

USER'S GUIDE MANUAL

for the
IMAGINATOR

MODEL I-100
RETROFIT GRAPHICS
DISPLAY BOARD

CLEVELAND CODONICS, INC.
CLEVELAND, OHIO

REV. A

Printed in the United States of America

Copyright © 1982 by CLEVELAND CODONICS, INC.

All rights reserved

No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual or otherwise, without the prior written permission of Cleveland Codonics, Inc., Cleveland, Ohio.

However, permission is granted to reproduce or abstract from the example programs supplied in the **User's Guide** section of this manual for inclusion within the user's programs.

Although every effort has been made to insure the correctness of this manual, Cleveland Codonics, Inc. assumes no responsibility for any errors that may appear in this manual. Cleveland Codonics, Inc. makes no commitment to update nor to keep current the information contained in this manual.

The information in this manual is subject to change without notice.

TABLE OF CONTENTS

INTRODUCTION	1
HOST COMMUNICATIONS REQUIREMENTS	3
WELCOME	5
GENERAL	9
COMMAND FORM AND FUNCTION, ASCII	11
COMMAND FORM AND FUNCTION, BINARY	27
EXAMPLES	41
THEORY OF OPERATION	53
REPLACEMENT PARTS	57
REFERENCES	59
MODIFICATIONS	61
APPENDIX	63
SCHEMATIC	67
WARRANTY	68

INTRODUCTION

The Imaginator is an intelligent, high efficiency, high resolution (504 by 247 pixel) graphics retrofit unit for your Heath/Zenith H/Z-19 terminal and H/Z-89 computer.

The Imaginator has its own onboard microcomputer to perform graphics processing independent of the host computer. This reduces the burden placed on the host processor and therefore improves execution speed.

A 128 character communications buffer further improves execution speed. This buffer permits the terminal and the host computer to perform their tasks asynchronously.

A graphics command may be entered by typing on the keyboard when the terminal is OFF-LINE or it may be sent via RS-232C from the host computer when the terminal is ON-LINE.

The Imaginator's transparent operation leaves all of the terminal's normal escape functions intact. The terminal's normal alphanumerics are totally independent of the Imaginator's graphics. The two displays can be overlayed on one another and may be individually altered under software control. Both alphanumeric and graphics images can be created in memory and restrained from being displayed on the screen. Once created they can be displayed instantaneously. Alternatively, the images may be displayed as they are created.

The graphics command processor (GCP) can be invoked to accept commands in either ASCII or BINARY format. ASCII mode has the advantage of easy user implementation of the graphics command language. All of the commands can be directly output by high level language programs which are executed in the host computer (e.g., PL/I, FORTRAN, PASCAL, BASIC, and of course ASSEMBLY languages). Standard, off-the-shelf, interpreters and compilers are all that are required (those languages need not have any special graphics instructions). No machine language driver programs are required.

The BINARY mode has the advantage of high efficiency. A minimum of information must be sent to specify an operation. Again, no special interpreters or compilers are required but machine language drivers are suggested (even these are not required) for efficiency.

An additional memory-mapped socket is provided for memory expansion. Up to 16K of E/P/ROM can be mounted and addressed by the GCP, or 8K of E/P/ROM and 8K of R/W RAM can be used. Custom programs can be downloaded from the host computer into this memory for fast independent execution.

GRAPHICS INSTRUCTION SET

EnterGraphicsMode
MoveTo (X,Y)
PointAt (X,Y)
LineTo (X,Y)
AreaTo (X,Y)
PriLineStyle (Z)
 30 Unique styles
SecLineStyle (Z)
 30 Unique styles
LineType (Z)
 On
 Off
 Complement
 Read Bit
 Toggle to Alternate LineStyle at Boundary
 Read Byte
DisplayToggle (Z)
 Enable/Disable Graphics
 Enable/Disable Alphanumerics
 Erase Graphics
 or any of the eight combinations
BringInProgram (Z₀,Z₁,...,Z₁₂₇)
JumpToProgram
ExitGraphicsMode

Cleveland Codonics, Inc. reserves the right to discontinue products and to change specifications at any time without incurring any obligation to incorporate new features in products previously sold.

HOST COMMUNICATIONS REQUIREMENTS

When operating at high baud rates, the graphics terminal will generally lag behind the host computer if asked to execute a succession of commands with long execution times (e.g., Erase, AreaTo, and LineTo commands). The Graphics Command Processor (GCP) will set the Request To Send RS-232C line false when the terminal's input communications buffer is nearly full, preventing a loss of data resulting from a buffer overflow. (The terminal's bell will tone to indicate a loss of data.) The GCP will reset the Request To Send line true when the buffer is ready to receive additional data.

Therefore, it is important that the host computer or MODEM is configured to respond to this signal. (The terminal needs no modification because it is manufactured with hardware handshaking capabilities.) A true RS-232C configuration will work fine, but often the typical RS-232C's handshaking portions are incomplete. Pin 4 of the 25-pin "D" connector on the back panel of the terminal is the Request To Send line (defined as Clear To Send at the computer end). A physical wire must connect the terminal's pin 4 with the computer's (MODEM's) pin 4.

The UARTs used in the host's RS-232C serial ports fall in two categories. Some UARTs, such as the INTEL 8251 Universal Synchronous / Asynchronous Receiver/Transmitter, respond directly to the Clear To Send signal. A high or low on the Clear To Send line with this type of UART will electronically disable or enable transmissions. This type of UART requires no further modifications.

The other type of UART has a software flag that represents the Clear To Send signal. Normally, the computer's operating system's Basic Input/Output System (BIOS) is responsible for interfacing with the serial port hardware. Generally, the BIOS will check to see if the transmitter is ready (TxRDY) before loading the UART with a character to transmit to the terminal. To add hardware handshaking, simply modify the BIOS to check the Clear To Send flag also. That is, make sure that TxRDY AND Clear To Send are both true before loading the UART with a new character to transmit.

Without this hardware handshaking, it is the programmer's responsibility to add software timing delays to prevent a buffer overflow.

Hardware handshaking will in no way detrimentally effect the operation of any of your existing programs. Software handshaking is still present when running the terminal in its standard alphanumeric mode. Assuming that the process executing in the host computer understands ctrl-S (stop transmitter) and ctrl-Q (start transmitter), it is possible to suspend graphics program output by typing a ctrl-S on the keyboard, when the terminal is on line.

The GCP supports only one directional hardware handshaking. It will send signals to control the host's serial channel transmitter, but will not respond to signals sent to the terminal's serial channel transmitter from the host.

This page intentionally left blank.

WELCOME

Welcome to the field of computer graphics. The human mind is the greatest known graphics processor in existence. Thoughts can be instantly conveyed by means of a picture. And in this time of information upheaval graphics is needed more than ever to enable one to assimilate it all. As a result computer graphics is one of the fastest growing disciplines in computer science.

Try typing in and executing the following demonstration programs. (We are assuming that you have access to a **BASIC** interpreter or compiler.)

Note that the Imaginator is assumed to be installed in a terminal that is serving as the console.

In case of error. If nothing appears to happen or something very strange happens once you have typed the RUN command give the terminal a hardware reset (right-SHIFT RESET) followed by a ctrl-C (or whatever command stops program execution in your particular version of **BASIC**). Type LIST and then double check the program for typing errors.

Enter and run this program first:

DEMONSTRATION 1.

```
00010 DEFINT X,Y
00020 PRINT CHR$(27);"1"
00030 PRINT "I0,N255,D3"
00040 PRINT "M";0;125
00050 FOR X=0 TO 500 STEP 2
00060 Y=100*SIN(X/13.27)+125
00070 PRINT "L";X;Y
00080 NEXT X
00090 PRINT "D6,E"
00100 STOP
```

Here's another one.

DEMONSTRATION 2.

```

00010  DEFINT A-Z
00020  PRINT CHR$(27);"1"
00030  PRINT "D3,I2,N255"
00040  FOR J = 1 TO 10
00050    X = 251
00060    Y = 126
00070    PRINT "P";X;Y
00080    FOR I = 0 TO 80 STEP 8
00090      X = 250-I
00100      Y = 125
00110      PRINT "L";X;Y
00120      X = 254
00130      Y = 121-I
00140      PRINT "L";X;Y
00150      X = 258 + I
00160      Y = 125
00170      PRINT "L";X;Y
00180      X = 250
00190      Y = 133 + I
00200      PRINT "L";X;Y
00210    NEXT I
00220  NEXT J
00230  PRINT "D6,E"
00240  STOP

```

Too simple? Try this one if you have some time.

This program requires the host computer to calculate over 30,000 coordinates so it takes quite a while to complete. Start this program and relax, read the rest of the **User's Guide**.

DEMONSTRATION 3.

```

00010  DEFINT F,I,L,N,O,X,Y
00020  DIM L(302)
00030  PRINT CHR$(27);"1";"D3,N255,I0,M0,0,A500247,I1"
00040  FOR I = 0 TO 301
00050    L(I) = 0
00060  NEXT I
00070  PRINT "P050023"
00080  OY = 23
00090  OX = 50
00100  FOR Y = 0 TO 100
00110    FOR X = 0 TO 300
00120      ZX = (X-150)*(X-150)/1790.5
00130      ZY = (Y-50)*(Y-50)/199
00140      Z = COS(ZX + ZY)/(SIN((ZX + ZY + .48)/82))
00150      NX = X + Y + 50
00160      NY = Y + Z + 20
00170      IF F = 1 THEN PRINT "M";NX;NY : F = 0 : GOTO 200

```

GRAPHDE.M03

```
00180 IF NY >= L(X + 1) THEN PRINT "P";OX,OY;"L";NX,NY : GOTO 200
00190 IF NY <= L(X + 1) THEN L(X) = L(X + 1) : GOTO 210
00200 L(X) = NY
00210 OX = NX
00220 OY = NY
00230 NEXT X
00240 F = 1
00250 NEXT Y
00260 PRINT "D6,E"
00270 STOP
```

This page intentionally left blank.

GENERAL

COMPUTER GRAPHICS BASICS

This is an introduction to the general concepts of computer graphics for those who may be unfamiliar with the field. Basically, a graphics terminal in its simplest form need only execute two commands: `MoveTo(X,Y)` and `LineTo(X,Y)`. A superset of commands can be formed from these two primitives.

Consider for the moment a hardcopy XY plotter. The `MoveTo(X1,Y1)` command in this case will lift the pen off the paper and move it to the absolute coordinate (X₁,Y₁). The `LineTo(X2,Y2)` command will drop the pen onto the paper and move it in a straight line to the absolute coordinate (X₂,Y₂) (i.e., it would draw a line segment from (X₁,Y₁) to (X₂,Y₂)).

In a CRT style graphics terminal the commands would be executed in a similar manner. The `MoveTo(X1,Y1)` command will move a virtual pointer to the absolute screen coordinate (X₁,Y₁). Nothing is written on the screen. The `LineTo(X2,Y2)` command writes a straight line on the screen from the absolute coordinate (X₁,Y₁) to the absolute coordinate (X₂,Y₂) by turning on the appropriate pixels (picture elements). Almost any geometrical shape can be created by a sequence of `MoveTo` and `LineTo` commands (e.g., a circle can be approximated by a many sided polygon). Several other primitive utility commands are convenient, such as some means to erase the screen and a command to reinitialize the graphics terminal. To take some of the burden from the applications programmer, this primitive instruction set is usually expanded.

IMAGINATOR SPECIFICS

The graphics screen memory is composed of 131072 bit arranged in a 512 by 256 array (although only 504

by 247 are user accessible and displayed). The positive X axis (horizontal axis) originates at the left of the screen and terminates at the right. The positive Y axis (vertical axis) originates at the bottom of the screen and ends at the top. Therefore, the origin (0,0) is located at the lower left of the screen. Since the alphanumeric screen is 80 characters wide and the graphics screen is 63 characters wide, the graphics screen's left starts at the alphanumeric's 9th character position.

To view the entire graphics screen, enable the 25th line, `ESC x 1` (`ESC [1 h` if in ANSI mode).

When the terminal is reset, either when powered up, a keyboard reset **right SHIFT-RESET**, or a software reset `ESC z` (`ESC [z` if in ANSI mode) the terminal will perform as though it were unmodified. It will execute all of the escape functions it did before the addition of the Imaginator—the functional existence of the Imaginator is transparent to the user. (At this time the graphic's video RAM will be cleared, and the line type will be ON; the primary line style will be solid, the secondary line style will be blank, and the virtual pointer will be assigned as (0,0).)

To invoke the graphics command processor (GCP), an "EnterGraphicMode" escape sequence is required. (When graphics or "EnterGraphics Mode" is referred to in this manual it should be connoted as a reference to the capabilities of the Imaginator, not the 33 special symbols stored in the terminal's character generator.)

The GCP can be invoked to accept commands in either ASCII mode or as seven bit binary words (BINARY mode). Both forms of each command will be accompanied by a functional description.

A command may be entered either by typing on the keyboard when the terminal is OFF LINE or it may be sent via RS-232C from the host computer when the terminal is ON LINE.

There is no good way to abort a command midway (e.g., delete and backspace won't erase a command). Obviously, a keyboard reset **right SHIFT-RESET** is one way to clear a half-created command, but is rather drastic. The GCP expects to receive commands and data in certain fixed sequences; once a command sequence is started it must be completed.

COMMAND FORM AND FUNCTION, ASCII

ASCII COMMAND FORMATS

A complete description of the form and function of each command follows.

Upper case characters **A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P** represent commands (some of these are unassigned).

X represents the absolute horizontal coordinate. It must be an integer between 0 and 999 inclusively, although it will be truncated to 503 if greater than 503.

64x8=

Y represents the absolute vertical coordinate. It must be an integer between 0 and 999 inclusively, although it will be truncated to 246 if greater than 246.

Z represents an operand. It must be an integer between 0 and 999 inclusively.

[opt. delim] represents an optional delimiter. A delimiter here is not required but may be included. If included it may be any number of ASCII characters except the characters 0,1,2,3,4,5,6,7,8,9.

[delim] represents a delimiter. A delimiter here is mandatory unless three consecutive numerals precede it (a delimiter is automatically assumed after a three digit number, additional delimiters are optional). The delimiter may be any ASCII character except 0,1,2,3,4,5,6,7,8,9.

It will be assumed in the remainder of this manual that the language **BASIC** is understood by the reader. However, only the most rudimentary of **BASIC** commands will be used to prevent undue confusion to a novice.

The following examples illustrate a typical command format.

A PointAt command: **P** [opt. delim] **X** [delim] **Y** [delim] may be created in **BASIC** as:

`PRINT "P";X;Y` The space will serve as the delimiter.

or

`PRINT "P",X,Y` The tab will serve as the delimiter, (note that in some **BASICs** a tab may be represented as a series of spaces. This format would then be inefficient.)

or

`PRINT "P"`
`PRINT X`
`PRINT Y` The carriage return/line feed will serve as the delimiter.

or

If **X** and **Y** are constants such as **X=25** and **Y=39**

`PRINT "P";25;39` The space will again serve as the delimiter.

or

`PRINT "P025039"` The leading zeros create three digit numbers so the delimiter is automatically inserted.

EnterGraphicsMode, ASCII

Command form: ESC 1

Command function:

This command signals the GCP to interpret all future information as graphics command/data. No graphics attributes are reinitialized. Commands and data will now be assumed to consist of the ASCII characters **A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P** and **0,1,2,3,4,5,6,7,8,9** respectively. ASCII mode has the advantage of easy user implementation of the graphics command language. All of the commands can be directly output by high level language programs which are executed in the host computer. No machine language driver programs are required. The ASCII mode has the disadvantage of inefficiency. On the average, twice as many characters must be sent to the terminal than in binary mode to perform the same operation. The disadvantage would be most evident when communications speed, rather than vector drawing speed or host processor speed, is the effective bottleneck.

EXAMPLE: 10 PRINT CHR\$(27);"1"

MoveTo (X,Y), ASCII

Command form: M [opt. delim] X [delim] Y [delim]

Command function:

The virtual pointer is assigned the absolute coordinate (X,Y). Nothing is written to the screen nor can it be interrogated.

EXAMPLE:

```
10 DEFINT X,Y
.
.
.
20 X = 25
30 Y = 210
40 PRINT "M";X;Y
```

PointAt (X,Y), ASCII

Command form: P [opt. delim] X [delim] Y [delim]

Command function:

The virtual pointer is assigned the absolute coordinate (X,Y). The Pattern byte (see the LineStyle commands) is rotated one position; if the carry contains a 0, the command is treated as a MoveTo, command. If the carry contains a 1, the pixel is interacted with according to the pending line type (see LineType command).

EXAMPLE:

```
10 DEFINT X,Y
.
.
.
20 X = 25
30 Y = 210
40 PRINT "P";X;Y
```

LineTo (X,Y), ASCII

Command form: L [opt. delim] X [delim] Y [delim]

Command function:

A line is drawn from, but not including, the virtual pointer's currently assigned absolute coordinate to the absolute coordinate (X,Y). The line drawn is subject to the current line style and line type attributes. This command will emulate a MoveTo command if the line style is 00000000 (execution time will be considerably longer however). At the completion of this command, the virtual pointer is assigned the absolute coordinate (X,Y).

EXAMPLE:

```
10 DEFINT X,Y
.
.
.
20 X = 25
30 Y = 210
40 PRINT "L";X;Y
```


AreaTo (X,Y), ASCII

Command form: A [opt. delim] X [delim] Y [delim]

Command function:

The area inside a regular rectangle is filled. The rectangle is defined as having the virtual pointer's currently assigned absolute address as one vertice and the absolute coordinate (X,Y) as the diagonally opposite vertice. Starting at, but not including, the virtual pointer's currently assigned absolute coordinate, a horizontal line is drawn to the opposite side of the rectangle. When possible, a second line starting at the original side of the rectangle is drawn adjacent to the first line (a rectangle with a height of 1 will only accept one line). This procedure is repeated until the rectangle is filled. The line drawn is subject to the current line style and line type attributes. This command will behave as a MoveTo command if the line style is 00000000 (execution time will be considerably longer however). At the completion of this command the virtual pointer is assigned the absolute coordinate (X,Y).

EXAMPLE:

```
10 DEFINT X,Y
.
.
.
20 X = 25
30 Y = 210
40 PRINT "A";X;Y
```

PriLineStyle (Z), ASCII

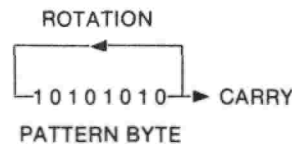
Command form: N [opt. delim] Z [delim]

Where: Z is a number between 0 and 999 inclusively. This number is converted to binary format whose least significant 8 bits are used to define the Primary Pattern.

Command function:

This command permits dashed or dotted lines to be automatically generated by the GCP.

Preceding any write to the graphics display, the pending Pattern byte is rotated one position. The least significant bit is rotated into the carry and is used to determine whether screen interaction is permitted or not. A logical 1 in the Pattern represents permission to interact with the pixel; a 0 disables interaction. The pending Pattern byte is then updated with the new rotated pattern. The least significant bit is the first to be tested to determine if interaction should occur. Therefore, the eight bit line style pattern is repetitively traced to the screen when drawing a line.



The LineStyle and LineType commands are totally independent of one another. The line style will equally effect any line type attribute (except READ BIT and READ BYTE). For instance, a line drawn with a 10101010 line style and a complement line type will complement every other pixel.

When short line segments are used to construct long lines (e.g., curves), they should be sent in a consecutive order. There is no guarantee that a line segment patched into the middle of an existing line will have a perfectly matched line style sequence. Of course, it is possible to reset the sequence by executing another LineStyle command.

The pending line style pattern is always reset to Primary when entering any graphics command.

Any portion of the graphics display may be selectively erased by executing an AreaTo command with a line style of 11111111 and an OFF line type.

EXAMPLE: 10 PRINT "N255"

SecLineStyle (Z), ASCII

Command form: O [opt. delim] Z [delim]

Where: Z is a number between 0 and 999 inclusively. This number is converted to binary format whose least significant 8 bits are used to define the Secondary Pattern.

Command function:

Identical to PriLineStyle (Z), ASCII

EXAMPLE: 10 PRINT "O170"

LineType (Z), ASCII

Command form: I [opt. delim] Z [delim]

Where:	Z	PIXEL ACTION
	0	ON
	1	OFF
	2	COMPLEMENT
	3	READ BIT
	4	TOGGLE TO ALTERNATE LINSTYLE AT BOUNDARY
	5	READ BYTE

Command function:

This command sets the type of line to be drawn, (Note, that a point is considered a short line and an area is considered a long line). Consider each pixel of the line individually for now.

The different line types are explained below.

ON—the pixel is turned on.

OFF—the pixel is turned off (i.e., erased).

COMPLEMENT—the pixel is complemented (i.e., the pixel is turned on if it was off and it is turned off if it was on).

READ BIT—The pixel is interrogated to determine whether it is on or off but is not otherwise effected. An ASCII 0 or 1 followed by a carriage return is transmitted to the host computer for a pixel that is respectively off or on.

This line type has some special restrictions.

This line type can only be used in conjunction with a PointAt command. LineTo and AreaTo commands will imitate a MoveTo command.

Note that if the terminal is OFF LINE this attribute will perform no function except that the PointAt, LineTo, or AreaTo command will act as a MoveTo command.

The line style will act as if it were set to solid (1111111) regardless of its actual value. (See LineStyle command). This is to prevent the host computer from getting trapped in an eternal wait loop for a terminal response if the line style contains a 0.

The process executing in the host computer that is responsible for reading the data sent by the terminal must be fast enough to keep pace. The terminal will transmit the data as fast as the baud rate selected will permit.

It is important that the host computer does not echo the terminal response (0 or 1 followed by a carriage return) back to the terminal. An echoed response will be treated by the GCP as command/data information. (This is really only true if the GCP is in BINARY mode, because in ASCII mode the 0 or 1 will be received when the GCP is expecting an opcode (A—P) and will therefore be assumed to be a delimiter.) See the **Examples** section of this manual to see how this can be implemented.

TOGGLE TO ALTERNATE LIFESTYLE AT BOUNDARY—This line type is a very simple, and therefore limited, algorithm that may be used for filling irregular polygons.

As the line is scanned, each pixel is interrogated in turn to determine whether it is on or off. If it is off it is written to according to the pending line style. A single on pixel will be left untouched, but the current line style pattern is exchanged with the alternate Pattern. For instance, if the line style is currently loaded with the Primary Pattern it will be reloaded with the Secondary Pattern, or if the Lifestyle is currently loaded with the Secondary Pattern it will be reloaded with the Primary Pattern. If two or more adjacent pixels are on they will be left untouched and line style pattern will NOT be exchanged. At the completion of the LineTo or AreaTo command the line style is reloaded with the Primary Pattern.

READ BYTE—The display byte is read and converted from binary to hexadecimal. The ASCII representation of this hexadecimal number is transmitted to the host computer. Display bytes are defined as 8 consecutive horizontal pixel locations. The beginning of a display byte is (X,Y) where X is 0,8,16,...,496 and Y is any integer between 0 and 246, inclusively. Each display byte is redundantly addressed by 8 coordinates. For example, to access the display byte beginning at (0,0) any of the following coordinates could be used: (0,0), (1,0), (2,0), (3,0), (4,0), (5,0), (6,0), or (7,0). The pixel at the beginning of the display byte is the least significant and the pixel at the beginning +8 is the most significant. Notice that this means that, visually, a pattern on the screen will appear in reverse significance with respect to its hexadecimal representation.

Leading zeros are transmitted (not suppressed).

This line type has some special restrictions.

This line type can only be used in conjunction with a PointAt command. LineTo and AreaTo commands will imitate a MoveTo command.

Note that if the terminal is OFF LINE this attribute will perform no function except that the PointAt, LineTo or AreaTo command will act as a MoveTo command.

The line style will act as if it were set to solid (11111111) regardless of its actual value. (See LineStyle command) This is to prevent the host computer from getting trapped in an eternal wait loop for a terminal response if the line style contains a 0.

The process executing in the host computer that is responsible for reading the data sent by the terminal must be fast enough to keep pace. The terminal will transmit the data as fast as the baud rate selected will permit.

It is important that the host computer does not echo the terminal response (00 to FF followed by a carriage return) back to the terminal. An echoed response will be treated by the GCP as command/data information. See the **Examples** section of this manual to see how this can be implemented.

DisplayToggle (Z), ASCII

Command form: D [opt. delim] Z [delim]

Where:	Z	ENABLE ALPHA	ENABLE GRAPHICS	ERASE GRAPHICS
	0	NO	NO	NO
	1	NO	NO	YES
	2	NO	YES	NO
	3	NO	YES	YES
	4	YES	NO	NO
	5	YES	NO	YES
	6	YES	YES	NO
	7	YES	YES	YES

Command function:

This command has two distinct functions. One function is to permit the user to block or not block the display of alphanumeric or graphics information to the entire screen. The other function of this command is to erase the entire graphics display memory. This command stays in effect even after executing an ExitGraphicsMode command.

EXAMPLE: 10 PRINT "D3"

This command would disable alphanumerics, enable graphics and erase the previous image.

BringInProgram (Z0), (Z1), ... ,(Z127), ASCII

Command form: B [opt. delim] Z0 [opt. delim] Z1 [opt. delim], ... ,Z127 [opt. delim]

Where: [opt. delim] in this case is any ASCII character except 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F.

AND

Where: Z is a double digit hexadecimal number between 00 and FF, inclusively. A leading zero must be present if a single digit number (i.e., 03 not 3). However, do not insert a leading zero in front of a two digit number (i.e., FF not 0FF).

Command function:

This command loads 128 bytes of data (Z0-Z127) into the expansion R/W RAM U9B. The data is converted from hexadecimal to binary format prior to loading into R/W RAM. Z0 is loaded into memory at address C001H, Z1 is loaded into memory at C002H, etc. After the 128th byte is loaded control is returned to the GCP for the next command.

This command is only useful if R/W RAM is mounted at U9B. Beware that once a BringInProgram command is initiated the GCP will expect at least 256 characters before accepting new commands (this is true regardless of whether R/W RAM is present at U9B or not).

EXAMPLE:

```
10 PRINT "B"
20 PRINT "00"
30 PRINT "00"
40 PRINT "00"
50 PRINT "C3"
60 PRINT "04"
70 PRINT "C0"
.
.
.
1290 PRINT "00"
```

This example of data entry is correct with regard to format but is quite inflexible and therefore not advocated as a good programming technique.

JumpToProgram, ASCII

Command form: J

Command function:

This command transfers control from the GCP to the program residing in U9B. Transfer is accomplished by a JMP (JUMP) to address C004H. Control may be given back to the GCP by a RET (RETURN) statement.

Before the transfer is made a test pattern is written to location C000H and then read back. The pattern must match or no transfer is permitted and control is returned to the GCP. Therefore, physical memory must be mounted at U9B and it must be valid at C000H. This prevents inadvertently jumping to a nonexistent program resulting in a runaway processor.

EXAMPLE: 10 PRINT "J"

ExitGraphicsMode, ASCII

Command form: E

Command function:

This command instructs the GCP to release control back to normal alphanumeric processing. All previously set graphics attributes will remain valid (i.e., no attributes revert back to default or reset values).

EXAMPLE: 10 PRINT "E"

This page intentionally left blank.


COMMAND FORM AND FUNCTION, BINARY

BINARY COMMAND FORMATS

The binary command formats are described with each command. X and Y coordinates and Z parameters are represented in binary notation. The ASCII character representing the binary number is transmitted to the terminal. For example, examine the bytes (P is the parity bit):

P 0 1 0 0 1 0 0	is represented by the ASCII character \$
P 0 1 0 0 0 0 0	is represented by the ASCII Space
P 1 1 1 1 1 1 1	is represented by the ASCII Delete

See the **Appendix** for binary to ASCII conversions.

More stringent conditions are placed on the syntax of commands in BINARY mode than in ASCII mode. In general delimiters are not required and are not permitted, but there is one exception. The binary codes P0000000 through P0001111 can serve as NOP (no operation) commands when used as opcodes. This permits the inclusion of dummy carriage-return and line-feed characters in transmissions. This is required because some high level languages insert their own carriage-returns regardless of whether the programmer requested one or not. For instance, some releases of **BASIC** automatically insert a carriage-return and line-feed if the user does not specify one before 255 consecutive characters are transmitted. Unpredictable results may result since this automatic carriage-return may occur when the GCP is expecting a valid operand. Therefore, it is important for the programmer to force occasional carriage-returns when the GCP expects an opcode (if they are to be ignored) before the automatic one is triggered. 

It is important to note that the binary codes P0000001 through P0001111 are valid when used as operands (P0000000 is never used because nulls are filtered out by the terminal and most operating systems)

Since the binary code P0000000 can never be used a simple data conversion needs to be performed when using MoveTo, PointAt, LineTo and AreaTo commands. The respective subroutines need to add an offset of 8 to X and an offset of 2 to Y.

- For instance, if the programmer wanted to MoveTo(0,0) a GOSUB 2000 would be executed (see MoveTo, BINARY).
- The programmer would set X=0 and Y=0 before the call.
- The MoveTo subroutine would effectively add 8 to X and add 2 to Y.
- The GCP will then subtract 8 from X and 2 from Y once it receives them.

Of course, these syntax restrictions only apply when in BINARY mode, these restrictions do not exist when in ASCII mode or when in the terminal's standard alphanumeric mode. The driver routines presented take all of these requirements into account.

To gain the most efficiency BINARY mode was really designed to be driven by assembly language routines. The routines should have the following features.

- They need to convert the X and Y coordinates or the Z parameter to the proper binary format.
- The command identifier needs to be appended to the opcode.
- The ASCII character that represents the binary word needs to be formed.
- The routines should not echo back ANY of the characters that are sent from the terminal. (As stated above, some characters can be echoed back without problem if the GCP expects an opcode, but it is simpler to unconditionally avoid echoing back any characters.) This is particularly true for data received from the terminal when the READ line types are set. Data should be read and processed but not echoed back.
- The routines should return control back to the calling program once the command and its data have been transmitted. When checking the serial channel status remember that the Imaginator expects that the Clear To Send signal is being monitored.

Example driver routines are included. More efficient driver routines can be written in assembly language but **BASIC** was chosen to help clarify the principles involved.

A note about the driver routines. These routines perform no X, Y, or Z parameter limit checking. For example, these routines would accept a value greater than 503 for the X coordinate without complaint and would pass an incorrect value to the terminal.

EnterGraphicsMode, BINARY

Command form: ESC 0

Command function:

This command signals the GCP to interpret all future information as graphics command/data. No graphics attributes are reinitialized. Commands and data will now be assumed to be seven-bit binary words (the parity bit is not used). The BINARY mode has the advantage of high efficiency because a minimum of information must be sent to specify an operation. Binary mode has the disadvantage of requiring the information to be condensed into a compact form by the host computer. Actually, this is a rather simple process, it requires only short subroutines. (Since the condensed information can cover the complete range from 00000001B to 01111111B inclusively, another problem may arise if the Basic Input Output System (BIOS) of the host's operating system filters out or modifies specific values. For instance, a DELETE may be changed to a BACKSPACE-SPACE-BACKSPACE. Or a TAB may be changed to a string of 8 spaces. The GCP would misinterpret this corrupted data with unpredictable results.)

EXAMPLE: 10 PRINT CHR\$(27);"0"

MoveTo (X,Y), BINARY

Command form:	Command Opcode	<u>7 6 5 4 3 2 1 0</u> P 1 1 0 1 X ₂ X ₁ X ₀
	First Operand	<u>7 6 5 4 3 2 1 0</u> P Y ₀ X ₆ X ₇ X ₆ X ₅ X ₄ X ₃
	Second Operand	<u>7 6 5 4 3 2 1 0</u> P Y ₇ Y ₆ Y ₅ Y ₄ Y ₃ Y ₂ Y ₁

P - parity

Command Function:

Identical to MoveTo (X,Y), ASCII

```

EXAMPLE:  10 DEFINT O,X,Y
              20 X = 25
              30 Y = 210
              40 GOSUB 2000
              .
              .
              .
              2000 REM MOVE TO COMMAND BINARY DRIVER
              2010 REM
              2020 REM X = X COORDINATE
              2030 REM Y = Y COORDINATE
              2040 REM
              2050 OPCODE = (X AND 7) OR &H68
              2060 OP1 = ((X AND NOT 7)/8 AND 63) + 1 + (Y AND 1)*64
              2070 OP2 = (Y AND 254)/2 + 1
              2080 PRINT CHR$(OPCODE);CHR$(OP1);CHR$(OP2)
              2090 RETURN
  
```

PointAt (X,Y), BINARY

Command form:	Command Opcode	<u>7 6 5 4 3 2 1 0</u> P 0 1 1 0 X ₂ X ₁ X ₀
	First Operand	<u>7 6 5 4 3 2 1 0</u> P Y ₀ X ₆ X ₇ X ₆ X ₅ X ₄ X ₃
	Second Operand	<u>7 6 5 4 3 2 1 0</u> P Y ₇ Y ₆ Y ₅ Y ₄ Y ₃ Y ₂ Y ₁

P - parity

Command function:

Identical to PointAt (X,Y), ASCII

```

EXAMPLE:  10 DEFINT O,X,Y
              20 X = 25
              30 Y = 210
              40 GOSUB 3000
              .
              .
              .
              3000 REM POINT AT COMMAND BINARY DRIVER
              3010 REM
              3020 REM X = X COORDINATE
              3030 REM Y = Y COORDINATE
              3040 REM
              3050 OPCODE = (X AND 7) OR &H30
              3060 OP1 = ((X AND NOT 7)/8 AND 63) + 1 + (Y AND 1)*64
              3070 OP2 = (Y AND 254)/2 + 1
              3080 PRINT CHR$(OPCODE);CHR$(OP1);CHR$(OP2)
              3090 RETURN
  
```

LineTo (X,Y), BINARY

Command form:	Command Opcode	<u>7 6 5 4 3 2 1 0</u> P 1 1 0 0 X ₂ X ₁ X ₀
	First Operand	<u>7 6 5 4 3 2 1 0</u> P Y ₆ X ₆ X ₇ X ₆ X ₅ X ₄ X ₃
	Second Operand	<u>7 6 5 4 3 2 1 0</u> P Y ₇ Y ₆ Y ₅ Y ₄ Y ₃ Y ₂ Y ₁

P - parity

Command function:

Identical to LineTo (X,Y), ASCII

EXAMPLE:

```

10 DEFINT O,X,Y
20 X = 25
30 Y = 210
40 GOSUB 4000
.
.
.
4000 REM LINE TO COMMAND BINARY DRIVER
4010 REM
4020 REM X = X COORDINATE
4030 REM Y = Y COORDINATE
4040 REM
4050 OPCODE = (X AND 7) OR &H60
4060 OP1 = ((X AND NOT 7)/8 AND 63) + 1 + (Y AND 1)*64
4070 OP2 = (Y AND 254)/2 + 1
4080 PRINT CHR$(OPCODE);CHR$(OP1);CHR$(OP2)
4090 RETURN

```


AreaTo (X,Y), BINARY

Command form:	Command Opcode	<u>7 6 5 4 3 2 1 0</u>
		P 1 0 1 1 X ₂ X ₁ X ₀
	First Operand	<u>7 6 5 4 3 2 1 0</u>
		P Y ₀ X ₆ X ₇ X ₆ X ₅ X ₄ X ₃
	Second Operand	<u>7 6 5 4 3 2 1 0</u>
		P Y ₇ Y ₆ Y ₅ Y ₄ Y ₃ Y ₂ Y ₁

P - parity

Command function:

Identical to AreaTo (X,Y), ASCII

EXAMPLE:

```

10 DEFINT O,X,Y
20 X = 25
30 Y = 210
40 GOSUB 5000
.
.
.
5000 REM AREA TO COMMAND BINARY DRIVER
5010 REM
5020 REM X = X COORDINATE
5030 REM Y = Y COORDINATE
5040 REM
5050 OPCODE = (X AND 7) OR &H58
5060 OP1 = ((X AND NOT 7)/8 AND 63) + 1 + (Y AND 1)*64
5070 OP2 = (Y AND 254)/2 + 1
5080 PRINT CHR$(OPCODE);CHR$(OP1);CHR$(OP2)
5090 RETURN

```

PriLineStyle (Z), BINARY

Command form:	Command Opcode	<u>7 6 5 4 3 2 1 0</u>
		P 1 1 1 0 * Z ₇ Z ₀
	First Operand	<u>7 6 5 4 3 2 1 0</u>
		P Z ₇ Z ₆ Z ₅ Z ₄ Z ₃ Z ₂ 1

* - don't care

P - parity

Command function:

Identical to PriLineStyle (Z), ASCII

```

EXAMPLE:  10 DEFINT O,Z
             20 Z = 3
             30 GOSUB 8000
             .
             .
             .
             8000 REM PRIMARY LINE STYLE COMMAND BINARY DRIVER
             8010 REM
             8020 REM Z = ATTRIBUTE
             8030 REM
             8040 OPCODE = (Z AND 3) OR &H70
             8050 OP1 = (Z AND 254)/2 OR 1
             8060 PRINT CHR$(OPCODE);CHR$(OP1)
             8070 RETURN

```

SecLineStyle (Z), BINARY

Command form:	Command Opcode	<u>7 6 5 4 3 2 1 0</u>
		P 1 1 1 1 * Z ₁ Z ₀
	First Operand	<u>7 6 5 4 3 2 1 0</u>
		P Z ₇ Z ₆ Z ₅ Z ₄ Z ₃ Z ₂ 1

* - don't care

P - parity

Command function:

Identical to SecLineStyle (Z), ASCII

EXAMPLE:

```

10 DEFINT O,Z
20 Z = 3
30 GOSUB 9000
.
.
.
9000 REM SECONDARY LINE STYLE COMMAND BINARY DRIVER
9010 REM
9020 REM Z = ATTRIBUTE
9030 REM
9040 OPCODE = (Z AND 3) OR &H78
9050 OP1 = (Z AND 254)/2 OR 1
9060 PRINT CHR$(OPCODE);CHR$(OP1)
9070 RETURN

```

LineType (Z), BINARY

Command form: Command Opcode 7 6 5 4 3 2 1 0
 P 1 0 0 1 Z₂ Z₁ Z₀

P - parity

Z ₂	Z ₁	Z ₀	PIXEL ACTION
0	0	0	ON
0	0	1	OFF
0	1	0	COMPLEMENT
0	1	1	READ BIT
1	0	0	TOGGLE TO ALTERNATE LINE STYLE AT BOUNDARY
1	0	1	READ BYTE

Command function:

Identical to LineType (Z), ASCII

```
EXAMPLE:  10 DEFINT O,Z
           20 Z = 3
           30 GOSUB 7000
           .
           .
           .
           7000 REM LINE TYPE COMMAND BINARY DRIVER
           7010 REM
           7020 REM Z = ATTRIBUTE
           7030 REM
           7040 OPCODE = Z OR &H48
           7050 PRINT CHR$(OPCODE)
           7060 RETURN
```

DisplayToggle (Z), BINARY

Command form:	Command Opcode	7 6 5 4 3 2 1 0
		P 0 1 0 0 A B C

P - parity

1 is logical true

A - Enable Alphanumerics

B - Enable Graphics

C - Erase Graphics

Command function:

Identical to DisplayToggle (Z), ASCII

EXAMPLE:

```

10 DEFINT O,Z
20 Z = 3
30 GOSUB 6000

.
.
.
6000 REM DISPLAY TOGGLE COMMAND BINARY DRIVER
6010 REM
6020 REM Z = ATTRIBUTE
6030 REM
6040 OPCODE = Z OR &H20
6050 PRINT CHR$(OPCODE)
6060 RETURN

```

BringInProgram (Z0), (Z1),..., (Z127), BINARY

Command form:	Command Opcode	<u>7 6 5 4 3 2 1 0</u>
		P 0 0 1 0 * * *

* - don't care

P - parity

Command function:

Identical to BringInProgram (Z0), (Z1),..., (Z127), ASCII

```

EXAMPLE:  10 DEFINT O
          20 GOSUB 10000
          .
          .
          .
          10000 REM BRING IN PROGRAM COMMAND BINARY DRIVER
          10010 OPCODE = &H10
          10020 PRINT CHR$(OPCODE)
          10030 PRINT "76"
          10040 PRINT "F5"
          10050 PRINT "F1"
          10060 PRINT "C9"
          .
          .
          .
          11300 PRINT "00"
          11310 RETURN

```

This example of data entry is correct with regard to format but is quite inflexible and therefore not advocated as a good programming technique.

JumpToProgram, BINARY

Command form:	Command Opcode	7 6 5 4 3 2 1 0
		P 1 0 1 0 * * *

* - don't care

P - parity

Command function:

This command transfers control from the GCP to the program residing in U9B. Transfer is accomplished by a JMP (JUMP) to address C001H. This command is otherwise identical to JumpToProgram, ASCII.

EXAMPLE:

```

10 DEFINT O
20 GOSUB 12000
.
.
.
12000 REM JUMP TO PROGRAM COMMAND BINARY DRIVER
12010 OP CODE = &H50
12020 PRINT CHR$(OP CODE)
12030 RETURN

```

ExitGraphicsMode, BINARY

Command form:	Command Opcode	<u>7 6 5 4 3 2 1 0</u>
		P 0 1 0 1 * * *

* - don't care

P - parity

Command function:

Identical to ExitGraphicsMode, ASCII

EXAMPLE:

```

10 GOSUB 1000
.
.
.
1000 REM EXIT GRAPHICS MODE COMMAND BINARY DRIVER
1010 PRINT CHR$(&H28)
1020 RETURN

```


EXAMPLES

We recommend that you try some of these examples.

Hands on experience is a must for learning any new subject. Refer to the command form and function sections for details. We recommend that you become acquainted with ASCII mode first before attempting BINARY mode. There is no way to damage the terminal by accidentally giving it an invalid command, so experiment.

EXAMPLE 1

The following example is meant to be entered by typing the commands directly on the keyboard rather than sending them from a host computer. Lock down the OFF-LINE key. Type the commands as they appear, for instance, type a space where a space is shown and a carriage return when a new line appears (the space and carriage-return will serve as delimiters).

TYPE ANYTHING	Type a few random characters.
CAPS-LOCK unlocked	
ESCx1	Enable 25th line, this permits the entire graphics display to be shown.
CAPS-LOCK locked down	It makes it easier to enter the remainder of the commands.
ESC1	EnterGraphicsMode command, ASCII.
D3	DisplayToggle command, disable alphanumerics, enable graphics, and erase the previous graphics image.
10	LineType command, the line type is ON. (Note that though a hardware or software terminal RESET will reinitialize the line type to ON, it is good practice to include initialization commands such as this one in your graphics programs. This permits graphics subroutines to be relocated without being concerned about the action of previously executed routines.)

N255	PriLineStyle command, 11111111 pattern (solid). Again, it is good practice to include this type of initialization command.
O0	SecLineStyle command, 00000000 pattern (blank). Again it is good practice to include this type of initialization command.
P50 0	PointAt (X,Y) command, X = 50, Y = 0
L50 200	LineTo (X,Y) command, X = 50, Y = 200 Notice that the line was drawn before the carriage return was keyed since 200 is a three digit number (i.e., the carriage return was really not needed).
P150 0	PointAt (X,Y) command, X = 150, Y = 0 The space between the 150 and the 0 is unnecessary since 150 is three digits long.
L150100	LineTo (X,Y) command, X = 150, Y = 100
M300 0	MoveTo (X,Y) command, X = 300, Y = 0 Notice that no point is drawn.
A400 200	AreaTo (X,Y) command, X = 400, Y = 200
P0 190	PointAt (X,Y) command, X = 0, Y = 190
L500 190	LineTo (X,Y) command, X = 500, Y = 190 The line type is ON.
I1	LineType command, Line type is OFF.
P0 150	PointAt (X,Y) command, X = 0, Y = 150
L500 150	LineTo (X,Y) command, X = 500, Y = 150
I2	LineType command, Line type is COMPLEMENT
P0 110	PointAt (X,Y) command, X = 0, Y = 110
L500 110	LineTo (X,Y) command, X = 500, Y = 110
I4	LineType command, Line type is TOGGLE TO ALTERNATE LINSTYLE AT BOUNDARY.
P0 70	PointAt (X,Y) command X = 0, Y = 70
L500 70	LineTo (X,Y) command, X = 500, Y = 70 The secondary line style is 00000000 (blank).
M0 30	PointAt (X,Y) command, X=0, Y = 30
A500 0	AreaTo (X,Y) command, X = 500, Y = 0 READ BIT and READ BYTE line types only operate when the terminal is ON-LINE.

D4	DisplayToggle command, disable graphics and Enable alphanumerics.
D2	DisplayToggle command, Enable graphics and Disable alphanumerics.
D0	DisplayToggle command, Disable graphics and Disable alphanumerics.
D7	DisplayToggle command, Enable graphics, Enable alphanumerics, and Erase previous graphics image.
E	ExitGraphicsMode command
TYPE ANYTHING	Normal alpha mode.

The following examples are written in **BASIC**.

The graphics terminal will receive its commands via RS-232C from the host computer so the terminal should be ON-LINE.

EXAMPLE 2

"GRAPHIC.EX2"

The following program will draw a simple XY axis with tick marks.

```
00010  DEFINT X,Y
00020  PRINT CHR$(27);"1"
00030  PRINT "D3"
00040  PRINT "I0"
00050  PRINT "N255"
00060  FOR X = 20 TO 500 STEP 10
00070  PRINT "M";X;"103";"L";X;"98"
00080  NEXT X
00090  FOR Y = 20 TO 240 STEP 10
00100  PRINT "M";"247";Y;"L";"252";Y
00110  NEXT Y
00120  PRINT "N170"
00130  PRINT "P250 240 L250 20 P20 100 L500 100"
00140  PRINT "D6"
00150  PRINT "E"
00160  STOP
```

Line 10 defines the variables X and Y as INTEGERS. The decimal point inserted in real numbers would act as an unintentional delimiter.

Line 20 sends an EnterGraphicsMode, ASCII command, ESC 1.

Line 30 sends a DisplayToggle command that turns the alphanumeric display off, the graphics display on and erases the graphics display memory.

Line 40 is a LineType command. The line type is defined as ON.

Line 50 is a PriLineStyle command. The primary line style pattern is defined as solid (11111111).

$$128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255$$

Lines 60, 70 and 80 form a program loop that draws tick marks on the horizontal axis. Line 70 consists of a MoveTo command ("M";X;"103") and a LineTo command ("L";X;"98").

Lines 90, 100 and 110 form a program loop that draws the tick marks on the vertical axis. Line 100 consists of a MoveTo command ("M";"247";Y) and a LineTo command ("L";"252";Y).

Line 120 is a PriLineStyle command. The primary line style pattern is defined as dotted (10101010).

$$128 + 0 + 32 + 0 + 8 + 0 + 2 + 0 = 170$$

Line 130 draws the horizontal and vertical axis. It consists of a PointAt command

("P250 240) a LineTo command (L250 20) a PointAt command (P20 100) and a LineTo command (L500 100").

Line 140 is a DisplayToggle command that instructs the graphics terminal to enable both alphanumeric and graphics displays.

Line 150 is an ExitGraphicsMode command.

Line 160 is the end of execution statement.

EXAMPLE 3

This program draws 256 lines each with a different line style. Though the patterns 00010001 and 00100010 are different, they appear identical when the pattern is repeated (e.g., when drawing a long line). Using this criteria to disqualify similar patterns, there remain 30 unique pattern styles.

```

00010  DEFINT A-Z
00020  PRINT CHR$(27);"1D3,10"
00030  LS = 0
00040  FOR I = 1 TO 32
00050  FOR J = 1 TO 8
00060  PRINT "N";LS
00070  PRINT "P";0;247-J*30
00080  PRINT "L";504;247-J*30
00090  LS = LS + 1
00100  NEXT J
00110  PRINT "D3"
00120  NEXT I
00130  PRINT "D5,E"
00140  STOP

```

" GRAPHIC.EX3 "

"GRAPHIC.EX4"

EXAMPLE 4

This program demonstrates a technique for generating cross hatched patterns easily. Bar charts are a typical application.

```

00010  DEFINT A-Z
00020  PRINT CHR$(27);"1"
00030  PRINT "D3"
00040  PRINT "M350,240"
00050  PRINT "I0"
00060  PRINT "N170"
00070  PRINT "A151,10"
00080  PRINT "I2"
00090  PRINT "A351,240"
00100  PRINT "I1"
00110  PRINT "A150,10"
00120  PRINT "I2"
00130  PRINT "A351,240"
00140  PRINT "I1"
00150  PRINT "N17"
00160  PRINT "A150,10"
00170  PRINT "D6,E"
00180  STOP

```

Line 40 moves the virtual pointer to (350,240).

Line 50 sets the LineType to ON.

Line 60 sets the PriLineStyle to dotted, 10101010.

Line 70 fills a rectangular area with (350,240) and (151,10) as the diagonally opposite vertices. Since 350-151 is not an even multiple of 2 (the numerical distance between two consecutive ones in the primary pattern byte, 10101010), the pixel pattern is diagonal. If the 151 was changed to 150, a pattern of vertical lines would have been drawn.

```

010101010
101010101
010101010
101010101
010101010

```

Line 80 changes the line type to Complement.

Line 90 draws another rectangular area on top of the one just drawn. This time the difference between old and new X coordinates (351-151) is an even multiple of 2. This results in a pattern of vertical interactions. Every other pixel is complemented, since the line style is still 10101010, resulting in the complete cancellation of every other row and a filling in of the remaining rows.

010101010		c c c c c		111111111
101010101		c c c c c		000000000
010101010	+	c c c c c	=	111111111
101010101		c c c c c		000000000

Line 100 changes the line type to OFF.

Line 110 makes an area pass over the existing pattern to selectively erase specific pixels.

Lines 130-160 are more of the same.

Line 170 is the erase screen and exit command.

It probably is now evident that combining area overlays using different line types and line styles can create some complicated but interesting patterns.

EXAMPLE 5

In case you are not convinced.

```

00010  DEFINT A-Z
00020  PRINT CHR$(27);"1"
00030  PRINT "D3"
00040  PRINT "M";"350";"240"
00050  PRINT "I0"
00060  PRINT "N1"
00070  PRINT "A151,10"
00080  PRINT "A352,240"
00090  PRINT "A151,10"
00100  PRINT "A350,240"
00110  PRINT "I2"
00120  PRINT "N170"
00130  PRINT "A151,10"
00140  PRINT "A351,240"
00150  PRINT "A150,10"
00160  PRINT "A350,240"
00170  PRINT "N17"
00180  PRINT "A150,10"
00190  PRINT "A352,240"
00200  PRINT "A150,10"
00210  PRINT "D6,E"
00220  STOP

```

EXAMPLE 6

Here's a different one.

```

00010  DEFINT X,Y
00020  PRINT CHR$(27);"1"
00030  FOR L = 1 TO 2
00040  PRINT "I2,N17,D2"
00050  FOR N = 30 TO 100 STEP 5
00060  PRINT "M";0;125
00070  FOR X = 0 TO 500 STEP 63
00080  Y = 100*SIN(X/N) + 125
00090  PRINT "A";X;Y
00100  NEXT X
00110  NEXT N
00120  NEXT L
00130  PRINT "D5,E"
00140  STOP

```

Obviously this program demonstrates the overlay principle, but there is another concept here. It is the fact that when strictly the complement line type is used, there is no loss of information on the screen. No matter how many times a pixel is overlayed its original state can be determined. There are some practical implications of this. Two independent images can be overlayed to form a single image that can be totally separated later. For instance, a crosshair cursor can be drawn over an existing image if done with the complement line type. It will always remain visible because it is complemented. The original image can be restored by recomplementing the crosshair cursor. Therefore, the crosshair cursor can be scanned across the original image without destroying the image, as long as it is drawn an even number of times in the same location before it is moved.

EXAMPLE 7

This program demonstrates an efficient method of generating graphics characters. Though graphics character generation is slower than alphanumeric character generation there are some advantages:

- The characters can be defined by the programmer (e.g., Latin and Greek symbols can be produced).
- The characters can be practically any size (a 5 by 5 matrix is the minimum for a well formed upper case character).
- The characters can be oriented in any direction—horizontal, vertical, inverted, diagonally, etc.
- The characters can be positioned anywhere. For example, a label can be centered directly under a graph's axis tick mark, rather than the closest character block position.

This program was written especially for generating upper case characters defined on a 5 by 5 grid. Label orientations are limited to horizontal (from left to right) and vertical (from bottom to top) directions. Any direction is possible, but the computations would

be severely slowed down since the trigonometric functions required are slow when written in **BASIC**.

MAIN PROGRAM

```

00010  DEFINT A-Z
00020  DIM CGEN(4,5)
00030  GOSUB 21000
00040  PRINT CHR$(27)"1"
00050  PRINT "D3,I0"
00060  LABEL$ = "0123"
00070  STARTX = 5
00080  STARTY = 100
00090  LDIR = 0
00100  GOSUB 20000
00110  PRINT "D6,E"
00120  STOP

```

GRAPHICS CHARACTER GENERATOR SUBROUTINE

```

20000  IF LDIR = 0 THEN ILDIR = 1 ELSE ILDIR = 0
20010  HEIGHT = 5
20020  COUNT = 0
20030  SPACE = 2
20040  FOR CHAR = 1 TO LEN(LABEL$)
20050  CHARCODE = ASC(MID$(LABEL$,CHAR,1)) - 48
20060  FOR I = 1 TO 5
20070  PRIPAT = CGEN(CHARCODE,I)
20080  PRINT "N";PRIPAT
20090  BASEX = STARTX + COUNT*ILDIR
20100  BASEY = STARTY + COUNT*LDIR
20110  PRINT"P";BASEX;BASEY
20120  X = BASEX + HEIGHT*-LDIR
20130  Y = BASEY + HEIGHT*ILDIR
20140  PRINT "L";X;Y
20150  COUNT = COUNT + 1
20160  NEXT I
20170  COUNT = COUNT + SPACE
20180  NEXT CHAR
20190  RETURN

```

CHARACTER GENERATOR TABLE

```

21000  CGEN(0,1) = 31
21010  CGEN(0,2) = 17
21020  CGEN(0,3) = 17
21030  CGEN(0,4) = 17
21040  CGEN(0,5) = 31
21050  CGEN(1,1) = 0
21060  CGEN(1,2) = 9
21070  CGEN(1,3) = 31
21080  CGEN(1,4) = 1
21090  CGEN(1,5) = 0
21100  CGEN(2,1) = 23

```

```

21110    CGEN(2,2) = 21
21120    CGEN(2,3) = 21
21130    CGEN(2,4) = 21
21140    CGEN(2,5) = 29
21150    CGEN(3,1) = 17
21160    CGEN(3,2) = 21
21170    CGEN(3,3) = 21
21180    CGEN(3,4) = 21
21190    CGEN(3,5) = 31
21200    RETURN

```

- LABEL\$ in the main program is the statement to be printed in graphics characters.
- STARTX and STARTY is the starting position for the label.
- LDIR (label direction) is the label orientation: 0 is horizontal; 1 is vertical.
- The character HEIGHT is 5, with 2 pixels (SPACE) skipped between each character.
- CHAR is the main loop which is indexed for each character in LABEL\$.
- CHARCODE stands for character code; it's purpose is to identify the character with its 5 by 5 pattern matrix (CGEN).
- The I loop performs the actual symbol drawing.
- Each of the 5 character generator (CGEN) matrix columns is in turn loaded into the primary line style pattern (PriLineStyle command).
- A short (5 pixel) line is drawn.
- LDIR and ILDIR (Inverse label direction) are multiplied with the draw coordinates to determine direction.
- I is then indexed for the next character column.
- The between character space is skipped before the next character is drawn.
- The subroutine returns to the calling routine once the complete label is drawn.
- The table is truncated here for brevity; available memory is the only restriction to its length.

EXAMPLE 8

This program reads a pixel from graphics memory and prints the result in alphanumeric mode. The result will be zero unless a pixel is turned on at (100,50).

```

00010  DEFINT B-Z
00020  PRINT CHR$(27);"1"
00030  PRINT "I3"
00040  PRINT "P100050"
00050  A$ = INPUT$(2)
00060  PRINT "E";A$
00070  STOP

```

Line 30 sets the line type to READ BIT.

Line 40 reads the pixel at (100,50).

Line 50 inputs the terminal's transmission of a 1 or 0 followed by a carriage-return without echoing the characters back to the terminal.

Line 60 exits graphics mode and prints the state of the pixel at (100,50).

EXAMPLE 9

The program reads a display byte from graphics memory and prints its value in alphanumeric mode. The result will be 00, unless a pixel is turned on between (96,50) and (103,50), inclusively.

```

00010  DEFINT B-Z
00020  PRINT CHR$(27);"1"
00030  PRINT "I5"
00040  PRINT "P100050"
00050  A$ = INPUT$(3)
00060  PRINT "E";A$
00070  STOP

```

Line 30 sets the line type to READ BYTE.

Line 40 reads the display byte between (96,50) and (103,50).

Line 50 inputs the terminal's hexadecimal transmission followed by a carriage-return without echoing the characters back to the terminal.

Line 60 exits graphics mode and prints the value of the display byte.

EXAMPLE 10

The following program is Demonstration program 1 rewritten to use the binary drivers described under **Command Form and Function, BINARY**.

```

00010  DEFINT X,Y
00020  PRINT CHR$(27);"0"
00030  Z = 0
00040  GOSUB 7000
00050  Z = 255
00060  GOSUB 8000
00070  Z = 3
00080  GOSUB 6000
00090  X = 0
00100  Y = 125
00110  GOSUB 2000
00120  FOR X = 0 TO 500 STEP 2
00130  Y = 100*SIN(X/13.27) + 125
00140  GOSUB 4000
00150  NEXT X
00160  Z = 6
00170  GOSUB 6000
00180  GOSUB 1000
00190  STOP

```

EXAMPLE 11

Demonstration program 1 is rewritten in ASCII mode and in **FORTRAN**.

```

                PROGRAM SINE
                INTEGER ESC,X,Y
                NO = 3
                ESC = 27
                WRITE(NO,10)ESC
                WRITE(NO,11)
                WRITE(NO,12)
                DO 200 X = 0,500,2
                Y = 100*SIN(X/13.27) + 125
                WRITE(NO,13)X,Y
200             CONTINUE
                WRITE(NO,14)
10              FORMAT(1X,A1,'1')
11              FORMAT(' I0,N255,D3')
12              FORMAT(' M0,125')
13              FORMAT(' L',2(I3,1X))
14              FORMAT(' D6,E')
                STOP
                END

```

(Most **FORTRAN** compilers permit the mode conversion required in the $Y = 100 \cdot \sin(X/13.27) + 125$ statement, if yours does not, then use the **FORTRAN IFIX** command.)

THEORY OF OPERATION

This section is for those reader interested in the electrical operation of the Imaginator. Information is also available on the software including the complete source code in the **Imaginator Source Code and Manual**.

Remove the schematic from the **Appendix** and place it where you can refer to it easily.

POWER SUPPLY

The Imaginator is interfaced to the rest of the terminal by means of two ribbon cable assemblies and a power harness. The power harness delivers unregulated +16VDC, -16VDC, +8.5VDC and ground. VR1 and R2 form a simple zener voltage regulator that supplies the -5VDC. VR2 is a 3 terminal voltage regulator used to supply the +12VDC. C2 is that regulator's input bypass capacitor to insure stable operation. VR3 is a +5VDC regulator that provides the bulk of the power to the board. C12 and C13 are used for that regulator's input and output bypass capacitors, respectively. Capacitors C10, C11 and C9 are used for bulk decoupling of the dynamic RAM's supply voltages. The remainder of the capacitors are placed near each IC to serve as decoupling capacitors. Reduced operating temperatures for VR2 and VR3 are maintained by heat-sinks. For additional safety, these integrated regulators have internal thermal shutdown circuitry. The power is distributed to the IC's via a grided network to minimize the effective inductance.

MICROCOMPUTER

The Z-80 microprocessor on the TERMINAL LOGIC board was moved to the Imaginator and in its place a 40 conductor ribbon cable was connected. The ribbon cable connects the Imaginator and the

TERMINAL LOGIC address buses, data buses, and control buses together (J4).

U4C, U5, and U6 are non-inverting buffers used to provide the supplementary source and sink drive required by the additional loads on the address bus. U35A and U35B are non-inverting buffers used to provide additional drive on the microprocessor's read (\overline{RD}) and write (\overline{WR}) lines.

E/P/ROM U9A contains the software for the graphics command processor (GCP). U8 is a 128 by 8 bit scratchpad R/W RAM used by the graphics command processor. U9B is a memory mapped socket that may be used for memory expansion.

GRAPHICS COMMAND PROCESSOR MEMORY EXPANSION

Jumpers E1 thru E13 allow the Imaginator to be reconfigured to accept up to 16K of E/P/ROM or a maximum of 8K of E/P/ROM and 8K of R/W RAM instead of the standard 2K E/P/ROM. The standard jumper configuration of E12-E13 connects pin 23 of socket U9A with Vcc. By changing the jumper to E12-E11 pin 23 is connected to address line 11. (In a similar manner E5-E7 can be changed to E5-E6 to reconfigure U9B.) This permits MOSTEK or INTEL family compatible 4K or 8K E/P/ROM to be used. A maximum of 16 K of E/P/ROM may be addressed by reconfiguring both jumper sets and adding an additional E/P/ROM at U9B.

Alternatively, U9B could be R/W RAM instead of E/P/ROM. Strapping E5-E4 connects pin 23 of U9B to the microprocessor write (\overline{WR}) line. This configuration is used for 2K R/W RAM. Strapping E9-E8 is used for larger 4K R/W RAM. Psuedostatic RAM can be implemented by strapping E2-E3. This

connects pin 1 of U9B to the microprocessor refresh (\overline{RFSH}) line. Refer to **Modifications** for specific strapping information.

MEMORY MAP

U22, U34B, U29B, and U28B form the memory map decoding logic. Address lines A_{15} , A_{14} , and A_{13} are decoded to divide the total address space of 64K into eight 8K banks.

The total address space is allocated as follows:

2000H-3FFFH, pin 14 of U22, is allocated to the graphics command processor E/P/ROM, U9A.

6000H-7FFFH, pin 12 of U22, is allocated to the graphics scratchpad R/W RAM, U8.

8000H-BFFFH, pin 10 OR pin 11 (U29B) of U22, is allocated to the graphics display dynamic R/W RAM, U10-U17.

C000H-DFFFH, pin 9 of U22, is allocated to the optional graphics command processor memory, U9B.

The rest of the address space 0000H-1FFFH, 4000H-5FFFH, and E000H-FFFFH, pin 7 OR pin 13 OR pin 15 (U28B) of U22, is allocated to memory on the TERMINAL LOGIC board.

The outputs of U22 are only enabled when pin 5 of U22 receives an active memory request (\overline{MREQ}) from the microprocessor.

INPUT/OUTPUT

An input/output request (\overline{IORQ}) is sent to the TERMINAL LOGIC board only when both \overline{IORQ} (pin 20 of U7) AND A_4 are low (U34C).

HARDWARE RESET

When the microprocessor receives a hardware reset (pin 26 of U7) it instinctively knows to output address 0000H and fetch an opcode. Normally 0000H-1FFFH is allocated to the E/P/ROMs on the TERMINAL LOGIC board. However, since the clear input of flip-flop U27A (pin 1) also receives the reset signal it causes the output of U34B to go high. This effectively moves a copy of the Imaginator's

E/P/ROM (U9A) at 2000H-3FFFH down to 0000H-1FFFH, while at the same time prevents a \overline{MREQ} (U28B) from being sent to the TERMINAL LOGIC board. The first instruction of the Imaginator's E/P/ROM (U9A) is a JMP to 2003H. The E/P/ROM program then accesses the R/W memory at 6000H-7FFFH (pin 12 of U22) which presets U27 (pin 4) and the memory map configuration returns to normal.

INTERRUPTS

The graphics command processor initializes the microprocessor to accept both non-maskable interrupts (\overline{NMI}) and software maskable vectored interrupts. The \overline{NMI} originates on the TERMINAL LOGIC board (pin 17 of U7). The software maskable interrupt request is output from multiplexor U38B pin 8. The multiplexor chooses between an interrupt generated on the TERMINAL LOGIC board (keyboard, asynchronous communications element or break key) or a horizontal retrace interrupt generated on the Imaginator. U21A first inverts the interrupt from the TERMINAL LOGIC board to convert to high true logic.

The horizontal retrace interrupt pulse is generated by U32A and U21C 10.24uS before the microprocessor's time slot begins to allow for interrupt latency. The horizontal retrace interrupt request becomes active when both U32A pin 5 AND U21C pin 6 are high (U38B pins 1 and 13). It is returned to an inactive state 5.12uS later when the inverted (U21C) secondary address line A_3 goes low. The horizontal retrace interrupt is used to signal when the microprocessor is permitted to access the graphics display R/W RAM.

The interrupt multiplexor (U38B) is controlled by the outputs of U40B (pins 8 and 9). The microprocessor can set the interrupt multiplexor (U38B) to pass an interrupt generated on the TERMINAL LOGIC board by performing an I/O write of port 00011000B. Specifically, this would happen with an active \overline{IORQ} (U21B and U28A pin 2) AND an inactive $\overline{M1}$ (U28A pin 13) AND A_4 high (U28A pin 1) while holding A_3 high (U40B pin 12). Alternatively, it can set the interrupt multiplexor to pass the Imaginator generated horizontal retrace interrupt by an I/O write to port 00010000B (as before except A_3 is low).

Once the microprocessor receives a software maskable interrupt it will request an interrupt vector to be placed on the data bus by making both \overline{IOQR} (U34D pin 12) and $\overline{M1}$ (U34D pin 13) active. The out-

put of U34D then enables the three-state buffers U35D and U41C. D_1 of the vector is determined by pin 9 of U40B with the other 7 bits permanently assigned as zero. The graphics command processor's interrupt routine will then access the graphics display memory located at 8000H to BFFFH causing the output of U29B to go low, presetting U40 (pin 10). The interrupt multiplexor (U38B) will then block horizontal retrace interrupts and will pass only interrupts generated on the TERMINAL LOGIC board.

GRAPHICS REFRESH ADDRESS COUNTERS

Synchronous binary counters (U19, U26, U31, U25) are cascaded to form the graphics display R/W RAM's 14 bit secondary address bus. These counters are synchronized with the vertical and horizontal sync pulses generated by the CRT controller on the TERMINAL LOGIC board (J2/J3 pins 6 and 8, respectively). The output of the non-inverting buffer U4A provides the clock signal used to synchronize these counters.

Counter U20 is used to disable the cascaded address counters from counting for a fixed number of clock pulses past the receipt of the horizontal sync pulse (U21E). This is done to center the graphics display on the screen. Since a low on either enable P or enable T disables the counter, the cascaded address counters are disabled until enable T (pin 10) of U19 goes high. When U20 receives a horizontal sync pulse (pin 9) it loads 0010 with the rising edge of the next clock pulse (pin 2). It then counts to 1111 at which time the ripple carry out goes high. The ripple carry out is inverted by U21D and input to the enable P pin on U20 (pin 7). This disables further counting and holds the ripple carry out high until the next horizontal sync pulse (i.e., the next display line).

X COUNTERS

Counters U19 and U26 are cascaded to form the 6 address lines A_0-A_5 , that cover the range 0 to 63. Note that these counters form addresses that are used to access bytes of data, and that the horizontal width of the graphics display is 512 pixels (i.e., 64 times 8). Since the horizontal sync pulse loads counter U26 with 1100 a ripple carry out will be generated on pin 15 of U26 when U19 and U26 reach the count of 63. This ripple carry out permits the cascaded counters U31 and U25 to increment with the next positive edge of the clock (pin 2).

Y COUNTERS

The next horizontal sync pulse restarts counter U20 and reloads counters U19 and U26 with 11000000 but does not effect U31 or U25. The latter two counters are loaded with 00000000 only when they receive a vertical sync pulse (pins 9 on U31 and U25). These counters count from 0 to 255 and logically represent the vertical axis of the graphics display.

DYNAMIC RAM ADDRESS MULTIPLEXORS

U18, U24, U30 and U36 are 4 to 1 multiplexors used to interface the graphics display's dynamic RAMs (U10-U17) with the microprocessor's address bus and the graphics counter's secondary address bus. These multiplexors have the dual task of selecting between the microprocessor's and secondary address buses and selecting between high and low order address bits on either bus. Select A (pins 14) are used to specify which address bus is to be selected and select B (pins 2) specifies the high or low order bits. The microprocessor's address bus is selected by U32B (pin 9) only when the CRT's electron beam is outside the graphics display region to prevent disturbing displayed graphics data.

ROW ADDRESS STROBE (RAS) AND COLUMN ADDRESS STROBE (CAS) LOGIC

The graphics display address counters and the microprocessor are not synchronized together, which requires the high/low address line selection to be controlled independently of each other. The multiplexor (U38A) that selects between these independent controllers is itself controlled by an output (pin 8) of U32B. U40A, U21F and U29D form the microprocessor's high/low address line controller. Address lines A_0-A_6 are selected until a positive edge of the microprocessor clock (U40A pin 3) latches an active read (\overline{RD}) OR write (\overline{WR}) (U29D) AND the graphics display RAM at 8000H to BFFFH (U40A pin 4) has been accessed. The microprocessor will access the graphics display RAM only after the states on the address bus are stable. The output of U29B will go low causing the output of U29C, the row address strobe (\overline{RAS}), to go low. The dynamic RAM's will latch addresses A_0-A_6 with the negative going edge of the row address strobe (\overline{RAS}). If the dynamic RAM's are accessed (pin 4 of U40A), the positive going edge of the microprocessor clock (pin 3 of U40A) will latch an active read (\overline{RD}) OR write (\overline{WR}) (U29D).

resulting in a high output (pin 6 of U40A). This causes the address multiplexor to select address lines A_7 - A_{13} . It also causes the output of U36 (pin 9) to switch states. This signal is used as the column address strobe (\overline{CAS}) after being delayed by the cascaded non-inverting buffers U35C and U35D. This delay is present to insure that the address lines A_7 - A_{13} have stabilized before the dynamic RAM's read them.

U33 and U39 make up a sequential circuit that generates the row address strobe and the address selector signal for the address multiplexor. The dot clock (U4B) and the load clock (U4A) are used to synchronize the generator with the graphics display RAM's secondary address bus counters and the shift register (U2).

Series terminating resistors R1 and R3-R11 are used to prevent undershoot by matching the low level source impedance with the line impedance.

DYNAMIC RAM ACCESS LOGIC

The dynamic RAM's are written to in 'early write' mode which permits the input and the output of each individual RAM to be connected without contentions (the outputs are internally placed in a high impedance state.) The write signal is generated by U3C when the graphics display RAM is selected (pin 9 of U3C) AND when the microprocessor issues an active write (\overline{WR}) (pin 10 of U3C). The dynamic RAM's data lines form the secondary data bus and are connected in parallel to the eight bit shift register (U2) and the octal bus transceiver (U37).

DATA BUS TRANSCEIVER

The octal bus transceiver is used to isolate the primary data bus from the secondary data bus. The two buses remain isolated from one another unless the microprocessor accesses the graphics display RAM at 8000H-BFFFH (pin 19 of U37). The read (RD) line from the microprocessor is used to determine the direction of data flow through the transceiver (pin 1 of U37).

GRAPHICS SHIFT REGISTER

The shift register (U2) loads these eight bits when the load clock is active (pin 15 of U2) AND with the positive going edge of the dot clock (pin 7 of U2). The shift register shifts one bit out with each subsequent

positive going edge of the dot clock. Since the data need only be stable on the secondary data bus when the shift register loads, an extra stage of pipelining is produced. The load clock and the dot clock are input to both shift registers (pins 15 and 7, respectively, of U1 and U2) before the non-inverting buffers U4A and U4B to avoid the propagation delay through those buffers.

ALPHANUMERIC SHIFT REGISTER

The 16 conductor ribbon cable carries the dot and load clock as well as the eight data lines from the character generator located on the TERMINAL LOGIC board. These signals are input to the shift register (U1) that was moved from the TERMINAL LOGIC board to the Imaginator. Unused lines 1, 6, 8, 9, and 16 of cable J1 are tied to the Imaginator's supply lines to prevent them from harmfully coupling with the high frequency clock signals carried on the ribbon cable.

DISPLAY ENABLE LOGIC

The output (pin 13) of shift register U1 is forced low when the clear input (pin 9 of U1) is brought low by the output of U23B. The state of address line A_7 (pin 12 of U23B) is latched by flip flop U23B with the positive going edge of \overline{IORQ} (pin 2 of U28A) AND NOT $\overline{M1}$ (pin 13 of U28A) AND address line A_4 (pin 1 of U28A) AND address line A_5 (pin 10 of U28C). The output (pin 13) of shift register U2 is forced low when either the output of U23A is low (pin 2 of U29A) OR the non-inverting output of U32B is low (pin 1 of U29A). The state of address line A_6 (pin 2 of U32A) is latched by flip flop U23A in the same manner as U23B.

The alphanumeric video output (pin 13) of U1 and the graphics video output (pin 13) of U2 are ORed together and then output to the TERMINAL LOGIC board (U3B).

REPLACEMENT PARTS

INTEGRATED CIRCUITS

CIRCUIT NUMBER	INTERNAL NUMBER	EXTERNAL NUMBER
U1	1550-0166	IC, 74LS166
U2	1550-0166	IC, 74LS166
U3	1540-0032	IC, 74S32
U4	1550-0367	IC, 74LS367
U5	1550-0367	IC, 74LS367
U6	1550-0367	IC, 74LS367
U7	1530-0080	IC, CPU, Z-80
U8	1560-0310	IC, 68A10
U9A	1560-0116	IC, EPROM, UNPROGRAMMED *
U10	1560-0216	IC, 4116N-4
U11	1560-0216	IC, 4116N-4
U12	1560-0216	IC, 4116N-4
U13	1560-0216	IC, 4116N-4
U14	1560-0216	IC, 4116N-4
U15	1560-0216	IC, 4116N-4
U16	1560-0216	IC, 4116N-4
U17	1560-0216	IC, 4116N-4
U18	1550-0153	IC, 74LS153
U19	1550-0163	IC, 74LS163A
U20	1550-0163	IC, 74LS163A
U21	1550-0004	IC, 74LS04
U22	1550-0138	IC, 74LS138
U23	1550-0074	IC, 74LS74
U24	1550-0153	IC, 74LS153
U25	1550-0163	IC, 74LS163A
U26	1550-0163	IC, 74LS163A
U27	1550-0074	IC, 74LS74
U28	1550-0011	IC, 74LS11
U29	1550-0008	IC, 74LS08
U30	1550-0153	IC, 74LS163A
U31	1550-0163	IC, 74LS163A
U32	1550-0074	IC, 74LS74
U33	1550-0163	IC, 74LS163A
U34	1550-0032	IC, 74LS32
U35	1550-0367	IC, 74LS367
U36	1550-0153	IC, 74LS153
U37	1550-0245	IC, 74LS245
U38	1550-0051	IC, 74LS51
U39	1550-0074	IC, 74LS74
U40	1550-0074	IC, 74LS74
U41	1550-0367	IC, 74LS367

VOLTAGE REGULATORS

CIRCUIT NUMBER	INTERNAL NUMBER	EXTERNAL NUMBER
VR1	1410-0100	ZENER DIODE, 1N751A
VR3	1580-0100	REGULATOR, +5V, LM340K-5
VR2	1580-0110	REGULATOR, +12V, LM340T-12

CAPACITORS

INTERNAL NUMBER	EXTERNAL NUMBER
1300-0110	CAP, TANT, 10uf
1350-0110	CAP, CER, 0.1uf
1350-0821	CAP, CER, 820pf

RESISTORS

INTERNAL NUMBER	EXTERNAL NUMBER
1200-4330	1/4W, 5%, 33 OHM
1200-2471	1/2W, 5%, 470 OHM

IC SOCKETS

INTERNAL NUMBER	EXTERNAL NUMBER
1100-0040	SOCKET, IC, 40 PIN
1100-0028	SOCKET, IC, 28 PIN
1100-0024	SOCKET, IC, 24 PIN
1100-0020	SOCKET, IC, 20 PIN
1100-0016	SOCKET, IC, 16 PIN
1100-0014	SOCKET, IC, 14 PIN

POWER HARNESS HARDWARE

INTERNAL NUMBER	EXTERNAL NUMBER
1110-0111	11-HOLE CONN. SHELL
1110-0211	11-PIN POLARIZING WAFER
1115-0000	CRIMP TERMINAL

HEAT SINKS

INTERNAL NUMBER	EXTERNAL NUMBER
1700-0010	HEAT SINK, LARGE
1700-0020	HEAT SINK, SMALL

HARDWARE

INTERNAL NUMBER	EXTERNAL NUMBER
1806-0500	6-32 X 1/2 SCREW
1806-0375	6-32 X 3/8 SCREW
1810-0632	6-32 NUT
1820-0600	#6 LOCKWASHER
1700-0000	CARD GUIDE

RIBBON CABLES

INTERNAL NUMBER	EXTERNAL NUMBER
1190-0001	40-COND. RIBBON ASSY.
1190-0002	16-COND. RIBBON ASSY.

HOOKEUP WIRE

INTERNAL NUMBER	EXTERNAL NUMBER
1690-1801	WIRE #18 AWG, WHT
1690-1802	WIRE #18 AWG, BLACK
1690-1803	WIRE #18 AWG, RED
1690-1804	WIRE #18 AWG, GRN
1690-1805	WIRE #18 AWG, YEL
1690-1806	WIRE #18 AWG, BLU
1690-1808	WIRE #18 AWG, ORG
1690-1810	WIRE #18 AWG, VIO

MISCELLANEOUS

INTERNAL NUMBER	EXTERNAL NUMBER
0500-0000	CIRCUIT BOARD
0100-0000	ASSY MANUAL
0100-0001	USER MANUAL
0110-0000	3-RING BINDER

H/Z-89 MODELS ONLY

INTERNAL NUMBER	EXTERNAL NUMBER
1110-0103	3-PIN CONN. SHELL
1110-0203	3-PIN POLARIZING WAFER
1710-0010	REAR BRACKET
1710-0011	FRONT BRACKET
1710-0012	TO-3 BRACKET
1120-0000	TO-3 SOCKET
1850-0025	NYLON SPACER
1190-0003	40-COND. RIBBON ASSY.
1190-0004	16-COND. RIBBON ASSY.

When ordering replacement parts be certain to specify the Internal Part Number.

Please mark all correspondence Attn: Parts Replacement.

* For replacement of programmed EPROMs give us the part number and revision number that are imprinted on the EPROM label.

REFERENCES

The following list of references is suggested for those who wish to explore the subject of computer graphics in greater detail. The first three references listed contain extensive bibliographies. Though the first two books listed have the same title, they are quite different in content.

William M. Newman, Robert F. Sproull. *Principles of Interactive Computer Graphics*. McGraw Hill Book Company, 1973.

William M. Newman, Robert F. Sproull. *Principles of Interactive Computer Graphics*. Second Edition. McGraw Hill Book Company, 1979.

Sylvan H. Chasen. *Geometric Principles and Procedures for Computer Graphic Applications*. Prentice-Hall, Inc., 1978.

Conrac Division, Conrac Corporation. *Raster Graphics Handbook*. Conrac Division, 1980.

Chris Rorres, Howard Anton. *Applications of Linear Algebra*. Second Edition. John Wiley and Sons, 1979. (This book has one chapter devoted to computer graphics.)

This page intentionally left blank.

MODIFICATIONS

The Imaginator may be easily reconfigured to permit the user to modify and enhance its present capabilities. Up to 16K of E/P/ROM can be accommodated by simply restrapping E1 through E13.

Alternatively, 8K of E/P/ROM and 8K of R/W RAM can be configured. This arrangement along with some of the special GCP commands permit the downloading of custom programs from the host computer into the graphics terminal. Fast custom character generators is a typical application.

NOTE: The information contained in this manual is not detailed enough to permit the user to take advantage of these enhancements. An additional manual, **Imaginator Source Code and Manual**, is required. However, the required strapping information is included below. A detailed pictorial description is included with memory expansion accessories.

MEMORY DEVICE	SOCKET U9A	SOCKET U9B
E/P/ROM		
2716	E12-E13	E5-E7, E14-E15
2732	E12-E11	E5-E6, E14-E15
2764	E12-E11	E2-E1, E5-E6, E14-E15
STATIC R/W RAM		
4118	NA	E5-E4
4802	NA	E5-E4
PSEUDO STATIC R/W RAM		
4816	NA	E2-E3, E9-E8

Only the indicated pads should be strapped. Unmentioned pads should have no connection. Non-volatile memory is mandatory in socket U9A, therefore the strapping for the R/W RAM is not applicable (NA).

This page intentionally left blank.

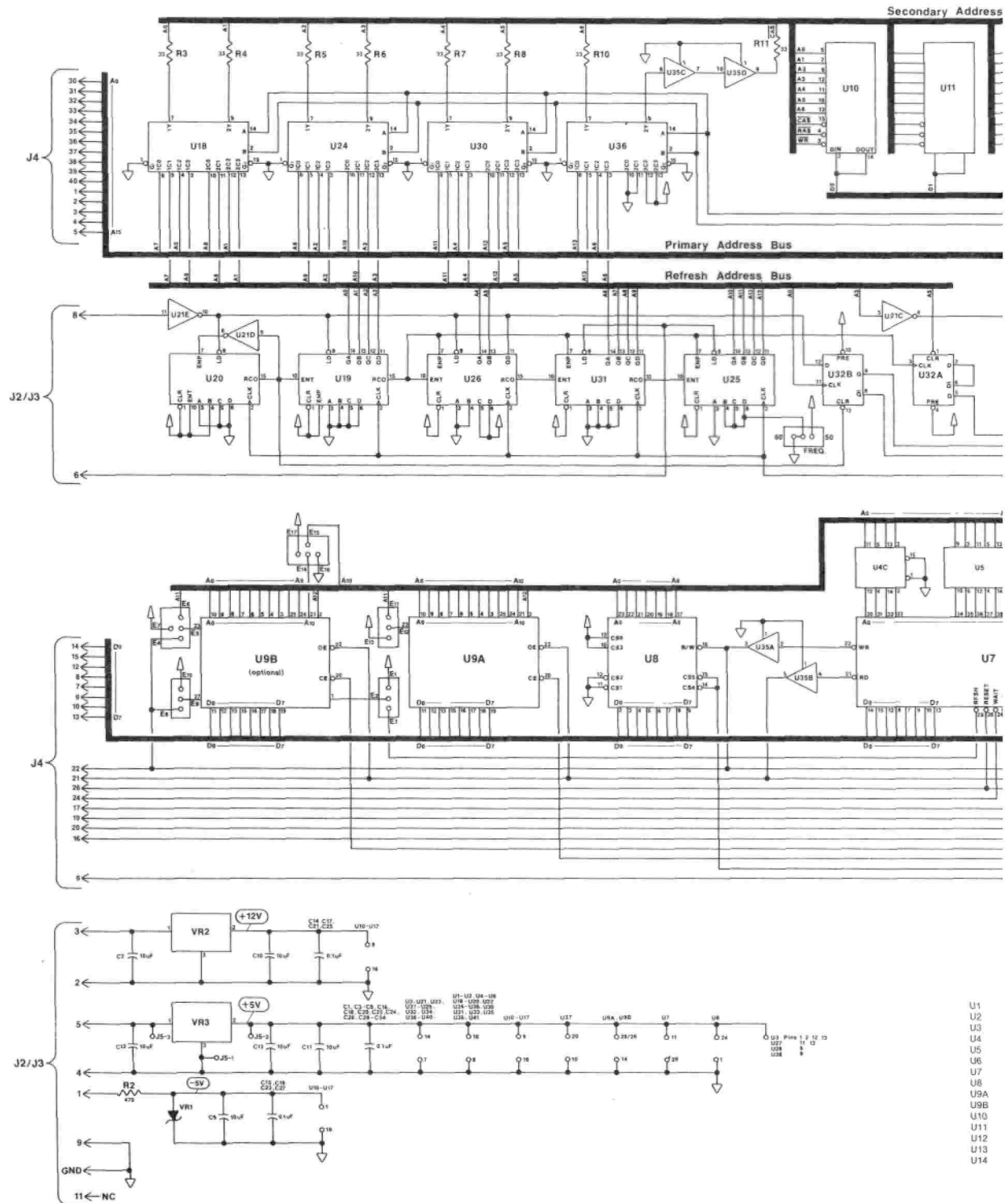
APPENDIX

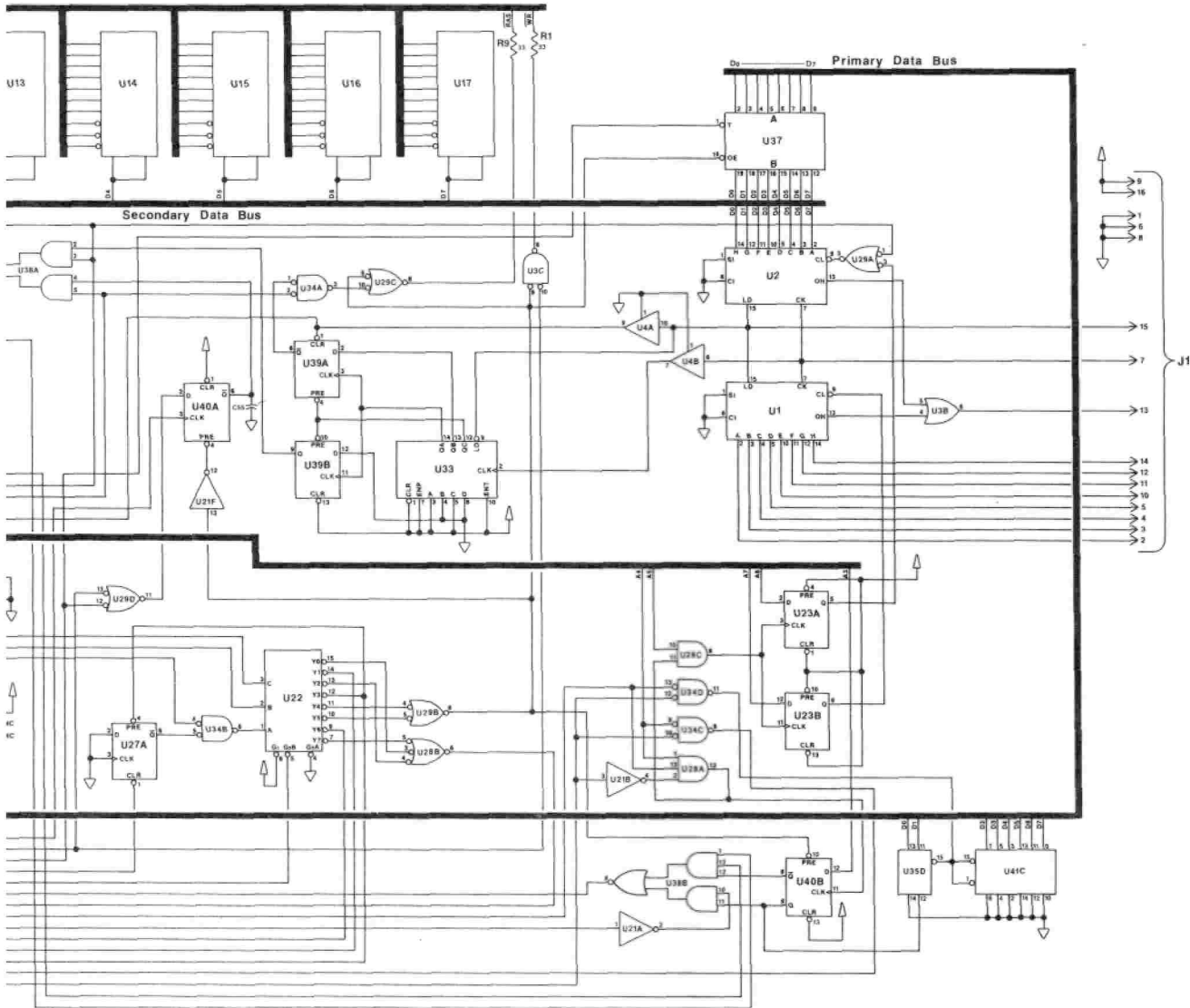
Alphanumeric Mode						Graphics Mode	
Decimal Code	Hex Code	Octal Code	Binary Code	Character	Control Keys	ASCII Mode	Binary Mode Operand
0	00	000	00000000	NUL	@	Unused Delimiters	Unused
1	01	001	00000001	SOH	A		No Operation
2	02	002	00000010	STX	B		
3	03	003	00000011	ETX	C		
4	04	004	00000100	EOT	D		
5	05	005	00000101	ENQ	E		
6	06	006	00000110	ACK	F		
7	07	007	00000111	BEL	G		No Operation
8	08	010	00001000	BS	H		
9	09	011	00001001	HT	I		
10	0A	012	00001010	LF	J		
11	0B	013	00001011	VT	K		
12	0C	014	00001100	FF	L		
13	0D	015	00001101	CR	M		BringInProgram
14	0E	016	00001110	SO	N		
15	0F	017	00001111	SI	O		
16	10	020	00010000	DLE	P		
17	11	021	00010001	DC1	Q		
18	12	022	00010010	DC2	R	Delimiters	BringInProgram
19	13	023	00010011	DC3	S		
20	14	024	00010100	DC4	T		
21	15	025	00010101	NAK	U		
22	16	026	00010110	SYN	V		
23	17	027	00010111	ETB	W		
24	18	030	00011000	CAN	X		
25	19	031	00011001	EM	Y		
26	1A	032	00011010	SUB	Z		
27	1B	033	00011011	ESC	[
28	1C	034	00011100	FS	\		
29	1D	035	00011101	GS]		
30	1E	036	00011110	RS	^		
31	1F	037	00011111	US	-		
32	20	040	00100000	SP			
33	21	041	00100001	!			
34	22	042	00100010	"			

Alphanumeric Mode					Graphics Mode		
Decimal Code	Hex Code	Octal Code	Binary Code	Character	Control Keys	ASCII Mode	Binary Mode Operand
35	23	043	00100011	#		Delimiters	DisplayToggle
36	24	044	00100100	\$			
37	25	045	00100101	%			
38	26	046	00100110	&			
39	27	047	00100111	'			
40	28	050	00101000	(ExitGraphicsMode
41	29	051	00101001)			
42	2A	052	00101010	*			
43	2B	053	00101011	+			
44	2C	054	00101100	,			
45	2D	055	00101101	-			PointAt
46	2E	056	00101110	.			
47	2F	057	00101111	/		Delimiters	
48	30	060	00110000	0		Data	
49	31	061	00110001	1			
50	32	062	00110010	2			
51	33	063	00110011	3			
52	34	064	00110100	4			
53	35	065	00110101	5			
54	36	066	00110110	6			
55	37	067	00110111	7			
56	38	070	00111000	8			
57	39	071	00111001	9		Data	
58	3A	072	00111010	:		Delimiters	
59	3B	073	00111011	;			
60	3C	074	00111100	<			
61	3D	075	00111101	=			
62	3E	076	00111110	>			
63	3F	077	00111111	?			
64	40	100	01000000	@		Delimiters	
65	41	101	01000001	A		AreaTo	
66	42	102	01000010	B		BringInProgram	
67	43	103	01000011	C			
68	44	104	01000100	D		DisplayToggle	
69	45	105	01000101	E		ExitGraphicsMode	
70	46	106	01000110	F			
71	47	107	01000111	G			
72	48	110	01001000	H			
73	49	111	01001001	I		LineType	
74	4A	121	01001010	J		JumpToProgram	
75	4B	113	01001011	K			LineType
76	4C	114	01001100	L		LineTo	
77	4D	115	01001101	M		MoveTo	
78	4E	116	01001110	N		PriLineStyle	
79	4F	117	01001111	O		SecLineStyle	
80	50	120	01010000	P		PointAt	JumpToProgram
81	51	121	01010001	Q		Delimiters	
82	52	122	01010010	R			
83	53	123	01010011	S			
84	54	124	01010100	T		Delimiters	

Alphanumeric Mode					Graphics Mode		
Decimal Code	Hex Code	Octal Code	Binary Code	Character	Control Keys	ASCII Mode	Binary Mode Operand
85	55	125	01010101	U		Delimiters	
86	56	126	01010110	V			
87	57	127	01010111	W			
88	58	130	01011000	X			
89	59	131	01011001	Y			
90	5A	132	01011010	Z			AreaTo
91	5B	133	01011011	[
92	5C	134	01011100	\			
93	5D	135	01011101]			
94	5E	136	01011110	^			
95	5F	137	01011111	_			
96	60	140	01100000	`			
97	61	141	01100001	a			
98	62	142	01100010	b			
99	63	143	01100011	c			
100	64	144	01100100	d			LineTo
101	65	145	01100101	e			
102	66	146	01100110	f			
103	67	147	01100111	g			
104	68	150	01101000	h			
105	69	151	01101001	i			
106	6A	152	01101010	j			
107	6B	153	01101011	k			
108	6C	154	01101100	l			
109	6D	155	01101101	m			
110	6E	156	01101110	n			
111	6F	157	01101111	o			
112	70	160	01110000	p			
113	71	161	01110001	q			
114	72	162	01110010	r			
115	73	163	01110011	s			PriLineStyle
116	74	164	01110100	t			
117	75	165	01110101	u			
118	76	166	01110110	v			
119	77	167	01110111	w			
120	78	170	01111000	x			
121	79	171	01111001	y			
122	7A	172	01111010	z			
123	7B	173	01111011	{			
124	7C	174	01111100				
125	7D	175	01111101	}			SecLineStyle
126	7E	176	01111110	~			
127	7F	177	01111111	DEL		Delimiters	

This page intentionally left blank.





115	4116-4	U30	74LS153
116	4116-4	U31	74LS163
117	4116-4	U32	74LS74
118	74LS153	U33	74LS163
119	74LS163	U34	74LS32
120	74LS163	U35	74LS367
121	74LS04	U36	74LS153
122	74LS138	U37	74LS245
123	74LS74	U38	74LS51
124	74LS153	U39	74LS74
125	74LS163	U40	74LS74
126	74LS163	U41	74LS367
127	74LS74	VR1	1N751
128	74LS11	VR2	LM340T-12
129	74LS08	VR3	LM340K-5

Copyright (C) 1982 by Cleveland Codonics, Inc.
All rights reserved. Worldwide

Cleveland Codonics, Inc.

SCALE:	APPROVED BY:	DRAWN BY:
DATE:		REVISED:
Imaginator		
DRAWING NUMBER		REV. B

WE WOULD LIKE YOUR COMMENTS ON THIS MANUAL

Did you find any errors in this manual? Where?

Was it complete, should some areas be covered in greater detail?

Was it the right level? Too simple? Too difficult?

Was it clearly written?

Please rate this document with respect to similar ones?

CLEVELAND CODONICS, INC.
P.O. Box 45259
Cleveland, Ohio 44145



YOUR 90-DAY LIMITED WARRANTY

Cleveland Codonics, Inc. warrants that during the first ninety (90) days after purchase, this product, when correctly assembled and used in accordance with our printed instructions, will meet published specifications.

For a period of ninety (90) days after purchase, Cleveland Codonics, Inc. will repair or replace (at our option) free of charge (excluding freight) any parts or assemblies that are defective in either materials or workmanship. This warranty covers only Cleveland Codonics, Inc. products. It does not include equipment used in conjunction with this product. We are not responsible for incidental or consequential damages, nor are we responsible for loss of business or profits.

EXCEPT FOR THE EXPRESS WARRANTIES CONTAINED HEREIN, CLEVELAND CODONICS, INC. DISCLAIMS ALL WARRANTIES ON THE PRODUCTS FURNISHED HEREUNDER, INCLUDING ANY AND ALL IMPLIED WARRANTIES FOR MERCHANTABILITY AND FITNESS. No agent, representative, dealer or employee of the company has the authority to increase or alter the obligations of this warranty. This warranty gives you specific legal rights and you may also have other rights which vary from state to state.

This warranty does not cover damage resulting from misuse, abuse, incorrect assembly, or unauthorized modifications.