. These are the latest versions.

Appendix D -- Traps

The MC68000 processor used in the CGC 7900 will detect a number of error conditions which may occur during a program. If a severe error occurs, the processor discontinues normal program execution and "traps." This causes the 7900 software to display information relating to the trap, and the system halts. A "crash" sound indicates that the program has crashed. The red indicator above the cursor keypad is illuminated.

Traps should not be encountered during normal operation. Users running preliminary or un-debugged software are more likely to encounter traps.

When the system traps, the bottom line of the Overlay displays the type of trap, the address at which the trap occurred, and seven words of data from the stack. These data may be helpful in determining the reason for the trap. When contacting Chromatics for assistance regarding a trap problem, please include all of the data displayed at the time the trap occurred.

After a trap, the system stops and does not acknowledge interrupts. The only way to recover from a trap is to press RESET. Pressing RESET turns off the red indicator on the keyboard; however, the trap message remains on the screen until erased. Pressing RESET followed by SOFT BOOT will clear the screen and recover from most traps.

If the program damaged any system data before trapping, it may be necessary to force a power-up Boot by pressing CTRL SHIFT RESET.

Traps are explained in more detail in the Motorola MC68000 User's Manual (available from Chromatics).

Traps are referenced by a number or a letter:

Trap type	Explanation
0	Bus Error (non-existent memory was accessed).
1	Address Error (attempt to fetch word data from an odd memory address).
2	Illegal Instruction.
3	Division By Zero.
4	CHK Instruction.
5	TRAPV Instruction.
6	Privilege Violation (attempt to execute privileged instruction while in User state).
7	Trace.
8	Line 1010 Emulator (illegal instruction).
9	Line 1111 Emulator (illegal instruction).
A	Spurious Interrupt.
B	Level 1 Interrupt Autovector.
C	Level 2 Interrupt Autovector.
D	Level 3 Interrupt Autovector.
E	Level 4 Interrupt Autovector.
F	Level 5 Interrupt Autovector.
G	Level 6 Interrupt Autovector.
H	Level 7 Interrupt Autovector (power-up interrupt).
P	Buffer Memory Parity Error.
*	Undefined Trap.

Appendix E -- Custom Modules, Cursors and Character Sets

This section discusses how to add custom features to the CGC 7900: modules, cursors and character sets.

Modules are commands which are loaded into memory (RAM or EPROM). When the system is booted, all modules are "linked" together in a fashion which allows any of them to be executed by ASCII code sequences. All of the features implemented in the 7900 are written as modules: this includes I/O drivers, Plot submodes, Mode, Escape and User codes. Using the information in this section, the user can write modules which perform custom functions, and link them into the CGC 7900 system.

Modules -- General

A module is a sub-program, written according to a list of guidelines so that it may be linked into the CGC 7900 firmware. There are seven types of modules:

B: Boot only. This module contains code which is executed at boot time, but not otherwise used by the system.

I: Input device. This is a driver which interfaces a physical input device to the system. Currently defined "I" modules include the keyboard and serial ports.

O: Output device. This is a driver which interfaces a physical output device to the system. Currently defined "O" modules include the serial ports, windows, and keyboard lights.

Mode: Mode code. This module performs functions which modify the attributes of a window. The window software calls Mode modules when it receives a Mode code sequence.

Plot: Plot submode. This module describes a Plot submode, and is called when a window receives a Plot code sequence.

Escape: Escape code. This module is called by the Escape code processor when an Escape code sequence is received. Escape codes generally alter the status of the entire system.

User: User code. This module is called by the Escape code processor when a User code sequence is received. User codes generally alter the status of the entire system, or cause execution of a controlling program (such as DOS or Idris).

Modules may exist in EPROM or in RAM. Most of the EPROM firmware in the 7900 consists of modules. For each Mode, Plot, Escape and User code sequence recognized by the 7900, there is a module in firmware which defines the actions taken by that code sequence. All modules in the system are "linked" whenever:

- the system is powered-up;
- CTRL SHIFT RESET is pressed;
- or when BOOT is pressed.

The process of "linking" means that the system is scanned for modules, the addresses of all modules are loaded into dispatch tables, and any necessary initialization is performed. Note that this "linking" is done sequentially, through EPROM, then through any RAM modules which may have been loaded by the user. This means that two or more modules may define the same code sequence, and the LAST one linked will be the one in the dispatch table after linking is complete. This makes it easy for a user module to re-define a system function, with the module replacing the firmware module which has the same identifying code at its beginning.

Mode, Plot, Escape and User modules may accept arguments. The required arguments are defined by the module, and the system automatically parses arguments before passing control to the module. This relieves the user from writing argument parsing routines, and insures that all arguments are parsed in a consistent manner.

A module must save any registers it modifies, and restore the registers before exiting. As described below, several registers are pre-loaded before the module is executed and provide the module with system status information.

The Linking Process

The system scans two areas for modules when linking is performed: first, each EPROM pair on the 7900 Raster Processor Board is checked. If a ROM Expander card is installed and its address is consecutive with the Raster Processor, firmware in the ROM Expander will also be checked. If an EPROM pair is installed, and if a valid module is found at the start of the EPROM, linking proceeds through the EPROM. Linking terminates when an invalid length descriptor or an invalid module type code (one not in the set B, I, O, etc.) is found. Linking then proceeds to the start of the next EPROM pair.

After all EPROM modules have been linked, the system checks for RAM modules. RAM modules, if they exist, must be loaded into system RAM at the address pointed to by RAMMDLE. (RAMMDLE is a pointer in CMOS memory, and its contents may be altered with the "Thaw" command. See Chapter 6 for details.) The default address for RAM modules is $1F000; this allows 4K of space for modules, when a single Buffer Memory card is installed.

If RAM modules exist, they must follow all the rules described in this section, with an additional provision: to indicate the presence of RAM modules, the bytes 'MDLE' must be loaded into RAM at the start of the first RAM module:

```
ORG.L    $1F000        Org where Thaw wants us to be
DC.L     'MDLE'        Indicate RAM modules here

etc.
```

The bytes 'MDLE' must NOT be included if a module is being put into EPROM.

Modules are expected to be "back to back," existing consecutively through memory. Thus, any tables or other data used by a module must be WITHIN the module, not after it.

The last module should end with the following statement:

```
DC.L  -1,-1
```

This indicates to the linker that no more modules follow.

Module Construction

Each module begins with a length descriptor. The length is used to determine where one module ends, and where the next one begins. The address of each module is determined at boot time during the linking process, and loaded into a dispatch table for future reference.

Next, a module contains one of the characters B, I, O, Mode (CTRL A), Plot (CTRL B), Escape (CTRL [), or User (CTRL U), to define the type of module. This is immediately followed by a character which uniquely identifies that module.

The makeup of the remainder of the module depends on the type of module you are writing. All of the seven types of modules are discussed in this section, and examples are provided.

Boot Modules

A Boot module is executed upon power-up, when the system RESET key is pressed, or when the system is booted (by pressing BOOT). A Boot module might be written to initialize a piece of hardware, or to pre-load the Case Table for interfacing to a certain host computer.

The Boot module contains a length descriptor (word), the character 'B' followed by a dummy character, and the code to be executed. It ends with a RTS instruction.

```
        ORG.L  $1F000     Org where Thaw says to Org
        DC.L   'MDLE'     Required for RAM modules

        DC.W  MdleEnd-IPC  This is our length
        DC.B  'B',0        Identify a Boot module

*
*   Boot code begins here...
*    .
*    .
*   and ends here.
*

        RTS

MdleEnd EQU   IPC
        DC.L  -1,-1     Make sure linking ends here
        END
```

Input/Output Modules

I/O modules define the interface between a physical device, such as a printer, and the logical device assignment structure in the 7900. The I or O module must be responsible for handling all transfers to or from the device, checking the status of the device (if applicable), and booting the device (if necessary). Note that the Boot section of an I or O module performs the function of a B module.

The I or O module begins with a length descriptor (word). This is followed by the character I or O, defining an input or output module, and a character between A and Z to identify this particular I/O module. This character (A-Z) is used in the "Assign" command to identify the physical device which is being assigned.

Two words of code must follow the identifying character. These must be either SHORT branches to Boot and Status sections of the module, or RTS instructions. The Status section is used in an I module to see whether a character is ready to be read. The Status code should return the Z flag SET if no character is available, or clear it if a character is available. If the Status code returns Z set, the system will not execute the main code. (The Boot and Status portions of the module must also terminate with a RTS.)

In an input module (type I), if input is being buffered, the Status code may return a "snapshot" of the oldest character in the buffer in D0.B. This feature is used by the STATIN routine in TERMEM, and is called by DOS to check for X-ON and X-OFF commands (to suspend output). Note that the character snapshot must not remove the character from the input buffer; the character should remain in the buffer until read by the main code of the I module.

The Status section is not used in an O module. An O module must not return until it has completed processing a character.

The main body of the module follows, terminated by a RTS. The system passes a character to an Output module in register D0, and expects an Input module to return a character in D0. The low 8 bits of the register are used.

Note that a single device capable of both input and output requires two modules, one I and one O. (Our device 'Q' below may have an I module and an O module associated with it.)

```
*
*  Sample Input module (type 'I') for a device named Q.
*
        ORG.L       $1F000          Org where Thaw says to Org
        DC.L        'MDLE'          Required for RAM modules

        DC.W        MdleEnd-IPC     Our length
        DC.B        'I','Q'         Input module for device 'Q'

        BRA.S       Qboot
        BRA.S       Qstat

*
*  Main code for inputting a character from device Q.
*

        MOVE.B      Qbuffer,D0      Get input data
        BSR         Stompbuf        Remove this data from buffer
        RTS


*
*  Code for booting device Q. (Executed at Boot time.)
*

Qboot   MOVE.B      #0,QCtrlPort    Initialize device Q.
        MOVE.B      #$FF,QCtrlPort
        MOVE.L      #QISR,Qvector   Load interrupt vector
        RTS

*
*  Code for checking status of Q.
*

Qstat   TST.W       Qbufstat        Check buffer status
        MOVE.B      Qbuffer,D0      Snapshot oldest character
        RTS

MdleEnd EQU         IPC
        DC.L        -1,-1           No more modules here
        END
```

Argument Parsing

The module types discussed below may have arguments associated with them. An argument is a set of characters or numbers which is passed to the module. Mode, Plot, Escape and User modules may accept arguments. The arguments required by a module are defined by that module, and are parsed by the system before the module is called. The module then simply picks up its arguments and processes them.

The system will parse ten types of arguments:

Arg type	Description
0:	No argument, just a placeholder.
1:	A single character.
2:	A string of characters, delimited by a space, comma, or semicolon.
3:	A string of characters, delimited by a semicolon ONLY.
4:	A signed 16-bit decimal number (or a number in Binary Coordinate form, if the system is in Binary Mode).
5:	A 16-bit hexadecimal number.
6:	A decimal number, or the X component of a coordinate defined by the cursor.
7:	A decimal number, or the Y component of a coordinate defined by the cursor.
8:	An X-Y coordinate pair (combination of 6 and 7; will accept a period for cursor position).
9:	Any number of coordinate pairs, delimited by a semicolon.
A:	A decimal number, scaled but without translation.

Argument types 6 through 9 are designed to parse coordinate data. Each of these will scale and translate arguments according to the scale factors in use in the window which executed the module.

In defining arguments, the system looks for two words immediately following the module's name. Up to eight argument types may be placed in these words. If no arguments are to be processed, fill both words with zero.

Peculiarities of each type of module are discussed below.

Register Setup for Modules

When a module is executed, several registers are pre-loaded for convenience. The following table defines what each register is used for, when each type of module is entered.

Module	Register Usage
B (boot)	No registers pre-loaded.
I (input)	No registers pre-loaded.
O (output)	No registers pre-loaded.
Mode, Plot	A0: Points to base of Window Table for the window which called the module (see Window Table description).
	A1: Points to Mode arguments (parsed before module execution begins).
	A3: Points to Plot arguments (parsed before module execution begins).
	D7: Contains window status for the window which called the module (see Window Table description).
Escape, User	A0: Points to base of the Window Table for window A (the Master Window).
	A1: Points to arguments (parsed before module execution begins).
	D7: Contains window status for window A and Escape code status.

Mode Modules

A Mode module is executed when a window receives the Mode code sequence identifying that module. Mode modules are expected to affect only the window which called them, although nothing in the system will prevent a Mode module from affecting other windows or other aspects of the system.

A Mode module consists of a length descriptor (word), followed by the MODE character (CTRL A, decimal 1), and a character which uniquely identifies the module. This character may be any ASCII character above '0' (hex $30).

The next long word in the Mode module defines a list of arguments, to be parsed and passed to the module. Each nybble (4 bits) of the long word specifies one of the ten argument types listed above.

The argument list in this long word is right-justified, and the least-significant nybble defines the FIRST argument to be parsed. The long word must be left-filled with zeros to indicate the end of the argument list. Up to 8 arguments may be defined in this long word.

Example:

```
DC.W  MdleEnd-IPC     Length Descriptor
DC.B  Mode,'L'        Name of Mode module
DC.L  $00000144       Argument list
```

This list would specify that the module requires two decimal numbers, followed by a single character.

Arguments to a Mode module are put onto the "A1 stack." That is, the Mode module may assume that its arguments are waiting for it at the address pointed to by (A1), and successive bytes. The stack may contain up to 16 long words. MODE codes are processed when they reach the screen (or other physical device).

Example Mode Module

```
*****************************************************
*                                                   *
* This sample Mode module does the same thing as    *
* the TERMEM Fill ON/OFF command (MODE F 1/0)       *
*                                                   *
*****************************************************


          ORG.L     $1F000        Org where Thaw says to Org
          DC.L      'MDLE'        Required for RAM modules

          DC.W      FillEnd-IPC   Length descriptor

          DC.B      Mode,'f'      Name of module
Mode      EQU       1             (Mode is Control-A)
Fillon    EQU       8             Bit 8 in D7

          DC.W      $0000         Our arg list
          DC.W      $0001         (just one character)

*
*  Code for module "MODE f" begins here.
*
*  The user would type MODE f <n>
*
*  where <n> is either 0 or 1.
*
*


          CMP.B     #'0',(A1)     Get the flag

          BEQ.S     FillOff       If it equals zero, kill Fill
          BSET      #Fillon,D7    Else turn it on
          RTS                     Then quit

FillOff   BCLR      #Fillon,D7    Turn off Fill mode
          RTS

FillEnd   EQU       IPC
          DC.L      -1,-1         Only if no more modules
          END
```

Plot Modules

A Plot module performs a graphics function. All but one of the plotting features in the 7900 firmware are written as Plot modules. Plot modules generally accept coordinate data as arguments, and perform some plotting function based on this data. The "linking" procedure used for all modules means that you can write your own Plot modules, link them in to add new features to the 7900 firmware, or replace any existing features. Plot codes, like Mode codes, are processed when they reach the current output device.

Plot modules begin with a length descriptor (word). This is followed by the PLOT character (CTRL B, decimal 2) and a character which identifies the module. The identifier may be any character above ASCII "@" (hex $40).

The next two words specify arguments to be passed to the Plot module. The FIRST time a Plot module is entered, it may need to execute a special sequence of instructions to initialize itself. A status bit tells the module whether it is being entered for the first time, or on subsequent calls.

The second word of the argument list is used to select arguments for the first call. The first word selects arguments for subsequent calls.

Once a Plot code sequence has been entered, execution of the Plot module begins. The module can then be repeatedly called; it will be executed automatically when it has enough arguments.

Example:

```
DC.W    ModEnd-IPC
DC.B    Plot, 'Z'

DC.W    $0011       Two chars for repeated calls
DC.W    $0008       Coord arg for first call
```

In this example, the Plot module requires a coordinate argument when it is first executed. After the coordinates have been entered, the system will accept two single characters before entering the module (this is the argument scheme used by Incremental Vector).

This process continues until another Plot submode is entered or until the user turns Plot mode off.

On entry to the Plot module (and other modules), certain registers are set up for convenience. One of these is D7, which contains the window status. The "Submode" bit of D7 (bit 17) is set when a Plot module is first entered, to indicate that "this Plot Submode was just entered." If a Plot module cares about whether the submode has just been entered, it must check this bit and perform any necessary initialization if the bit is set. Then, it must clear the "Submode" bit.

If the Plot module does not do anything special when it is first entered, duplicate the argument list in the first and second word:

```
DC.W    ModEnd-IPC
DC.B    Plot,'Z'

DC.W    $0008          A coord for repeated calls
DC.W    $0008          and also for the first call.
```

Note that Plot modules may only parse FOUR arguments before execution begins; Mode modules could have up to eight.

To prevent conflicts between Plot code and Mode code arguments, Plot arguments are stacked on the "A3 Stack" and may be picked up from the addresses pointed to by A3. The module must not alter A3 or any other registers. Up to 32 long words are allocated for Plot module arguments.

Remember that more than one window at a time may be in the same Plot Submode. A Plot module should not store any local data which could interfere with another window calling the Plot module. Local data should be stored in the window table, or otherwise be localized to the window.

Example PLOT Module

```
*
*     An example of how a Plot module should be set up.
*
          ORG.L    $1F000
          DC.L     'MDLE'

          DC.W     ModEnd-IPC
          DC.B     Plot,'Z'
Plot      EQU      2              (Plot is Control-B)

          DC.W     $0008          One coordinate pair normally,
          DC.W     $0088          Two pairs the first time through.

Submode   EQU      17
          BTST     #Submode,D7    Did we just enter the submode?
          BEQ.S    Norm           No, it's a normal entry
*
*     Submode was just entered.  We can set up
*     in this block of code if necessary.
*
          BCLR     #Submode,D7    Prepare for next entry
*
*         (Setup code goes here)
*
          RTS

*
*
*     Come here if submode was NOT just entered.
*
*
Norm      MOVEM.L D0-D2/A3,-(SP)  Save registers
          MOVE.W  (A3)+,D0        Get X argument into D0
          MOVE.W  (A3),D1         Get Y argument into D1

*
*     Now, do something with the arguments here....
*     Maybe plot a vector or something.
*
          MOVEM.L (SP)+,D0-D2/A3  Restore registers
          RTS
```

Escape and User Modules

Escape and User modules are identical to each other, and are similar to the Mode modules discussed earlier. An Escape or User module begins with a length descriptor (word), followed by the ESC ($1B hex) or USER ($15 hex) character, and a single ASCII character which identifies the module. The identifier may be any character above ASCII '@' ($40 hex).

The next two words define an argument list, exactly like a Mode module. All arguments (up to eight) are parsed by the Escape Code Processor and are passed to the Escape or User module. Arguments are found on the A1 stack (pointed to by A1). This does not conflict with the stack of Mode arguments because Escape and User codes are processed by a different routine than Mode codes.

Remember that Escape and User codes have identical priority in the 7900 code processing scheme. Escape and User modules are not specific to a window, so they should be used to implement functions affecting the entire machine. Escape and User codes are trapped and processed by the Escape Code Processor.

Example ESCAPE code Module

```
*
*    A sample Escape code module to play with the
*    lights on the keyboard.  This could also have
*    been written as a User module.
*
         ORG.L   $1F000
         DC.L    'MDLE'          Identify a module is here

         DC.W    MdleEnd-IPC     Length descriptor
         DC.B    Esc,'X'         Escape X is our sequence
Esc      EQU     $1B

         DC.W    $0000
         DC.W    $0004           We want one decimal #

         MOVE.W  (A1),Keybrd     Send it to the keyboard
Keybrd   EQU     $FF8080         (Keyboard address)

         RTS

MdleEnd  EQU     IPC
         DC.L    -1,-1

         END
```

Window Tables

512 bytes of data are allocated as a Window Table for each active window. These bytes hold the current status of the window, including such items as color, blink, scale, window limits, and most other window attributes. Window attributes are usually set by Mode code sequences, and Mode modules will often want to alter items in a Window Table.

The following chart lists the location of each item in the window table, by giving an offset into the table where each item may be found. An item can be altered by a module by using the listed offset as a displacement from (A0), since A0 is pre-loaded with the base of the Window Table.

Example:

```
FrgCol   EQU      $8E              Offset in table
                                   for Foreground color

         MOVE.W   FrgCol(A0),D0    Get color into D0
```

In this table, entries are marked with .B, .W, or .L, to indicate the appropriate data size (where possible).

```
$00 .L    TVALUE:   Temporary storage area (reserved)
$04 .L    Arglst:   Mode argument list
$08 .L    Parglst:  Plot argument list
$0C .L    STATUS:   copy of D7 status long word
$10 .L    ^Argstk:  pointer to Mode argument stack
$14 .L    Argdsp:   Mode dispatch address
$18 .L    ^Prgstk:  pointer to Plot argument stack
$1C .L    Prgdsp:   Plot dispatch address (submode)

      Window variables are stored beginning here
              and occupy one word each.

$20 .W    Window Variable A
$22 .W    Window Variable B
   .
   .
$5C .W    Window Variable _
$5E .W    Window Variable ^
```

The following items are also Window Variables a-z
but are used for system data as well.

$60	.W	AcursX:	Overlay cursor X position
$62	.W	AcursY:	Overlay cursor Y position
$64	.W	CursX:	Bitmap cursor X position
$66	.W	CursY:	Bitmap cursor Y position
$68	.W	AwindX0:	Overlay window upper left corner X
$6A	.W	AwindY0:	Overlay window upper left corner Y
$6C	.W	AwindX1:	Overlay window lower right corner X
$6E	.W	AwindY1:	Overlay window lower right corner Y
$70	.W	WindX0:	Bitmap window upper left corner X
$72	.W	WindY0:	Bitmap window upper left corner Y
$74	.W	WindX1:	Bitmap window lower right corner X
$76	.W	WindY1:	Bitmap window lower right corner Y
$78	.W	CharXZ:	Character X raster size
$7A	.W	CharYZ:	Character Y raster size
$7C	.W	CharDX:	Character delta X after write
$7E	.W	CharDY:	Character delta Y after write
$80	.W	CharXM:	Character X multiplier
$82	.W	CharYM:	Character Y multiplier
$84	.W	XVmin:	Virtual X minimum value
$86	.W	YVmin:	Virtual Y minimum value
$88	.W	Tabcol:	Tab stop spacing
$8A	.W	Vecwid:	Vector width
$8C	.W	BkgCol:	Background color
$8E	.W	FrgCol:	Foreground color
$90	.W	PlaneE:	Planes enabled
$92	.W	CursCol:	Cursor color (not used, reserved)

Some miscellaneous items...

$94	.W	OldX:	Rubber band X position
$96	.W	OldY:	Rubber band Y position
$98	.W	XScl:	Virtual X Scale value
$9A	.W	YScl:	Virtual Y Scale value
$9C	.L	Charadr:	Character set base address
$A0	.L	Endbuf:	End of arguments for virtual coordinate
$A4	.L	Plotdot:	Dispatch address for dot plotting
$A8	.L	Plotvect:	Dispatch address for vectors

The following five items are used for raster processor operations in the window.

$AC .W	WXsrc:	X source raster operating point
$AE .W	WYsrc:	Y source raster operating point
$B0 .W	WDXsrc:	Delta X for source raster
$B2 .W	WDYsrc:	Delta Y for source raster
$B4 .W	Wctrl:	Control bytes for rasters

The next three items are for argument storage.

$B6-$F5	Curstg:	32 words for cursor pixel storage
$F6-$135	Argstk:	Argument stack for Mode args (32 words)
$136-$1B9	Pargstk:	Argument stack for Plot args (64 words)

The following four bytes contain the current Overlay color, blink and transparency attributes.

$1BA	AentSt:	Transparency
$1BB	AentBC:	Background color
$1BC	AentFC:	Foreground color
$1BD	AentCh:	ASCII character position
$1BE-$1FF		Reserved for future expansion

Window Status and ESCAPE Code Status

As shown before, register D7 is pre-loaded with status information when certain types of modules are executed. The bits in D7 are defined as follows:

Bit	Name	Meaning
0	modeF:	Mode flag used by window processor
1	plotF:	Plot flag used by window processor
2	Moredat:	Control bit used by argument scanners
3	Negnum:	Control bit used by argument scanners
4	visctrl:	SET when visible ctrls are on
5	Pltmode:	SET when in a plot submode (not alpha)
6	Overlay:	SET when Overlay on (not Bitmap)
7	Curson:	SET when cursor is on
8	Fillon:	SET when fill is on
9	Blinkon:	SET when blink is on
10	Rollon:	SET when roll is on
11	OvrStrk:	SET when overstrike is on
12	Binmode:	SET when binary mode is on
15	Rubron:	SET when rubber band is on
16	Patton:	SET when patterns are on
17	Submode:	SET when Plot submode just entered
18	Cursin:	SET when Bitmap cursor in screen RAM
19	BinTwo:	Flag for binary coordinate parser
20	VScale:	SET when scaling is on
21	A7on:	SET when A7 character set active
23	Local:	SET in LOCAL mode
24	Full:	SET in FULL duplex
25	Escflg:	Escape code processor flag
26	Usrflg:	Escape code processor flag
27	Create:	SET if Create is on
28	LitP0:	Escape code processor flag
29	LitP1:	Escape code processor flag
30	Literal:	SET if Literal Create is on
31	Escdone:	Escape code processor flag

Bit 0 is the least significant bit. Unused bits are reserved.

Custom Character Sets

The 7900 allows user-defined character sets to be used in the Bitmap in place of the two character sets supplied with the system. An entry in each Window Table (Charadr) points to the base of the character set for that window, and the size of the font (X by Y pixels) may also be defined for each window. The character font dimensions may be up to 16 in the X direction, and 256 in the Y direction.

Since the character set address for a window is stored in the Window Table, it will default back to the normal character set whenever BOOT or SOFT BOOT is executed.

The character set for the Overlay is stored in high-speed PROM, and is not alterable through software.

The following program is a module, designed to be linked into the 7900 system software. It will install a custom character set in any window which receives a MODE i command. This program is an example ONLY. It does not include a complete character set definition. The data set which should accompany this program would be too long to fit into the standard 7900 memory, unless you do one of the following: (1) change the ORG address, which requires changing address RAMMDLE with the Thaw command, (2) change the height of the character set to reduce the data required, or (3) install additional memory above address $20000.

```
*
*    Sample module to install a new character set.
*    The set is installed in a window by the command:
*
*        MODE i X, Y, <addr>
*
*    To return to the standard set, use  SOFT BOOT.
*
*****************************************************************
*
*    What follows is the character font definition.
*
*    It is arranged as 128 regular characters, followed by 128
*    A7 characters.  All 256 characters should be defined.
*
*    Assume the font is set in a field X by Y.  (Default is
*    X = 6 and Y = 8.)  Then each character requires 8 words
*    of data, one for each Y scan; and 6 bits of each word
*    are used.  The active 6 bits are left-justified in the
*    16-bit word.
*
*
*    Example: the character "A"
*
*                                    bit #
*
*                                    111111              Hex value
*                                    5432109876543210
*
*                                1   .XXX............     $7000
*                                2   X...X...........     $8800
*                                3   X...X...........     $8800
*            Word #              4   XXXXX...........     $F800
*        (one per Y scan)        5   X...X...........     $8800
*                                6   X...X...........     $8800
*                                7   X...X...........     $8800
*                                8   ................     $0000
*
*
*    The X by Y field must include any necessary
*    spacing between characters or between lines.
*
*
*    Memory requirements:
*
*    This module will require 256 * Y words.
*    This much memory must exist between address
*    "RAMMDLE" and the physical end of memory.
*    If necessary, use Thaw to alter the address of "RAMMDLE" (be
*    sure to put it where nothing will stomp on it!)
*
*
```

```
*
*
*    This Scale Factor will left-justify a 6-bit number in a
*    16-bit field, by shifting 10 bits.  All of the numbers in
*    this database are 6-bit for convenience, and S justifies
*    them properly.
*
S       EQU     1024

*
*
BASE    ORG.L   <known address>         This is where it all begins.
*
*    This is the regular character set.
*
    DC.W    $24*S,$34*S,$3C*S,$2C*S,$24*S,$04*S,$04*S,$07*S     ^@
    DC.W    $0C*S,$10*S,$08*S,$05*S,$1D*S,$07*S,$05*S,$05*S     ^A
*     .
*     .    etc. for regular set.
*     .
    DC.W    $01*S,$0E*S,$10*S,$00*S,$00*S,$00*S,$00*S,$00*S      ~
    DC.W    $0A*S,$15*S,$0A*S,$15*S,$0A*S,$15*S,$0A*S,$00*S    DEL
*
*
*
*
*    Alternate (A7) character set begins here.
*
    DC.W    $1F*S,$15*S,$15*S,$1F*S,$15*S,$15*S,$1F*S,$00*S     ^@
    DC.W    $08*S,$15*S,$02*S,$08*S,$15*S,$02*S,$00*S,$00*S     ^A
*     .
*     .    etc. for A7 set.
*     .
    DC.W    $01*S,$02*S,$02*S,$04*S,$08*S,$10*S,$10*S,$20*S      ~
    DC.W    $00*S,$00*S,$00*S,$00*S,$00*S,$00*S,$00*S,$3F*S    DEL
*
*

        END
*
*
```

Vector-Drawn Character Format

The command format for vector-drawn characters is on page 9-26. These characters are formed by a vector list, which is an arbitrary number of short (bold) vectors. All printable characters from the exclamation point ($21 hex) through the tilde (~, $7E hex) are described by entries in a vector list. The list is pointed to by a long word in CMOS called **VChrset**, address $E4021E. **VChrset** will normally point to the default vector list in PROM, but it can be altered to point to a custom set.

Each vector-drawn character lies within a 16 x 16 matrix. The cursor is assumed to be at the upper left corner of the matrix, so the matrix points are numbered (in hexadecimal):

```
 0123456789ABCDEF
0
1
2
.
.
E
F
```

Within the vector list, each 16-bit word defines one vector of the character. Four bits each are used for X1, Y1, X2 and Y2 for that vector. For example, consider the letter H:

```
 0123456789ABCDEF
0X.........X.....
1X.........X.....
2X.........X.....
3X.........X.....
4X.........X.....
5XXXXXXXXXXX.....
6X.........X.....
7X.........X.....
8X.........X.....
9X.........X.....
AX.........X.....
B................
C................
D................
E................
F................
```

In the diagram, Xs correspond to points to be drawn in the character. Three vectors are needed: from Ø,Ø to Ø,A; from Ø,5 to A,5; and from A,Ø to A,A. The vector list for this character would be:

```
$ØØØA
$Ø5A5
$AØAA
$ØØØØ
```

The zero word signals the end of the list for that character. The definition for each character may contain an arbitrary number of vectors.

The default character set only uses part of the available 16 by 16 matrix. Most characters in the default set fit within an 8 by 12 matrix, including lower-case descenders. If you use a "non-standard" size, the Set Intercharacter Spacing command will force the proper spacing.

Algorithm Description

When told to produce a character, TERMEM scans the vector list starting at the address in **VChrset.** It assumes that the description for the exclamation point character ($21 hex) is at the start of this list. Then it scans forward, looking for the zero words that separate the character descriptions, until the correct character is reached.

Each data word describing the character is separated into the four-bit values for X1, Y1, X2 and Y2. These values are multiplied by the current X and Y character size parameters in the Window Table. Then, the coordinate data is rotated according to the angle given in the Vector-Drawn Character command. The current cursor position is added to these rotated XY values, and a bold vector is drawn in the current foreground color and vector width. This process is repeated until the terminating zero word is found in the vector list.

Next, the cursor location is updated. If proportional spacing is ON, the largest X value in the character list plus 2 is used (this implies that the characters must be left-justified in the 16 by 16 matrix for proportional spacing to work properly). If proportional spacing is OFF, the current X intercharacter spacing is used. In either case, this value is multiplied by the current X character size, rotated and added to the current cursor position.

The Space character does not have a descriptor in the vector list, so the current intercharacter spacing is used to update the cursor position.

The following DOS program fragment will load a custom vector-drawn character set. Since **VChrset** is in CMOS and the vector list is in RAM, you will have to either save the default value of **VChrset** and restore it at the end of the program, or press CTRL SHIFT M1 M2 RESET to return to the default character set.

****

```
VChrset     EQU     $E4021E             pointer to data set

            ORG.L   $1C3C               run in DOS area

Start       MOVE.L  VChrset,-(SP)       save default pointer
            MOVE.L  #Table, VChrset     load the new pointer

            (program goes here)

            MOVE.L  (SP)+, VChrset      reload default pointer
            CLR.L   DO
            CLR.L   D1
            RTS                         return to DOS


Table       DC.W    xxxx,xxxx...        data for !
            DC.W    0                   terminator

            DC.W    xxxx,xxxx...        data for "
            DC.W    0
*
* And so forth, for all ASCII characters
*
            DC.W    xxxx,xxxx...        data for tilde (~)
            DC.W    0

            END     Start               execute at label "Start"
```

Installing a New Cursor

The 7900 Bitmap cursors, Plot and Alpha, are each described by a set of data. This set is pointed to by pointers in the CMOS area, one pointer for the Plot cursor and one for the Alpha cursor.

```
Plotcur   EQU   $E4016A
Alphcur   EQU   $E4016E
```

The cursor descriptor data is a list of up to 32 long words. Each long word describes the displacement of one pixel of the cursor, with respect to the center pixel of the cursor. The list is terminated with a zero word. Since this zero word is part of the descriptor, the center pixel of the cursor is always ON.

The displacements are given as addresses in Bitmap memory. Each pixel in Bitmap memory corresponds to a word (two bytes) of memory, so an X displacement of one pixel is produced by an address displacement of two. (Positive X displacement is to the right.) Similarly, a Y displacement of one pixel corresponds to an address change of 2048 bytes (1024 pixels per Y line of the screen, times two bytes per pixel. A positive Y displacement is in the down direction.

A sample cursor might look like this, where X's correspond to pixels included in the cursor:

```
 X
XXX
 X
```

The data list for this cursor would be:

```
   +2  (the pixel to the right of center)
   -2  (the pixel to the left of center)
+2048  (the pixel below center)
-2048  (the pixel above center)
    0  (the center pixel, and end of the list)
```

To install a new cursor, first define it in the form above. Store this data in memory. Then, alter the pointer in CMOS (either Plotcur or Alphcur) so that it points to your data. Note that if you store your cursor in RAM other than CMOS, the description will vanish when system power is turned off, but the CMOS pointer will remain! This will cause you to have NO cursor at all. To reload CMOS defaults, use CTRL SHIFT M1 M2 RESET.

```
**********************************************************************
*
*    Sample program PUTCURS
*
*    Installs a new cursor as the Bitmap plot cursor.
*
*    This program stores its cursor descriptor in upper CMOS
*    memory, unused by current 7900 software.  This may not be
*    compatible with future 7900 releases.
*
*
*
        ORG.L   $1C3C           We run in DOS area

PUTCURS MOVE.L  #HiCMOS,A2      Point to some unused CMOS
        MOVE.L  #Cursor,A3      Point to our new cursor descriptor

PUTloop MOVE.L  (A3)+,(A2)+     Copy a long word into CMOS
        TST.L   -4(A2)          Was it zero?
        BNE.S   PUTloop         No, continue copying

        MOVE.L  #HiCMOS,Plotcur   Set up pointer to new cursor

        CLR.L   D0              Flag no error occurred
        CLR.L   D1              (We don't check for colon on line)
        RTS                     Return to DOS


HiCMOS  EQU     $E40E00         CMOS area (unused in TERMEM 1.4)
Plotcur EQU     $E4016A         Plot cursor pointer

Cursor  DC.L    -4*1024         New cursor descriptor
        DC.L    -4*1024+2
        DC.L    -4*1024-2
        DC.L    -2*1024-4
        DC.L    -2*1024+4
        DC.L    -4
        DC.L    +4
        DC.L    +6
        DC.L    +8
        DC.L    -2*1024+8
        DC.L    -2*1024+10
        DC.L    -2*1024+12
        DC.L    -2*1024+14
        DC.L    -4*1024+10
        DC.L    -4*1024+12
        DC.L    -4*1024+14
        DC.L    -4*1024+16
        DC.L    4*1024
```

```
        DC.L    4*1024+2
        DC.L    4*1024-2
        DC.L    2*1024-4
        DC.L    2*1024+4
        DC.L    2*1024+8
        DC.L    2*1024+10
        DC.L    2*1024+12
        DC.L    2*1024+14
        DC.L    4*1024+10
        DC.L    4*1024+12
        DC.L    4*1024+14
        DC.L    4*1024+16
        DC.L    0               (end of list)

        END     PUTCURS
```

TERMEM Jump Tables

This section describes the utilities available in CGC 7900 PROM firmware. Each of these routines may be accessed through a subroutine call (JSR) to the appropriate address. BSR will not work, because the jump tables will be located beyond a 16-bit displacement from your program.

Registers used in each routine are defined in this section. Unless noted, the routine only alters registers as necessary to return values to the caller. The notation "D1.W" means that the low word (16 bits) of D1 are used by the routine, D0.B means the low byte of D0, and so on.

Name:
BOOT

Address:
$80004C

BOOT boots the system. It does not return to the caller. One reason for calling BOOT would be to link in any RAM modules you have loaded into system memory.

Name:
CHARIN

Address:
$80000C

Entry:
D1.W = logical device number to read from

Exit:
Zero flag SET if no character was available.
Zero flag CLEAR and character in D0.B if available

CHARIN is the system character-in routine. It reads a character from a logical input device. Reading from device 0 will get a character from the keyboard (if available), device 1 will be the serial port (these are the default connections).

To wait for a character from CHARIN, use a loop on the EQ condition:

```
Loop  JSR     CHARIN    Get a character, if any
      BEQ.S   Loop      No character yet
```

Escape and User codes are not processed by CHARIN, but are treated as normal characters. See CTRLOUT, CTRLIN and ESCPROC below.

Name:
CHAROUT

Address:
$800008

Entry:
D0.B = character to pass to logical device.
D1.W = logical device number (0 to 4 are defined).

CHAROUT is the system character-out routine. It passes a character to a logical output device. The system's device assignment structure will then pass the character on to a physical device, if possible. Logical output device 0 is normally used to put characters on the screen, and device 1 is normally connected to the serial port. If the character is part of a Mode or Plot code sequence, it will be processed when it reaches a Window (physical device). Escape and User codes are NOT processed by CHAROUT, but are treated as normal characters. To process Escape and User codes, see CTRLOUT, CTRLIN and ESCPROC below.

Name:
CTRLIN

Address:
$800014

Entry:
D1.W = logical device number to read from.

Exit:
Zero flag SET if no character was available.
Zero flag CLEAR and character in D0.B if available.

CTRLIN is like CHARIN, but processes Escape and User codes before returning a character. It does this by calling CHARIN and then ESCPROC. If an Escape or User code was entered, it will be "eaten" by ESCPROC and the zero flag will be set, indicating that no character was available.

Name:
CTRLOUT

Address:
$800010

Entry:
D0.B = character to pass to logical device.
D1.W = logical device number.
Zero flag MUST be cleared.

CTRLOUT is like CTRLIN, but processes Escape and User codes before passing the character on to the logical output device. It does this by calling ESCPROC (see below), then calling CHAROUT.

NOTE:

Do not call CTRLOUT if the zero flag is set -- your character will be ignored.

Name:
ESCPROC

Address:
$800018

Entry:
D0.B = character to process.
Zero flag MUST be cleared.

Exit:
Zero flag SET if character was "eaten" by ESCPROC.
Zero flag CLEAR if character is still available.

ESCPROC handles Escape and User code processing. It is used by CTRLIN and CTRLOUT, and may also be called directly. ESCPROC detects Escape and User codes, and processes them by setting the zero flag to indicate that the character was processed (and thus should not be considered available). After the Escape or User code is detected, ESCPROC will process subsequent characters to satisfy the argument list of that particular Escape or User function, then execute the function. This will normally be transparent to the user.

Note that ESCPROC also processes the Create Buffer. Characters must flow through ESCPROC (or CTRLIN, or CTRLOUT), or they will not be put into the Create Buffer.

Name:
Keystuff

Address:
$800088

Entry:
D0.W = character to put into keyboard buffer

Keystuff puts 8-bit characters into the keyboard input buffer. These characters will be read from the buffer along with any characters which were actually typed. Keystuff also checks if the M1 and/or M2 keys are active (signified by bits 10 and 11 being set), and performs translations if so. To insert a normal 8-bit character into the keyboard buffer using Keystuff, mask off these bits:

```
AND.W   #$FF,D0     Make 8-bit only
JSR     Keystuff    And put into buffer
```

If the keyboard buffer is full, the character is discarded.

Name:
NOISE

Address:
$800054

Entry:
A0 points to a tone descriptor block (14 bytes).

Exit:
A0 is incremented past block.

NOISE feeds data to the sound generator. 14 bytes are loaded sequentially into the tone chip. These bytes go into registers 0 through 13 of the tone chip (a General Instruments AY-3-8910), and control the following attributes:

Register #	Purpose
0	Fine Tune A (8 bits)
1	Coarse Tune A (4 bits)
2	Fine Tune B (8 bits)
3	Coarse Tune B (4 bits)
4	Fine Tune C (8 bits)
5	Coarse Tune C (4 bits)
6	Noise Period (5 bits)
7	Output Enable
8	A Amplitude (5 bits)
9	B Amplitude (5 bits)
10	C Amplitude (5 bits)
11	Envelope Period Fine (8 bits)
12	Envelope Period Coarse (8 bits)
13	Envelope Shape/Cycle Control (4 bits)

The tone generator has three voices, A, B, and C, each of which can be programmed to produce tone or noise. If a given voice is programmed for both tone and noise, noise will usually dominate. Tone and/or noise are enabled by register 7:

7	6	5	4	3	2	1	0
X	X	An	Bn	Cn	At	Bt	Ct

A zero on any of the "n" bits enables noise from that channel, and a zero on any of the "t" bits enables tone from that channel. Unused channels are turned off by writing ones in the desired bits.

Registers 8, 9 and 10 control the output amplitudes:

7	6	5	4	3 2 1 0
X	X	X	A	manual level ctrl

A one in bit 4 specifies the channel's amplitude to be controlled by the envelope generator (Auto mode). If bit 4 is a zero, the amplitude is fixed by the value in bits 0-3.

The envelope generator is controlled by register 13:

7	6	5	4	3	2	1	0
X	X	X	X	cont	attk	alt	hold

Bits 0-3 describe the envelope with "continue," "attack," "alternate" and "hold." See General Instruments literature for the envelope waveforms. The 7900 Hardware Reference Manual also has more information on the sound generator.

Name:
PLRRCT

Address:
$800068

Entry:
D0.W = radius
D1.W = angle in integer degrees

Exit:
D0.W = X
D1.W = Y

PLRRCT performs polar to rectangular conversion, using SIND0 and a technique similar to the one described in SIND0, below.

Name:
READJOY

Address:
$800070

Entry:
A1.L = pointer to joystick X, Y or Z axis.

Exit:
D0.W = value read from joystick.

READJOY returns the current value of a joystick axis as a 10-bit number in the range 0 to 1023. A1 must be set to the address of one of the joystick axes, as follows:

X	$FF80C6
Y	$FF80CA
Z	$FF80CC

Name:
PRTDEC

Address:
$800058

Entry:
D0.W = decimal number to convert to ASCII.
A1.L = pointer to buffer where ASCII goes.

Exit:
A1 is incremented past the string.

PRTDEC prints a decimal number as an ASCII string. The string is placed into memory at (A1)+.

Name:
Scankey

Address:
$800098

Entry:
D0.W = character to be converted and buffered.

Scankey converts an 8-bit code (produced by the upper half of the keyboard) into a stream of 7-bit codes, then stuffs them into the keyboard input buffer. See Keystuff, described above.

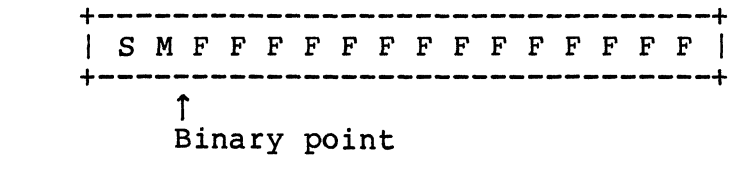
Name:
SIND0

Address:
$800064

Entry:
D0.W = angle in integer degrees.

Exit:
D0.W = sine of that angle.

SIND0 takes the sine of an angle and returns the value as a 14-bit fraction. The form of the fraction is:



Where:

S is the sign of the value (1 is negative).

M is the mantissa (zero except if the value is one or negative).

F are the fractional bits.

The following example uses SIND0 to compute Y*SIN(Theta). D0 is the angle Theta, and Y is in the lower word of D1. The value is returned in D1.

```
JSR     SIND0    Get sine of theta
MULS    D0,D1    Y=Y*SIN(theta)
ASL.L   #2,D1    Adjust for 14-bit fraction
SWAP    D1
EXT.L   D1       Clear garbage from high word
```

Name:
STATIN

Address:
$80008C

Entry:
D1.W = logical device number to read from.

Exit:
D0.B = snapshot of oldest character (see below).
Zero flag CLEAR if character available, SET if not.

STATIN is similar to CHARIN and CTRLIN, in that it returns the status of an input device reflected in the zero flag. STATIN will also return a snapshot of the oldest character in the input buffer (if possible; some devices are not buffered). STATIN does not actually remove a character from a buffer. A character returned by STATIN will be returned again by subsequent calls to STATIN, and will also be returned by a call to CHARIN. Only after CHARIN or CTRLIN have been called, is the character removed from the device's input buffer.

Name:
TERMEM

Address:
$800040

TERMEM is the entry point for the Terminal Emulator, the main operating program in the CGC 7900. Programs can jump to this address to terminate execution without returning to DOS. TERMEM does not return to the caller. When entering TERMEM, it remains in the same status it was last in (half duplex, full duplex, or local).

Name:
TCRSON

Address:
$8000A4

TCRSON will turn the cursor on in the currently specified cursor color. It determines whether the 7900 is addressing the Overlay or Bitmap and which mode it is in (Plot or Alpha) and turns on the appropriate cursor.

Name:
TCRSOFF

Address:
$8000A8

TCRSOFF determines whether the 7900 is addressing the Overlay or Bitmap and which mode it is in (Plot or Alpha) and turns off the appropriate cursor.

Plotting Functions

Many of the plotting primitives in the 7900 may be accessed through jump tables. Plot functions are specific to a window, which means the system must know which window to use for executing the plot routine. Data from the Window Table determines such things as the color of the plotted figure, and whether or not the figure will be filled.

Before calling any of these plot routines, the data to be plotted must be scaled to screen coordinates, between 0 and 1023. Data outside this range will be plotted unpredictably unless clipping is enabled. If clipping is enabled, out-of-range data will not be plotted.

All of the plot routines discussed below need register A0 to point to the base address of the current Window Table. This will be done automatically if your program is linked as a module, but if you are writing a transient, you must load A0 in your program. If plotting in window A is desired, you may set up A0 by this code:

```
BtmGWin  EQU      $C40          Pointer to W table base
         MOVE.L   BtmGWin,A0    Get pointer
```

Each Window Table occupies 512 bytes. If D0.L contains the window number (0 through 7, for windows named A through H), the following code would point A0 to the base of any Window Table:

```
MOVE.L   BtmGWin,A0     Get pointer
ASL.L    #5,D0
ASL.L    #4,D0          D0=512*D0
ADD.L    D0,A0          Add to base
```

Some of the functions described below require an argument list, which is passed on the "A3 stack." The values passed to the routine are pointed to by (A3), and the words following (A3). An area of the Window Table called **Pargstk** (Plot Argument Stack) is normally used to pass arguments, or your program can use other RAM for this purpose. To load the Pargstk area with four values for a vector, use the code on the next page.

```
Pargstk EQU     $136              Offset in W table for plot args

        LEA     Pargstk(A0),A3    Point to Pargstk
        MOVE.W  X1,(A3)
        MOVE.W  Y1,2(A3)          Load XY values on A3 stack
        MOVE.W  X2,4(A3)
        MOVE.W  Y2,6(A3)
        JSR     FVECT             Draw a vector
```

Before calling any of these plot routines, the data to be plotted must be scaled to screen coordinates, between 0 and 1023. Data outside this range will be plotted unpredictably.

These four routines plot figures according to the attributes of the Window Table pointed to by A0. Plotting will occur in the Overlay or Bitmap, with or without patterns, according to the current status of the window.

Name:
ARC

Address:
$80007C

Entry:
A3.L = pointer to X, Y, radius, start, delta (words)
A0.L = pointer to Window Table
D7.L = Window Table status

Name:
BVECT (Bold Vectors)

Address:
$800074

Entry:
A3.L = pointer to X1, Y1, X2, Y2 (words) and 4 scratch words
A0.L = pointer to Window Table
D7.L = Window Table status (see section A.10)

Name:
CIRCLE

Address:
$800078

Entry:
A3.L = pointer to X, Y, radius (words)
A0.L = pointer to Window Table
D7.L = Window Table status

Name:
CURVE

Address:
$800084

Entry:
A3.L = pointer to X1, Y1, X2, Y2, X3, Y3, X4, Y4 (words)
A0.L = pointer to Window Table
D7.L = Window Table status

Name:
FVECT

Address:
$800060

Entry:
A3.L = pointer to X1, Y1, X2, Y2 (words)
A0.L = pointer to Window Table

FVECT plots a vector in the Overlay or Bitmap, from (X1, Y1) to (X2, Y2). FVECT vectors through the Window Table entry called Plotvect, which holds the address of a routine to plot a vector in the Overlay or Bitmap (see the discussion of Plotdot above). The address in Plotvect is loaded by any of the "MODE O" or "MODE T" commands. The current foreground color of the window (stored in the window table) is used unless patterns are active, OR UNLESS THE VECTOR IS HORIZONTAL. If Y1=Y2, a special fast vector routine is used which writes part of the vector through proprietary Color Status hardware. The color produced by Color Status is determined by loading the Color Status Foreground latch, a word at address $E40016. To use FVECT properly, you must load the Window Table and the Color Status latch with your desired color.

Name:
PLOTXY

Address:
$80005C

Entry:
D0.W = X value
D1.W = Y value
A0.L = pointer to Window Table

PLOTXY plots a single dot in the Overlay or Bitmap, at the XY coordinate specified by D0 and D1. PLOTXY vectors through the Window Table entry called **Plotdot**, which holds the address of a routine to plot a dot in the Overlay or Bitmap. Plotdot also holds the address of a routine which plots patterns in the Bitmap, if patterns have been enabled. This address is loaded by the "Mode O" or "Mode T" commands. The current foreground color of the window is used unless patterns are active.

NOTE:

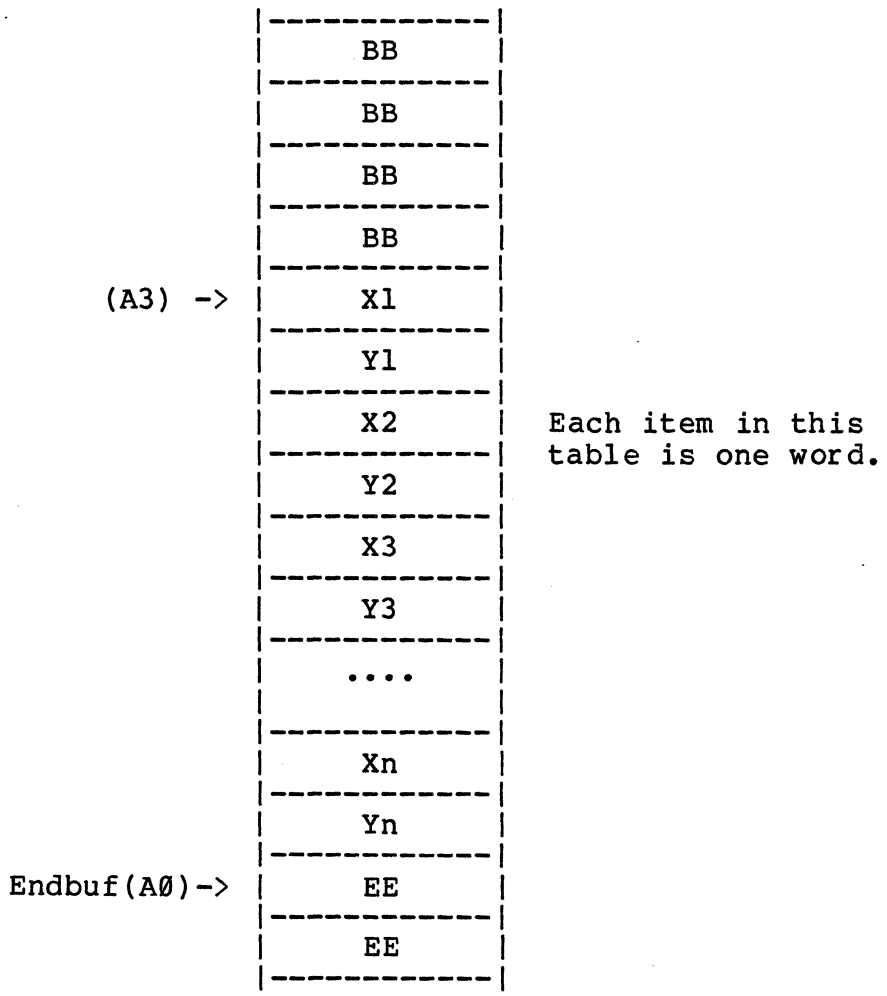
Both FVECT and PLOTXY make use of the optional Hardware Vector Generator, if one is installed in your system.

Name:
POLYG

Address:
$800080

Entry:
A3.L = pointer to coordinate pairs (words)
A0.L = pointer to Window Table
D7.L = Window Table status

POLYG is similar to the other plot routines described above, except that it can accept a variable-length argument list. The beginning of the list is pointed to by A3.L, and the end of the list is pointed to by an entry in the Window Table named Endbuf. Endbuf is a long word, and it holds the address of the word PAST the last coordinate in the polygon argument list.



Pairs of words before and after the list (marked BB and EE above) are used as scratch areas by POLYG, and your program should allow room for them.

The following code might be used to call POLYG:

```
Pargstk   EQU     $136              Offsets in W table
Endbuf    EQU     $A0

          LEA     Pargstk(A0),A3    Point to arg stack
          MOVE.L  A3,-(SP)          Save pointer
          MOVE.W  X1,(A3)+
          MOVE.W  Y1,(A3)+          Load coordinates
          MOVE.W  X2,(A3)+           onto A3 stack
          MOVE.W  Y2,(A3)+
            .
            .
            .
          MOVE.W  Xn,(A3)+          Put last values
          MOVE.W  Yn,(A3)+
          MOVE.L  A3,Endbuf(A0)     Set up end of list
          MOVE.L  (SP)+,A3          Retrieve pointer to
                                     start of list
          JSR     POLYG             Do polygon
```

Inline Calling Sequences

Name:
INLINE

Address:
$80A00C

Entry:
A1.L = pointer to input buffer to be used
(must be at least 84 characters long).
D1.W = Logical Input Device number to read from.
D7.B = control bits (see below).

Exit:
Zero flag SET if the user hit DELETE.
Zero flag CLEAR if the user hit RETURN.

INLINE is the 7900's general-purpose input routine, used by DOS, the Monitor, and Thaw. It reads a line of up to 83 characters from the user, allowing character editing, Recall Last Line, etc. Bits in D7 control INLINE as follows:

Bit	Meaning if SET
5:	Treat line-feeds as logical line separators (newline character).
3:	Echo the input line to the screen after RETURN is pressed (in expanded form, Modes and tabs executed normally).
2:	Process Escape and User codes as they are entered.
1:	Use "A7" character set for control-characters displayed in compressed form.
0:	Do not display the characters as they are entered (you can't see what you type). The line will NOT be put into the Recall Buffer.

DOS uses D7 equal to $0E, D1 equal to zero, and A1 pointing to the DOS input buffer in low RAM. The end of the user's input line is indicated by a RETURN character ($0D) in the buffer. Note that INLINE does not echo a RETURN/LINEFEED when the user enters RETURN.

Name:
INLINE1

Address:
$80A018

Entry:
A1.L = pointer to input buffer to be used.
D1.W = Logical Input Device number to read from.
D2.L = length of input buffer.
D7.B = control bits as described for the INLINE routine.
A4.L = pointer to table of line termination characters.

Exit:
Zero flag CLEAR if user hit RETURN; D0.L undefined.

Zero flag SET if the user entered a line delete character sequence. D0.L = index into termination table of characters causing delete (i.e. the instruction MOVE.L 0(A4,D0.L),D0 will put the terminating characters into D0).

INLINE1 is used to read lines of non-standard length. The length of the input buffer is given in D2.L. INLINE1 will read up to **length-1** characters into the buffer, with the end of the line being flagged by the RETURN character ($0D). INLINE1 also provides for changing the list of characters which delete an input line (DELETE, BREAK and CTRL C are the standard line termination characters).

Note that lines longer than 255 characters will not be stored in the Recall Buffer.

As an example, suppose you are writing an editor. You want to be able to edit lines of up to 150 characters. Furthermore, you want the characters CTRL X, INS LINE and DEL LINE to cause INLINE to abort input. The following is a program fragment to accomplish this.

```
*
* define input buffer storage
*

LINELEN EQU     150+1                   150 characters + CR
INPBUF  DS.B    LINELEN                 Input buffer

*
* define termination table
* (you must ensure that it is on an even boundary!)
*

TRMTBL  DC.L    $18                     CTRL-X
        DC.L    $00013E32               Insert line (MODE > 2)
        DC.L    $00013C32               Delete line (MODE < 2)
        DC.L    0                       End of table marker

*
* read a line
*

        MOVEQ.L #0,D1                   Read from Device 0
        MOVEQ.L #$0E,D7                 Control bits
        LEA     *+(TRMTBL-(IPC+2)),A4   Pointer to termination table
        LEA     *+(INPBUF-(IPC+2)),A1   Pointer to input buffer
        MOVE.L  #LINELEN,D2             Line Length
        JSR     INLINE1                 Read line
        BEQ     trmn8ed                 If LINE DELETE char entered
        BRA     retin                   Return entered
        .
        .
        .

trmn8ed CMP.L   #4,D0                   See if INSERT LINE entered
        BEQ     ...                     If yes
        CMP.L   #8,D0                   See if DELETE LINE entered
        BEQ     ...                     If yes
```

Name:
INLINE2

Address:
$80A01C

Entry:
Registers as set up by INITINL (routine described below)

Exit:
Same as INLINE1

INLINE2 is a low level routine used to get characters from the specified device and process them until **RETURN** or a line termination character is entered. Its use will be demonstrated in the INLINE3 example below.

Name:
INLINE3

Address:
$80A020

Entry:
Registers as set up by INITINL or returned by INLINE3.
D0.B = character to process.

Exit:
If D0.B was not a line termination character or RETURN, then D2.L=0; Zero flag is SET. Registers are updated for another character.

If D0.B was RETURN, then D2.L=1, Zero flag CLEAR.

If D0.B was a line termination character, then D2.L=2; Zero flag is CLEAR; D0.L=index into termination table.

Important! Between calls to INLINE3, the only registers that may be altered are D0, A1 and A5. Changing any other registers will most likely crash the system. Also, INLINE3 will destroy D0, A1 and A5.

INLINE3 is the workhorse of the INLINE system. It accepts characters one at a time and processes them, performing all editing functions.

As an example of how these routines tie together, see the program fragment below. It shows how the DOS editor (should) display lines for editing. Assume that A5 points to the line to be edited and that it is terminated by a **RETURN.** The code on the following page will display the line, then let the user modify it:

```
*
* save pointer to char as INLINE3 eats things
*
    MOVE.L   A5,-(SP)

*
* set up inline for a new line:
*
    MOVEQ.L  #0,D1                  Read from device 0
    MOVEQ.L  #$0E,D7                Control bits
    LEA      *+(TRMTBL-(IPC+2)),A4  Pointer to termination table
    LEA      *+(INPBUF-(IPC+2)),A1  Pointer to input buffer
    MOVE.L   #LINELEN,D2            Length of input buffer
    JSR      INITINL                Initialize the INLINE system

*
* get characters from EDIT buffer and feed to INLINE
*
X1  MOVE.L   (SP)+,A5               Get pointer to char
    MOVE.B   (A5)+,D0               Get the character
    CMP.B    #CR,D0                 End of the line?
    BEQ.S    X2                     If it is....
    MOVE.L   A5,-(SP)               Remember:  INLINE3 munches reg's
    JSR      INLINE3                Feed char to INLINE3
    BRA.S    X1

*
* let user edit the line
*
X2  JSR      INLINE2

*
* INLINE2 returns EQ/NE.  Take appropriate action....
*
```

Name:
INITINL

Address:
$80A024

Entry:
A1.L = pointer to input buffer to be used.
D1.W = Logical Input Device number to read from.
D2.L = length of input buffer.
D7.B = control bits.
A4.L = pointer to table of line termination characters.

Exit:
Registers as required by INLINE3.

INITINL is used to initialize all information as required by the INLINE system.

Name:
INLHOME

Address:
$80A028

Entry:
Registers as set up by INITINL.

Exit:
Cursor moved to home position of line.

After a line has been output (see INLINE3 example), INLHOME could be used to put the cursor on the first character of the line before allowing the user to edit the line. The Chromatics editors using INLINE put the cursor at the end, but it is only a matter of taste.

Appendix F -- ASCII Codes

This section includes a chart of the ASCII (American Standard Code for Information Interchange) characters, and a chart of the character fonts provided in the 7900.

The 7900 uses some control-codes in ways not defined by ASCII. See Appendix A for the 7900's allocation of control-codes.

Standard ASCII Character Set

Value	Code	Character	Value	Char	Value	Char	Value	Char
0 $00	CTRL @	Null	32 $20		64 $40	@	96 $60	`
1 $01	CTRL A	Start of Heading	33 $21	!	65 $41	A	97 $61	a
2 $02	CTRL B	Start of Text	34 $22	"	66 $42	B	98 $62	b
3 $03	CTRL C	End of Text	35 $23	#	67 $43	C	99 $63	c
4 $04	CTRL D	End of Transmission	36 $24	$	68 $44	D	100 $64	d
5 $05	CTRL E	Enquiry	37 $25	%	69 $45	E	101 $65	e
6 $06	CTRL F	Acknowledge	38 $26	&	70 $46	F	102 $66	f
7 $07	CTRL G	Bell	39 $27	'	71 $47	G	103 $67	g
8 $08	CTRL H	Backspace	40 $28	(	72 $48	H	104 $68	h
9 $09	CTRL I	Tab	41 $29	)	73 $49	I	105 $69	i
10 $0A	CTRL J	Linefeed	42 $2A	*	74 $4A	J	106 $6A	j
11 $0B	CTRL K	Cursor Up	43 $2B	+	75 $4B	K	107 $6B	k
12 $0C	CTRL L	Form Feed	44 $2C	,	76 $4C	L	108 $6C	l
13 $0D	CTRL M	Return	45 $2D	-	77 $4D	M	109 $6D	m
14 $0E	CTRL N	Shift Out	46 $2E	.	78 $4E	N	110 $6E	n
15 $0F	CTRL O	Shift In	47 $2F	/	79 $4F	O	111 $6F	o
16 $10	CTRL P	Data Link Escape	48 $30	0	80 $50	P	112 $70	p
17 $11	CTRL Q	Device Control 1	49 $31	1	81 $51	Q	113 $71	q
18 $12	CTRL R	Device Control 2	50 $32	2	82 $52	R	114 $72	r
19 $13	CTRL S	Device Control 3	51 $33	3	83 $53	S	115 $73	s
20 $14	CTRL T	Device Control 4	52 $34	4	84 $54	T	116 $74	t
21 $15	CTRL U	Negative Acknowlege	53 $35	5	85 $55	U	117 $75	u
22 $16	CTRL V	Synchronous Idle	54 $36	6	86 $56	V	118 $76	v
23 $17	CTRL W	End Trans. Block	55 $37	7	87 $57	W	119 $77	w
24 $18	CTRL X	Cancel	56 $38	8	88 $58	X	120 $78	x
25 $19	CTRL Y	End of Medium	57 $39	9	89 $59	Y	121 $79	y
26 $1A	CTRL Z	Substitute	58 $3A	:	90 $5A	Z	122 $7A	z
27 $1B	CTRL [	Escape	59 $3B	;	91 $5B	[	123 $7B	{
28 $1C	CTRL \	File Separator	60 $3C	<	92 $5C	\	124 $7C	|
29 $1D	CTRL]	Group Separator	61 $3D	=	93 $5D	]	125 $7D	}
30 $1E	CTRL ^	Record Separator	62 $3E	>	94 $5E	^	126 $7E	~
31 $1F	CTRL _	Unit Separator	63 $3F	?	95 $5F	_	127 $7F	DEL

Regular and Alternate (A7) Character Fonts