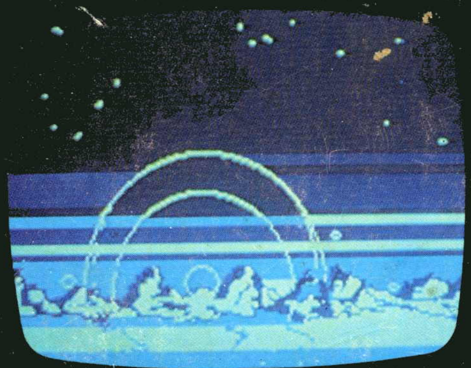
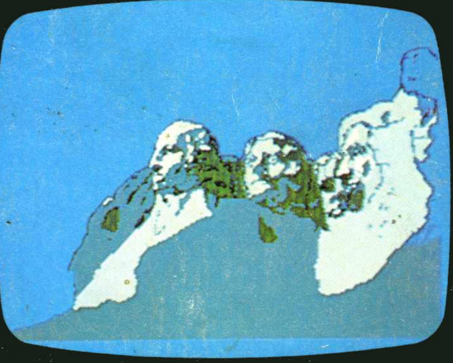
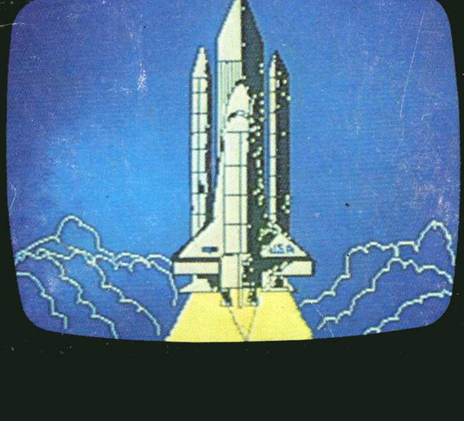
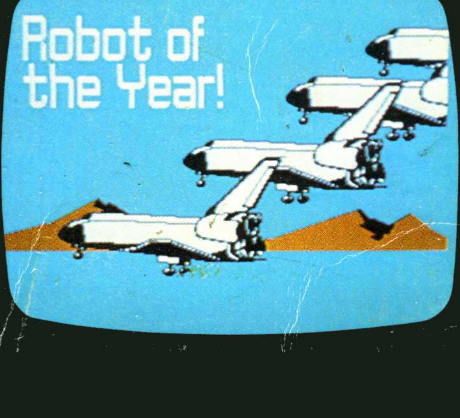
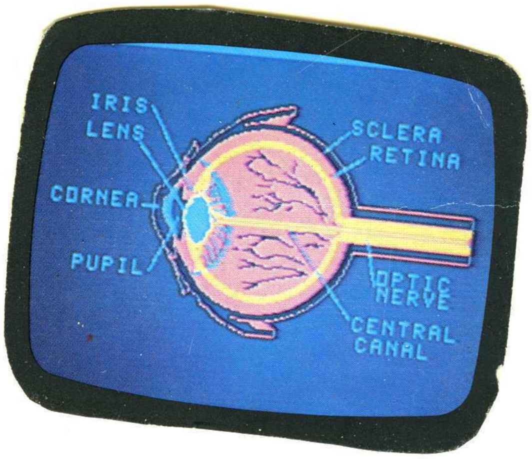
datamax presents.... living art



Rec 5-11-83


IRIS
LENS
SCLERA
RETINA
CORNEA
PUPIL
OPTIC NERVE
CENTRAL CANAL

UV-1 ZGRASS GRAPHICS SYSTEM
OPERATOR MANUAL

REAL TIME DESIGN, INC.

MARCH 1981

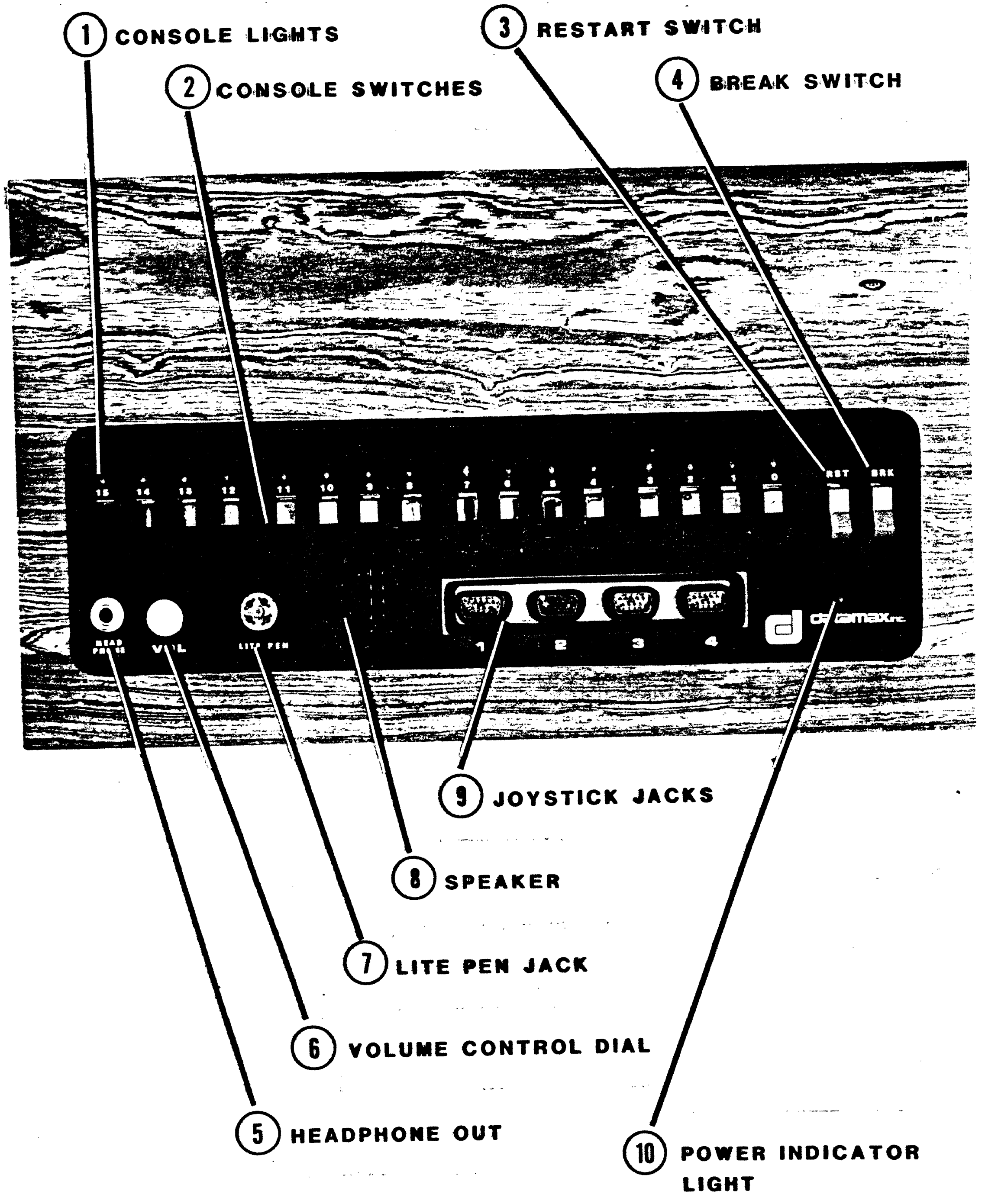
UV-1 ZGRASS GRAPHICS SYSTEM

OPERATOR MANUAL

CONTENTS

HARDWARE FEATURES

FRONT PANEL



HARDWARE FEATURES

FRONT PANEL

1. CONSOLE LIGHTS (16)
   Assigned numbers 15-0, user programmable on 2 different output ports (see glossary under PORT).

2. CONSOLE SWITCHES (16)
   Assigned numbers 15-0, user programmable on 2 different input ports (see glossary under PORT).

3. RESTART SWITCH (Lefthand of 2 Red Switches)
   Same result as RESTART command: clears memory and restarts ZGRASS if you answer "Y" when asked. Typing "N" avoids memory clear in the case of accidental restart.

4. BREAK SWITCH (Righthand of 2 Red Switches)
   Same result as pressing BREAK key on terminal or CTRL+C: stops currently running MACRO(s), clears control characters.

5. HEADPHONE OUT (1/4" Phone Jack)
   Monaural, 8 ohms impedance, cuts out speaker, volume controlled by volume control dial. Monitors audio output of 3-voice synthesizer accessed via MUSIC and PORT commands (see glossary).

6. VOLUME CONTROL DIAL
   Controls audio output gain of either headphone out or speaker.

7. LIGHT PEN JACK
   See parts list for plug specifications. Software to follow.

8. SPEAKER
   Monitors audio output of 3-voice synthesizer accessed via MUSIC and PORT commands (see glossary). Volume controlled by volume control dial, cut out by headphones.

9. JOYSTICK JACKS (4)
   For standard accessory joysticks, accessed as user programmable device variables (see glossary under

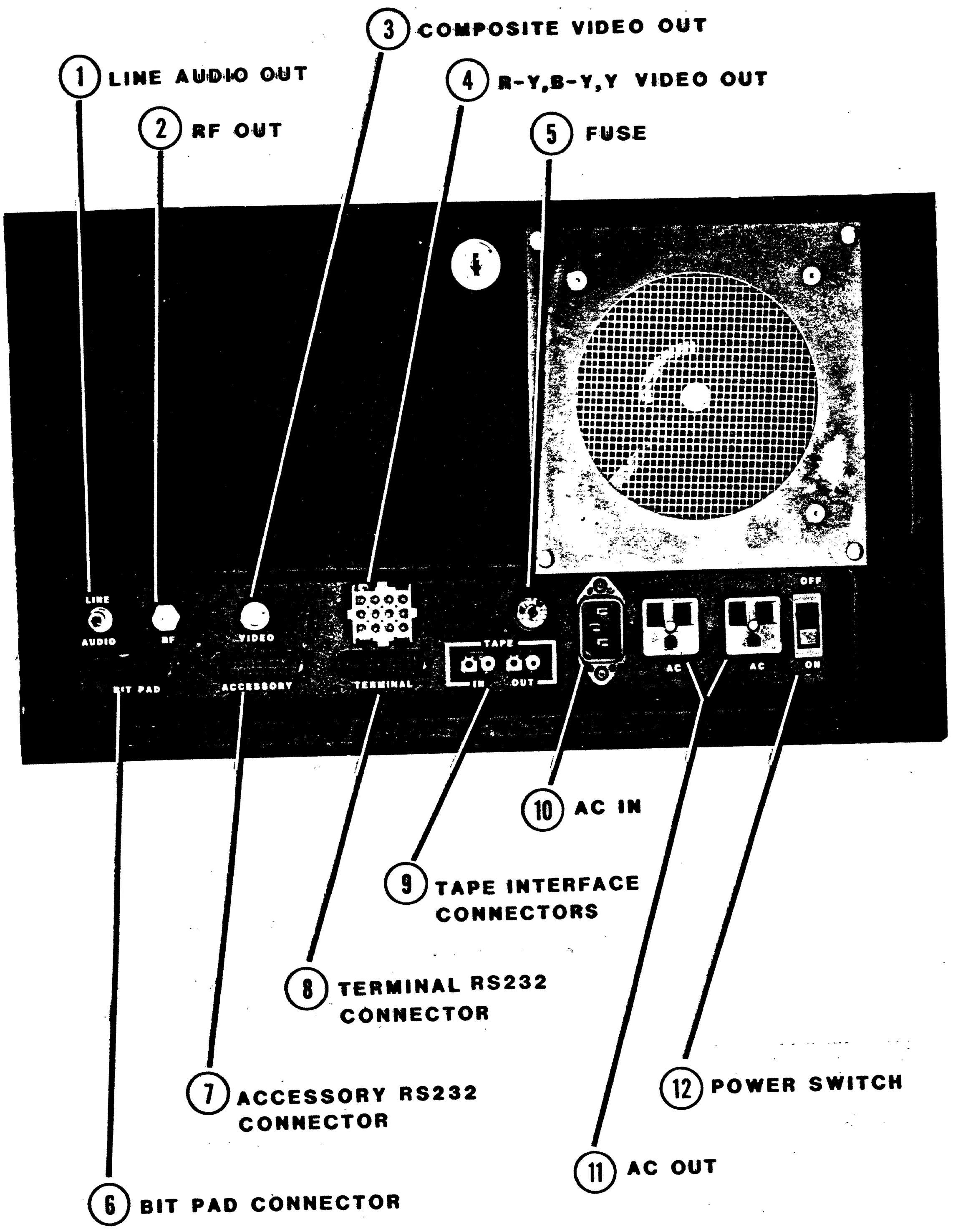
JOYSTICK and DEVICE VARIABLES).

10. POWER INDICATOR LIGHT
    Power switch is on the rear panel.

End of FRONT PANEL DESCRIPTIONS

HARDWARE FEATURES

REAR PANEL



HARDWARE FEATURES

REAR PANEL

1. LINE AUDIO OUT (RCA Phono Connector)
   Approximately 1 volt P-P, 200 ohms impedance or higher. Outputs audio from 3-voice synthesizer, accessed via MUSIC and PORT commands (see glossary).

2. RF OUT ("F" Connector)
   Outputs RF audio/video to antenna input on standard TV receiver on VHF channel 3 or 4, home videocassette recorder,etc.

3. COMPOSITE VIDEO OUT (BNC Connector)
   1 volt P-P, 75 ohms impedance. Outputs composite, NTSC standard video to video monitor, videotape recorder, switcher,etc. (Contact DATAMAX for future availability of Genlock option.)

4. R-Y,B-Y,Y VIDEO OUT (Mate'n'Lock Connector)
   Outputs R-Y,B-Y,Y video to DATAMAX RGB monitor which contains RGB converter board (also available from DATAMAX).

5. FUSE (MDA-1)

6. BIT PAD CONNECTOR (Parallel Interface)
   Currently compatible with Summagraphics Bit Pad One (TM Summagraphics Corp.). Any other tablet currently uses Accessory RS232 interface with user-developed software.

7. ACCESSORY RS232 CONNECTOR (Serial Interface)
   For connecting disk drive, printer, plotter, modem, additional computer,etc. To set Baud Rate, see Baud Rate Selection section of this manual. (Also see glossary under RS232)

8. TERMINAL RS232 CONNECTOR (Serial Interface)
   For connecting Hazeltine, ADM3A(+), ActIV terminals or emulators. To set Baud Rate, see Baud Rate Selection section of this manual. To set Editor keys to your terminal and your taste, see glossary for TERMINAL command (see also RS232). NOTE: The terminal used with the UV-1 must have both upper and lower case character

capability. In ZGRASS, your general mode is upper case with lower case being used only for local variables, etc. This means that the ZGRASS echo is the reverse of the upper/lower case functions of your terminal so that when you type something in unshifted, lower case mode, the characters you see on your terminal will be upper case.

9. TAPE IN/OUT CONNECTORS (AUDIOTAPE INTERFACE)
(2) 1/8" Mini Phone Jacks for audio data IN/OUT and (2) Submini Phone Jacks for DC Control Signals OUT for remote motor control. 2,000 Baud Rate. See Audiotape Interface section of this manual for connections and procedures. See glossary under PORT for direct program control of DC signals to control other devices such as film cameras, time-lapse videotape recorders, etc.

10. AC IN JACK (AC 120V 60 HZ 150 WATTS MAX)
(If you want to put a line protector on the AC line coming into the UV-1 : to compute line protection for individual or a group of devices, add 10-20% safety factor to the TOTAL power requirement and then get a unit that covers that amount.)

11. AC OUT JACKS (2)
For AC feed to peripherals. (Note: peripherals drawing AC through the UV/1 raise the line protection requirement for it.)

12. POWER ON/OFF SWITCH
Red power indicator light is on the front panel.

End of REAR PANEL DESCRIPTIONS

INITIAL SET UP

Now that you've got your UV-1 ZGRASS Graphics System out of the box, and

BEFORE YOU PLUG IT INTO AC POWER,

open up the back and make sure that none of the boards or connectors inside have come loose in shipping.

With the power still OFF, set the TERMINAL interface Baud Rate according to your terminal's requirements. See Baud Rate Selection section of this manual for that procedure. 19200 is the recommended speed, although 9600 is acceptable.

When you've got the Baud Rate set properly, close up the back of the computer; plug in the UV-1, terminal, and video monitor to AC power and make appropriate connections.

POWER-ON your terminal and video monitor.

When these are warmed up, POWER-ON the UV-1. The "ZGRASS!" prompt will appear at once on the terminal and the video monitor will show a white screen. Throw the SOFTWARE RESTART switch and the console lights will blink (first RESTART only).

If any of these three devices don't come up, turn to the Troubleshooting section of this manual before continuing with system tests.

1. JOYSTICKS AND JOYSTICK JACKS

At least one handcontrol, called a 'joystick' is included as a standard accessory with your UV-1 ZGRASS Graphics System. Programming control is accessed according to the number of the joystick jack (Front Panel, numbered 1- 4), NOT the number on the joystick knob. So either plug each joystick into the correspondingly numbered jack or ignore the knob numbers altogether.

The knob on top of the joystick can be moved in 8 directions, forward, backward, left, right, and at 45 degree angles. This control is accessed through the DEVICE VARIABLES $X1, $X2, $X3, $X4 (X-AXIS movement) and $Y1, $Y2, $Y3, $Y4 (Y-AXIS movement).

Knob rotation is accessed via $K1, $K2, $K3, $K4 and the trigger is accessed via $T1, $T2, $T3, $T4.

With the following test program, you can see the values of the various joystick DEVICE VARIABLES change in response to manipulation of the joystick controls. This tests both the individual joystick itself and the jack it is plugged into. (See glossary under JOYSTICK and DEVICE VARIABLES).

```
JEST=[PROMPT "WHICH JOYSTICK/JACK FOR TESTING?"
INPUT A
SKIP A
PRINT "JOY #",1,$X1,$Y1,$K1,$T1;SKIP 0
PRINT "JOY #",2,$X2,$Y2,$K2,$T2;SKIP 0
PRINT "JOY #",3,$X3,$Y3,$K3,$T3;SKIP 0
PRINT "JOY #",4,$X4,$Y4,$K4,$T4;SKIP 0]
```

If that joystick and jack work, hit BREAK or CNTRL+C, plug into another jack and test that one. Play musical joysticks/jacks until you've tested all of them.

2. CONSOLE LIGHTS AND SWITCHES

The (16) console switches are turned on by depressing the lower rocker, throwing the upper rocker out toward you. This test program will turn on the light above the switch thrown. Press BREAK or CTRL+C to get out.

```
BLINK=[PORT 38,PORT 20
PORT 39 ,PORT 21
SKIP -2]
```

To see how the switches work, enter and run:

```
SWITCH=[PRINT PORT 20;SKIP 0]
```

Starting with all switches off, turn on one at a time beginning with 0. Your terminal will display PORT 20's binary value as read from each switch individually.

SWITCH #(PORT 20)	7	6	5	4	3	2	1	0
SWITCH #(PORT 21)	15	14	13	12	11	10	9	8
PORT VALUE READ	128	64	32	16	8	4	2	1

By changing the program to read PORT 21 instead, you can do the same for Switches 8-15. Turn on more than one switch per set at once and the summed value will be read via the PORT # for that set (e.g. with a set of switches all on, the value for that PORT will be 255). See glossary under PORT.

4. 3-VOICE SOUND SYNTHESIZER OUTPUT

```
TONE=[PORT 22,15
A=A+1
PORT 17,A
SKIP -2]
```

Enter and run this program to test Line Audio Out, Speaker, Headphone Output and Volume Control with the whooping tone that results. To stop the tone, press BREAK or CTRL+C.

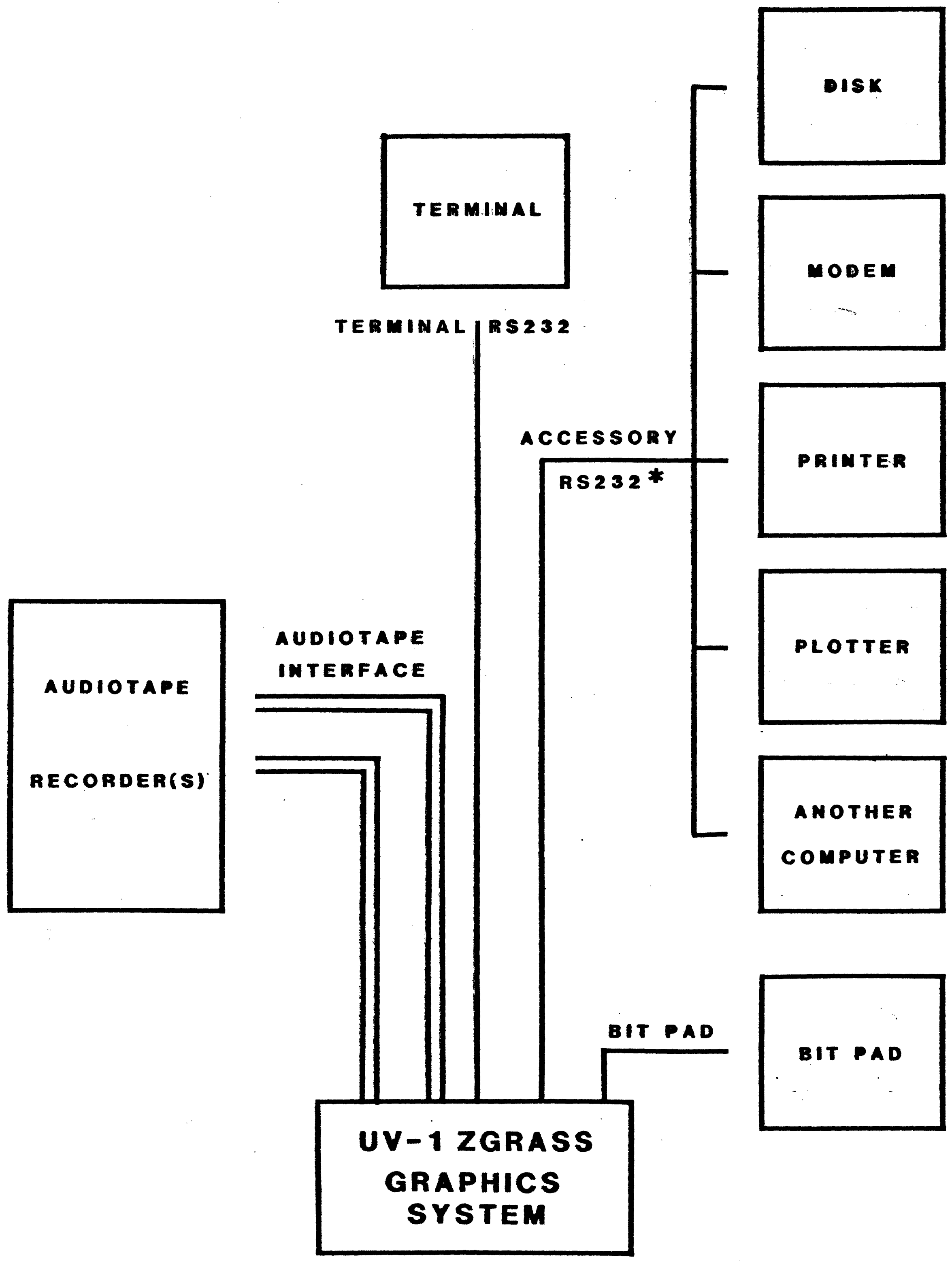
INTERFACES

Many of the features which make the ZGRASS Graphics System a powerful tool are those which support flexible interfacing with other data and video devices.

Some of these features are the programmability of DC control signals and console switches, the 3-voice sound synthesizer, hardware interfaces and software for interactive controls such as joysticks, light pen, and tablet, as well as an extra card space for such user-specified additions as voice synthesizer or interface for the Sandin Digital Image Processor.

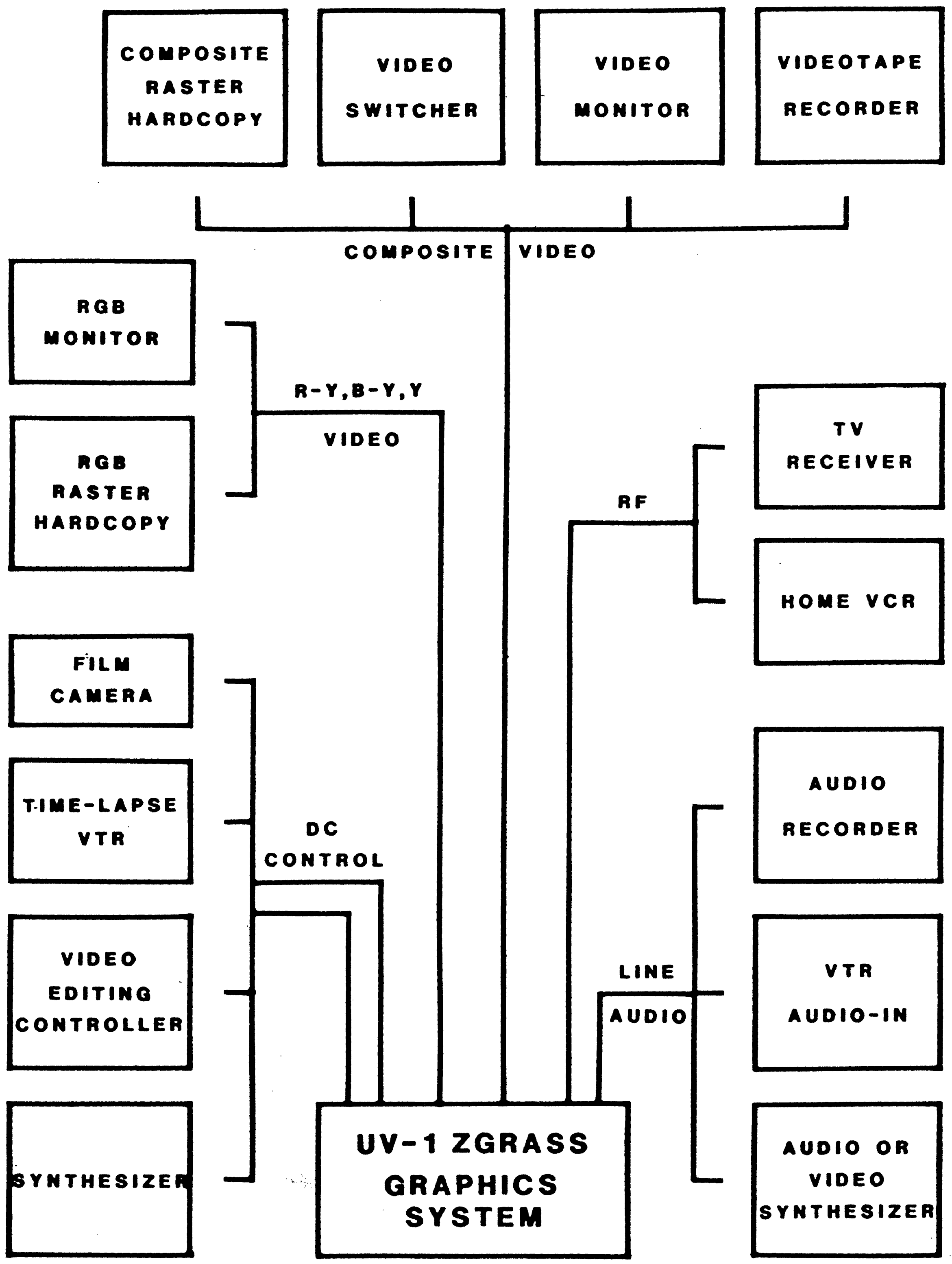
NOTE: Currently the UV-1 is compatible with PERSCI DISK model 277 with 1070 interface and direct or via modem with UNIX (tm Bell Laboratories) if specified. Contact DATAMAX for current interfaces and for future availability of interfaces with mini-disks, Winchester-type disks, etc.

DATA INTERFACES



*SOFTWARE MUST BE PROVIDED BY EXPERIENCED USER.

VIDEO INTERFACES

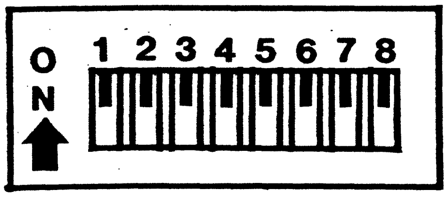


BAUD RATE SELECTION

To make the UV-1 ZGRASS Graphics System communicate properly with such peripherals as a terminal, disk drive, printer, plotter, modem, etc., you must set the (8) Baud Rate Selector Switches accordingly.

MAKE SURE AC POWER IS OFF BEFORE PROCEEDING!!!

To set Baud Rate for the TERMINAL and ACCESSORY RS232 interfaces, open the back of the computer and identify the LOGIC board (labeled as such, probably the topmost board, and has (2) 50-pin ribbon cable connectors attached to it). First, disconnect the lefthand ribbon cable connector (careful not to bend any pins!), spread the card ejectors and then gently pull the LOGIC board far enough out of the rack to get at the Baud Rate Selector Switches which look like this:



BAUD RATE SELECTOR SWITCHES

Switches 1-4 control the ACCESSORY interface.

Switches 5-8 control the TERMINAL interface.

Now set the switches to the proper Baud Rate for your peripherals according to this table. NOTE: the settings shown apply to either set of 4 switches, either 1,2,3,4 for ACCESSORY or 5,6,7,8 for TERMINAL.

BAUD RATE		SWITCH SETTINGS
50	=	on-on-on-on
75	=	off-on-on-on
110	=	on-off-on-on
134.5	=	off-off-on-on
150	=	on-on-off-on
300	=	off-on-off-on
600	=	on-off-off-on
1200	=	off-off-off-on
1800	=	on-on-on-off
2000	=	off-on-on-off
2400	=	on-off-on-off
3600	=	off-off-on-off
4800	=	on-on-off-off
7200	=	off-on-off-off
9600	=	on-off-off-off
19200	=	off-off-off-off

EXAMPLES: to set TERMINAL switches to 9600 Baud Rate, you would set switches 5,6,7,8 to on,off,off,off settings respectively and to set ACCESSORY switches to 300 Baud Rate, you would set switches 1,2,3,4 to off,on,off,on settings respectively.

Now gently slide the LOGIC board back into the rack. If it does not go back in easily, gently jiggle it a bit from side to side until it slips in...DON'T FORCE IT. Reconnect the lefthand ribbon cable connector. Now close up the back of the computer before you turn on the power again.

See glossary for TERMINAL command to reconfigure Editor keys.

AUDIOTAPE INTERFACE

The Audiotape Interface allows you to store files (MACRO, STRING, ARRAY, SWAP MODULE, screen dump) on audio tape and then read files back into memory. The Audiotape Interface transmits/receives computer data-as-audio at approximately 2,000 Baud Rate.

As a file is read into memory, it is checked for errors. Should any occur, the next copy of the file specified will be read automatically. In order to take advantage of this feature, be sure to specify that several copies be made of each file you want to store.

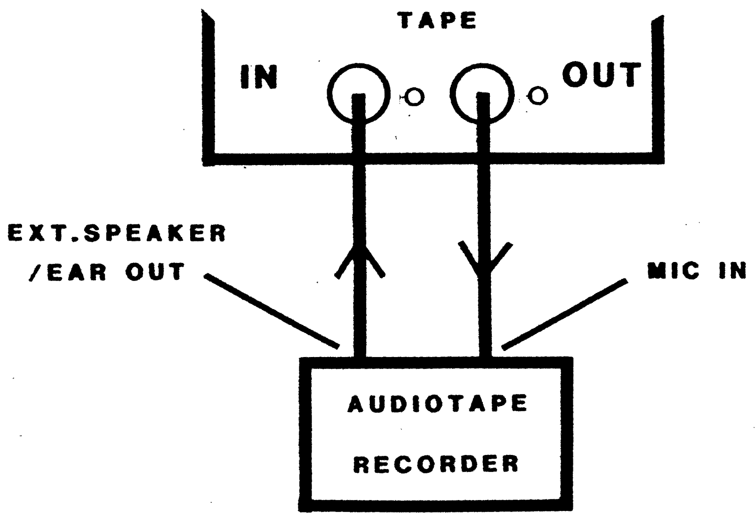
Different models of audio tape recorders have a variety of controls and connections, so experiment with interfacing your recorder(s) and test the read/write operations with a simple program before attempting to store anything important.

Use standard, shielded coaxial audio cables for Audiotape Interface connections with the correct connectors on both ends.

For details about direct control of the DC remote motor control signals, see glossary under PORT.

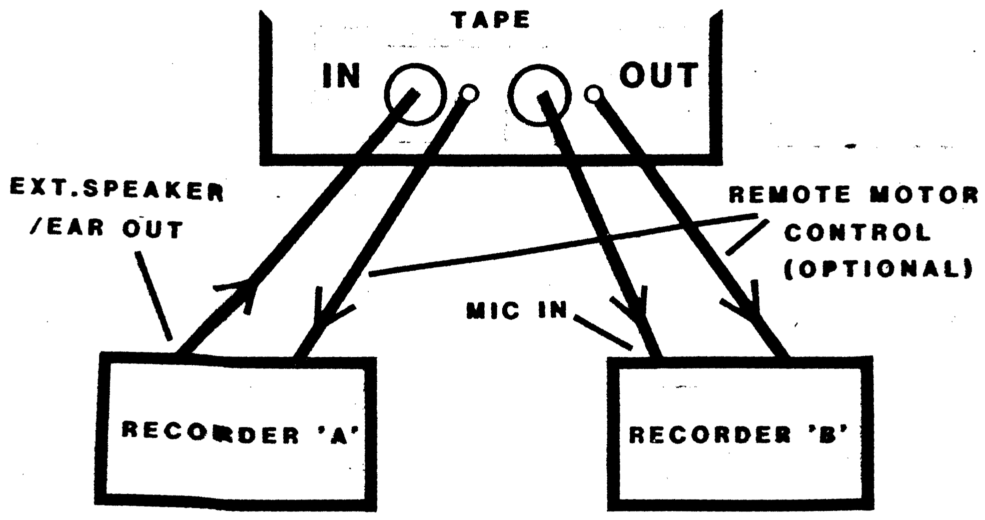
1. CONFIGURATIONS

A. To use ONE AUDIO RECORDER for BOTH play and record (read and write), connect audio cables as follows:



With this configuration, reading files into memory and writing files onto tape from memory are separate tasks usually punctuated by changing tapes.

B. To use TWO AUDIO RECORDERS, one to play and one to record (primarily useful for editing/copying tapes), connect audio cables as follows:



This configuration enables you to easily read files into memory from Recorder A (the 'player'), edit them, and then write the new versions on another tape in Recorder B (the 'recorder').

2. TUNING

A. To RECORD files on audio tape:

If your recorder has a record-volume control dial, set it in the upper half of its range.

If your recorder has a VU meter, set record level at 0db.

B. To PLAY files into memory:

Set playback volume control in the upper half of its range.

Set tone control (if present) to 'High' or treble bias.

BEFORE PROCEEDING FURTHER, READ

ABOUT GETTAPE AND PUTTAPE COMMANDS

IN THE GLOSSARY

3. PROCEDURES

A. RECORD

Find the location on the audio tape where you want to store the file and its copies.

Type in:

```
PUTTAPE NUMBER,FILENAME,STRING
```

start recording, and then press the RETURN key on your terminal.

When the Record operation is completed, the attention mark will be returned. Note that the number of copies you specify is displayed in the LED's and counts down as the copies are put on tape. If the transfer is taking about a minute or so, you may have misspelled the FILENAME and are getting a dump of the screen onto tape unintentionally.

B. PLAY

You can either set the tape at the exact location just before the stored file you want to bring into memory OR simply start at the beginning and have the computer search the tape for the file you want.

Type in:

```
GETTAPE FILENAME
```

Start the recorder in 'play' and then press the RETURN key on your terminal.

If you press CTRL+N before you press RETURN, you will get a complete directory listing of the contents of the tape.

Use BREAK to prematurely stop the GETTAPE process.

Switches:

.ERR	accept the file even if an error is read.
.ANY	get the next file whatever its NAME is on the tape and read it in with the NAME you specify.

AUDIOTAPE INTERFACE TROUBLESHOOTING

IF YOUR FILES ARE NOT BEING PROPERLY RECORDED ON THE TAPE:

Check recording levels. If computer data is actually recorded on the tape, you should be able to hear spurts of loud squeals and burbles in playback when the ext.spkr/ear connector is unplugged.

Check cable connections.

Are you using adaptors? Try direct connect with proper connectors on each end of the cable.

Do computer or other signals already exist on the tape? It is generally advisable to record computer data on a blank or well-erased tape.

If you're using one recorder only, maybe you're getting crosstalk between the play and record circuits. Try disconnecting the plug in the ext.spkr/ear jack when recording. (To avoid confusion, color code your cables.)

De-magnetize record heads on the recorder.

Check your understanding of the PUTTAPE command.

IF YOUR FILES ARE NOT BEING PROPERLY READ INTO MEMORY:

Check recorder playback volume control and playback tone control.

Check cable connections.

Are you using adaptors? Try direct connect with proper connectors on each end of the cables.

Perhaps there are errors in the stored file. Try GETTAPE.ERR which will read the file in as best it can.

Check recorder output voltage level.

Try another tape recorder.

Check your understanding of the GETTAPE command and options.

VIDEO

One of the most useful features of the UV-1 ZGRASS Graphics System is its ability to output real-time computer graphics as an NTSC standard video signal and interface with video equipment ranging from home videocassette recorders to educational video environments and commercial production switchers.

1. CHOOSING YOUR VISUAL FEEDBACK DEVICE

The UV-1 ZGRASS Graphics System hardware and software has been optimized for accessible, interactive control structures and an environment of rich, rapid feedback. Your terminal gives you feedback about your program, the status of variables,etc., and for graphic feedback you have the choice of TV receiver, RGB monitor, or composite video monitor.

The form of visual feedback you provide yourself depends to a high degree on what form of presentation/distribution you will be using. An educational environment using 3/4" videotape has different requirements from a corporate environment using photographic hardcopy from an RGB plotter.

Feeding RF to a TV receiver has the virture of lowest cost and it interfaces with all home video formats but gives the least precise color feedback.

An RGB monitor gives the sharpest color feedback, is indispensible for working with RGB plotters, slides, etc., but is a specialized device not always immediately interfaced with video equipment.

A composite video monitor provides feedback that is representative of how your graphics will look to the world of NTSC video, live and recorded. It is also a fairly generalized device, interfacing easily with recorders, switchers, etc.,and is available in a wide price range.

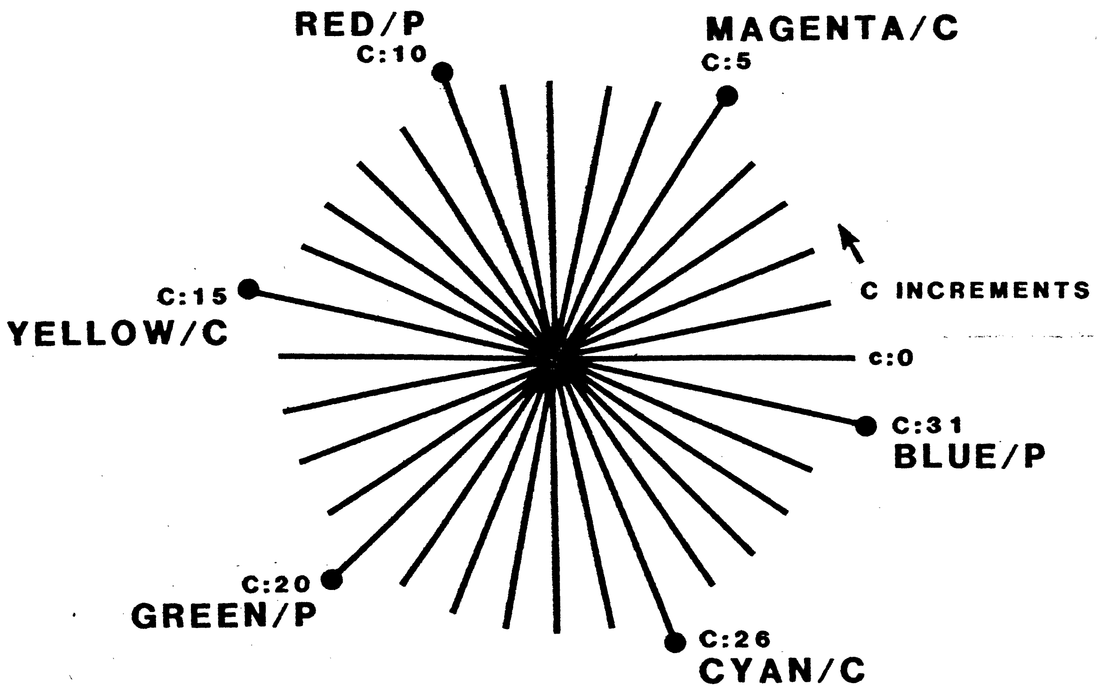
(See Interfaces section of this manual)

2. COLOR FEEDBACK

The actual 'color' that you see on the video screen at any given point is determined by a number of components. The range of visually distinguishable COLORS VALUES is 0-255. Each one can be analyzed according to this formula:

COLOR = C*8+G
where C = chrome value component (0-31 range)
      G = grey value component (0-7 range)

This diagram shows the progression of chrome component values and identifies the video primaries and complements to +/- one value accuracy. Their various locations closely approximate a vector scope display of video colorbars.



For each video primary and complementary (identified above by /P and /C) chrome value, there is a particular grey value that, when combined, produces a visually fully saturated COLOR VALUE. These are:

BLUE : GREY VALUE of 1
RED : GREY VALUE of 2
MAGENTA : GREY VALUE of 3
GREEN : GREY VALUE of 4
CYAN : GREY VALUE of 5
YELLOW : GREY VALUE of 6

You will notice, however, that grey values are somewhat interactive, depending on relative screen area, proximity, field grey value, etc.

This program generates the closest possible (for ZGRASS) approximation of NTSC colorbars.

```
CBARS=[CL;A=-149;C=0;$HB=21
$RO=0;$R1=82;$R2=43;$R3=249
$LO=7;$L1=213;$L2=126;$L3=164
IF A<115,BOX A=A+45,0,46,202,C=(C+1)\3+1;SKIP 0]
```

If you run this program and turn down the chroma or color control on your video monitor, you will see the stepped grey value scale. See glossary under COLOR, COLOR MAP, COLOR MODES, COLORS, DISPLAY MODES.

4. CENTER OF THE SCREEN

Draw this box on the screen: BOX 0,0,320,202,1

The area of this box represents the ZGRASS addressable area, i.e. that portion of the video raster where you can use ZGRASS commands to draw and manipulate graphics. (The color of the margins outside the addressable area can be assigned via the $BC DEVICE VARIABLE.) The area of this box also represents the default WINDOW (see glossary).

Now draw this box: Box 0,90,50,50,3

You can see that any portion of a graphic element located outside this addressable area will be clipped.

The large box also shows you how the alignment (i.e. the centering of the video raster on the CRT) on your particular receiver or monitor is biased. The alignment of TV receivers and monitors is rarely perfect, with a range of variations in direction and extent, most however seem slewed to the viewer's left.

Now you have the choice of allowing for your device's bias when programming your graphics, having your receiver or monitor aligned, or using a monitor with underscan capability so you can see the entire video raster. Whichever choice you make you still should allow for the diversity of alignment biases. (Alignment adjustment is a standard service any TV repairperson can provide and some monitors have these controls user-accessible.

(See glossary under CENTERING, CENTER XCOOR,YCOOR and COORDINATES.)

5. COMMUNICATIONS TOOL

Because of the UV-1 ZGRASS Graphics System's fairly unique compatibility with the world of NTSC video, many easy and interesting interfaces with the television communication world are possible for live and recorded computer graphics. Besides the standard role of creative/production/education tool, there are other possibilities.

Educational: Record a graphic sequence on the video channel of a videotape along with audio synthesis soundtrack from the Line Audio Out and then record the program data which executes that sequence onto the second audio track from the Audiotape Interface Out. If your audience does not have access to a UV-1, use the TEXT command to turn the listing into graphics and record it as video. Or you could record yourself giving a mini-tutorial or friendly greeting on one of the audio channels. This turns one videotape into a self- contained, triple-channel education/communication package.

Live Television: Many cable TV channels are switched and logged by computer. The UV-1 could add ongoing, live pattern generation or other periodic graphics to a cable TV channel's menu.

Watch this space for future developments!

TROUBLESHOOTING

IF POWER INDICATOR LIGHT DOES NOT COME ON WHEN POWER SWITCH is thrown and no output appears on terminal or video monitor, check fuse.

IF NOTHING APPEARS ON THE TERMINAL, but the computer looks alive check Terminal RS232 connection, check Baud Rate setting, RESTART ZGRASS, check gain control on the terminal, turn the terminal off and on again. Try operating terminal in LOCAL mode if possible.

IF THERE IS NO OUTPUT TO THE VIDEO MONITOR (without graphics commands the screen will be white), check connections, try alternate feedback mode to isolate malfunction (e.g. try RF if no composite video output).

IF USEMAP SAYS 'ZAPPED' OR ERROR #27 OCCURS REPEATEDLY, there has been some kind of confusion in memory and it is best to salvage what you can and then RESTART ZGRASS. Memory will be more efficiently and smoothly allocated if files are retrieved in order of descending storage size. USEMAP tells you the number of bytes each file takes up.

IF MEMORY TRASHES UNPREDICTABLY, you might have a problem with static electricity, which can cause malfunctions in electronic devices, especially computer memory. Pile carpets and dry air contribute to the creation of static electricity. Many terminals have non-metal, anti-static chassis. If static electricity causes a memory dump, simply do a software RESTART and begin again, and remember to back-up your work more often.

DATAMAX UV-1

Zgrass GLOSSARY & Zgrass LESSONS

October 27, 1981

****

-ADM 5 Dumb Terminal Video Display Unit	Lear Siegler, Inc. Data Products Div. Anaheim, CA
-Micropolis Disk Drive	Micropolis Corporation Canoga Park, CA
-Bit Pad One Data Tablet/Digitizer	Summagraphics Corporation Fairfield, CT
-Mini-Winchester Disk Drive	International Memories,Inc. Cupertino, CA

DATAMAX UV-1
Zgrass GLOSSARY

October 27, 1981

DATAMAX UV-1 Zgrass

RESIDENT Zgrass COMMANDS & FUNCTIONS

October 27, 1981

***GRAPHICS/ ARRAYS:**

- ARRAY
- ARRAY.FLOAT
- ARRAY.STR
- BOX
- CENTER
- CLEAR
- CLEAR.CRT
- CLEAR.WINDOW
- DISPLAY
- LINE
- PATTERN
- PATTERN.FILL
- MMOVE
- MMOVE.UP
- POINT
- POINT.SNAP
- SCALE
- SCROLL
- SHRINK
- SNAP
- STRIPE
- TEXT
- WINDOW
- WINDOW.BOX
- WINDOW.FULL

***DISK:**

- DBAKS
- DCREATE
- DDELETE

(continued)

***DISK:**

- DDELETE.BAK
- DFETCH
- DFETCH.ZAP
- DGET
- DGET.BAK
- DGET.OR
- DGET.XOR
- DINIT
- DINIT.WINCH
- DLOAD
- DLOAD.CLEAR
- DLOAD.ZAP
- DLOOK
- DPUT
- DPUT.TV
- DSETUP
- DSETUP.RESET
- DUSEMAP
- DYANK

***INPUT/ OUTPUT:**

- CONTROL P
- GETDISK
- GETTAPE
- INPUT
- INPUT.NAME
- INPUT.STR
- PERSCI
- PORT
- PRINT
- PRINT.FORCE

(continued)

***INPUT/ OUTPUT:**

- PROMPT
- PROMPT.FORCE
- PUTDISK
- PUTTAPE
- RS232
- TABLET
- TERMINAL

***MATH:**

- ARCCOS
- ARCSIN
- ARCTAN
- COSINE
- EXP
- FLOAT
- FLOAT.DEG
- FLOAT.LEN
- FLOAT.OFF
- FLOAT.RAD
- INT
- LN
- LOG
- POWER
- SINE
- SQRT
- TANGENT

***MISCELLANEOUS:**

- COMPILE
- CONTROL

(continued)

***MISCELLANEOUS:**

- DELETE
- EDIT
- LOOPMAX
- RESTART

***PROGRAM FLOW:**

- .B
- .F
- GOTO
- IF
- JUMP
- RETURN
- SKIP
- STOP
- TIMEOUT
- WAIT

***STRING MANIPULATION:**

- ASCII
- LEN
- LPAD
- STRING
- SUBSTR

***USER INFO:**

- ADDRESS
- ANYARGS
- CORE
- HELP
- STATUS
- USEMAP
- VERSION

DATAMAX UV-1 Zgrass
INDEX OF BUZZWORDS & IDIOSYNCRASIES
October 27, 1981

BUZZWORDS
(common computer terms)

- ADDRESS
- ALGORITHM
- AND
- ARGUMENT
- ARGUMENT LIST
- ASSIGNMENT
- ASSIGNMENT OP-ERATOR
- BIT
- BYTE
- BYTE ARRAY
- CALL
- CLIPPING
- COMMAND
- COMMENT
- CONCATENATION
- CONSTANT
- CONTROL CHAR-ACTERS
- EXCLUSIVE OR
- EXECUTE
- FILES
- FLOATING POINT
- FRAME BUFFER
- FUNCTION
- INDEX
- INDIRECTION
- INFINITE LOOP
- INTEGER
- INTERRUPT
- ITERATION
- LOGICAL OPERATOR
- LOOP
- MEMORY
- NUMBER
- NUMERIC VARIABLE
- OR
- OVERFLOW
- PIXEL
- PLOP
- PRECEDENCE
- PUNCTUATION
- RADIANS
- RANDOM
- RECURSION
- RELATIONAL OPERATOR
- RESOLUTION
- REVERSE PRIORITY
- SEMANTICS
- SNAPPED PIX
- STRING
- STRING VARIABLE
- SWITCH
- SYNTAX
- TRUTH TABLES
- VALUE
- VARIABLE
- WRAP AROUND
- XOR

IDIOSYNCRASIES
(special Zgrass terms)

- ABBREVIATION
- CENTERING (of graphics primitives)
- COLOR
- COLOR MAP
- COLOR MODES
- COORDINATES
- CURSOR
- DEVICE VARIABLES
- DISK
- DISPLAY MODES
- ERROR NUMBER
- EXPRESSION
- JOYSTICK
- LABEL
- LOCAL VARIABLE
- MACRO
- NAME
- NEXTLINE
- OPERATOR
- PORTS
- PRIORITY WRITE
- SCREEN
- SWAP COMMAND or FUNCTION

DATAMAX UV-1 Zgrass
SWAP COMMANDS and SWAP FUNCTIONS
October 27, 1981

***GRAPHICS/ ARRAYS:**

✓ ELLIPSE
CMPARA
FONT
→ PANORAMA - not Listed in Glossary
✓ TXT
✓ WRAP

***ERROR HANDLING:**

GETERROR

***STRING MANIPULATION:**

BUMP
MATCH
REPLACE
SLDDR
SLDIR
SZAP

***DISK:**

DCHECK
DCOPY
DDSMAP
DFORMAT
DMATCH
DRENAME
DZAP

***MISCELLANEOUS:**

XR
ZAP1
ZAP 2

***PROGRAM FLOW:**

***USER INFO:**

DEBUG
SHOW
WHATSIS

ZGRASS Glossary of: BUZZWORDS, COMMANDS, FUNCTIONS IDIOSYNCRASIES, SWAP COMMANDS, SWAP FUNCTIONS, SWITCHES, AND ESOTERICA

October 27, 1981

Note: BUZZWORDS are common computer terms. IDIOSYNCRASIES are concepts and features peculiar to or specially modified for ZGRASS. SWAP COMMANDS and SWAP FUNCTIONS have to be gotten from disk or tape first. SWITCHES modify commands. The ESOTERICA are the advanced features for experienced programmers.

ABBREVIATION
Idiosyncrasy

you can abbreviate COMMAND, FUNCTION, VARIABLE, and MACRO NAMEs. For example:

```
     PRINT 5
```

is the same as:

```
     PR 5
```

This can cause confusion if you are not careful when you abbreviate NAMEs.
Example:

```
     TRY1=6
     TR=2
```

will cause TRY1 to be equal to 2 because TR is a valid abbreviation for TRY1.
To verify this:

```
     PRINT TR,TRY1
```

ADDRESS
Esoteric Buzzword

the number which corresponds to the location of data in MEMORY.

ADDRESS(NAME)
Esoteric Function

returns an INTEGER which represents the ADDRESS of the NAME.
Example:

```
     SAM=5
     PR ADDRESS(SAM)
```

returns a number corresponding to SAM's address in decimal.

ALGORITHM
Buzzword
is a method you use to solve a problem.

AND
Buzzword
works on BITs. It makes 1's AND'ed with 1's equal to 1; and all other combinations produce 0.
AND table using 2 BITs:

	12	13	14	15
AND	00	01	10	11
00	00	00	00	00
01	00	01	00	01
10	00	00	10	10
11	00	01	10	11

The AND COLOR MODES are 12-15. The AND DISPLAY MODES are 3,13,23,...,133,143.

ANYARGS()
Esoteric Function
returns 0 if no ARGUMENTs left in the ARGUMENT list passed to a MACRO and 1 if there are ARGUMENTs left in the ARGUMENT list.
Example:

```
     ADDEMUP=[SUM=0
     IF ANYARGS()==1,INPUT A;SUM=SUM+A;SK 0
     PRINT SUM]
     ADDEMUP 5,10,15,20
```

ADDEMUP will add up all the arguments passed to it, and then print the total, which is 50 in this case.

ARCCOS(NUMBER)
Function
returns the inverse cosine of NUMBER.

ARCSIN(NUMBER)
Function
returns the inverse sine of NUMBER.

ARCTAN(NUMBER)
Function

returns the inverse tangent of NUMBER.

ARGUMENT
Buzzword

is computer talk for the stuff between commas that you give to a COMMAND, FUNCTION, or MACRO. (Actually, the first ARGUMENT has a space or '(' to its left and the last has a NEXTLINE, ';' or ')' to its right, but there are always commas in between ARGUMENTS). ARGUMENTS must be VARIABLEs, NUMBERs, or EXPRESSIONs. Generally speaking, the presence of an ARGUMENT does not mean anyone is disagreeing about anything.
Note: superfluous spaces between ARGUMENTs and at the end of the line are not allowed. CTRL+Y will place a "!" at the end of each line marking the NEXTLINE so you can tell if there is an extra space between the last ARGUMENT and the NEXTLINE.

ARGUMENT LIST
Buzzword

is the list of ARGUMENTS that you give (pass) to a COMMAND, FUNCTION, or MACRO. You assign the passed ARGUMENTS to VARIABLEs in a MACRO by using the INPUT COMMAND (see INPUT).
Esoteric Note:
VARIABLEs are passed by NAME. Complex EXPRESSIONs (A+6-2) are EXECUTEd when they are passed. If you want to pass a VALUE, and the value is in a single VARIABLE (not an expression), use the "?" OPERATOR.
For instance:

```
     A=10
     PRINT A,A=100
```

will print 100,100. Since the ARGUMENTs are scanned before they get to PRINT.

```
     A=10
     PRINT ?A,A=100
```

will print 10,100
It is especially important to note that if LOCAL VARIABLEs are passed by NAME (no "?"), the called MACRO will not be able to access the LOCAL VARIABLE of the calling MACRO. If you must pass by VALUE, the following is an example of how to do it:

```
     FEE=[a=100
     FOO ?a]

     FOO=[INPUT b
     PRINT b*b]
```

Using "a+0" will also force evaluation for numerical VARIABLES. For STRINGs use "?" (for example, ?ABC), or CONCATENATE a null string. (i.e., ABC&[])
This problem shows up in global VARIABLES too. Compare:

```
     TOM=[A=100
     SAM A]

     SAM=[A=10
     INPUT B
     PRINT B*B]
```

will print 100 whereas:

```
     TOM=[A=100
     SAM A+0]
```

will print 10000
If you want to force passing by VALUE, use the "?" OPERATOR. ZGRASS needs to be able to pass by NAME so the ASSIGNMENT OPERATOR can be used in EXPRESSIONs and so certain FUNCTIONs (like TABLET, for example) can return more than one VALUE.

ARRAY NAME,NUMBER
Command

creates a FLOATING POINT array with elements referenced by NAME(0), NAME(1),...,NAME(NUMBER-1). ARRAYs up to four dimensions are allowed.
Example:

```
     SHOW=[ARRAY JANE,200
     A=0
     JANE(A)=1%100
     A=A+1
     IF A<10,SK -2
     CLEAR.C
     USEMAP
     A=0
     PRINT "JANE("&A&")="&JANE(A)
     A=A+1
     IF A<10,SKIP -2]
     SHOW
```

When you run SHOW, it will first create the ARRAY JANE, then assign a RANDOM number to each element in JANE, then generate a USEMAP listing so you can see the size of JANE, and finally print out the first ten elements. If you change ARRAY JANE to ARRAY.INT JANE, you will notice USEMAP lists JANE as about half as big. For another ARRAY example see INDIRECTION.

ARRAY.INT NAME,NUMBER
Command

creates a FIXED POINT array with elements referenced by NAME(0), NAME(1),...,NAME(NUMBER-1). ARRAYs up to four dimensions are allowed. (See definition of INDEX.)
Examples:

```
     ARRAY ROOTS,10
```

will create a 10 element array referenced by ROOTS(0),...,ROOTS(9).

```
     CARS=[ARRAY BUICK,100
     A=0
     BUICK(A)=1%320
     A=A+1
     IF A<100,SK -2
     A=0
     BOX 0,0,BUICK(A),BUICK(A+1),7
     A=A+2
     IF A<100,SK -2]
```

will fill an array, BUICK, with 100 RANDOM VALUES and use them to draw 50 BOXes.

```
     ARRAY CHECKER,10,10
```

will create a 100 element array referenced by CHECKER(0,0),CHECKER(0,1),...,CHECKER(9,9).
For another example, see INDIRECTION.

ARRAY.STR NAME,NUMBER
Esoteric Command

creates a STRING array with string elements referenced by NAME(0), NAME(1),...,NAME(NUMBER-1). ARRAYs up to four dimensions are allowed. To store STRING ARRAYs on tape or disk, you need to use GTSTRING/PTSTRING or GDSTRING/PDSTRING, SWAP MODULES which are not yet available.
Example:

```
     ARRAY.STR ATHRUZ,26
     ALPH=[I=0
     ATHRUZ(I)=ASCII(I+65)
     PRINT "ATHRUZ("&I&")="&ATHRUZ(I)
     IF (I=I+1)<26,SK -2]
```

This MACRO will fill the STRING ARRAY ATHRUZ with the letters A-Z and print them out. For another ARRAY example, see INDIRECTION.

ASCII(NUMBER)
Esoteric Function

returns a one character STRING corresponding to NUMBER, an ASCII value. ASCII is the coding system for characters, numbers and punctuation. Refer to a standard ASCII table for specific values. The STRING COMMAND takes characters and returns their ASCII values.

Example:

```
    NUMS=[K=48
    ZEROTONINE=ZEROTONINE&ASCII(K)
    IF (K=K+1)<58,SK -1
    PRINT ZEROTONINE]
```

The ASCII values for the characters 0-9 are 48-57. This MACRO CONCATENATES the characters 0-9 and then prints them out as "0123456789".

ASCII VALUES FOR CONTROL CHARACTERS, NUMBERS, CAPITAL LETTERS, SMALL LETTERS, AND SYMBOLS

00 NUL	21 NAK	42 *	63 ?	84 T	105 i
01 SOH	22 SYN	43 +	64 @	85 U	106 j
02 STX	23 ETB	44 '	65 A	86 V	107 k
03 ETX	24 CAN	45 -	66 B	87 W	108 l
04 EOT	25 EM	46 .	67 C	88 X	109 m
05 ENQ	26 SUB	47 /	68 D	89 Y	110 n
06 ACK	27 ESC	48 0	69 E	90 Z	111 o
07 BEL	28 FS	49 1	70 F	91 [	112 p
08 BS	29 GS	50 2	71 G	92 \	113 q
09 HT	30 RS	51 3	72 H	93]	114 r
10 LF	31 US	52 4	73 I	94 ^	115 s
11 VT	32 SP	53 5	74 J	95 _	116 t
13 CR	34 "	55 7	76 L	97 a	118 v
14 SO	35 #	56 8	77 M	98 b	119 w
15 SI	36 $	57 9	78 N	99 c	120 x
16 DLE	37 %	58 :	79 O	100 c	121 y
17 DC1	38 &	59 ;	80 P	101 d	122 z
18 DC2	39 '	60 <	81 Q	102 e	123 {
19 DC3	40 (	61 =	82 R	103 f	124 ¦
20 DC4	41)	62 >	83 S	104 g	125 }

ASSIGNMENT
Buzzword

Examples:

```
    A=100
```

This assigns the VALUE 100 to the VARIABLE A.

```
    LETTERS="ABCDEFG"
```

assigns the STRING "ABCDEFG" to the VARIABLE LETTERS.

```
     LONGSTRING="THIS IS A  VERY VERY  LONG STRING
     WITH NEXTLINES  AT  THE  END  OF  EVERY LINE.
     NOTICE  YOU   CAN  HAVE   NEXTLINES,  COMMAS,
     PERIODS, AND ANY  OTHER PUNCTUATION  EXCEPT A
     DOUBLE QUOTE IN THIS CASE."
```

Note that you can assign very long STRINGs to VARIABLES.

```
     NULLSTRING=""
```

A VARIABLE can have a NULL STRING as its VALUE.

```
     ROOT=(-B+SQRT(B*B*A*C))/2
```

EXPRESSIONs can be assigned to a VARIABLE.
You can put ASSIGNMENTs in EXPRESSIONs:

```
     TOM=[IF A<160,BOX 0,0,A=A+10,A,3;SKIP 0]
```

ASSIGNMENT OPERATOR
Buzzword

is the '=' sign.

.B
Switch

NAME.B means run NAME in the background over and over again interleaved with other .B MACROs, if any, until CTRL+C or STOP NAME is seen. You will notice that when you .B a MACRO the ">" cursor is still there which means you can issue COMMANDs from the keyboard, EXECUTE other MACROs, or .F MACROs, all of which take precedence.
Example:

```
     BOX 0,0,36,36,1
     BOX 0,18,4,8,3
     SNAP APPLE,0,4,48,48
     CLEAR
     ANIMATE=[DISPLAY APPLE,X=X+$X1,Y=Y+$Y1,0]
     ANIMATE.B
```

will move the APPLE (a SNAPped picture element) under the control of the first JOYSTICK until further notice. Try COMPILing ANIMATE to see the APPLE move faster. Then try typing in other COMMANDs and see the .B MACRO stop while the COMMAND is EXECUTEd.

```
     COMPILE ANIMATE,FASTER
     FASTER.B
```

Any MACRO called by a .B MACRO will be executed as if it were a single line, that is, without interleaving with other .B MACROs.

To interleave .B MACROs with regular MACROs, use CTRL+A.

BIT
Buzzword

is a single binary value, either 0 or 1. There are two BITs for each PIXEL on the screen. Since one BIT can specify one of two NUMBERS, two BITs can specify four NUMBERS, which is why four COLORs can be displayed on the screen at any one point. There are eight BITs in a BYTE, and, in this system, sixteen BITs in an INTEGER and thirty two bits in a floating point number.

BOX XCENTER,YCENTER,XSIZE,YSIZE,COLORMODE
Command

draws a filled rectangle of the dimensions XSIZE by YSIZE, centered at XCENTER,YCENTER with drawing mode specified by COLORMODE (see COLOR MODES for the 21 options). If used as a function, a -1 is returned if the bit is entirely off the screen; and if an OR or XOR mode is used, a 0 is returned if nothing non-zero was written over and a 1 is returned if something was written over.
Example:
```
     BOX 0,0,80,60,1
```
draws a rectangle centered at 0,0 which is 80 PIXELS wide, 60 PIXELs high, and is drawn in COLORMODE 1. If you draw a BOX which as a whole can't fit on the screen, it will be CLIPPED to the edges of the screen. For example:
```
     BOX 150,90,100,100,1
```
will put a 60X60 BOX in the upper right corner.

BUMP STRING,NUMBER
Esoteric Swap Command

increments the ASCII code of the last non-null character in a string by a specified numeric value.
Example:
```
     TEST="ABCDE"
     BUMP(TEST,2)
     PRINT TEST
```
prints out the string "ABCDG"
Note: BUMP does not cause the re-assignment of the STRING so:
```
     TEST="ABCDE"
     BARB=TEST
     BUMP(TEST,2)
     PRINT TEST,BARB
```
will print "ABCDG" twice. First, CONCATENATE BARB with a STRING to avoid this, if necessary: BARB=BARB&[]

BYTE
Buzzword

a BYTE is the amount of MEMORY needed to hold a single character. Computers generally store one BYTE at each MEMORY location. ZGRASS lists the amount of MEMORY a NAMEd thing takes up in BYTEs when you use the USEMAP command.

BYTE ARRAY
Buzzword

if the values you want to store are limited to the range of 0-255 and you are very short on memory, you can use the STRING command as a way to store single byte values instead of characters. The STRING command can then be thought of as accessing the string as a BYTE ARRAY. If you place a zero in your BYTE ARRAY and attempt to store the string on the disk, it will only store as far as the zero. Be careful also not to print the string because some characters turn the terminal off, clear the screen, etc. This way of saving memory is for expert users only.

CALL
Buzzword

is what you do to cause the execution of a MACRO, COMMAND, or FUNCTION; that is, specifying its NAME and ARGUMENTs. ZGRASS has no CALL COMMAND since specifying a NAME plus ARGUMENTS is enough to call the MACRO, FUNCTION or COMMAND.

CENTERING (of Graphics Primitives)
Idiosyncrasy

The centering of even-numbered dimensions is biased to the upper right. The lower left hand corner of the upper right quadrant is the center pixel. For example, given a BOX centered at 0,0 which is 6 PIXELs wide on the X-axis, and 4 PIXELs high on the Y-axis, the left X would be -3, bottom Y -2, right X 2, top Y 1.

```
         <-----6 PIXELS wide----->
                     Y axis
                       ^
         |---|---|---|---|---|---|
         |   |   |   | 1 |   |   |
         |---|---|---|---|---|---|
         | -3| -2| -1| 0 | 1 | 2 |
   -X  - |---|---|---|---|---|---|-> X axis
         |   |   |   | -1|   |   |
         |---|---|---|---|---|---|
         |   |   |   | -2|   |   |
         |---|---|---|---|---|---|
                       |
                      -Y
```

You can see that the center PIXEL in this 6X4 box is located in the lower left hand corner of the upper right hand quadrant.

CIRCLE XCENTER,YCENTER,DIAMETER,0\1,COLORMODE
Command

draws a circle (specify 0 for border only, 1 for filled circle) centered at XCENTER, YCENTER with the specified DIAMETER using the COLORMODE indicated.

CLEAR
Command

clears the TV screen (not the computer's memory). See FRAME BUFFER. RESTART clears the computer's memory, not the TV screen.

CLEAR.CRT
Command

clears the CRT screen.

CLEAR.WIND
Command

clears the graphics WINDOW.

CLIPPING
Buzzword

refers to the action of displaying only a portion of a LINE, SNAP, or BOX if part of it exceeds the screen or window boundaries. Example:

```
     BOX 120,80,100,100,3
```

will put a BOX in the upper right corner and throw away parts exceeding 159 in the X direction and 101 in the Y.

CMPARA(A1,B1)
Esoteric Swap Function

returns values depending on the comparison of two ARRAYs (usually used to compare SNAPs). The values returned are:

```
     0 if all the BITs of A1<=A2
     1 if all the BITs of A1==A2
    -1 if all the BITs of A1>=A2
    -2 otherwise
```

Example:

```
     BOX 0,0,20,20,1
     SNAP FIRST,0,0,20,20
     BOX 0,0,20,20,3
     SNAP SECOND,0,0,20,20
     PRINT CMPARA(FIRST,SECOND)
```

prints 0 because all of FIRST is 01 PIXELs which are all less than or equal to all of SECOND's 11 PIXELs. If the second box were drawn in COLOR MODE 2, the result would be -2.

COLOR
Idiosyncrasy

The 256 COLORs available in ZGRASS form an abbreviated spectrum. You can get four COLORs on the screen at any one point. The default COLOR VALUES are white (7), red (91), green (165), and blue (8). By using the DEVICE VARIABLES $L0 through $L3 you can change the currently available palette of 4 COLORs. The VALUE of $L0 is 7 (white). The VALUE of $L1 is 90 (red), etc. See COLOR MAP for how ZGRASS keeps track of these four COLORs.

COLOR MAP
Idiosyncrasy

The COLOR MAP is the way ZGRASS translates COLORs 0-3 into the 256 available COLOR VALUES. The hardware looks at the values of $L0-$L3 before it writes a PIXEL to the screen. If it is writing a 0 it uses the COLOR VALUE (0-255) stored in $L0. If it is writing a 1, it uses the COLOR VALUE stored in $L1, and so on. To change the COLOR MAP so 1 refers to yellow instead of red, assign:

```
     $L1=127
```

There are actually two COLOR MAPs, the $L's and the $R's. You get to the $R's by setting $HB. See DEVICE VARIABLES.

Example:

```
     CBARS=[CLEAR;A=-149;C=0;$HB=21
     $R0=0;$R1=82;$R2=43;$R3=249
     $L0=7;$L1=213;$L2=126;$L3=164
     IF  A<115,BOX  A=A+45,0,46,202,C=(C+1)\3+1;SK
     0]
```

This will make a set of colorbars for tuning your TV.

COLOR MODES
Idiosyncrasy

The possible values for COLOR MODES are 0-21. You may need to study your truth tables for PLOP, XOR, OR, AND, PRIORITY, and REVERSE-PRIORITY logical operations to really understand what's going on. Look under PLOP, XOR, etc. for their respective truth table.

COLOR MODE	MEANING:
0	PLOP with COLOR 00 (white)
1	PLOP with COLOR 01 (red)
2	PLOP with COLOR 10 (green)
3	PLOP with COLOR 11 (blue)
4	XOR screen with COLOR 0 (no change)
5	XOR screen with COLOR 1
6	XOR screen with COLOR 2
7	XOR screen with COLOR 3
8	OR with 00 (no change)
9	OR with 01 (if white or red, turn red if green or blue, turn blue)
10	OR with 10 (if white or green, turn green, if red or blue, turn blue)
11	OR with 11 (turn blue)
12	AND with 00 (turn white)
13	AND with 01 (if white or green turn white, if red or blue, turn red)
14	AND with 10 (if white or red, turn white, if green or blue, turn green)
15	AND with 11 (no change)
16	PRIORITY WRITE 01 (if white or red turn red, if green stay green, if blue stay blue)
17	PRIORITY WRITE 10 (if white, red or green turn green, if blue stay blue)
18	REVERSE-PRIORITY 01 (red, green, and blue turn red, and white stays white)
19	REVERSE-PRIORITY 10 (green and blue, turn green, red stays red, and white stays white)
20	Increment COLOR (if white turn red, if red turn green, if green turn blue, if blue turn white)
21	Decrement COLOR (if white turn blue, if red turn white, if green turn red, if blue turn green)

COMMAND
Buzzword

there are three types of COMMANDs: system COMMANDs, SWAP COMMANDs, and ones you define yourself, called MACROs. System COMMANDs are built-in and are listed by the HELP COMMAND. Swap COMMANDs function like System COMMANDs except they must first be gotten from tape or disk.

COMMENT
Buzzword

it is helpful to have COMMENTs in your MACROs to tell how they work. In ZGRASS, a line which starts with a '.' is taken as a COMMENT. You can also have COMMENTS on lines where there are COMMANDs by using a ';' and then a '.'. Examples:

```
     .THIS LINE IS TAKEN AS A COMMENT
     LINE 6,-70,1;.THIS LINE HAS A COMMAND TOO
```

COMPILE NAME,NEWNAME
Command

takes a MACRO called NAME, and creates a compiled MACRO called NEWNAME. Compiled MACROs are larger but run faster. They cannot be stored on disk or tape.
Note: several COMMANDs; EDIT, CORE, HELP, LOOPMAX, ONERROR, and USEMAP if included in a MACRO will cause your MACRO not to be able to be COMPILEd and you will get ERROR #59.
Example:

```
     TALL=[ARRAY LONGNAME,200
     INDEX=0
     LONGNAME(INDEX)=SQRT(INDEX)
     INDEX=INDEX+1
     IF INDEX<200,SKIP -2]
```

TALL will take approximately 15.5 seconds to run.

```
     COMPILE TALL,FASTER
```

FASTER will take approximately 3.5 seconds to run. The compiler figures out NAME references, SKIPs, GOTOs, and figures out OPERATORS and parentheses. You will see better improvements in compiling when you have long programs with lots of arithmetic and/or long NAMES, or lots of LOCAL VARIABLES. COMPILing BOX COMMANDS, on the other hand, gives a less dramatic speed increase because the time is spent mostly drawing to the screen, not figuring out the ARGUMENTS. You can't store COMPILEd MACROs on disk or tape.

COMPRESS FONTARRAY,NAME
Swap Command

compresses the snaps in a FONTARRAY and creates a new FONTARRAY called NAME. COMPRESS allows single-color characters to be displayed with text in any color and also halves the space required. Any character in the font with more than one color will not be COMPRESSed.

CONCATENATION
Buzzword

is joining STRINGs together with the '&' operator. Examples:

```
     PRINT "A"&"B"&"C"  prints ABC
     PRINT "A"&10       prints A10
     N="MOON"
     S="SHINE"
     PRINT N&S          prints MOONSHINE
```

CONSTANT
Buzzword

Examples:

```
     PRINT 'THIS is a constant or literal STRING'
     PRINT 33.75
     PRINT 1.23E17
```

Constants, unlike VARIABLES, never change. You can have both NUMBERS and STRINGS as constants.

CONTROL CHARACTERS
Buzzword

are single character requests you type on the keyboard by holding the key marked CTRL down (as you would the shift key) and at the same time pushing any key from A to Z. See the CONTROL COMMAND for the listing of the CONTROL CHARACTERS.

CONTROL(NUMBER)
Esoteric Function

returns the current value of the CONTROL CHARACTER identified by NUMBER. For instance, to see if CTRL+Y is on:

```
     PRINT CONTROL(25)
```

if CONTROL+Y is on, the answer will be 1, and if it is off, 0.

CONTROL CHAR.	NUM.	TYPE	DESCRIPTION
A	1	S	;Editor delete line; also allows .B MACROs to be interleaved with regular MACROs
B	2	*	;Resets COLORs to WRGB and $TV,$MW, $MR, and $ML to 0
C	3	S	;Stops currently running MACRO(s) and clears CONTROL characters
D	4	T	;Single step in MACROs on/off with CTRL+X gives single step and listing and in the Editor moves lines
E	5	S	;Editor exit and update and stops PATTERN command
F	6	S	;Editor copy lines
G	7	*	;Turn off all CTRL characters (set to 0)
H	8	S	;Editor Cursor Control
I	9	S	;Repeats the last command line once same as TAB
J	10	S	;Editor Cursor Control
K	11	S	;Editor Cursor Control
L	12	L	;Editor Cursor Control
M	13	S	;Carriage return
N	14	T	;Beep on/off for CR
O	15	T	;Supress/allow printing on CRT
P	16	T	;Echo CRT on printer, if any
Q	17	T	;Start/Halt printing on CRT
R	18	S	;Editor delete character or continuously repeat last command line if not in EDIT
S	19	S	;Editor set move pointers
T	20	S	;Editor delete move pointers
U	21	*	;Line erase (outside the editor)
V	22	T	;Allows auxillary RS232 input in parallel with keyboard RS232 input
W	23	T	;Twenty line mode on/off waits for return key to print 20 more lines
X	24	T	;List on/off as MACRO EXECUTES
Y	25	T	;A "!" is put at the end of every line which has a CR (in editor, also use CTRL+T)
Z	26	S	;Stop MACRO in progress and accept lines till return key alone typed
[	27	*	;Not used
\	28	*	;Switch upper/lower case
]	29	*	;Cancel/enable break button on terminal (also allows terminal to be unplugged without causing break to happen)
^	30	*	;Not used
-	31	*	;Not used

TYPES:

T is a toggle switch which you can turn on (1) or off (0) only by keyboard action.

S can be set (1) by keyboard action. You can set these to any number including zero with the CONTROL NUMBER1,NUMBER2 command below.

* means this CONTROL CHARACTER is not accessible through the CONTROL COMMAND.

Note: the CONTROL Characters which are used by the EDITor can be reset by using the TERMINAL Command.

CONTROL NUMBER1,NUMBER2
Esoteric Command

Like CONTROL (NUMBER) but it writes NUMBER2 in the CONTROL CHARACTER indicated by NUMBER1. Use to set CONTROL CHARACTERs in a MACRO. (Setting CONTROL CHARACTERS B,G,U to 1 doesn't do anything, however.) CONTROL CHARACTERS used only in EDIT (F,H,J,K,S,T) may be used by you for your own purposes outside of EDIT. Characters A,D,N,O,P,Q,V,W,X,Y are set to one by an odd number of user CTRL key presses and cleared to zero by even presses. The rest are set by one or more user presses and cleared by system actions.

Examples:

CONTROL 3,1;.Will cause a CTRL+C to happen programatically

CONTROL 16,1;.Will cause whatever comes out on the CRT to be printed on the printer, if any.

CONTROL 15,1;.Will cause whatever you type on the computer terminal to be not printed to the CRT until CONTROL 15,0 is EXECUTEd.

CONTROL 24,1;.Will cause listing of lines as they EXECUTE until CONTROL 24,0 is EXECUTEd.

COORDINATES
Idiosyncrasy

are the values across the X (horizontal) axis and up and down the Y (vertical) axis. The COORDINATES range from -32768 to 32767. With the default WINDOW in effect the visible X-COORDINATES range from -160 to 159, and the Y-COORDINATES range from -100 to 100. See WINDOW.

CORE
Command

tells you how much memory you have in BYTES in how many fragments. The first number is the hexadecimal ADDRESS which you should ignore. A BYTE will hold one character so if you have a

MACRO on tape that is 500 BYTES long (USEMAP will give its length once it's in memory), CORE has to show a fragment with a least 500 BYTES for you to GETTAPE the MACRO without getting ERROR #27 (not enough memory space).

CORE()
Function

returns the size of the largest block of MEMORY left and also prints the CORE map. (You can supress the printing with CONTROL 15,1.)
Example:

```
     A=CORE()
```

will print a list of the available memory

```
     PRINT A
```

will print 4064 if this is done right after RESTART.

COSINE(NUMBER)
Function

returns the cosine of NUMBER.

CURSOR
Idiosyncrasy

is the little box over a character in EDIT. The next thing you do in EDIT (insert, delete, etc.) will be done at the CURSOR position.

DEBUG
Esoteric Swap Command

Refer to the Swap Module creation documentation, a separate package.

DELETE NAME0,NAME1,NAME2,...NAMEn
Command

deletes the NAME/s (VARIABLE, ARRAY, STRING) from memory and reclaims the memory for further use. Certain things cannot be deleted (DEVICE VARIABLES, the VARIABLES A-Z, system COMMANDS, and FUNCTIONS) so an appropriate ERROR message accompanies illegal deletion requests. Never DELETE anything that is referenced in a COMPILEd MACRO unless you have already DELETEd that COMPILEd MACRO or intend not to use it again.
Example:

```
     GONE="WITH THE WIND"
```

USEMAP will tell you that there is a STRING called GONE in MEMORY.

```
     DELETE GONE
```

USEMAP will now show you that GONE is gone.

DEVICE VARIABLES
Idiosyncrasy

are special VARIABLES starting with a '$' that access system features. You use them just like other VARIABLES. Most DEVICE VARIABLES (except COLOR VARIABLES) are set to 0 when the system is turned on or reset.

VARIABLE:	Description:	Range:

Screen COLOR VARIABLES:

$L0	COLOR 0 left	0-255
$L1	COLOR 1 left	0-255
$L2	COLOR 2 left	0-255
$L3	COLOR 3 left	0-255

(left means left half of screen set by $HB)

$R0	COLOR 0 right	0-255
$R1	COLOR 1 right	0-255
$R2	COLOR 2 right	0-255
$R3	COLOR 3 right	0-255

('right' means right half of screen, set by $HB)

$HB	Horizontal Color Boundary	0-44
$BC	Border Color	0-3

```
0     set Border to $L0
1     set Border to $L1
2     set Border to $L2
3     set Border to $L3
```

JOYSTICK control VARIABLES:

$X1-$X4	X of JOYSTICKs 1-4	-1,0,1
$Y1-$Y4	Y of JOYSTICKs 1-4	-1,0,1
$K1-$K4	knob value of JOYSTICKs 1-4	-128 to 127
$T1-$T4	trigger value of JOYSTICKs 1-4	0 or 1

DISK information:

$DS	has disk number set by DSETUP	-3 to 7
$DV	disk verify on write: 0 = on	

System Timers:

NOTE: system timers are suspended by tape I/O and floppy disk I/O operations

$Z0-$Z9 decremented by 1 every 1/60 second until 0

```
     $TK system time in 1/60's seconds
         up to 60
     $SC in seconds up to 60
     $MN in minutes up to 60
     $HR in hours up to 24
     $DA in days up to 32767
     $ST in seconds up to 32767
Example:
     CLOCK=[PR $HR,':',$MN,':',$SC,':',$TK;SK 0]
     CLOCK.B
```

Terminal Control:

```
     $RS if non-zero, allows the 8th bit through
         from the RS232 ports; if zero, the 7th
         bit is always 0
```

256K Screen Memory Controls (for an example, see SCREEN):

```
     $TV sets the screen the TV uses to 0-15
     $MW sets the screen the computer writes to
         and if $ML=0, reads from 0-15
     $MR sets the screen the computer reads from
         if $ML=1
     $ML if 1, allows read and write to
         be from different screens; if 0,
         forces $MW to be used for both read
         and write
```

Math Control:

```
     $RD if 0, use degrees;
         if 1, use radians
```

Graphic Control:

```
     $DX  is the X offset for all graphic commands
     $DY  is the Y offset for all graphic commands
```

Memory Allocation:

```
     $BF if non-zero, attempt to do a best-fit
         allocation, which takes longer but
         reduces memory fragmentation
```

Number Formatting:

```
     $KZ if 1, allows trailing zeroes after
         decimal point
```

DISK
Idiosyncrasy

A DISK (also called FLOPPY DISK or WINCHESTER DISK) is the best place to store information. Since it is a much more complex device than an audio tape recorder, several commands are necessary to manage it. You must occasionally do housekeeping on your disk to keep it from filling up. (Esoteric note: the disk software uses 512-byte sectors.) The umpteen disk commands are grouped as follows:

Resident Commands:

to choose a disk, use DSETUP
to reset the disk, use DSETUP.RESET
to get a disk file, use DGET
to put a disk file, use DPUT
to put out a screen dump, use DPUT.TV
to delete a disk file, use DDELETE
to initialize a disk, use DINIT
to initialize a Winchester, use DINIT.WINCH
to tell what is on the disk, use DUSEMAP
to create a submap name, use DCREATE
to get into a submap, use DSETUP
to load a whole floppy (or the Winchester diskmap), use DLOAD
to unload (write) a whole floppy, use DLOAD.ZAP
to clear the floppy in memory without writing it, use DLOAD.CLEAR
to lookup a file, use DLOOK
to get a file you DLOOK'd up, use DYANK
to check the disk, use DFETCH (with no arguments)
to load a specific sector, use DFETCH
to write a specific sector, use DFETCH.ZAP
to delete all the BAKS, use DBAKS

Swap Commands:

to check a disk, use DCHECK
to copy a disk to another disk, use DCOPY
to rename a file name, use DRENAME
to delete a whole submap, use DDSMAP
to match file names, use DMATCH
to read/write individual bytes, use DZAP
to format a disk, use DFORMAT

DBAKS
Command

deletes all BAK files on the disk. DPUT automatically creates BAK files for you and these take up space. You can individually delete them with DDELETE.BAK or delete them all at once with DBAKS.

DCHECK DRIVENUMBER
Swap Command

reclaims any 'lost' sectors on the disk specified by the DRIVENUMBER. Sectors can get lost if you push the red RST button during a DPUT or get an error during a DPUT. DCHECK does not verify the integrity of the data on the disk. See DCOPY and DFETCH.

DCOPY SOURCEDISK,NEWDISK
Swap Command

copies the SOURCEDISK onto the NEWDISK clobbering all previous information on the NEWDISK. The NEWDISK does not have to be DINIT'd but it must have been DFORMAT'd. DCOPY also verifies the information on the SOURCEDISK and NEWDISK (if $DV=0) as it is copying. You should backup disks with DCOPY fairly often (every couple hours of working) since floppies are not super-reliable. You can see the disk sectors numbers in the display lights.
Example:

```
    DCOPY 0,1
```

copies what is on disk 0 to disk 1.

Copies can be made from DLOAD'd disks.

DCREATE SUBMAPNAME,[MESSAGE]
Command

creates a submapname on the disk. Submaps allow you to have several independent groupings of disk files on the same disk, thus allowing the same name to be in different submaps. Once you DCREATE a submapname, you will see it in DUSEMAP. You then use DSETUP with a disk number and a submapname to make all disk commands reference only files within that submap (the exception is that if the command cannot find a name it looks in the normal (unnamed) map so it is easy to get swaps and common macros). If you DSETUP for a particular submapname without having DCREATE'd it, you may not be able to find DPUT'd files unless you are very good at remembering since the submapname will not show up in the normal disk map. DCREATE automatically puts you in the submapname you specified.
Examples:

```
    DSETUP 1            ;setup for disk 1
    DCREATE JOB77       ;.create submap JOB77
```

DINIT MAXNAMECOUNT
DINIT.WINCH
Command

reserves space on a formatted disk, starting at sector 0 (the outermost sector on the disk) for the directory (we call it the DISKMAP) of the contents of the disk. This command initializes the disk, erasing all previous information in the currently DSETUP'd drive, reserving space for the number of entries specified by MAXNAMECOUNT. Kinds of entries are: MACROs, ARRAYs, SNAPs, monitor SCREEN dumps, STRINGs, etc.

It is important to plan the intialization of your disk. If you do not plan for enough entries, you may run out of space for names in the directory before you run out of actual space on the disk, in which case you'll get the "DISKMAP FULL" error message. Likewise, if you allocate too much space for names in the directory, you could be wasting valuable disk space.

To calculate how much directory space should be reserved, use a ratio of 4 entries per sector of directory space. Each entry requires 128 bytes to store the entry name, type, size, comments, and pointer to the entry's actual location on the disk. In addition to the 4:1 ratio, allow several sectors for overhead.

For instance, a SCREEN dump (saving on disk all information currently displayed on the monitor SCREEN) uses 16K bytes (or 32 sectors) of a disk. Based on 32 sectors per dump, you can only store 11 screen dumps on one side of a disk. To optimize usable space on the disk, initialize the directory for 19 entries, so that 8 sectors are used for the directory information and more than 370 sectors remain for storage of screen dumps.

Suppose you will be storing a lot of little strings and macros. In that case, you'd want to have a large directory of roughly 300 entries, using almost 78 sectors for the directory, leaving about 300 sectors free storage space on the disk.

If you are in doubt about the kind of entries you'll be storing on a disk, a suggested value for MAXNAMECOUNT is 200, which should allow adequate directory space and storage space for general purposes.

It is not necessary to initialize a disk (using DINIT) if you use DCOPY, since the directory information will be copied with the rest of the disk.

Examples:

```
     DINIT 300
     DUSEMAP          ;prints 307 free sectors
     DINIT 100
     DUSEMAP          ;prints 357 free sectors
     DINIT 20
     DUSEMAP          ;prints 377 free sectors
```

DINIT.WINCH intializes the current Winchester drive (-1, -2, or -3).

See Zgrass Lesson 5 for more information on disk and tape storage.

DLOAD
DLOAD.ZAP
DLOAD.CLEAR
DLOAT.SET
Command

DLOAD takes the current disk and if it is a floppy, loads in into SCREEN MEMORY, screens 4-15. Then, all references to that DISK will be done from MEMORY. When you are through, be sure to DLOAD.ZAP if you don't want to lose all changes to the disk. If it's the Winchester, DLOAD loads the DISK MAP into screens 4-15. All Winchester writes go to both places, so DLOAD.ZAP for the Winchester is ignored. DLOAD.CLEAR disables the DISK in MEMORY without writing it out. DLOAD.ZAP copies what is loaded into SCREEN MEMORY onto the disk in the currently DSETUP'd drive. DLOAD and DLOAD.ZAP are a good way of making lots of copies of the same floppy disk--just DLOAD the master and then switch disks and do a DLOAD.ZAP for each copy. DLOAD.SET forces screens 4-15 to think they were DLOAD'd from drive 0, and is used only if you or the system DLOAD.CLEAR by accident.

DLOOK FILENAME,ARRAYNAME
Esoteric Command

looks up the information necessary for DYANK and stores it in the ARRAY specified. The ARRAY should already be created by the ARRAY command and should have 18 elements. Do not change the physical disk or drive number before you do the

DYANK (you can change submaps, though). The reason for DLOOK and DYANK is to allow you to remove the overhead of going through the disk map in time-critical applications by doing the lookup ahead of time.
Example:

```
     ARRAY BOOPSIE,18
     DLOOK SNIFFLEPIX,BOOPSIE
       sometime later:
     DYANK SNIFFLEPIX,BOOPSIE
```

DMATCH(STRING)
DMATCH(STRING,TYPE)
Esoteric Swap Function

uses same syntax as the MATCH function for strings to match names in the disk map (or current submap). DMATCH returns the matched name as a string or a null string if no match is found. Each time you do a DMATCH, it will resume looking in the directory where it left off. DSETUP resets the matching to the first name in the disk map. The optional type allows you to match only certain file types (see WHATSIS for types; screen dumps are type 38).
Examples:

```
     DEEZ=[DS 0
     ABC=DMATCH([D*])
     IF ABC#[],PR ABC;SKIP -1]
```

The above will print all disk file names on disk 0 starting with D.

```
     PRPIX=[DS 0
     ABC=DMATCH([[A-Z]*],38)
     IF ABC#[],PR ABC;SKIP -1]
```

The above will print all disk files on disk 0 that are screen dumps.

DPUT NAME,[MESSAGE]
DPUT NAME
DPUT.TV NAME,[MESSAGE]
DPUT.TV NAME
Command

puts NAME out on the disk with the message indicated. Messages can be any string and are used for documentation only. DUSEMAP shows them. If there is already a file with the same name and type, the message can be omitted and the old message will be copied over. (If the types don't match, you will get error #81 indicating it.) DPUT automatically creates BAK files (which you can get at with DGET.BAK and delete with DDELETE.BAK or DBAKS). If a BAK file is present already, it is automatically deleted when you DPUT again. DPUT.TV must be used to put out a screen dump.

DRENAME OLDNAME,NEWNAME,[NEW MESSAGE]
Swap Command

renames the oldname to the newname on the disk with the new message.
Example:

```
DRENAME MANKIND,PERSONKIND,[A NEW MESSAGE]
```

DSETUP DISKNUMBER
DSETUP DISKNUMBER,SUBMAPNAME
Command

DSETUP does several things. First, it sets the disk to be the "current" one; that is, the one used by most disk commands. 0 and 1 are the upper sides of the disks in the drives marked 0 and 1 respectively. 4 and 5 are the lower sides of 0 and 1 respectively. If you are lucky enough to have two double disk drives, the numbers of the second ones are 2 and 3 (upper) and 6 and 7 (lower). If you are even luckier and have a Winchester disk, it is configured as -1, -2, -3, each holding about 2 megabytes. (Esoteric note: DSETUP also caused DMATCH to start looking from the first name in the disk map.)
Second, if the SUBMAPNAME is supplied, the disk commands are all directed to reference only file names within the indicated submap. (DGET will look at the normal disk map after a failure in the current submap, however). You cannot get a file from another submap nor put a file out into another submap without changing the submapname with DSETUP.

DUSEMAP
DUSEMAP FILENAME
Command

lists all the names on the disk (under the current submap, if any.) If a FILENAME is specified, just that name's map information is printed.

DYANK NAME,ARRAYNAME
Esoteric Command

uses the ARRAYNAME set up by DLOOK to get the FILE from the disk and call it NAME without the overhead of going through the disk map. Do not change disks between DLOOK and DYANK. See DLOOK for an example.

DZAP SECTORNUMBER,BYTENUMBER
DZAP SECTORNUMBER,BYTENUMBER,VALUE
Esoteric Swap Command/Function

like ZAP but works on disk information. The disk is formatted into 384 sectors of 512 bytes each (Micropolis floppy only). Sector 0 holds a byte map indicating used sectors and sectors 1-n have the disk map information. Sectors n+1 through 383 have data. This is a dangerous command since you can permanently confuse a disk if you DZAP it. There is more documentation on the disk formats in the Swap Module Documentation, a separate package.
Example:

```
     PRBYTEMAP=[A=0;K=0
     IF K#255,PR K=DZAP(0,A=A+1);SKIP 0]
```

the above will print out the bytes in sector 0 of the disk until a -1 byte is seen. The zero bytes represent free sectors and the one bytes mark used sectors.

DISPLAY NAME,XCENTER,YCENTER,DISPLAYMODE,ROTATION
Command

takes a SNAPped NAME and writes it at the center indicated using DISPLAYMODE. If not specified, ROTATION is assumed to be 0. Rotation 1 means rotate 90 degrees; rotation 2 means rotate 180 degrees; rotation 3 means rotate 270 degrees. Refer to DISPLAY MODES for the details on the 74 different writing modes. (A SNAPped NAME is actually an ARRAY specially created by the SNAP COMMAND and is essentially an exact copy of an area of screen memory.) You can use DISPLAY for animation. Say there is an apple drawn at the center of the screen which fits inside a rectangle of 48X48 PIXELs. The following code will draw it, SNAP it. and move it on a JOYSTICK.

```
     CLEAR
     CIRCLE 0,0,40,1,1
     BOX 0,17,4,8,3
     SNAP APPLE,0,0,48,48
     .LEAVE EXTRA WHITE AROUND FOR ERASING
     MOVE=[DISPLAY APPLE,X=X+$X1,Y=Y+$Y1,0
     SK -1]
     MOVE
```

Note: The largest square area you can SNAP in one piece is 125X125 PIXELs (or about 15625 PIXELs or 1/4 of the screen.)

DISPLAY.SCREEN 0-15,XCENTER,YCENTER,DISPLAYMODE,ROTATION
Command

sames as DISPLAY but uses contents of the specified SCREEN (0-15) to DISPLAY on the current SCREEN instead of a SNAP.

DISPLAY MODES
Idiosyncrasy

the possible values for DISPLAY MODE are between 0 and 144. You may need to study your truth tables for PLOP, XOR, OR, AND, and PRIORITY logical operations to really understand what's going on. There are 8 logic modes, mentioned above, which we combine with 15 filters (0,10,20...140) to come up with 120 DISPLAY MODES:
0,1,2,3,4,5,6,7,10,11,12,13,14,15,16,17,...,
140,141,142,143,144,145,146,147

Logic MODES	Meaning:
0	PLOP the SNAPped NAME on the screen
1	XOR the SNAPped NAME with the screen
2	OR the SNAPped NAME with the screen
3	AND the SNAPped NAME with the screen
4	PRIORITY WRITE red(01)covers white(00) green(10)covers white and red blue(11)covers white, red and green
5	PLOP SNAP only on the screen colors mentioned in the filter
6	XOR SNAP only with the screen colors mentioned in the filter
7	OR SNAP only with the screen colors mentioned in the filter

FILTERS:	DISPLAY only this COLOR in SNAPped NAME:
0	everything
10	white (00)
20	red (01)
30	green (10)
40	blue (11)
50	red and blue
60	green and blue
70	red and green
80	white and blue
90	white and red
100	white and green
110	white, red and green
120	white, red and blue
130	white, green and blue
140	red, green and blue

The equation for figuring out a specific DISPLAY MODE is:

DISPLAYMODE=LOGICMODE+FILTER

Note: Modes 8-9, 18-19, 28-29, etc do not exist.

EDIT NAME
Command

edits the MACRO specified.

EDIT CONTROLS:

Left Arrow	;Move cursor left
Right Arrow	;Move cursor right
Up Arrow	;Move CURSOR up
Down Arrow	;Move CURSOR down
TAB	;Delete line
NEXTLINE	;insert a line
ESC	;Delete a character
HOME	;insert a character
CTRL+S	;set copy/move pointers
CTRL+T	;clear copy/move pointers
CTRL+D	;move
CTRL+F	;copy
CTRL+E	;Update and exit from editor
BREAK	;Exit editor without updating

There are only 80 characters visible on a line. More are permissable, but you may not see them in edit mode unless you split the line into two separate lines by inserting a carriage return.

ELLIPSE ANGLE,XCEN,YCEN,XSIZE,YSIZE,TYPE,COLORMODE
Swap Command

draws an ellipse centered at XCEN,YCEN with XSIZE as the width and YSIZE as the height in the specified COLOR MODE. Set TYPE to 1 to get a solid ellipse, and 0 to get just the outline. ANGLE is the tilt off the X-axis in DEGREES, unless you tell the system to use RADIANS by setting $RD to 1.

Examples:

```
    ELLIPSE 0,-50,50,80,40,1,1
    ELLIPSE 0,50,50,80,40,0,1
```

The first draws a solid ellipse, the second just its outline.

```
    ELLIPSE 45,50,-50,80,40,1,2
```

draws a solid ellipse tilted off the X-axis at 45 degrees.

ERROR NUMBER
Idiosyncrasy

an ERROR NUMBER is printed on the CRT if something has gone wrong in your MACRO, or if you try to do something like dividing by zero which is not allowed. Refer to the following list for a clue to what went wrong:

ERROR #:	Explanation:
2	;System error - RESTART system
3	;System error - RESTART system
4	;System error - RESTART system
20	;Operand (VARIABLE, Number, etc.) expected but not seen
21	;Something other than a legal NAME on the left side of an ASSIGNMENT
22	;Can't do this conversion, only strings and numbers may be converted to each other
23	;Arithmetic overflow (number too big to convert to INTEGER or exceeds FLOATING POINT range)
24	;You tried to divide by zero
27	;Out of memory space, DELETE something
28	;More than 128 characters typed before a NEXTLINE
30	;Too many ARGUMENTs for this COMMAND
31	;Funny SYNTAX
32	;Extra stuff on line
33	;Illegal character after COMMAND name
34	;This NAME should be a MACRO but it isn't
35	;Can't find this NAME
36	;More RETURNs than MACRO calls
37	;Can't find this LABEL
38	;This NAME can't be DELETEd for system integrity reasons
39	;Not enough ARGUMENTs for this COMMAND
40	;No such COLORMODE or DISPLAYMODE
41	;Illegal character in NAME (must be a followed by letters or digits)
42	;Unbalanced parentheses
43	;Number expected but you forgot it!
45	;This NAME already exists
46	;Illegal special VARIABLE NAME
47	;ARRAY reference out of bounds
48	;More than 4 dimensions specified in ARRAY COMMAND
50	;No such SWITCH with this COMMAND
51	;Fraction too small (arithmetic underflow)
52	;Invalid ARGUMENT value (example: SQRT(-1)
53	;EDIT only works on MACROs (STRINGS)
54	;Only A-Z allowed in CONTROL COMMAND
55	;Too many digits after decimal point (12 maximum)
56	;Negative value not allowed here

57	;Null STRING not allowed here
58	;Negative ARGUMENT not allowed here
59	;Can't COMPILE this COMMAND
60	;Duplicate LABEL
61	;INTEGERS only for COMPILEd SKIPs
62	;Too many lines for COMPILER
63	;Illegal LABEL SYNTAX
64	;ONERROR in LOOP
65	;LOOPMAX exceeded
66	;System STRING error
67	;Too many ARGUMENTS
69	;Must be in MACRO for this COMMAND
70	;Can't CMPARA ARRAYs of different sizes
71	;Transmit error over auxillary RS232 Port
72	;Disk Byte map messed up
73	;No such file
74	;Feature not implemented
75	;Disk error
76	;Too many SKIPs, GOTOs, IFs to COMPILE (max is 99)
77	;Disk full
78	;Disk track seek error
79	;Disk read error
80	;Disk write error
81	;Can't back up one file type over another type which have the same name
82	;Disk not loaded
83	;Use DDSMAP to delete an entire submap ~~PATTERN.FILL~~
84	;A disk already DLOAD'D can't DLOAD unless DLOAD.CLEAR is done first
85	;Must be submap for DDSMAP
86	;Too many turns in PATTERN or PATTERN.FILL
87	;FIXED POINT stack underflow
88	;FIXED POINT stack overflow
89	;FLOATING POINT stack underflow
90	;FLOATING POINT stack overflow
91	;The first argument of the STRIPE command can only be 0-15
92	;Can't DLOAD.ZAP unless a disk is DLOAD'd

EXCLUSIVE OR
Buzzword

See XOR.

EXECUTE
Buzzword

is computer talk for doing a COMMAND, MACRO, or ASSIGNMENT. It has nothing to do with killing anything. (See CALL)

EXP(N)
Function

returns the value of e (2.71828) raised to the power N.
Examples:

```
     PR EXP(2)
prints 7.38905
     PR EXP(1)*EXP(1)
prints 7.3891
```

EXPRESSION
Idiosyncrasy

is:
1. a CONSTANT (12, 'foo', for example)
2. a NAME (TOM, $X1, POOHBAH, for example)
3. a combination of OPERATORs and CONSTANTs or VARIABLEs (+6, ?B, -ABC, FF+1, 'tom'&'sam', Beer*4, for example).
4. a FUNCTION or MACRO call (SIN(a)+COS(b)), MAX(k,F+E,Beer), etc).

Expressions can be simple or complex. Actually, anything syntactically correct in ZGRASS is an EXPRESSION. Arithmetic EXPRESSIONs result in numbers being generated and are a mix of arithmetic OPERATORs (+,-,/,\,*,?,&&,¦¦), parentheses, numbers, and VARIABLEs. STRING EXPRESSIONs are a mix of STRING OPERATORs (",',[,],{,},&,@,?) and STRING VARIABLEs. FUNCTIONS which return NUMBERS or STRINGs can also be parts of EXPRESSIONs. ZGRASS attempts to convert NUMBERS to STRINGS and STRINGS to NUMBERS when it can, so a STRING like ABC='1234' can legally be used in PRINT ABC+ABC or PRINT ABC&ABC, and so on. COMMANDS are EXPRESSIONs too. Most return the value 1 but some, like ANYARGS, SINE, RETURN, can return other values as well. The basic idea is to combine small EXPRESSIONs to make larger ones. Examples:

```
     A
     A+1
     A+B*C
     (A+B)*c
     SIN(ABC)+COS(ABC)
     C=A+BOX(10,10,20,30,5)
     etc.
```

.F
Esoteric Switch

is a way of telling a MACRO to EXECUTE every 1/60 second. Such MACROs should be short since they take precedence over regular and .B MACROs.
Example:

```
TIMESUP=[timer=timer+1
IF timer==180,PRINT '3 SECS ARE UP';timer=0]
TIMESUP.F
```

Unfortunately, unless COMPILEd, this takes about 6.2 seconds to do. See TIMEOUT.

FILES
Buzzword

is what things stored on disk or tape are called. FILENAMES are the NAMEs of FILES, of course. Never use abbreviations for FILENAMES!

FLOATING POINT
Buzzword

is computer talk for numbers bigger than 32767 and smaller than -32768 (16 BIT INTEGER range). Numbers outside this range and those with decimal points must be stored and computed specially for esoteric computer reasons. The trade-off is that the range of the numbers available for FLOATING POINT calculation becomes enormous, but the accuracy starts to slip after a while. Fractions are always converted to FLOATING POINT. The name, by the way, comes from the decimal point floating around according to the POWER of ten the number has to be raised to in order to print it out to six digits of accuracy. It is also called 'scientific' notation; and, if you are not a scientist or engineer, you will probably not need to worry about it. You can convert to whole numbers with the INT FUNCTION.

FONT STRING,ARRAYNAME,SNAPNAME,YOFFSET,LEFTX,RIGHTX
Esoteric Swap Command

is used to create and maintain ARRAYS of characters or symbols to be used with the TEXT COMMAND. Each time it's used, the FONT COMMAND adds one character (or symbol) to a FONT ARRAY if it has not been previously defined in that ARRAY or replaces it if it has.
The ARGUMENTs are:

STRING is a single character. This character is used to identify this entry in the ARRAY. When this character is used in a STRING in the TEXT COMMAND, the corresponding character or symbol in the 'SNAPNAME' is displayed on the screen.

ARRAYNAME is the NAME of the FONTARRAY. If this NAME already exists, the character and SNAP are added to it, replacing a previous entry having the same identifying character, if necessary. If the NAME doesn't exist, it's created.

SNAPNAME is the NAME of the SNAP to be copied into the FONTARRAY. This can be a SNAP of any character or symbol, of any size or COLOR. If it is really large, you can't have many in one FONTARRAY before you run out of space.

YOFFSET is a number used along with the Y-COORDINATE in the TEXT COMMAND to determine the Y-COORDINATE used in displaying this character. A negative number drops the character below the line of text, a positive number raises it. This option is used for characters such as lower case g or p, which should drop below the line of text or superscripts which should go up some.

LEFTX
RIGHTX are numbers from 0 to 4. They identify the type of the left or right edge of a character. The type of the right edge of one character, and the left edge of the next, are used in the TEXT COMMAND to look up a horizontal spacing value in a two-dimensional ARRAY.
For example:

	LEFT				
		o	/	\	1
	0	1	2	3	4
RIGHT 0	*	*	*	*	*
o 1	*	2	3	4	5
/ 2	*	3	4	5	6
\ 3	*	4	5	6	7
1 4	*	5	6	7	8

The value found represents a number of pixels. This value, along with the horizontal spacing

constant given in the TEXT COMMAND, are used to determine the horizontal spacing between characters. The TEXT COMMAND has a built-in ARRAY with all entries of zero. Users may create their own ARRAYS and use them in the TEXT COMMAND to override the built-in ARRAY. A zero in the row column means use the default spacing. Typically you would define less space between two o's than between two l's or two m's.
Example:

Char	LEFTX	RIGHTX
O	3	3
A	4	4
L	4	1
T	1	1
C	2	3
G	3	2
Y	1	1

Here, the spacing between "L" and "C" would be 3 and the spacing between "C" and "L" is 7.

FRAME BUFFER
Buzzword

is used to store the images on the screen. Each PIXEL on the screen is represented by 2 BITs at a location in MEMORY. Changing that MEMORY location will change a specific PIXEL on the screen. There is 16K of screen RAM which means the FRAME BUFFER in ZGRASS has a RESOLUTION of 320 by 201 with 2 BITs per PIXEL.

FUNCTION
Buzzword

is a COMMAND or MACRO that returns a value and is used as part of an EXPRESSION. Actually all COMMANDs and MACROs return values of 1 unless something else is specifically returned. Lots of programming languages use the term FUNCTION so we use it here as a gesture towards programmer solidarity.
Examples:

```
GREED=SIN(AVARICE)
FUNNYARRAY(MAX(A,B,C)=-9999
```

MAX is taken as a user defined FUNCTION which returns the largest of three numbers. See the RETURN Command for how MAX is written.

GETERROR(NUMBER)
Esoteric Swap Function

if NUMBER==0, returns the ERROR number that last occurred. Usually used in conjunction with ONERROR to figure out programatically what ERROR condition arose. Cannot be used outside of the MACRO in which the ERROR occurred.

if NUMBER==2, returns the COMMAND line in ERROR as a STRING. It can be used in conjunction with GETERROR(0) to pinpoint the part of the COMMAND in ERROR and point it out friendly-like to the user of your MACRO.

Example:

```
     BAD=[ONERROR 1
     BOX 0,0,"!",1,3
     PRINT "OK"
     RETURN
     1 PR "ERROR #"&GETERR(0),"ON LINE:",GETERR(2)
     RETURN]
```

will catch the ERROR ("!" is an invalid INTEGER) and print out:

```
ERROR #22 ON LINE: BOX 0,0,"!",1,3
```

GETTAPE FILENAME
Command

gets the FILENAME from tape. May be a MACRO, ARRAY, or a 16K screen dump (see PUTTAPE). When you

```
     GETTAPE FILENAME
```

you get a complete directory listing of everything else which is on the tape before FILENAME. You can also see your file being read in by looking at the lights above the switches. If a read error occurs. the next copy will be read (see PUTTAPE). Use the red RST button to prematurely stop GETTAPE.

Example:

```
     GETTAPE FOOD
```

will search through the tape until it finds FOOD, then print out:

```
     STRING NAME: FOOD
     LENGTH: NUMBER (IN BYTES)
     A DESCRIPTIVE MESSAGE ABOUT THE FOOD
```

If it reads a copy of the file which has errors it will print out '***BAD READ***', and look for another copy.

Switches:

.ERR	accept the file even if an error is read
.ANY	get the next file, whatever its NAME is, on tape and read it in with the NAME

	you specify
.OR	OR's the screen if doing a screen dump read
.XOR	XOR's the screen if doing a screen dump read

GOTO LABEL
Command

causes the line which begins with LABEL to be EXECUTEd next. LABELs begin with numbers.
Examples:
These are valid LABELs:

```
     10
     1NOW
     2small
     30000
```

Example:

```
     SQUARES=[A=80
     1AGAIN BOX 0,0,A,A,A/10
     IF (A=A-10)>0,GOTO 1AGAIN
     PRINT "IS THIS ART?"]
```

HELP
Command

gives a list of the resident COMMANDS and FUNCTIONS available along with their ARGUMENTs.

HELP COMMAND NAME
Command

gives information on using a specific COMMAND.

IF CONDITIONAL,COMMAND
Command

if the CONDITIONAL is satisfied the COMMAND following is EXECUTEd. Otherwise, control is skipped to the next line. A CONDITIONAL is a EXPRESSION which evaluates to 0 (false) or 1 (true). Expressions using RELATIONAL OPERATORS evaluate to true or false, and the rest of the line (including ';'s) is EXECUTEd if the condition is true. Anything that evaluates to 0 or 1 can be used as part of an IF statement. Note that IF must always be followed by a space.
For example:

```
     IF A==10,PRINT A;.will  print the  value of A
     if it is equal to 10

     FIXUP=[PR "I'M YOURS"]
     IF 1,FIXUP;.this will always happen
     IF FLAG,B=C+D;.this will happen if FLAG==1

     IF SIN(BRADIANS)*1.25<=.7,DRAW
```

The last example shows that complex EXPRESSIONs are allowed in an IF statment. Note: "equals" as a RELATIONAL OPERATOR is "==", and a single "=" is the ASSIGNMENT OPERATOR, even in IF statements. For example:

```
     IF A=B,FOO
```

EXECUTES FOO only if B=0.

INDEX
Buzzword

is the NUMBER indicating which ARRAY element is being picked. The INDEX in ABC(4) is 4. ARRAYs can have multiple indices if they are multidimensional; for example, CHECKERBOARD(8,8), which has indices (0,0),(0,1),...,(7,7) allowing 64 elements.

INDIRECTION
Buzzword

allows one NAME to hold another NAME as a STRING to be used as a reference. '@' is the indirection OPERATOR. Examples:

```
     TOM=12
     SAM="TOM"
     PRINT @SAM
```

this prints 12

```
     MKARRAY=[PR "ARRAY NAME PLEASE"
     INPUT.STR ANAME
     PR "HOW MANY ARRAY ELEMENTS?"
     INPUT n
     CLEAR.CRT
     ARRAY @ANAME,n
     PRINT ANAME,"HAS ELEMENTS 0 TO",n-1
     TMP=@ANAME
     i=0
     PROMPT ANAME&"("&i&")=?"
     INPUT q
     TMP(i)=q
     IF(i=i+1)<n,SK -3
     PRINT "ARRAY",ANAME,"HAS THE VALUES:"
     i=0
     PRINT ANAME&"("&i&")="&TMP(i)
     IF (i=i+1)<n,SK -1]
```

When you EXECUTE MKARRAY, first it makes an ARRAY with ANAME of size n, then the user inputs values

for each ARRAY element, and finally the contents of the ARRAY is printed out.
The detail to notice is:

```
     TMP=@ANAME
```

This is a shortcut for dealing with ARRAY elements in a general program, so that each element can be accessed as TMP(0), TMP(1),...,TMP(n). We could skip the assignment of @ANAME to TMP and instead build a string:

```
     @(ANAME&"("&1&")")
```

which is the same as

```
     TMP(1)
```

Unfortunately, the building of strings through CONCATENATION is time-consuming.

INFINITE LOOP
Buzzword

is a LOOP which has no intention of ever stopping. Such a LOOP is an error if you want the MACRO it's in to stop or are using it as a FUNCTION which is supposed to return a value. It can be useful, though, as a MACRO run under .B or .F mode or something you want to get out of by using CTRL+C. The LOOPMAX COMMAND can be used to catch infinite loops.

INPUT NAME1,NAME2,...,NAMEN
Command

gets the VALUE from the user or the ARGUMENT list passed to the MACRO and stores the VALUE as a number in NAME.
Examples:

```
     ABS=[INPUT a
     IF a<0,RETURN -a
     RETURN a]
     PRINT ABS(-10)
```

prints out 10

```
     PRINT ABS(10)
```

prints out 10

```
     ASK=[PROMPT "WHAT'S YOUR AGE?"
     INPUT AGE
     PRINT "YOU ARE",AGE*12,"MONTHS OLD AT LEAST"]
```

if EXECUTEd by typing:

```
     ASK 33
```

the PROMPT is suppressed.
if EXECUTEd by typing:

```
     ASK
```

the PROMPT is printed, and you have to supply the ARGUMENT by typing it in.
Note: if you are passing a VARIABLE (rather than a number, as above), make an EXPRESSION of it by

adding 0 or using the "?" OPERATOR so its VALUE is passed rather than its NAME. This is particularly important when passing LOCAL VARIABLEs and ARRAY references.

INPUT.NAME NAME
Command

gets a STRING of characters from the user or the ARGUMENT list passed to the MACRO and checks it for valid SYNTAX, and then puts it into NAME as a STRING.
Example:

```
WHO=[PROMPT "TYPE YOUR FIRST NAME:"
INPUT.NAME NAME1
PRINT NAME1"IS A FUNNY NAME!"]
```

Note: Do not use INPUT.NAME to pass VARIABLEs to called MACROS if it is the value of the VARIABLE you want to pass. Use INPUT.STR to pass a STRING in a VARIABLE to a called MACRO.

INPUT.STR NAME1,NAME2,...,NAMEN
Command

gets a STRING of characters and then puts it into NAME. This option is good for reading an entire line from the terminal, including commas. It must also be used to pass a STRING with commas or spaces as an ARGUMENT, in which case it should be enclosed in quotes or other STRING delimiters.
Examples:

```
MAILINGLIST=[PROMPT "TYPE IN A NAME, ADDRESS,
AND PHONE # FOLLOWED BY A BLANK LINE"
CR={
}
PROMPT "MORE:"
INPUT.STR INFO
IF INFO#{},LIST=LIST&INFO&CR;SK -2]
```

Note: when passing LOCAL STRING VARIABLEs to MACROS, make EXPRESSIONS out of them by CONCATENATing them with a null string or by using the "?" OPERATOR in front of the NAME so that the VALUE of the STRING is passed rather than the NAME of the STRING.

INT(NUMBER)
Function

FUNCTION which returns the INTEGER part of a number. INT(5%8) will give 5, 6, or 7 without the fractional part, for example.

INTEGER
Buzzword

An integer in ZGRASS is a number between 32767 and -32768. It is very easy for the computer to store and deal with numbers in this range so they are used often. Fractions and decimal points are not allowed in INTEGER arithmetic.

INTERRUPT
Esoteric Buzzword

The ZGRASS System is programmed to EXECUTE a chunk of special code every 1/60 of a second, when the code is "interrupted" by the vertical sync of the TV scan. .F causes a macro to run every 1/60 second.

ITERATION
Buzzword

is the process of solving things by doing LOOPS. Typically, in computing, ITERATION means doing things incrementally. For instance, a computer would probably walk over to the wall by accurately measuring the distance between it and the wall, computing the exact number of steps needed, and then it would take a step, see if all the steps it had to take were taken yet, and take another if not. If it made a mistake, it might crash into the wall. People, of course, do things through feedback, and often you can program that way with computer systems that are significantly better connected to you than the average payroll-check stomper (like ZGRASS is, of course). To draw 100 RANDOM sized BOXes on the screen, you could type in 100 different BOX COMMANDs, or write a MACRO which would do it. For example:

```
SQUARES=[B=0
BOX -150%150,-90%90,1%50,1%30,1%8
IF (B=B+1)<100,SK -1]
```

JOYSTICK
Idiosyncrasy

is the gadget with the knob and the trigger that is connected to the ZGRASS machine. You can have up to four joysticks. The first one's knob is known as $K1, its X value as $X1, its Y value as $Y1, and its trigger value as $T1 (see DEVICE VARIABLES).

JUMP ADDRESS
Esoteric Command

Refer to the Swap Module creation documentation, a separate package.

LABEL
Idiosyncrasy

GOTO 1THIS causes ZGRASS to move to whatever line begins with the LABEL 1THIS. LABELs in ZGRASS start with numbers to differentiate them from NAMES which cannot start with numbers. LABELs also cannot contain punctuation. You can't have one GOTO in a MACRO go to a LABEL in another MACRO.

LEN(STRING)
Esoteric Function

returns the length of a character STRING. If the ARGUMENT is a null STRING, 0 is returned.
Example:

```
     PRINT LEN("abcdef")
```

prints the VALUE 6

LEN.NUMBER
Esoteric Function

makes the system print out NUMBER to n decimal places. The default is 6.

LINE XCOORDINATE,YCOORDINATE,COLORMODE
Command

draws a line from the previous line endpoint used in the current MACRO to the endpoint specified by the XCOORDINATE and YCOORDINATE in the COLORMODE indicated. LINE X,Y,4 will move the endpoint without drawing anything and can be used to set the first endpoint if you do not want the first LINE to start at (0,0). See COLOR MODES. Each MACRO has its own place to store the last endpoint used and it is set to zero when the MACRO is called.
Example:

```
     LINE 50,-30,1
```

draws a line from 0,0 to 50,-30.

```
     LINE -80,20,2
```

draws a line from 50,30 to -80,20.

```
     LINE 50,50,4
     LINE 50,-50,3
     LINE -50,-50,3
     LINE -50,50,3
     LINE 50,50,3
```

draws a rectangle outline.

```
ZIGZAG=[LINE -160%159,-100%100,0%15;SK 0]
```

ZIGZAG will draw RANDOM lines of different COLORs all over the screen

LOCAL VARIABLE
Esoteric Idiosyncrasy

a VARIABLE which starts with a lowercase (a-z) letter. LOCAL VARIABLEs are known only to the MACRO they are in and are deleted automatically when the MACRO returns. They help save memory and are really useful in .B, .F, and RECURSIVE MACROs.

LOGICAL OPERATOR
Buzzword

returns a truth value (0 or 1). ZGRASS has logical "AND" and "OR". The "AND" OPERATOR is '&&'. The logical "OR" OPERATOR is '¦¦'. They are useful in many situations, one of which is combining conditionals in IF statements. Examples:

```
BEEPTHEJEEP=[CONTROL 14,1]
IF A==10&&B==20,BEEPTHEJEEP;.done if A is 10
and B is 20
IF A==10¦¦B==20,BEEPTHEJEEP;.done if either
is true
```

LOOP
Buzzword

is a series of COMMANDS done over and over. If the loop never stops, it is called an INFINITE LOOP. LOOPs in ZGRASS are constructed with IF's, GOTOs and SKIPs or with .B and .F. You can always get out of a LOOP with CTRL+C. CTRL+Z allows you to get out of a LOOP to do something and then get back in by pressing the RETURN key. A loop is an example of ITERATION.
Examples:

```
INFINITELOOP=[PRINT A=A+1;SK 0]
```

is a loop which will not stop because it doesn't have an end condition. CTRL+C will stop it.

```
LOOPWHICHSTOPS=[A=0
PRINT A
A=A+1
IF A<10,SKIP -2]
```

LOOPWHICHSTOPS prints 0 through 9 and stops.

LOOPMAX NUMBER
Esoteric Command

allows you to catch INFINITE LOOPs by setting a maximum for the NUMBER of SKIPs and GOTOs that can occur before ERROR #65 is caused. Macros which contain a LOOPMAX command cannot be COMPILEd.

Make sure when you use LOOPMAX that you set it up outside the loop or it won't work correctly.
Example:

```
     TEST=[CONTROL 1,0;.SET CTRL+A TO ZERO
     PRINT "HIT CTRL+A"
     ONERROR 1SLOW
     LOOPMAX 100
     IF CONTROL(1)#1,SK 0
     RETURN
     1SLOW PRINT "YOU DIDN'T HIT CTRL+A FAST
     ENOUGH!"]
```

LN(NUMBER)
Function

returns the natural log of NUMBER.

LPAD(STRING,CHARACTER,FIELDWIDTH)
Esoteric Function

returns a pointer to the STRING, padded on the left with a specified CHARACTER so that it fits within a given FIELDWIDTH.
Examples:

```
     PR LPAD("ABC","*",6)
```

prints out the STRING "***ABC"

```
     PR LPAD("EXAMPLE","*",5)
```

prints out the STRING "AMPLE"

```
     LEFTX=[A=2
     PR LPAD(A,'X',5)
     A=A*10;IF A<=20000,SK -1]
```

takes each VALUE of A and pads it on the left with X's until each number is printed in a field of 8 characters. Usually used with blanks, not X's.

```
               XXXX2
               XXX20
               XX200
               X2000
               20000
```

Given JOHN is an ARRAY with 20 numbers representing USA money:

```
     DOLLARSANDCENTS=[TOTAL=0
     A=0
     TOTAL=TOTAL+JOHN(A)
     PRINT LPAD(FORMAT(JOHN(A),2),' ',20)
     A=A+1
     IF A<20, SKIP -2
     PRINT LPAD('-','-',20)
     PRINT "TOTAL=",LPAD(FORMAT(TOTAL,2),' ',13)]
```

As another example:

```
     PR FORMAT(10/4,3)
```

prints out the STRING "2.500".

MACRO
Idiosyncrasy

is a STRING that contains legal ZGRASS COMMANDS. Most programming languages call such things 'programs' or 'subroutines'. MACROs are user-defined COMMANDS. You can pass ARGUMENTs to MACROs with the INPUT COMMAND and return values with the RETURN COMMAND. You define a MACRO just like you define a STRING (with an ASSIGNMENT to a NAME or by using EDIT).

MATCH(OTEXT,MTEXT,LOWER,UPPER)
Esoteric Swap Function

Search for the occurrence of MTEXT, a STRING, within a specified range of OTEXT, another STRING. If a MATCH is found, the returned displacement value is relative to the beginning OTEXT, the first character being the 0th one. -1 is returned if a MATCH was not found within the specified limits. The search for a MATCH may proceed from either direction. If UPPER is greater than or equal to LOWER a forward search is made. If UPPER is less than LOWER a backward search is made. (That is, the characters are still matched left to right bu the pointer backs up on failure to match instead of advancing.) MTEXT does not necessarily have to contain all the characters of the desired MATCH but rather, may use the following expression symbols:

?	(wild card) MATCH any one character
*	MATCH all characters
*text	MATCH all characters preceding actual text
text*	MATCH text and all remaining characters following text
text1*text2	MATCH all characters between text1 and text2
[chars]	MATCH first occurrence of any one of the characters with the '[',']'s. All the expression symbols lose their special meaning when appearing within square brackets.
[char-char]	MATCH any character within the range specified. [0-9] is the same as specifying [0123456789]. The minus sign

	loses its special meaning when specified as first or last character within the square brackets.
\	ignore the following character's special meaning
¦	anchor MATCH to beginning or end of OTEXT depending on whether the anchor symbol occurs first or last within MTEXT

Examples:

```
                                                ANSWER:
PR MATCH("VACATION","CAT",0,7)                    2
PR MATCH("VACATION","CAT",0,3)                   -1
PR MATCH("ABABCDAB","?A",0,10)                    1
PR MATCH("ABABCDAB","?A",4,10)                    5
PR MATCH("ABABCDAB","?A",10,0)                    5
PR MATCH("ABABCDAB","?A",4,0)                     1
PR MATCH("SIGNAL","*",0,20)                       0
PR MATCH("WHAT TIME?","ME\?",0,10)                7
PR MATCH("SIGNAL","*",20,0)                       0
PR MATCH("SIGNAL","*",3,20)                       3
PR MATCH("THIS IS A TEST","[AEIOU]",3,7)          5
PR MATCH("THIS IS A TEST","[A-H]",15,0)          11
PR MATCH("GRAPHICS","¦",0,10)                     7
PR MATCH("COMPUTER GRAPHIX","G*X",5,20)           9
```

MEMORY
Buzzword

is computer storage which is divided into BYTES. ZGRASS has 320K BYTES of MEMORY. 32K is ROM (Read Only Memory) where the resident code for ZGRASS is stored. 256K is Screen RAM (Random Access Memory that feeds the TV screen). 32K is RAM used to store MACROS, ARRAYS, SWAP MODULES, SNAPS, and VARIABLES. USEMAP shows usage of the 32K RAM. CORE tells you how much of the 32K RAM you have free.

MMOVE SOURCE,DESTINATION,LENGTH
MMOVE.UP SOURCE,DESTINATION,LENGTH
Esoteric Command

Uses the Z-80 LDIR (block memory move) instruction to move the number of bytes given by LENGTH from the SOURCE to the DESTINATION. It's good for esoteric manipulations of screen memory. Beware, you can also scramble user memory easily. Very Esoteric Note: MMOVE does a LDDR Z-80 instruction. MMOVE.UP does a LDIR Z-80 instruction. The first argument is HL, second DE, and the third is BC.

Examples:
```
    NB
    MMOVE 31600,32000,15200
```
moves image down
```
    MMOVE.UP 16384+80,16384,16000
```
moves image up

```
    MMOVE 31998,32000,15200
    MMOVE 31919,32000,15000
    MMOVE 21919,22000,5000
    MMOVE 31920,32000-16384,15000
```
The last one shows the use of XOR (works only if XOR set by drawing a box with XOR -- as NB does.) Also works with OR if last box was OR'd. Note that screen source is addressed at 0 by subtracting 16384, which kicks in the special XOR and OR hardware. Note that if $ML is 1, you can use $MR and $MW to do clever copying between screen pages. See SCREEN.

NAME
Idiosyncrasy
is any set of symbols starting with a letter that has a VALUE (TOM=5, SAM="HOWDY", for example) or an ARRAY of VALUEs (ARRAY WOMEN,13, for example). A NAME must start with a letter (or '$') and has only letters and numbers (0-9) and '$'s in it. The rule is that a STRING is not a name if it starts with a number. In this case, it is either a NUMBER or a LABEL (LABELs must be the first thing on a line, of course). If it starts with a letter, it is a NAME. Any kind of punctuation ends the NAME. A NAME is also an EXPRESSION, although a very simple one. NAMES joined together with numbers and other NAMEs using punctuation (+,-,/,*,(,),etc.) are EXPRESSIONS. If a NAME begins with a lowercase letter, it is LOCAL and is known only to the MACRO in which it occurs. For example, sam=5.

NEXTLINE
Idiosyncrasy
is the code ZGRASS uses to represent the end of a line. It is generated by the RETURN key. Sometimes it is known as the 'carriage return' or 'CR' from the old days or 'RETURN' on most keyboards (not to be confused with the RETURN COMMAND, of course). This character is at the end of every line in a MACRO except possibly the last. It is also the key which tells ZGRASS you are

finished typing in the line you have been typing. If you hit CTRL+Y and then list out a MACRO, you will see a '!' marking the position of each NEXTLINE. NEXTLINE also advances the 20-line printout mode started by CTRL+W.
Note: you cannot have any spaces before the NEXTLINE. CTRL+Y is good for verifying that no spaces exist between the last character on the line you've typed and the NEXTLINE. In edit, also use CTRL+T.

NUMBER
Buzzword

Examples:

```
1778
1.5
-44.3
3.5E6 (3.5 million)    → Exponential
-2E-9 (-2 trillionths)   u
```

NUMERIC VARIABLE
Buzzword

is a VARIABLE which has a NUMBER as its value. USEMAP will tell you it is a NUMNAME.

ONERROR LABEL
Esoteric Command

sets up a transfer to LABEL when an ERROR occurs. You can turn off ONERROR by specifying no LABEL (ONERROR by itself turns the normal ERROR CODES back on). You normally put a ONERROR LABEL before a statement that is likely to cause an ERROR. You can only have one ONERROR setup per MACRO at a time, but you can change it in the MACRO anytime. MACROs which have ONERROR commands cannot be COMPILEd. See LOOPMAX and GETERROR for examples of ONERROR. Note: this COMMAND precludes you from EXECUTing a MACRO NAMEd "ONE" due to the ABBREVIATION POLICY. This is a common mistake.

OPERATOR
Idiosyncrasy

is what glues NUMBERs and NAMEs into EXPRESSIONS. OPERATORs take the values they operate on and return a single value. Each OPERATOR has a precedence, that is, a pecking order for evaluation.

OPERATOR:	MEANING:	PRECEDENCE:
@	indirect	9
?	value	9
-	unary minus	8

!	absolute value	8
+	unary plus	8
%	random	7
/	division	6
\	modulus	6
*	multiply	6
+	add	5
-	subtract	5
&	concatenate	5
==	equals	4
<	less than	4
>	greater than	4
<= or =<	less than or ==	4
>= or =>	gr. than or ==	4
# or <>	not equals	4
&&	logical AND	3
¦¦	logical OR	2
=	assign	1
()	parentheses	0

OPERATORS with higher PRECEDENCE are done before ones with lower PRECEDENCE and ones with equal PRECEDENCE are done from left to right. Examples:

```
     2+3*4    equals 14
     (2+3)*4  equals 20
     -7*3+2   equals -19
```

OR
Buzzword

works on BITs. It makes BITs OR'ed with 1's equal to 1, and leaves BITs OR'ed with 0's the same as they were.
OR table using 2 BITS:

	8	9	10	11
OR	00	01	10	11
00	00	01	10	11
01	01	01	11	11
10	10	11	10	11
11	11	11	11	11

The OR COLOR MODES are 8-11. The OR DISPLAY MODES are 2,12,22,...,132,134.

OVERFLOW
Buzzword

is what happens when the range of a CONSTANT, VARIABLE, or EXPRESSION is too large or too small. For instance, many DEVICE VARIABLES represent a single BYTE of information which gives a range of 0-255 or -128 to 127. Exceeding this range causes WRAP-AROUND so 256 is actually 0, 257 is 1, 258 is 2. INTEGERs overflow after 32767 and under -32768, which causes ERROR # 23.

PATTERN X,Y,XOFFSET,YOFFSET,SNAPNAME
Command

like PATTERN.FILL but uses PIXELs out of a SNAPNAME to fill within a bounded area. X and Y indicate the starting point for the pattern fill. The area is filled with SNAPNAME as if its lower left corner were positioned at 0,0. XOFFSET and YOFFSET are used to change this orientation. The following example illustrates the use of the pattern fill with and without offsets.
Example:

```
     OPART=[CLEAR
     BOX -130,0,24,24,3
     BOX -130,0,18,18,2
     BOX -130,0,12,12,1
     BOX -130,0,6,6,0
     SNAP SQR,-130,0,24,24
     BOX 0,0,180,100,1
     BOX 0,0,178,98,0
     BOX 40,0,50,40,1
     BOX 40,0,48,38,0
     BOX -40,0,50,40,1
     BOX -40,0,48,38,0]
     OPART
     PATTERN 40,0,0,0,SQR;.NO OFFSET
     PATTERN -40,0,14,-15,SQR;.OFFSET ON X AND Y
     PATTERN 0,0,0,0,SQR
```

PATTERN can be aborted with CTRL+E.

PATTERN.FILL X,Y,FILLCOLOR
Command

is used to fill a bounded area with a solid color. X and Y are the coordinates of a point interior to the boundary. FILLCOLOR can be 0,1,2,3 referring to the four COLORs $LO-$L3 (and $RO-$R3 if $HB is SET). Refer to PATTERN for filling areas with a pattern. Example:

```
     BOX 0,0,80,80,1
     BOX 20,20,40,40,0
     BOX -20,0,36,76,0
     BOX 18,-20,40,36,0
```
Creates an L shaped area in red ($L1).
```
     PATTERN.FILL 10,-20,3
```
will fill the area bounded by the red outline with blue starting at the point 10,-20.

PIXEL
Buzzword

is the smallest thing you can change on the screen. The POINT COMMAND will fill one pixel with a COLOR. The screen is divided into 64640 pixels (320*201). There are 320 PIXELs horizontally and 201 vertically. The center of the screen is (0,0) and the PIXELs are numbered -160 to 159 horizontally (X direction) and -100 to 100 vertically (Y direction). The POINT FUNCTION will give you the COLOR VALUE of a PIXEL.

Due to the fact that the PIXELs are not quite square, circles are somewhat elliptical and squares are somewhat rectangular on most TV sets; this is a non-adjustable hardware constraint.

PLOP
Buzzword

means that whatever COLOR you write with (00-11) will cover whatever is on the screen. So:
```
     Y PLOP X equals Y
```
For Example:
```
     00 PLOP 10 equals 00
     10 PLOP 11 equals 10
```
PLOP table using 2 BITs.

	PLOP with:			
	0	1	2	3
PLOP	00	01	10	11
00	00	01	10	11
01	00	01	10	11
10	00	01	10	11
11	00	01	10	11

The PLOP COLOR MODES are 0-3. The PLOP DISPLAY MODES are 0,10,20,30,...,130,140.

POINT(XCOOR,YCOOR)
Function

returns the value (0-3) of the PIXEL ADDRESSed by the two COORDINATES given. 0 means that COLOR (00) is at that ADDRESS on the screen, 1 means that COLOR (01) is there, etc. If the ADDRESS is outside the current WINDOW area, a -1 is returned.
Example:
```
     BOX 0,0,40,40,1
     PRINT POINT(18,15)
```
prints 1
```
     PRINT POINT(50,70)
```
prints 0

POINT XCOORDINATE,YCOORDINATE,COLORMODE
Command

draws a point at XCOORDINATE,YCOORDINATE in the COLOR MODE specified. A POINT is one PIXEL in size. See COLOR MODE.
Examples:
```
     POINT 80,30,1
```
draws a red point at 80,30
```
     POINT 40,20,2
```
will draw a green point at 40,20.
```
     SPIRAL=[angle=0;radius=0
     x=radius*SIN(angle)
     y=radius*COS(angle)
     POINT x,y,1
     angle=angle+18
     radius=radius+.5
     SKIP -5]
```
SPIRAL draws a spiral starting at 0,0. Press CTRL+C to stop it.

POINT.SNAP(SNAPNAME,XCOOR,YCOOR)
Esoteric Function

returns the value (0-3) of a PIXEL in SNAPNAME ADDRESSed by XCOOR and YCOOR. XCOOR and YCOOR are relative to the center (0,0) of the SNAP. -1 will be returned if the PIXEL is outside the SNAP.
Example:
Create a three-color SNAP using the macro PSNAP
```
     PSNAP=[CLEAR
     BOX -10,0,10,20,3
     BOX 0,0,10,20,2
     BOX 10,0,10,20,1
     SNAP FLAG,0,0,30,20]
```
use TEST to find the color value of the PIXELs in the SNAP FLAG
```
     TEST=[PROMPT"INPUT X,Y POSITIONS IN SNAP"
     INPUT x,y
```

```
          PRINT POINT.SNAP(FLAG,x,y)
          SKIP -3]
```

POINT.SNAP SNAPNAME,XCOOR,YCOOR,COLORMODE
Esoteric Command

changes the PIXEL at XCOOR,YCOOR in SNAPNAME in the COLORMODE specified. XCOOR and YCOOR are relative to the center of the SNAP which is 0,0. You have to DISPLAY the SNAP to see the changes, of course.

Example:

The following MACRO will change any of the PIXELs in FLAG which are red (COLOR 01) to blue (COLOR 11).

```
     COLORCHANGE=[CLEAR
     BOX -10,0,10,20,3
     BOX 0,0,10,20,2
     BOX 10,0,10,20,1
     SNAP FLAG,0,0,30,20
     PR"WATCH THE FLAG CHANGE COLORS"
     y=(FLAG(1)/2);sy=-y
     PR sy,y
     x=FLAG(0)/2
     sx=-x
     c=POINT.SNAP(FLAG,sx,sy)
     IF c==1,POINT.SNAP FLAG,sx,sy,3
     IF c==2,POINT.SNAP FLAG,sx,sy,1
     IF c==3,POINT.SNAP FLAG,sx,sy,2
     IF (sx=sx+1)<=x,SK -4
     DISPLAY FLAG 0,0,0
     IF (sy=sy+1)<y,SK -7
```

The X size of FLAG is stored in FLAG(0). The Y size is stored in FLAG(1). This information is used to determine the setup for two nested loops which will go PIXEL by PIXEL through the SNAP FLAG looking for 01 PIXELs and changing them to 11, 10 PIXELs to 01, and 11 PIXELs to 10 PIXELs. After an entire horizontal line of the SNAP has been evaluated, FLAG will be DISPLAYed.

PORT(NUMBER)
Esoteric Function

returns the VALUE read at the PORT NUMBER identified.

Example:

```
     PR PORT(20)
```

will print the value of the switches 0-7. If switches 0,1,2,3 are down, 15 will be printed.

PORT NUMBER1,NUMBER2
Esoteric Command

writes NUMBER2 to the PORT identified by NUMBER1.
Examples:

```
     A=0
     COUNT=[PORT 38,A
     A=A+1
     WAIT 1
     SKIP -3]
```

will cause the lights to count in binary, one per second until WRAPAROUND occurs after 255.

```
     FASTERCOUNTDOWN=[PORT 38,A
     $Z0=32767
     PORT 38,$Z0
     IF $Z0#0,SK -1]
```

$Z0 is a system timer decremented by 1 every 1/60th second. When $Z0 hits 0, it is no longer decremented.

PORTS
Idiosyncrasies

are hardware ADDRESSes for DEVICEs and various input and output gadgets. Some are massaged and put into DEVICE VARIABLES (like the JOYSTICKS & COLOR VARIABLEs). Some are accessed with COMMANDS (like RS232).
Example:

```
     PRINT PORT(N)
```

will print what the value is at PORT N is.

```
     PORT N,K
```

will set PORT N to the VALUE in K

OUTPUT PORTS: (write only)

PORT #:	FUNCTION:
10	Vertical Blanking Line
12	Magic Register
16	Master Oscillator
17	Tone A Frequency
18	Tone B Frequency
19	Tone C Frequency
20	Vibrato
21	Tone C VOLUME and Noise Modulation Control
22	Tone B Volume and Tone A Volume
23	Noise Volume
25	Expand Register
38	Lights 0-7 (Bit 0=Light 0)
39	Lights 8-15 (Bit 0=Light 8)
40	Controls Tape Motor Switch 1=on 0=off

Bit 0=Motor 1, Bit 1=Motor 2

INPUT PORTS: (read only)

Bit values for each Joystick	
Bit 0	UP on (Y)
Bit 1	Down on (Y)
Bit 2	Left on (-X)
Bit 3	Right on (X)
Bit 4	Trigger
Bits 5-7	not used

16	Joystick 1 ($X1,$Y1,$T1)
17	Joystick 2 ($X2,$Y2,$T2)
18	Joystick 3 ($X3,$Y3,$T3)
19	Joystick 4 ($X4,$Y4,$T4)
20	Switches 0-7 (bit 0=switch 0)
21	Switches 8-15 (bit 0=switch 8)
28	Knob Joystick 1 ($K1)
29	Knob Joystick 2 ($K2)
30	Knob Joystick 3 ($K3)
31	Knob Joystick 4 ($K4)
46	Tablet data

INPUT/OUTPUT PORTS: (read and write)

32	Terminal RS232 data
33	Accessory RS232 data
34	Terminal RS232 control
35	Accessory RS232 control
36	NCUDAT(9511 chip) data port
37	NCUCOM(9511 chip) status port
41	Tape Data Bit 0=Data

POWER(NUMBER1,NUMBER2)
Function

returns NUMBER1 raised to the POWER of NUMBER2.
Example:

```
    PRINT POWER(5,3)
```

prints 125

PRECEDENCE
Buzzword

is the pecking order for the evaluation of OPERATORS in EXPRESSIONs. See OPERATOR for the PRECEDENCE order in ZGRASS.

PRINT THING
Command

THING (a NUMBER, ARRAY VALUE, EXPRESSION etc.) is converted to a STRING, if possible, and printed followed by a NEXTLINE. Several STRINGs can be used. If you separate them by commas, a space is printed between them. If you do not want the

space. separate them with &'s. Stuff in quotes can also be used (like PRINT "THE ANSWER IS:",A). PRINTS (and PROMPTS) are suppressed if there are ARGUMENTs passed to the MACRO.
Examples:

```
     PRINT 5
will print 5
     A=1;B=333
     PRINT A&B
will print 1333
     ME=7
     PRINT 5,ME
will print 5 7
     PRINT "A"&"B"&"C"
will print ABC
     PRINT "FOOT("&1&")="&"BIGTOE"
will print FOOT(1)=BIGTOE
```

PRINT.FORCE THING
Command

like PRINT but forces printing whether or not an ARGUMENT list is passed to the MACRO.
Example:

```
     FLUBB=[
     PRINT.F "I WAS FORCED TO PRINT THIS"
     INPUT n]
     FLUBB 4
```

will print "I WAS FORCED TO PRINT THIS" even though you directly passed the value 4.

PRIORITY WRITE
Idiosyncrasy

means a COLOR 00-11, will write over another COLOR if it is greater than or equal to that COLOR.
So: X PRIORITY Y equals MAX(X,Y)
For example:

```
     10 PRIORITY 11 equals 11
     01 PRIORITY 00 equals 01
```

ZGRASS has two PRIORITY WRITE COLOR MODES 16 and 17. See REVERSE PRIORITY.

PRIORITY		16	17	
WRITE	00	01	10	11
00	00	01	10	11
01	01	01	10	11
10	10	10	10	11
11	11	11	11	11

PROMPT THING
Command

just like PRINT but does not print the NEXTLINE at the end.

PROMPT.FORCE THING
Command

like PROMPT but forces printing whether or not an ARGUMENT list is passed to the MACRO.

PUNCTUATION
Buzzword

is any typed character which is not a letter, or number, or '$'. Many PUNCTUATION symbols are OPERATORS.

PUTTAPE NUMBER,FILENAME,STRING
Command

puts FILENAME (MACRO, STRING, ARRAY, SWAP MODULE, SCREEN dump) on the tape the number of times indicated by NUMBER under the NAME of "FILENAME". The last ARGUMENT is a message to be put with the tape directory header which will print back when scanning the tape using GETTAPE. See GETTAPE. The reason for printing a file several times is to safeguard against errors. An error detection code is stored with each entry on tape and if an error is detected by GETTAPE, it will try the next copy automatically for you. Press the RESET button to stop PUTTAPE.
Example:

```
    PUTTAPE 3,PARMESAN,[THIS IS A SNAP OF CHEESE]
```

puts out the SNAP ARRAY PARMESAN three times on tape with the message indicated.

PUTTAPE.TV -
Command

a 16K dump of the screen will be put out on tape under the FILENAME.

RADIANS
Esoteric Buzzword

PI RADIANS is defined as equal to 180 degrees. One degree is equal to 3.14159/180 RADIANS. One RADIAN equals 180/3.14159 degrees. (1 RADIAN = 57.296 DEGREES) SINE, COSINE, and TANGENT take values in DEGREES. ARCTAN returns values in DEGREES. The system default is DEGREES. If you want to use RADIANS instead, set $RD to 1.

RANDOM
Buzzword

is a way of choosing a NUMBER in a range so that the NUMBER is not predictable. The RANDOM OPERATOR in ZGRASS is '%'. 10%100 means pick a NUMBER between 10 and 100 (but not including 100). Each time the % OPERATOR is used, the answer should be different, because it is RANDOM, although sometimes it's the same.

RECURSION
Buzzword

see RECURSION.

RELATIONAL OPERATOR
Buzzword

returns the value of 1 if the condition is true, 0 if false. RELATIONAL OPERATORs are used in IF statements mostly but can be used in other contexts as well since they are OPERATORS just like the arithmetic ones.
The RELATIONAL OPERATORS in ZGRASS are:

OPERATOR:	MEANING:
==	equals
<	less than
>	greater than
<= or =<	less than or equals
>= or =>	greater than or equals
# or <>	not equals

See IF COMMAND for examples.

REPLACE(BIGSTRING,OLDSTRING,NEWSTRING,NUMBER)
REPLACE(BIGSTRING,OLDSTRING,NEWSTRING,NUMBER,LOWER,UPPER)
Esoteric Swap Function

Search for the occurrence of OLDSTRING in BIGSTRING from the beginning of BIGSTRING and replace OLDSTRING with NEWSTRING. NUMBER specifies how many times to attempt replacement. The string with the replacement is returned. BIGSTRING is not modified. The matching of OLDSTRING is accomplished in the same manner as in the MATCH routine. You can use expression symbols, as described in the MATCH FUNCTION. If LOWER and UPPER are present, they indicate the start location to search and the end location. If UPPER is less than LOWER, the search is done backwards (that is, from UPPER down one by one to LOWER).

Examples:

```
     PR REPLACE("ABA","A","-*-",1)
prints out "-*-BA"
     PR REPLACE("ABA","A","-*-",1,5,0)
prints out "AB-*-"
     PR REPLACE("SUNSHINE","SUN","MOON",3)
prints out "MOONSHINE"
     PR REPLACE("UNIVERSITY OF ILLINOIS AT CHICAGO
     CIRCLE","*","UICC",1)
prints out "UICC"
     PR REPLACE("THIS IS A VERY EASY
     TEST","[AEIOU]","-",20,10,0)
prints out the string "TH-S -S -  VERY EASY TEST"
     NOISE=[BEEP THE JEEP]
     PR REPLACE(NOISE,"EEP","UNK",2)
prints out "BUNK THE JUNK".  NOISE is unchanged.
     NOISE=REPLACE(NOISE,"EEP","UNK",2)
prints out "BUNK THE JUNK", and assigns this
NEWSTRING to NOISE.
```

RESOLUTION
Buzzword

is the measure of the number of PIXELS on the TV screen. The RESOLUTION of ZGRASS is 320 by 201.

RESTART
Command

clears memory and restarts ZGRASS if you answer by pressing the 'y' key. This is a software way to push the red RST button. 'N' or any other key press will not clear memory. This option is there since system failure often results in automatic restarts, and typing "N" in prevents you from losing everthing in MEMORY.

RESTART STRING
Command

will RESTART the system and then automatically execute the STRING. Example:

```
RESTART [DGET DOIT;DOIT]
```

NOTE: Some differences between RESTART and RESTART STRING:

With RESTART STRING, the $VARIABLEs do not reset, the STRIPE command is still in effect. and a DLOAD'd disk is still there. Also, RESTART STRING does not ask for 'y' or 'n'.

RETURN
Command

returns control to the calling MACRO. Same as running off the end of a MACRO.

RETURN VALUE
Command

returns the VALUE indicated and control to the calling MACRO. Useful for creating user defined FUNCTION calls which return values.
Example:

```
MAX=[INPUT a,b,c;.NOTE THE local VARIABLES
IF a<b,IF b<c,RETURN c
IF a>b,IF a>c,RETURN a
RETURN b]
```

This will return the maximum of the three parameters passed and could be used in:

```
BIGGEST=MAX(OF,THESE,THREE)
HONEY=MAX(CRUNCH1,CRUNCH2,KISS)
NUWAVE=MAX(?a,?b,?c)
```

The last is an example of passing LOCAL VARIABLEs. NOTE that, for a rather esoteric design deficiency, you cannot pass back a local string with RETURN unless the MACRO is COMPILEd or you CONCATENATE it with a null string.

REVERSE-PRIORITY
Buzzword

means a COLOR 00-11, will write over another COLOR if it is less than or equal to that COLOR.
So:

X REVERSE PRIORITY Y equals MIN(X,Y)

For example:

10 REVERSE PRIORITY 11 equals 10
10 REVERSE PRIORITY 01 equals 01

ZGRASS has two REVERSE PRIORITY COLOR MODES 19

(01-red), and 18 (10-green). See PRIORITY WRITE.

REVERSE		19	18	
PRIORITY	00	01	10	11
00	00	00	00	00
01	00	01	01	01
10	00	01	10	10
11	00	01	10	11

RS232(NUMBER)
Esoteric Function

returns the INTEGER value of the RS232 PORT indicated by NUMBER. If NUMBER==0, the terminal port is read, if NUMBER==1, the accessory RS232 PORT is read. 0 is returned if no character is at the PORT.
Examples:

```
     GETAKEY=[PRINT "PRESS A NUMBER KEY"
     A=RS232(0)
     IF A==0,SK -1
     IF A>47&&A<58,PRINT "YOU PRESSED THE ",A-48,"
     KEY"]
```

(Note: this will not work well in .B or .F MACROS because key presses are automatically sent to normal (or"calculator mode").

```
     ANYBODYTHERE=[A=RS232(1)
     IF A#0,PRINT "WAKEUP"]
```

This will print "WAKEUP" if a device or other computer is trying to talk to ZGRASS over the accessory PORT. Note: since 0 indicates no character, you cannot receive an ASCII null character. Also note that the high bit of each character is set to 0 automatically, unless you set $RS to 1. Use the PORT command for more direct conrol.

RS232 NUMBER1,NUMBER2
Esoteric Command

If NUMBER==0, then write to the terminal. If NUMBER1==1, then write to the accessory RS232 PORT. NUMBER2, a VALUE from 0-255, is written to the PORT chosen.
Example:

```
     RS232 0,7
```

will make the terminal beep. A table of ASCII values in decimal will help you with this COMMAND. See ASCII. Note: you can transmit an ASCII null

character by:
```
     RS232 1,0
```

SCALE XSCALE,YSCALE,SNAPNAME,XCENTER,YCENTER,DISPLAYMODE
Command

takes SNAPNAME and scales it on its X and Y axes using XSCALE and YSCALE, and then writes it to the screen at XCENTER,YCENTER with the specified DISPLAYMODE. The range for XSCALE and YSCALE is -128.00 to 127.00. A negative scale factor will give a mirror image. SCALing by 1 on both axes will give the original SNAP. SCALing by 0 will result in ERROR #24.
Example:
Connect up JOYSTICK #1.
```
     .LARGE
     CLEAR;TEXT 0,0,2,0,1,0,0,"FROG"
     SNAP RRR,14,5,31,10
     X=8;Y=8
     PR "PRESS TRIGGER TO SEE MORE"
     IF $T1==0,SK 0
     IF X#0,CLEAR;SCALE X,Y,RRR,0,0,0
     IF (X=X-1)>-8,Y=X,SK -3
```
A SNAP called RRR is made of the word FROG by writing it on the screen using the TEXT Command. Press the trigger to continue. The next image you see is a SCALEd version of the SNAP RRR 8 times the size of the original. By pressing the trigger, you get RRR SCALEd by 7. This will continue on from 6,5,4,...,-6,-7. By SCALing with a negative number, you can reverse your image.

SCALE.SCREEN XSCALE,YSCALE,0-15,XCENTER,YCENTER,DISPLAYMODE
Command

same as SCALE but uses contents of screen 0-15 instead of a SNAP name.

SCREEN
Idiosyncrasy

UV-1's have 16 screens of 16K bytes (320 X 201 PIXELs) each. These are known as SCREENs 0-15. $TV is set to 0 on start-up and when changed, a different 16K screen is shown on the television. $MW controls which screen the computer writes to (and reads from in the case of non-PLOP color and display modes), so you can be building an image on one screen while seeing another. If $ML (memory lock) is set to 1, $MR is used for reads and $MW for writes, thus allowing more complex screen writes.

CTRL+B resets $TV, $MW, $ML, and $MR to zero, as does RESTART.

If a disk is DLOAD'd, $MW and $MR are used modulo 4, since DLOAD uses screens 4-15. For example:

```
    DRAWSCREENS=[M=0;A=5
    1BEG $TV=M;$MW=M;CLEAR
    CIR 0,0,A,1,20;CIR 0,0,A+10,1,20;
    CIR 0,0,A+20,1,20
    A=A+10
    IF (M=M+1)#16,GOTO 1BEG]

    CYCLE=[LIM=15;N=0;S=1
    1AGAIN $TV=N
    IF (N=N+S)#LIM,GOTO 1AGAIN
    S=-S
    IF LIM==15,LIM=0;GOTO 1AGAIN
    LIM=15;GOTO 1AGAIN]

    DRAWSCREENS
    CYCLE
```

DRAWSCREENS uses $MW to write to each of the 16 screens. Setting $TV allows you to watch the screens on the TV screen as they are being drawn on. This step is optional. CYCLE uses $TV to flip through the screens.

SCROLL XCEN,YCEN,XSIZE,YSIZE,XMOV,YMOV,DISPLAYMODE,FCOLOR
Command

moves an area of the screen centered at XCEN,YCEN of XSIZE, YSIZE dimensions with DISPLAYMODE in the direction defined by XMOV,YMOV using FCOLOR to fill in the old area. FCOLOR can be any one of the 4 COLORs 0-3. For example:

```
    SIDEWAYS=[XMOVE=1;YMOVE=1
    TEXT -100,-50,3,3,1,0,1,"HELLO"
    TEXT -50,0,3,3,2,0,1,"HELLO"
    TEXT 0,50,3,3,3,0,1,"HELLO"
    1AGAIN SCROLL 0,0,320,200,XMOVE,YMOVE,0,0
    IF (XMOVE=XMOVE+1)<24,GOTO 1AGAIN]
```

SEMANTICS
Buzzword

The meaning of a COMMAND as opposed to its SYNTAX.

SHOW FONTARRAY,CHARACTER,YOFFSET,XLEFT,XRIGHT,XSIZE,YSIZE
Swap Command

puts the information concerning the character in the FONT ARRAY specified in the variables indicated. See FONT.

SHRINK XSCALE,YSCALE,NAME,XCENTER,YCENTER,XSIZE,YSIZE
Command

is like SNAP but shrinks or expands the part of the screen it is SNAPping. Only positive VALUEs for XSCALE and YSCALE will work. XSIZE and YSIZE are the dimensions of the area to be shrunk.
Example:

```
BOX 0,0,320,201,1
BOX 50,0,40,201,2
BOX 0,50,320,60,6
SHRINK .25,.3,SCREEN1,0,0,320,201
CLEAR
DISP SCREEN1,0,0,0
SHRINK .25,.3,SCREEN2,0,0,320,201
DISPLAY SCREEN2,0,0,0
CLEAR
DISP SCREEN1,-50,0,0
DISP SCREEN2,50,0,0
```

SINE(NUMBER)
Function

returns the sine of NUMBER.

SKIP NUMBER
Command

skips the given NUMBER of lines (excluding the one you are on). It transfers control by counting the NUMBER of NEXTLINE's indicated. SKIP 0 hangs in place, SKIP 2 skips the next 2 lines, SKIP -3 goes back 3 lines. SKIP 999 is the same as RETURN and SKIP -999 will get you back to the beginning of the MACRO. SKIP does not allow LABELs. Use GOTO with LABELs.
Examples:

```
SKIP 0;.GOES TO THE BEGINNING OF THIS LINE
SKIP 2;.SKIPS THE NEXT TWO LINES
SKIP -3;.GOES BACK 3 LINES
SKIP 1;.GOES TO THE NEXT LINE

TOGO=[m=10
PRINT m,"TO GO"
IF (m=m-1)>0,SKIP -1
PRINT "NO MORE"]
```

SNAP NAME,XCENTER,YCENTER,XSIZE,YSIZE
Command

takes the PIXELs in the area indicated and saves them in an ARRAY called NAME. The DISPLAY COMMAND is used to redraw the ARRAY somewhere else. The SCALE COMMAND is used to scale and redraw the ARRAY somewhere else. NAME(0) gets the XSIZE and NAME(1) gets the YSIZE for your use. Example:

```
     FLASH=[s=28
     BOX 0,0,s,s,5%8;IF (s=s-2)>2,SK 0
     SNAP ART,0,0,32,32
     DISP ART,x=x+$X1,y=y+$Y1,0;SKIP 0]
     FLASH
```

FLASH will draw some BOXes, make a 32X32 SNAP called ART, and finally allow the user to move the SNAP around on the screen using JOYSTICK #1.

NOTE: The largest square area you can SNAP in one piece is 125X125 PIXELs (or about 15625 PIXELs or 1/4 of the screen).

SNAPPED PIX
Buzzword

is a special ARRAY which contains PIXELs from an area on the screen specified by a SNAP COMMAND. See SNAP and DISPLAY.

SQRT(NUMBER)
Function

returns the square root of NUMBER.

STATUS XCENTER,YCENTER
Command

returns the X,Y COORDINATES of the current center of the screen in XCENTER,YCENTER. See WINDOW.CENTER.

STATUS LEFTX,BOTTOMY,RIGHTX,TOPY
Command

returns two X COORDINATES and two Y COORDINATES which decribe the boundaries of the current WINDOW. See WINDOW.

STOP NAME
Command

is used to selectively halt the EXECUTion of a MACRO or COMPILEd MACRO running in .B or .F mode. A MACRO/COMPILEd MACRO can stop itself or any other MACRO.

STRING
Buzzword

is a collection of characters (numbers, letters, punctuation) delimited (enclosed) by single or double quotes or balanced square '[]' or curly '{}' brackets. If you have to use a string delimiter within a STRING, make sure it is delimited by a different string delimiter or things will get very confused (most likely, it will consider the rest of your MACRO as part of the STRING). Examples:

```
"THIS IS A STRING"
"PRINT A*B*C
SKIP -1;.THIS STRING COULD BE A MACRO TOO"
[1234]
PRINT [']; .A QUOTE IN A STRING
```

STRING(NAME,NUMBER)
Esoteric Function

returns the INTEGER which represents the character in the position indicated by NUMBER. Can be used to access STRINGs as BYTE ARRAYs.
Example:

```
TYPE=[
PRINT "INPUT A STRING OF CHARACTERS"
INPUT.STR CHAR
A=0
B=STRING(CHAR,A)
IF B==0,SK -4
PRINT B,"IS ASCII FOR",ASCII(B);A=A+1;SK -2
SK -6]
```

This prints out the decimal ASCII values of the string of characters which you input and are stored in CHAR. If you input "ABC", you should get 65,66,67. The listing of characters stops when it encounters the null (INTEGER value 0) at the end of CHAR (and every STRING).

STRING NAME,NUMBER1,NUMBER2
Esoteric Command

puts NUMBER2 into the STRING "NAME" offset by the number of BYTES in NUMBER1.
Example:

```
LETTERS="ABCDE"
STRING LETTERS,3,50
PRINT LETTERS
```

will print ABC2E
Note: allowing NUMBER1 to exceed the length of the STRING can clobber innocent MEMORY and lead to software failures. You can use a STRING as a BYTE ARRAY only if you have first made it large enough

by CONCATENATION or ASSIGNMENT. This command and the ASCII command are potentially useful for communication over the accessory RS232 PORT.

STRING VARIABLE
Buzzword

is a NAME that has a STRING as its VALUE.

STRIPE STRIPENUM,0-15,LINENUM,COL0,COL1,COL2,COL3
STRIPE.OFF
Esoteric Command

used to change the left COLORMAP part way down on the screen. The STRIPENUM (range 0-15) is an index into a special 80 byte system table of 5 byte entries. The LINENUM can range from 0 to 196. It indicates how far down the screen the change should start. The colors change (50 is 1/4 down, 100 halfway, 160 is 4/5 down, etc.) The LINENUM indicates approximately where COL0 gets changed. COL1 gets changed the next video line, COL2 the next, and COL3 the next (the hardware does not support this useful function well and leaves only 11 microseconds to change each element during a scanline). COL 0-3 are numbers in the range 0-255, representing the respective colors the pixel values should show on the screen. You must leave at least eight lines between stripes. Furthermore, unless you want the screen to flash, make sure the LINENUMs get larger as the STRIPENUM gets larger. You can cancel STRIPE by STRIPE.OFF. Note that it is normal for the stripes to temporarily undo during disk access or when using the TABLET, BREAK, AND CTRL+C.
NOTE:

RESTART STRING

will clear memory without changing stripes.
Example:

```
.ALTERNATE DEMOSTRIPE
CLEAR;$L0=0;$L1=94;$L2=166;$L3=14
BOX -100,0,50,200,1
BOX 0,0,50,200,2
BOX 100,0,50,200,3
NEW0=$L0+1;.            NEW VALUE FOR $L0
NEW1=$L1-1;.            NEW VALUE FOR $L1
NEW2=$L2-1;.            NEW VALUE FOR $L2
NEW3=$L3-1;.            NEW VALUE FOR $L3
A=0;.                   STRIPE NUMBER VARIABLE
B=28;.                  LINECHNGNUM VARIABLE
STRIPE A,B,NEW0,NEW1,NEW2,NEW3
A=A+1;B=B+28
```

```
NEW0=NEW0+1;NEW1=NEW1-1
NEW2=NEW2-1;NEW3=NEW3-1
IF A<6,SK -4
```

Use STRIPE.OFF to clear STRIPEs from the screen.

SUBSTR(MYSTRING,BEGIN,END)
Esoteric Function

returns a STRING value that is the subset of MYSTRING specified by the BEGIN and END displacement values. If the END value extends beyond the end of MYSTRING, the substring simply contains all the characters of MYSTRING following BEGIN. A null string is returned if the value of BEGIN extends beyond the end of MYSTRING.
Examples:
PR SUBSTR("ABCDEF",0,2) prints out the string "ABC"
PR SUBSTR("ABCDEF",4,20) prints out the string "EF"

SWAP COMMAND or FUNCTION
Idiosyncrasy

is a COMMAND or FUNCTION written in assembly language which must first be gotten into memory from disk or tape.

SWITCH
Buzzword

is an option for COMMANDS, FUNCTIONS, and MACROS. The only switches defined for MACROs are .B and .F which cause the MACRO to be EXECUTED in the background and foreground respectively. Many COMMANDS and FUNCTIONS (INPUT, ARRAY, etc.) have SWITCHes which are given as separate entries in this glossary. SWITCHes are always preceded by the NAME they are modifying and a period.
Examples:

```
INPUT.STR SAM
ARRAY.INT FOO,123
DEATHWEAPON.B
```

SYNTAX
Buzzword

is the form of a language, its spelling, punctuation, words, etc. (Contrast with SEMANTICS.)

TABLET(X,Y)
Function

returns the X,Y values of the TABLET pen position in X and Y, and the value of the pen push (0=not pushed but on surface, 1=pushed, -1=off surface). If you have a four-button cursor, the value returned also indicates which button was pushed.
Example:
```
     PXY=[A=TABLET(X,Y)
     X=X/6;Y=Y/6
     IF A==1,BOX X,Y,4,4,3
     IF A==0,BOX X,Y,4,4,5;BOX X,Y,4,4,5
     SKIP -4]
```
This will put a blue BOX (if the pen or yellow button on the cursor is pushed) or a flashing red BOX at the pen's or cursor's current location. The X and Y range is:
```
     -1100 < X,Y < 1100
```
Divide X and Y by 6 to SCALE them to Zgrass X and Y coordinate range. Of course, any VARIABLE NAME can be used instead of X and Y. NOTE that if TABLET returns a -1, you should not rely on the values of X and Y.

TANGENT(NUMBER)
Function

returns the tangent of NUMBER.

TERMINAL
Esoteric Command

TERMINAL bypasses the keyboard and puts the CRT directly in connection with the accessory RS232 PORT so you can connect up to another computer system as a terminal. BREAK gets you back to ZGRASS.

TERMINAL ARG0,ARG1,...ARG9
Esoteric Command

allows user to specify one of three terminals with ARG0. Set ARG0 to 0 for Hazeltines, 1 for ADM3As, 2 for ACTIVs. Then, up to 9 decimal ARGUMENTS may be entered. ARG1 allows you to define an additional key for rubout outside EDIT (ESC or underscore work well). ARG2-9 specify the EDIT keys:

	Maps into:	
ARG2	CURSOR Right	08 (^H, Backspace)
ARG3	CURSOR down	09 (^I, Tab)
ARG4	CURSOR Up	010 (^J, Linefeed)
ARG5	CURSOR Left	011 (^K)

ARG6	INSERT char	94 (^)
ARG7	Delete Char	18 (^R, HOME)
ARG8	Delete Line	01 (^A, CLEAR)
ARG9	Extra for RUB	127 (RUBOUT)

Examples:

```
    SETUP=[TERMINAL 2,95,8,11,26,24,94,9,27,95]
    SETUP
```

Sets up for an ACTIV using underscore as an alterative for DEL (rubout) both in and out of EDIT. It also specifies the arrow keys for cursor left, right, up, and down in EDIT. Delete character, in this example, is TAB and delete line is ESC. You need an ASCII table for your terminal to use this command successfully.

```
    ADM3A=[TERMINAL 1,95,12,10,11,8,30,27,9,95]
    ADM3A
```

Sets up for an ADM3A.

TEXT XLEFT,YLOWER,HORSP,VERSP,FCOLOR,BCOLOR,DMODE,TSTRING, FONTARRAY1...FONTARRAYn
Swap Command

is used to generate strings of text or arbitrary figures on the TV screen. The size of the text, the styles, the colors, and spacing are all user-definable through the FONT COMMAND and the TEXT COMMAND itself.

ARGUMENT:	Description:
XLEFT	is the X COORDINATE where TSTRING is to begin.
YLOWER	is the bottom row of PIXELs on which TSTRING is to be displayed.
HORSP	is any positive or negative INTEGER or zero. It represents a constant spacing factor in PIXELs to be inserted between characters.
VERSP	is an INTEGER which signifies the number of pixels to move up (+) or down (-) on seeing a NEXTLINE in TSTRING.

FCOLOR is the foreground color of the character (0-3).

BCOLOR is the background color of the character (0-3).

DMODE is the DISPLAY MODE. Any ZGRASS DISPLAY MODE can be used.

TSTRING is the STRING to be displayed. Every character in the STRING should have been previously defined in a FONT ARRAY named in the next operand. If a character isn't found in one of the named ARRAYs, the character is ignored, and no warning is given.

FONTARRAY1,FONTARRAYn are the NAMEs of the FONT ARRAYs to be used. The ARRAYs are searched in the order given. The number of ARRAYs that can be entered is only limited by the number of characters you can type on a line. The default FONTARRAY is used if none is specified.

For example:

```
     WRITEIT=[
     X=-100;Y=50
     TEXT X,Y,3,4,1,0,0,"THIS IS A TEXT"
     TEXT X,Y-20,3,4,2,0,0,"WITH DIFFERENT COLORS"
     TEXT X,Y-40,6,4,3,0,0"AND VARIABLE SPACING"]
```

This example uses the default FONTARRAY.

TEXT.ROT 0-3,plus same arguments as TEXT
Command

0-3 specifies the rotation of the text; 0 = no rotation; 1 = 90 degrees; 2 = 180 degrees; 3 = 270 degrees. For example:

```
     ROTATETEXT=[TEXT.ROT 1,-100,-50,3,3,1,0,0,
       "TURN YOUR HEAD"
     TEXT.ROT 2,20,0,3,3,2,0,0,"AROUND"
     TEXT.ROT 3,100,60,3,3,3,0,0,"TO READ THIS!"]
```

TEXT.SPACE SPACEARRAY,plus same arguments as TEXT
Command

SPACEARRAY, a text-spacing array described in FONT, is used to affect the spacing of characters.

TEXT.SPROT 0-3, SPACEARRAY,plus same arguments as TEXT
Command

does both .SPACE and .ROT.

TIMEOUT NUMBER
Esoteric Command

wait for NUMBER/60 seconds and then return.
Example:

```
     FOO=[TIMEOUT 300
     PRINT "5 SECONDS UP"]
     FOO.F
```

Every 5 seconds "5 SECONDS UP" will be printed.
Works only with .F macros.

TRUTH TABLES
Buzzword

See AND, OR, PLOP, PRIORITY, REVERSE-PRIORITY, and XOR.

TXT X,Y,XSIZE,YSIZE,FCOLOR,BCOLOR,DISPLAYMODE,CHARSTRING
Swap Command

prints CHARSTRING on the TV screen starting at X,Y with the character size specified by XSIZE,YSIZE, in FCOLOR with BCOLOR as the background COLOR using the specified DISPLAYMODE. The smallest values for XSIZE,YSIZE are 1,1 which means that the characters will be 5 PIXELs wide and 7 PIXELs high. The largest character can take up 4K or the largest available chunk of memory.
Examples:

```
     TXT 0,0,1,1,1,0,0,"SMALLTEXT"
```

this will print "SMALLTEXT" starting at 0,0 with 5X7 characters in red (01) with a white background (00) using the PLOP DISPLAYMODE.

```
     TXT -50,-30,2,2,3,1,1,"SMALLTEXT * 2"
```

will print "SMALLTEXT * 2" starting at -50,-30 with 10X14 characters in blue (11) with a red background (01) using the XOR DISPLAYMODE.

USEMAP
Command

gives a list of NAMEs currently in use and the number of BYTEs they take up.

VALUE
Buzzword

is typically a NUMBER or STRING. PRINT will always tell you the value of a CONSTANT or a VARIABLE.

VARIABLE
Buzzword

is a NAME you can use to hold a VALUE. Any NAME in ZGRASS that can be put on the left side of a '=' is a VARIABLE and its VALUE can be varied by that ASSIGNMENT (which is why it's called a VARIABLE instead of a CONSTANT, of course). USEMAP will give you information about your VARIABLES. NOTE: VARIABLES A-Z and the DEVICE VARIABLES are built into the system and are not listed in USEMAP.

VERSION
Function

returns the VERSION number of the current ZGRASS software you have. Example:

```
     Print VERSION ()
```

WAIT NUMBER
Command

waits the specified NUMBER of seconds before continuing by doing a SKIP 0 until the time is up. Example:

```
     NEST=[A=0
     A=A+10
     BOX 0,0,A,A,7
     WAIT 2
     IF A<200,SK -3]
```

This will draw a BOX waiting approximately 2 seconds before starting another. To wait a fraction of a second, use a System Timer which counts in 1/60 seconds.

```
     TENPERSECOND=[$Z0=6
     IF $Z0#0,SK 0
     PR "XX"
     SK -3]
```

this will print "XX" ten times per second

WHATSIS(NAME)
Esoteric Swap Function

returns an INTEGER value for the type represented by the NAME.

Values:	Meaning:
2	;Null NAME
8	;STRING NAME
14	;NUMBER NAME
16	;ARRAY NAME
18	;COMPILEd MACRO NAME
20	;SWAP MODULE

Example:

```
     MACROONLY=[GETTAPE SUE
     A=WHATSIS(SUE)
     IF A#8, SK -2]
```
This will set A to SUE's type. If you PUTTAPEd a SUE that was a SNAP and a SUE that was a MACRO, waiting for A to equal 8 would allow you to skip the SNAP called SUE.

WINDOW XLEFT,YBOTTOM,XRIGHT,YTOP
Command

creates a window in the ZGRASS screen with XLEFT as the left side, YBOTTOM as the bottom side, etc. CLIPPING is done for all drawing COMMANDS. Windows are CLIPPED to the screen and use the same COORDINATE system unless changed by CENTER. WINDOW.FULL resets the WINDOW to full screen. (Screen dumps are not subject to the WINDOW command.)
Example:
```
     CLEAR
     WINDOW -40,-60,40,60
     VIEW=[BOX -160%159,-100%100,20,20,5%8
     SKIP -1]
```

WINDOW.BOX XCENTER,YCENTER,XSIZE,YSIZE
Command

is the same as WINDOW except you specify it like BOX using XCENTER,YCENTER to mark the center and XSIZE,YSIZE to specify the dimensions of the WINDOW.

WINDOW.CENTER XCOOR,YCOOR
Command

changes the center of the screen, default of which is 0,0. See STATUS.
Example:
```
     BOX 10,10,20,20,1
     WINDOW.CENTER 160,100
     BOX 10,10,20,20,1
```
WINDOW.CENTER will change the center of the screen to the lower left corner to allow displays with COORDINATES in the range:
```
     X axis 19840-20159
     Y axis 19900-20101
```
This could be useful for roaming around large databases like the map of a city. Use STATUS to find the current screen WINDOW.CENTER.

WINDOW.FULL
Command

resets the WINDOW to full screen.

WRAP XCEN,YCEN,XSIZE,YSIZE,XMOVE,YMOVE,DISPLAYMODE
Swap Command

moves an area of the screen centered at XCEN,YCEN of XSIZE,YSIZE dimensions in the direction defined by XMOVE,YMOVE around onto the originally defined area by wrapping around using the specified DISPLAYMODE. For example:

```
MOVEIT=[
A=0;B=10
BOX 0,0,B,B,20;IF (B=B+10)<100,SK 0
1BEG WRAP 0,0,320,200,A*2,-A*2,0
IF (A=A+1)#25,GOTO 1BEG]
```

WRAP AROUND
Buzzword

is the phenomena that causes OVERFLOWed VARIABLES to print as weird numbers. If a DEVICE VARIABLE OVERFLOWs at 255, 256 WRAPS AROUND to 0, 256 to 1, etc. This is the same as modulus arithmetic with base 256.

XOR
Buzzword

(also called 'exclusive or') is a LOGICAL operation used to draw PIXELs on the screen. What gets drawn is a value from 0-3 and is computed by the XOR function of what there was on the screen with what you give it to write there. The reason for this complexity is that a couple of neat tricks are made possible by XOR. First, if you draw anything on the screen with XOR (COLOR MODES 4-7) or DISPLAY a SNAPped picture element with DISPLAY MODE 1, you can erase it by simply drawing or DISPLAYing it again the same way. In other words, two XOR's is the same as nothing. Second, by setting $L3=$L2 (and $R3=$R2 if you mess with $HB), you can make anything written with COLOR 1 pass 'behind' anything written with COLOR 2 (you have to try it to believe it).

XOR table using 2 BITs:

	4	5	6	7
XOR	00	01	10	11
00	00	01	10	11
01	01	00	11	10
10	10	11	00	01
11	11	10	01	00

The XOR COLOR MODES are 4-7. The XOR DISPLAY MODES are 1,11,21,...,131,141.

XR NUMBER1,NUMBER2
Esoteric Swap Command

ZGRASS uses three stacks to manage subroutining. Normally, you can pass about 80 ARGUMENTS and go about 14 levels deep before running out of stack space. The XR COMMAND allows you to reorganize some of your RAM for stack space. NUMBER1 indicates the total number of ARGUMENTS you wish to pass. 800 is the maximum for NUMBER1. NUMBER2 is the level to which you want to nest RECURSIVE subroutines. 120 is the maximum for NUMBER2. The largest values for NUMBER1 and NUMBER2 use up quite a bit of MEMORY. If your RECURSIVE routines exceed the limits you set using XR, the system will most likely print an ERROR message relating to the effects of wanton memory writes. There isn't any ERROR-checking on stack overflows in this mode. If you need more stack space, you can use the XR command again, but the old space is not reclaimed so it is best to RESTART first. Each MACRO/CPL invocation temporarily uses about 110 BYTEs so if you go 100 deep, 11000 additional BYTEs will be chewed up at the deepest level. So a MACRO like TOM below could easily use half your MEMORY.

Example:
```
     XR 400,100
     K=0
     TOM=[INPUT A,B,C,D;K=K+1;CORE;WAIT 1
     IF K<100,TOM 1,2,3,4]
```
Notice how CORE is decreasing as your MACRO is EXECUTing.

ZAP1(INTEGER)
Esoteric Swap Function

takes the INTEGER as an ADDRESS and returns a 8-BIT value.
Example:

```
     DUMP=[A=0
     PRINT ZAP1(A)
     A=A+1
     IF A<32767,SKIP -2]
```

This will print a decimal dump of the ZGRASS code for anyone who is into machine code disassembling.

ZAP1 INTEGER1,INTEGER2
Esoteric Swap Command

puts INTEGER2 (8-BIT VALUE) into the space addressed by INTEGER1.
* Be careful, this command can wipe out the system.

ZAP2(INTEGER)
Esoteric Swap Function

takes the INTEGER as an ADDRESS and returns a 16 BIT value.
Example:

```
     PRINT ZAP2(1)
```

prints out the first location the code jumps to on RESTART.

ZAP2 INTEGER1,INTEGER2
Esoteric Swap Command

puts INTEGER2 into the 16-bit area ADDRESSed by INTEGER1.

DATAMAX UV-1
Zgrass LESSONS

October 27, 1981

DATAMAX UV-1 Zgrass LESSONS

October 27, 1981

LESSON 0 READ ME FIRST

Zgrass is a graphics programming language. It is probably closer to BASIC than any other language, yet it is much more flexible and general than BASIC. The presumption in these lessons is that you already know how to program BASIC (at least BALLY BASIC) and are familiar with loops, IF's, GOTO's, variables, and so on, and are ready to learn what makes Zgrass tick. The essential differences between Zgrass and BASIC are:

1. Zgrass allows any number of programs and subroutines, each named, and they can run in series or parallel. BASIC has one unnamed program and a lot of GOSUB's.

2. Zgrass has an interactive full-screen editor. BASIC edits with line numbers.

3. Zgrass has good ways of passing arguments to subroutines; BASIC has none.

4. Zgrass can construct programs and run them with string manipulation features; BASIC cannot.

5. Zgrass has excellent debugging aids: single step, run-time listing, and error trapping; BASIC doesn't.

6. Zgrass has fast, advanced graphics commands; most BASICs use peek and poke.

7. BASIC has FOR/NEXT. Zgrass does without.

In order to learn Zgrass, you will have to explore it. Fortunately, this is not hard and is very rewarding. These lessons are to help you start exploring. They don't teach you how to program or write games, they just present the tools to you and encourage you to build your own. The first six lessons concentrate on defining the playing field. Pay close attention!

If you find a word being used that you do not understand, consult the Glossary. Once you get through the lessons, read the Glossary in detail. You'll find yourself understanding some of the esoterica. Some of the advanced features you may never use or understand and it may take awhile for you to see why some of the diversions from BASIC were necessary. Before long, however, you will find going back to BASIC unbearable.

Just to get you started, there's a test program called NB in the system. Press the red RST on the UV-1 front panel and answer Y. Then type the two letters NB and press the RETURN key. An image will appear. Type NB and press RETURN again. The image will undo itself. For more action, type NB.B and press RETURN. You can stop this by pressing the CTRL (called CONTROL on some keyboards) key, holding it down and simultaneously pressing the C key. Have fun!

End of Lesson 0.

LESSON 1 GETTING STARTED

When Zgrass first starts up, you see a '>' on the terminal screen. This is the "attention mark" and it means Zgrass is waiting for you to type something. To make sure it's listening, press the RETURN key. It should put an attention mark on the next line (if not, push the RST button on the UV-1 front panel). Whenever there is an attention mark, you can type a COMMAND.

Aside from graphics output, the primary means of communication from Zgrass to you is the PRINT COMMAND. It is your window into Zgrass. Whenever you what to find out what something will evaluate to, type in PRINT plus that thing. Separate PRINT and the thing by a single space and press the RETURN key to end the command. (The RETURN key performs the same function as the GO key in BALLY BASIC, you might observe.) Unless you've already pressed the RETURN key, mistakes can be corrected by typing RUB to erase and re-typing correctly. Try these:

```
        PRINT 5
        PRINT 5*5
        PRINT 5*2+2
        PRINT 5*(2+2)
        PRINT (5*2)+3
        PRINT 100/3
```

The above examples illustrate using Zgrass as an overweight pocket calculator. When doing arithmetic in Zgrass, you must observe the PRECEDENCE of OPERATORS. The OPERATORS above are +,* and /. To discover the other operators in Zgrass and their precedence, look up both OPERATOR and PRECEDENCE in the Glossary now.

Zgrass has some operators that don't exist in BASIC. The random number operator is the percent sign: %. It takes the two numbers on either side of it and yields (returns) a number randomly chosen between them. The lower bound is sometimes chosen but the upper never is, although it can get very close. Try:

```
    PRINT 1%5
```

several times. Notice that you get fractions.

Another operator in Zgrass is the ASSIGNMENT operator, "=". You can type:

```
    PRINT A=10%100
```

and the number randomly chosen between 10 and 100 will be printed and stored in VARIABLE A. Normally you do not want to see the printout everytime you store something in a variable so you leave out the PRINT:

```
    A=10%100
```

Zgrass does not like extraneous spaces, except at the beginning of a line, so

```
    A = 10%100
```

will generate ERROR #20. Spaces on the end of a line are tough to see because you can't tell where the NEXTLINE character is. If you press CTRL down and press Y at the same time (hereafter referred to as CTRL+Y) an "!" will be printed where the NEXTLINE's are so you can see them. Another CTRL+Y turns this feature off, so it is called a toggle after its similarity to toggle light switches.

In any event, you can always find out the current value of A by typing:

```
    PRINT A
```

The concept of an EXPRESSION is central to Zgrass. All the things you have typed following the PRINT's above are expressions. PRINT always gives the value of an expression. The smallest expression is a single number or variable and larger expressions are made up of smaller expressions glued together with operators. In fact, even "PRINT 10" is an expression, as is every legal thing you can type in ZGRASS besides CTRL characters. You can verify that "PRINT 10" is an expression by:

```
    PRINT PRINT 10
```

which will print a 10 then a 1. The one is the value of the expression "PRINT 10" which the leftmost PRINT gives. It prints a 1 because the PRINT command and all other commands which have nothing more meaningful to evaluate to give 1's to indicate "success".

Evaluating to a value is often referred to as "returning" a value depending on the context. In these lessons, we will talk about returning values, which should never be confused with the RETURN key. To avoid confusion between the RETURN command which returns values and the RETURN key, we normally call the character generated by the RETURN key a "NEXTLINE." That's why we say CTRL+Y puts "!"'s where NEXTLINES are instead of where RETURN's are. (Many people refer to NEXTLINES as "carriage returns" which makes no sense whatsoever on a cathode ray tube terminal.)

To drive home the point of returning values which PRINT prints out, try:

```
    PRINT PRINT PRINT 10
```

Any Zgrass command can have its expressions put in parentheses instead. Try:

```
    PRINT(10)
    PRINT(PRINT(PRINT(10)))
```

Note the lack of spaces. The 10 is called an "argument" by computer folk and is an indication that computer languages were developed by mathematicians, not social psychologists or artists. At any rate, arguments are always separated by commas:

```
    PRINT 10,20,30
    PRINT(10,20,30)
```

The parentheses are used to clarify the nesting. The only reason we do not require them always is that they are a pain to type all the time (if we did require parentheses, by the way, you would be able to type spaces anywhere, but we don't so you can't). People seem to have an inordinate amount of trouble accepting two different formats for enclosing arguments to commands. Please make sure you re-read the above few paragraphs until they are clear to you.

Try this last PRINT evaluation:

```
    PRINT(PRINT(10)+PRINT(5))
```

Each print inside the parentheses returns a 1 after printing its argument and the sum of the two 1's is printed by the leftmost PRINT which prints 2. (The 1 it returns is lost because it is not assigned or passed on to anything.)

There are, of course, more simple ways to arrive at one and one are two, but understanding the connections between arguments, commands and values returned is critical to developing a feel for zooming around in Zgrass and seeing how Zgrass is far more powerful than BASIC.

WHAT COMMANDS DO

In addition to printing 10 on the terminal, PRINT 10 returns a 1 to whatever called it because it had nothing better to return than "success." That's reasonable because it did its work as a side effect of returning that 1. Commands generally are interesting because of what they do rather than what they return. For example, type:

```
    HELP
```

Look at the last line. Now type (make sure there's no space between the parentheses):

```
    PRINT HELP()
```

and you'll see a 1 there. The point of this is that commands cause other things to happen besides returning a value. FUNCTIONs are commands that return a value but otherwize don't change anything. It is possible to have expressions which contain both commands and functions since the distinction is, in essence, artificial, but such constructions are confusing, especially to someone else trying to read your programs.

Functions are generally used as parts of complex expressions and were invented because once you get past about ten operators, you simultaneously run out of punctuation symbols and the capacity to remember what they do. Functions have names like commands (you are probably familiar with SIN, COS, TAN, LOG, and SQRT, for example) and you can create your own functions and commands in Zgrass quite easily. We'll show you how in the next lesson.

Meanwhile, let's do some graphics commands. Try:

```
    BOX 0,0,300,200,3
    BOX 75,0,20,100,2
    BOX -75,0,20,100,1
    BOX 0,0,200,10,0
```

The BOX command draws a filled-in rectangle defined by its arguments:

```
    BOX XCENTER,YCENTER,XSIZE,YSIZE,COLORMODE
```

X and Y are the horizontal and vertical coordinates and have the ranges:

```
    -160<X<159
    -100<Y<100
```

The last argument to BOX is the COLORMODE. There are lots of these, but for now, assume 0 is white, 1 is red, 2 is green and 3 is blue.

The POINT command allows you to draw points on the TV screen. An individual point (from now on called a "pixel" to avoid confusion with the POINT command) is sometimes hard to see, especially if you are using a regular TV. Try:

```
    BOX 0,0,320,200,3;POINT 100,20,0
```

Look up the POINT command if the arguments are not obvious to you.

While looking up the POINT command, you might have noticed a POINT function,too. When the COLORMODE is not specified, Zgrass assumes you want it returned to you as a value. The values range 0-3 if the coordinate is on-screen and -1 is returned if the pixel specified is off-screen. Try:

```
    CLEAR
    BOX 0,0,100,100,3
    BOX 0,0,50,50,2
    POINT 0,0,1
```

and:

```
    PRINT POI(0,0),POI(0,1),POI(30,-30),POI(200,200)
```

So, POINT used as a function only returns a value while POINT used as a command changes the screen

One last command for this lesson: CLEAR. CLEAR erases the TV screen. CLEAR.CRT (or CL.C for short) erases the terminal screen. The ".CRT" is called a switch. Switches are used to modify some system commands and are separately documented in the Glossary.

End of Lesson 1.

LESSON 2 WRITING MACROS

It's pretty difficult to find a programming language that doesn't deal with numbers coherently. However, the use and handling of alphanumeric text (called "character strings") so fundamental to our natural language communication, is, in fact, a clumsy add-on to most popular programming languages. Only the most primitive and ad-hoc constructs are available to the user in BASIC or FORTRAN, for example. Zgrass, on the other hand, uses character strings as its way of building and storing user programs so string manipulation is as much a part of the language as numerical computation.

STRINGs (look up the definition in the Glossary if it's a new concept to you) are defined much the same as numbers except you need to specify delimiters (special punctuation) which first, indicate that they are strings and second, say where they end. The string delimiters in Zgrass are:

```
     ",',[,],{ and }
```

the last four of which can be nested. Try:

```
    PRINT "HELLO!"
```

or

```
    PRINT [HELLO!]
```

To assign a string to a variable, type:

```
    A=[HELLO!]
```

and then try:

```
    PRINT A,A,A,A,A
```

Unlike in BASIC, you do not need to use a '$' to indicate a string variable; Zgrass is pretty good at figuring it out. You may use '$' in a name if you find it comfortable. For the time being, avoid names that start with a $ and have exactly two additional characters to eliminate possible conflict with DEVICE VARIABLE names (see LESSON 7 and the Glossary). Also, do not use $CHAR1 as a name, since that is used by the TEXT command.

Note that there are spaces between the HELLO!'s when printed as above. PRINT always puts a space after the thing it prints. To eliminate the spaces, try:

```
    PRINT A&A&A&A&A
```

The '&' is the concatenation operator and it creates a string expression.

As stated before, Zgrass stores your programs as strings. We call programs in Zgrass "MACROs." So, to create a macro, you assign the string that contains the commands to a variable. Try:

```
    Q=[BOX 0,0,100,100,2
    BOX 50,50,100,100,3]
```

(Note the plus sign (+) which appears when a multi-line string is being typed and Zgrass is waiting for you to type the matching delimiter.)

To run this macro, just type Q and press the RETURN key.

You should wonder why the letter Q was used instead of A. If you use A as a name above, you will get ERROR #39 (try it). The reason is very important and the result of a great design difference between Zgrass and other languages: abbreviation. Abbreviation allows you to interact with Zgrass faster than if you had to spell everything out all the time.

What you are doing in assigning the string to Q is creating a macro, as we said. You can also think of it as creating your own command named Q. There aren't any system commands starting with Q so Zgrass can tell right away that it is your command you want to run. Since Zgrass allows you to abbreviate, you will notice that H is enough for HELP, P for PRINT, etc. (HELP will tell you what the first command in each alphabetical grouping is and you can always get at it with its first letter.) Since H by itself always gets to HELP, you cannot have a macro named H and expect to be able to run it. You can have a string variable or number variable named H without any problems, but you can't run it if it's supposed to be a macro.

A few examples might help clarify:

```
    P=100
    PRINT P
    P P
    P P 10
```

The PRINT P and P P are the same. However, P P 10 is the same as PRINT PRINT 10 or PRINT(PRINT(10)). It's that innocent little space between the P and the 10 that causes Zgrass to look for commands instead of variables. And it always looks at system commands first. Watch out for extraneous spaces!

Why all this confusion? It's to help you avoid lots of typing. You can create a macro like this one:

```
 QUANTIZED$SECOND$DEGREE$SPACE$INVADERS=[BOX 0,0,100,100,3]
```

and run it by typing Q, assuming it's your only thing starting with a Q. Even if you are a good typist, you will appreciate abbreviation once you get used to it. HELP can tell you what the system names are, and you can avoid abbreviations of them for macro names. If you do name a macro BO, which is a abbreviation for BOX, just assign it to another name:

```
    MYBO=BO
```

and run MYBO.

Particularly error-prone names are WAIT, TIME, GET, PUT, and ONE. Use HELP to figure out what commands these are abbreviations for.

The command USEMAP lists all the names currently in use. Once you start using names longer than one letter, they show up in the USEMAP command. Try the following:

```
    ABC=10
    DEF=[STRINGYTHINGY]
    USEMAP
```

DELETE will remove them:

```
    DEL ABC
    USEMAP
```

Let's do a simple macro:

```
    MANYBOXES=[CLEAR
    A=300
    10 A=A-3
    IF A>0,BOX 0,0,A,A,A\4;GOTO 10]
```

Run MANYBOXES. Notice that a comma is used to delimit the conditional part of the IF command. The semi-colon allows a second command (GOTO) to be in the scope of the IF command. The '\' is the modulus operator and assures the COLORMODE will not exceed 3.

If you are a terrible typist and cannot get this program in straight, persevere. The next lesson will show a much more humane way to create macros. Line numbers are used only as labels for GOTO's and have nothing to do with line ordering. In fact, GOTO labels can have letters too, as long as they start with a number. 1AGAIN, 3DEATH, 4TEEN are all legal labels.Another example:

```
        RANDOMCROSSES=[CLEAR
        1MORE X=-160%159
        Y=-100%100
        LINE X-10,Y,4
        LINE X+10,Y,3
        LINE X,Y-10,4
        LINE X,Y+10,3
        GOTO 1MORE]
```

This macro puts little crosses on the screen until you press CTRL+C or BREAK. The 4 as colormode above places the starting point of the line to be drawn each time. See the LINE command in the Glossary.

Note that if you like to have your programs formatted so that labels are at the left edge and commands indented a little, you can use spaces as the first characters of the lines you choose. You cannot use tabs, however. Use of spaces does chew up memory, but you can do it.

Zgrass encourages you to write several small macros rather than one large program as BASIC requires you to do. Small macros are easier to configure as software tools. In case you are tempted to write enormous programs, Zgrass, for internal reasons, limits you to 99 SKIPS, GOTOS and IFS in a single macro. Macros cannot exceed 4000 characters either. Even the EDIT command (next lesson) works best on small programs. So break your bad BASIC habits!

End of Lesson 2.

LESSON 3 EDITING

Editing is the act of creating and changing a character string. In the process of developing a complex graphics simulation, for example, you do a lot of editing. In BASIC, editing is done with line numbers. Zgrass allows you to roam around a page of text on your terminal screen with the EDIT command.

To start, type:

```
    EDIT TEST
```

The screen should clear. Just start typing the following:

```
    PR "THIS IS LINE 1"
    PR "THIS IS LINE 2"
    PR "THIS IS LINE 3"
```

Note that you do not put brackets or other string delimiters in. EDIT does that automatically.

You move the cursor with the arrow keys. Now practice getting the cursor under any character you choose.

Change the quotes in the first line to single quotes by positioning the cursor under them one at a time and typing a single quote each time. You can change any character by typing over it.

Now press RETURN. There's an open line now. Type:

```
            THIS IS THE NEW LINE 2
```

Move the cursor down and change old line 2 to line 3, etc.

To change the 1 in the first line to ONE, position the cursor under the 1 and type ONE. You have wiped out the end " though, so type it in again. It's easy to type characters at the end of a line.

To change it to "THIS IS NOT LINE ONE", position the cursor under the L, press the HOME key and type NOT plus a space. Move the cursor down a line to get out of insert mode.

To delete the word NOT, position the cursor under the N and press ESC three times. Type it one time more to delete the extra space.

Some other EDIT functions:

-To delete a whole line, press the TAB key.
-To exit from EDIT, CTRL+E.
-To exit but ignore all changes made in EDIT, press BREAK.
-To insert before the first line, press the HOME key
 and type away.
-To insert a NEXTLINE in the middle of a line,
 press the HOME key first.

EDIT also allows you to move and copy lines or parts of lines. In order to do this, you must set two pointers, one indicating the start and one past the last character to be copied or moved by positioning the cursor and typing CTRL+S each time. A single quote will appear temporarily to indicate the pointer. (If you make a mistake, type CTRL+T to erase the pointers.) Then position the cursor again to where you want the text to go and type CTRL+D to move the text there or CTRL+F to copy it there. If you confuse things horribly, press BREAK and try EDIT again.

After a while, using EDIT will become easy and you will be forever spoiled.

End of Lesson 3.

LESSON 4 MORE ON MACROS

All the macros in this lesson should be typed in with EDIT. Remember to only type the characters between the square brackets when in EDIT. Try:

```
        TRIANGLES=[CLEAR
        PRINT "LOOK AT THE TV!"
        SIZE=0
        1UP SIZE=SIZE+2
        LINE 0,SIZE*2,4
        LINE -SIZE,-SIZE,1
        LINE SIZE,-SIZE,2
        LINE 0,SIZE*2,3
        IF SIZE<50,GOTO 1UP]
```

Run TRIANGLES. If there are any errors, check your macro carefully and EDIT it.

You can speed up TRIANGLES a bit by COMPILing it:

```
COMPILE TRIANGLES,CT
```

and then run CT as you would any macro or command.

For too many reasons to go into here, Zgrass has no FOR/NEXT construct. You must build loops out of IF's and GOTO's, explicitly testing conditions. The iteration and testing step can be combined, if you prefer:

```
IF (SIZE=SIZE+2)<50,GOTO 1UP)
```

although this would necessitate SIZE to be initially set to 2 instead of 0, and the removal of the "SIZE=SIZE+2" in the line starting with 1UP, in this particular case. It's ok to have a label without anything following it, by the way.

You should be wondering about that assignment inside the IF. It's possible because Zgrass does not use the same operator for assignment and logical equals as BASIC does. Zgrass uses '==' (double equals) for logical equality testing. It's another minor deviation from BASIC to get used to. Note its use:

```
        COUNTDOWN=[A=10
        1LESS PRINT A
        A=A-1
        IF A==5,PR "HALFWAY THERE"
        IF A>0,GOTO 1LESS
        PRINT "BLASTOFF"]
```

Run this one and then edit it so the A==5 is A=5. The loop will never end since A is continually assigned 5. To further clarify:

```
        PRINT 5==5
        PRINT 5==6
```
So, "true" is 1 and "false" is zero.

IF takes the rest of the line if the stuff between it and the comma evaluates to non-zero. Therefore, IF 1,PRINT "HI!" would always print HI!

The next macro will draw sweeping lines of random colors forming an ellipse 300*150 pixels:
```
       SWEEP=[ANGLE=0
       CLEAR
       X=SIN(ANGLE)*150
       Y=COS(ANGLE)*75
       LINE X,Y,4
       LINE -X,-Y,1%4
       IF (ANGLE=ANGLE+2)<180,SKIP -4]
```
Note the SKIP -4 in the last line. It is a shorthand way of doing GOTO's without the need for labels. It goes back four lines. SKIP 2 would go down two lines (not skip the next two lines as you might think, that would be SKIP 3). SKIP 0 keeps executing the same line it is on.

As in BASIC, the INPUT command is used to get responses from the user of the macro:
```
       HOWMANYBOXES=[CLEAR
       PROMPT "NUMBER OF BOXES TO DRAW?"
       INPUT QUAN
       XCENTER=-160
       BOX XCENTER=XCENTER+20,0,18,18,3
       IF (QUAN=QUAN-1)>0,SKIP -1]
```
If you answer with too big a number for QUAN, boxes will be drawn off-screen.

To wrap the boxes around in a grid, try:
```
       GRIDBOXES=[CLEAR
       PROMPT "NUMBER OF BOXES TO DRAW?"
       INPUT QUAN
       YCENTER=80
       XCENTER=-160
       BOX XCENTER=XCENTER+20,YCENTER,18,18,3
       IF (QUAN=QUAN-1)==0,RETURN
       IF XCENTER==140,YCENTER=YCENTER-20;SKIP -3
       SKIP -3]
```
after running GRIDBOXES, try:
```
        GRIDBOXES 30
```
and notice that no prompt appears. If a macro is passed arguments like the 30 above, the PRINT and PROMPT commands are automatically suppressed. You can tell if there are arguments passed with the ANYARGS command, just look it up.

Now try:
```
   SINECURVE=[PROMPT "WHAT'S THE OFFSET TO BE?"
   INPUT OFFSET
   X=-160
   ANGLE=0
   POINT OFFSET+X,SIN(ANGLE)*80,3
   ANGLE=ANGLE+2
   IF (X=X+1)<159,SKIP -2]
```

with:
```
    SINECURVE 0
    SINECURVE 30
```

To show how macros can be used as subroutines for other macros, try:

```
   MANYSINES=[PROMPT "FIRST OFFSET?"
   PROMPT "LAST OFFSET?"
   PROMPT "HOW MANY?"
   INPUT FIRST,LAST,QUAN
   INCREMENT=(LAST-FIRST)/QUAN
   SINECURVE FIRST
   IF (FIRST=FIRST+INCREMENT)<=LAST,SKIP -1]
```

You can answer the questions by typing:
```
   MANYSINES
```

or specify arguments:

```
    MANYSINES 0,100,5
```

If you choose to speed up SINECURVE, you can compile it, but remember to change MANYSINES to refer to the compiled name:
```
    COMPILE SINECURVE,FASTSINE
```

As a further note, you can also input strings. Use the INPUT.STR command:
```
      NAMES=[PROMPT "WHAT'S YOUR NAME?"
      INP.STR XXXX
      PRINT "THAT'S FUNNY!, ",XXXX," IS MY NAME TOO!"]
```

End of Lesson 4.

LESSON 5 STORING MACROS ON TAPE AND DISK

With luck, by this point, you may have some macros worth saving. If you do not have a disk yet, or want to send someone a tape copy, you need to know how to use GETTAPE and PUTTAPE.

Zgrass's audio tape storage has some advanced features when compared with standard BASIC tape handling. Zgrass allows you to easily store several copies of a macro so that if an error is detected while reading it back, the next copy can be automatically retrieved. It also will print out a directory of the tape as it is looking for a file by name, if you wish.

PUTTAPE works on macros, arrays, swap modules and screen dumps. So far, you've only used macros. You cannot PUTTAPE compiled macros or number variables. Screen dumps are 16K byte blocks which are memory dumps of the screen, useful for storing pictures instead of the instructions to draw the pictures. A screen dump gets PUTTAPED when you use the .TV switch and a name.

The required cable hookups for tape storage are described in the hardware manual for the UV-1.

To store a macro on tape, you need to specify three things:

```
   PUTTAPE NUMBER,MACRONAME,[SOME DESCRIPTIVE MESSAGE]
```

where the NUMBER is the number of times to write the file out (2 or more is suggested), and the message in brackets is supposed to be descriptive so that when you look at the tape three months from now you know what it's for. You can see the number count down in binary in console lights 8-15 as the copies are being put on tape.

For example, to PUTTAPE a macro called SHIRLEY, get the tape deck ready, type in the following line but don't press the RETURN key yet:

```
   PUTTAPE 2,SHIRLEY,[SHIRLEY DRAWS PINK ELEPHANT HATS]
```

set the tape deck in RECORD with the tape moving and then press RETURN. When you get the attention mark back, stop the deck.

Since Zgrass programming usually involves creating several macros as software tools, it's a good idea to have macros for each major task that do nothing but GETTAPE and

PUTTAPE all the pieces. You cannot conveniently reclaim data space on an audio cassette, so always work in the mode of reading all the parts in, change them as necessary for debugging and write them out in order.

To GETTAPE a file named SHIRLEY, simply type:

```
    GETTAPE SHIRLEY
```

Press the RETURN key and start the audio tape up, making sure that the cables are connected, of course.

If there is an error detected in the tape read, you will see

```
    BAD-AUTO RETRY
```

and the next copy will be gotten. You can test this feature by turning the audio level down all the way during a GETTAPE for an instant.

It's actually a good idea to rewind your tape, and type GETTAPE XXXXX after you've PUTTAPED your macros, before you RESTART the system. Assuming XXXXX is not a name on the tape, GETTAPE will scan the entries, print the directory information, verify the integrity of the data (if it doesn't say BAD DATA, it's ok), and keep going. If you encounter some errors, you can re-PUTTAPE the macros again. Get out of GETTAPE or PUTTAPE by pressing the red button marked "RST" on the UV-1 front panel.

If you are thorough and methodical about saving your Zgrass macros, arrays and so on, you will not lose your temper when the power company glitches, or you find a bug in Zgrass that erases all your macros in memory.

USING THE MICROPOLIS DISK AND DGET/DPUT

The Micropolis 5" floppy disk system supplied with Zgrass units has a lot of software support. All disk commands begin with a D. and often only require two characters (for example, DU works for DUSEMAP, DG for DGET, etc.). These disks will hold approximately 180,000 bytes on each surface giving a total of over 700,000 bytes of storage on-line.

The primary difference between tape and disk is random access. Disks have diskmaps (sometimes called directories) which tell you what is on them and the system where the data is stored on the disk. Unlike audio tape, access to different parts of the disk is easy so it can jump around alot. You do not have to store stuff sequentially as you do with audio tape.

Another difference is deletion. You can update stuff simply by putting it out again. The DPUT command automatically keeps one backup for you for safety. Of course, the disk is much faster than audio tape as well.

One note: names stored on disks are generally referred to as 'files' so do not get confused by the terminology. Files are just things stored on the disk, nothing more.

The first thing you should do is insert your system disk into drive 0. You do this by gently pushing the disk in the slot, keeping the label up and the little notch to the left. Then push the lever with the blue on it down until it catches. Then type:

```
     DSETUP 0
     DUSEMAP
```

and you will see a listing of all the swap modules on your system disk. Now put a blank disk in drive 1. Type:

```
     DSETUP 1
     DINIT 200
```

This will erase the disk and then initialize it to accept a maximum of 200 names. Type DUSEMAP and you will see how many sectors (each is 512 bytes) are left.

If you get an error when trying to DINIT, the disk may not be formatted, in which case you must first DFORMAT it. See the Glossary.

Now type:

```
     DS 1  (the system starts up with disk 0 setup)
     SAM=NB
     DPUT SAM,[SAM IS A COPY OF NB]
     DUSEMAP
```

and you will see SAM in the disk map. If you RESTART then type:

```
     DGET SAM
     USEMAP
```

you will see SAM in memory.

If you DPUT SAM and then type DUSEMAP, you will see SAM and a backup copy of SAM (type is listed as BAK). The second and subsequent times you DPUT something you do not have to include a message (the stuff in square brackets) unless you want to change the message. DPUT always maintains one and only one BAK.

You can DGET a BAK with DGET.BAK NAME.

You can delete a name on the disk with DDELETE NAME. If you want to save space on the disk, you can get rid of BAKs with DDELETE.BAK NAME. There is also a command DBAKS which removes all BAKs from the disk. You only need to remove BAKs if you need the space, which is not normally the case.

The reverse side of disk 0 is called disk 4, so you use DSETUP 4 to get at it. Of course, it must have been DINIT'd at some previous time. The reverse of disk 1 is called disk 5. If you have two Micropolis dual drives, you get disks 2,3 and 6,7 as well.

The Glossary has descriptions of the rest of the disk commands. One concept that has to be explained in detail, though, is the submap. Submaps are essentially little disk maps which you can use to partition your disk into areas specific to individual projects you are working on. Once you are working within a submap, all disk commands reference only that submap and cannot get from or put into any other submap. The only exception to the rule is that if DGET cannot find a name within the submap, it will go look through the regular diskmap (but not any other submaps) for the name, so you can DGET swap modules and any other 'tools' you often use that are stored in the regular disk map.

The way you get into a submap the first time is to type:

```
     DCREATE SUBMAPNAME,[MESSAGE]
```

For example, try:

```
     DCREATE PAINT,[PAINT PROGRAM SUBMAP]
     DUSEMAP
```

Nothing is there at present. If you DPUT a name, it will show up with DUSEMAP. To get back to the regular disk map, type:

```
     DSETUP n
```

where n is the disk you're using. To get back to the submap PAINT, type:

```
     DSETUP n,PAINT
```

The regular disk map holds entries for all the submaps so you can tell what their names are. You can actually store things under a submapname without having created the submap with DCREATE but you will not know they are there unless you possess an extraordinary memory. You have to pay attention when you change disks, in particular, so you do not DPUT into submaps which have not been DCREATE'd on that disk. It would take an enormous amount of overhead to have Zgrass check each time for whether the submap has been created.

DDSMAP, a swap module, will remove a submap and all its entries.

The disk commands turn off the rest of the system while operating so you cannot type ahead as you normally can.

There are also two other artifacts: the stripe command temporarily freezes and the system timer device variables are suspended during disk access time. Control characters are not listened to either, so if you want to use CTRL+W with DUSEMAP, make sure you press the keys before the command starts going.

You should also check out DLOAD.

Please read about the other disk commands in the Glossary at your leisure.

End of Lesson 5.

LESSON 6 DEBUGGING

This lesson is about what to do when your program doesn't do what you expect it to do. There are several classes of reasons for unexpected behavior. The first class involves SYNTAX errors.

Syntax errors are essentially errors in spelling, punctuation, abbreviation or specifying arguments to a command. Syntax errors can always be fixed with EDIT. Examples of syntax errors are:

Is this an ERROR

Spelling errors:
```
    HILP instead of HELP
    help instead of HELP   (is caplock key on?)
    POINT instead of POINT  (zero instead of O)
```
Punctuation errors:
```
    POINT 14;30,4  (';' should be a ',')
    SKIP-2          (needs a space)
    POINT (14,30,2) (if parentheses, no space allowed)
    TEST="PRINT '"'"  (double quote inside double quotes)
    IF A=10,PRINT "huh?" (this always prints. Use '==')
```
Be particularly careful to match parentheses and brackets. Quotes, single and double, cannot be nested like brackets and parentheses can. Although you might at first be more comfortable with quotes from experience with BASIC, strings defined with brackets (both curly and square) seem easier to locate when you're debugging.

RECKLESS ABBREVIATION:

```
    DOODLE CR1,CR2,CR3,CR4,CR5,CR6,CR7,CR8,CR9,CR10,CR11
```
will cause confusion because CR1 is an abbreviation for CR10 and CR11 as well as, in this case, probably a variable name itself. Use names like CR01 if you have a names like CR10.
```
    SIN FIRST,ORIGINAL
```
SIN is a system function so you can't use it for a macro name. If you get into this situation, reassign the macro name by typing:
```
    SIN1=SIN
```
You can delete SIN then, if you want to clean things up. (Astute note: since you created SIN1 after SIN, the reference to SIN in DELETE SIN will find the first SIN, not SIN1. If you typed DELETE SIN again, it would get rid of SIN1, of course. Since there is no way to pass system command names as arguments, you can't actually delete the SINE command, even if you try.)

MISSING ARGUMENTS:

Often you omit something a command wants as an argument. HELP gives you the arguments in brief for each system command and function, and the Glossary gives information in detail. Sometimes unbalanced parentheses or brackets will confuse the argument scanner, other times it's a misplaced comma or semi-colon. Proofreading is essential in Zgrass since there is not nearly the redundancy present in English or BASIC.

OTHER TYPES OF ERRORS:

There are three other types of errors you will encounter: logic errors, incorrect assumptions, and running out of space and time. We will discuss all of these in detail.

Incorrect assumptions arise from poor documentation or instruction on our part and bad guessing by you. It is impossible to describe everything a programming language does or can do (which is why they're so intriguing, of course) so the best we can do is get you to a level at which you can tell whether it is your problem or a shortcoming of Zgrass when the unexpected happens.

DEBUGGING STEPS:

First, check for syntax errors. If there are none apparent, check that the commands you are using actually do what you expect them to do. If possible, use the command outside a macro in its simplest form. Use numbers instead of expressions since the error will often be in your expression, not in the command, something you should discover as soon as possible.

Always debug in interpretive mode (non-compiled). Anything that works as a macro should work as a compiled macro (if it doesn't, it's our fault unless it's documented).

Macros which make you unhappy do so because they generate errors or don't do what they're supposed to. The former are easier to detect because Zgrass points them out and gives you an error number which you can look up in the Glossary. You can list the program as it is executing with CTRL+X and you will see the last lines executed before the error is generated. You can print the values of variables and see what's wrong. A second CTRL+X cancels list mode, as will pressing CTRL+C or BREAK.

If you are still having trouble, once you get to the part of the program causing the error, press CTRL+D. This puts you into single step mode. Each time Zgrass sees a NEXTLINE or semi-colon, it prints a '#' and acts like it's in attention mode. You can type commands to see what's in variables, and/or press RETURN to continue one step at a time. A second CTRL+D plus a RETURN gets you out of single step.

There is also a once-only CTRL+D. You get it by pressing CTRL+Z. A RETURN key press gets you back into the macro.

Since it is easy to EDIT in Zgrass, put PRINT's in crucial places to test your variables in a loop. You can then use the print control characters (CTRL+O, CTRL+Q) to alter execution or printing of the variables. CTRL+O will suppress output to the terminal but otherwize allow the macro to execute so if you put a PRINT in a loop and don't want to see it all the time, type CTRL+O to turn printing off, another one to resume printing. CTRL+Q stops not only the printing but the execution as well when your macro is trying to print. It's good for waiting while you scratch your head trying to figure out the problem. Another CTRL+Q will resume printing. You can't type anything when stopped with a CTRL+Q, by the way, as you can in CTRL+Z or CTRL+D modes.

Block mode →

CTRL+W is also useful for seeing a terminal screen-full of information at a time. Press the RETURN key to get 20 lines of text at a time after you press CTRL+W. Another CTRL+W will get you out of this mode. (We used CTRL+W in Lesson 1 to page through the HELP command.)

SPACE AND TIME PROBLEMS:

Zgrass has a lot of memory for a small system. If you use a lot of arrays, system swap modules, or many, many large macros, you can run out of space. The only recourse is to delete some of the stuff that's taking up all the space. You may have to design some graphics sequences to GETTAPE or DGET parts while running, deleting stuff that is no longer needed.

Getting things to run faster is the real problem, though. The compiler speeds things up, of course. If you're really pressed for time, try modifying your macros to

do less computation, use less array
references, etc. Games programmers get paid very well for their cleverness.

You now have the tools necessary to debug your programs. It still isn't easy, but it's most of what we call programming.

End of Lesson 6.

LESSON 7 DEVICE VARIABLES AND PORTS

Devices (also sometimes called "peripherals") are hardware gadgets that hang off the computer system and do the communication with humans or other computers. So far you've communicated with Zgrass by typing on a terminal and it talked back with graphics on the TV and characters on the terminal. There are several other devices available in Zgrass some of which you access via DEVICE VARIABLES and others you read/write via PORTs. Ports are a more primitive way of accessing the hardware. Device variables have been provided for hardware features that share ports in a complex way and for timing-related software/hardware programming. Simpler devices (like the LED lights and switches, for instance) can be set or read easily by the PORT command so the overhead in providing device variables for them is not justified.

Device variables do not show up in USEMAP. They all start with a dollar sign and have two more letters.
You can have your own variable names with dollar signs as the first character, so you might want to always have three or more characters following the dollar sign to avoid conflict with device variables. Many device variables are acted upon by the system every 1/60 second.

For instance, find a joystick and plug it into the leftmost joystick hole in the front panel. There are four device variables which report the values received at each hole sixty times a second. For hole 1 (notice that the number on top of the joystick knob only corresponds if you take the care to plug it into the correct hole), the device variables are as follows:

```
    $T1 is zero if the trigger is out, 1 if in.
    $K1 ranges from -128 to +127 when you turn the knob as
        you would a volume control.

    $X1 is 0 if the knob is in the center position,
           1 if the knob is pushed to the right and
          -1 if the knob is pushed to the left
    $Y1 is 0 if the knob is in the center position,
           1 if the knob is pushed away from you and
          -1 if the knob is pulled toward you
```

assuming you hold the joystick with the trigger pointing away from you.

The easiest way to discover the values of a device variable is to print it in a loop while changing its values:

```
    RANGE=[PRINT $T1;SKIP 0]
```

and so on.

Let's write a simple drawing routine which puts a point down whenever the trigger is pulled in:

```
    DRAW=[IF $T1==1,POINT X=X+$X1,Y=Y+$Y1,3
    SKIP -1]
```

This program needs improvement, though. You can't tell where the point is going to be unless the trigger is in. You also can't erase any points put down in error. Try:

```
    DRAW=[POINT X=X+$X1,Y=Y+$Y1,3
    IF $T1==0,POINT X,Y,0
    SKIP -2]
```

This DRAW will erase the point you see at the current accumulated X,Y position unless the trigger is in. It's still pretty hard to see the dot, though, so let's put a cursor in:

```
    DRAW=[K=0
    X=X+$X1;Y=Y+$Y1
        BOX X,Y,1,20,5
        BOX X,Y,20,1,5
    IF (K=K+1)\2==1,SKIP -2
    IF $T1==1,BOX X,Y,3,3,$K1/64+2
    SKIP -5]
```

This DRAW uses several tricks. First, COLORMODE 5 is used with LINE to allow the red lines to flash without overwriting already drawn blue points. If we used COLORMODE 1, the red lines would erase the blue points. Secondly, COLORMODE 5 applied twice erases the red lines. Color modes 4-7 are done by special hardware in the system and will be discussed in detail in the next lesson. The four LINE commands are done twice by the testing of K modulo 2 which evaluates to 0 or 1 each time. Finally, the $K1/64+2 evaluates to 0,1,2, or 3 and sets the COLORMODE of the point drawn so you can erase with 0 or draw with red, green or blue by turning the knob.

It's less easy to figure out how to use device variables you set instead of read. The hue and brightness associated with the four colors on the screen are set by Zgrass sixty times a second based on what values you have stored in $L0, $L1, $L2 and $L3. Print out the values of these variables:

```
    PRINT $L0,$L1,$L2,$L3
```

Zgrass automatically puts these values in $L0-$L3 when you restart the system or type CTRL+B.

The colors are arranged in 8 brightness levels of 32 colors yielding a choice of 256 colors. Setting $L0 to 127, for example, turns the screen to yellow. Try:

```
    COLORS=[$L0=$L0+$X1;SKIP 0]
```

Let's make finding your favorite colors easier:

```
    CHOOSY=[PRINT "MOVE THE KNOB SIDE TO SIDE
       TO CHANGE COLOR
    AND TURN IT TO CHANGE BRIGHTNESS
    PULL TRIGGER TO PRINT VALUE"
    A=0
        A=A+$X1*8;IF A<0,A=0
        IF A>248,A=248
        $L0=A+$K1/32+4
    IF $T1==1,PRINT $L0
    SKIP -4]
```

The $K1/32+4 evaluates to a number in the range of 0-7 plus a fraction. Since $L0 is used as an integer value, the fractional part is tossed away.

Now, look in the Glossary under DEVICE VARIABLES. Of primary interest are $HB and $BC. Try setting them to numbers within their ranges and see the results. When you set $HB to 20, for example, the screen now shows the right half colors $R0-$R3. Draw a set of color bars:

```
     CBARS=[CL;A=-149;C=0;$HB=21
     $R0=0;$R1=82;$R2=43;$R3=249
     $L0=7;$L1=213;$L2=126;$L3=164
     IF A<115,BOX A=A+45,0,46,202,C=(C+1)\3+1;SKIP 0]
```

You can easily make a pocket watch out of Zgrass with the system time device variables:

```
    CLOCK=[PRINT "INPUT HOUR,MINUTE,SECOND"
    INPUT $HR,$MN,$SC
    PRINT $HR,$MN,$SC;SKIP 0]
```

There are also ten system timers $Z0-$Z9 which you can use to control things over time. Many device variables are used for setting options in the software. See the Glossary for complete details. $RD, for example, if 0, sets the system to use degrees for angle movement; if 1, radians are used. All device variables except the $L0-$R3 and $HB are set to zero on RESTART, so the default settings are the ones that correspond to the device variable being 0. See the examples under DEVICE VARIABLES.

PORTS:

Use of the PORT command and function are documented in the Glossary.
You must use ports to get at the console switches, light up

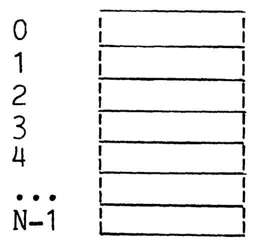
the console lights, and to directly access the music synthesizer. The rest of the ports you diddle at your own software risk--you can't break the hardware by setting ports, of course.

End of Lesson 7

LESSON 8 ARRAYS (Optional Lesson)

Computer programming languages have been largely built around what computers do well. One thing they do well is manipulate lists of things. A string is a list of characters, as you know, one after another. Also common are lists of numbers, and these are called ARRAYs. (There are also string arrays in Zgrass, but let's ignore them for now.)

An array with N elements looks like:



Before we get into the mechanics of accessing individual array elements, let's discuss the benefits of using arrays.

First, it takes at least 16 bytes to store the name and value of a named numeric variable. So 100 named variables take up at least 1600 bytes. Second, you have to get fancier than you presently know how to change all 100 of them in a loop with less than 100 different assignment statements, and it's not very fast anyway. Besides, it's a pain to type in so many names.

To create an array, use the array command:

```
ARRAY STUFF,100
```

STUFF now has 100 elements called STUFF(0), STUFF(1), ...,STUFF(99). To print out the values of STUFF, you have to write a macro:

```
PRSTUFF=[N=0
PRINT STUFF(N)
IF (N=N+1)<100,SK -1]
```

Run PRSTUFF. Of course, all values are 0. Try:

```
STUFF(91)=12345
PRSTUFF
```

You'll see the 12345 in element 91. We'll pretty up the array printing later on.

The major negative aspects of using arrays are that first, STUFF(91) is not a very good name, compared with GUNANGLE or BIGNURSE, for example, and, second, you wind up typing a lot of parentheses.

Geometrical problems lend themselves to array-based solutions. We will show you how to rotate a pyramid in 3-D using arrays. Let's look at the x,y,z coordinates and the color. The base is blue, and the lines going to the apex are red and green:

X	Y	Z	color
50	0	0	4
0	0	50	3
-50	0	0	3
0	0	-50	3
0	50	0	1
0	0	50	1
0	0	-50	4
0	50	0	2
-50	0	0	2

We will store the 40 numbers above in an array called PYR. First, create it:

```
        ARRAY PYR,40
        X values are in PYR(0), PYR(4), PYR(8), ... ,PYR(36)
        Y values are in PYR(1), PYR(5), PYR(9), ... ,PYR(37)
        Z values are in PYR(2), PYR(6), PYR(10),... ,PYR(38)
        Colors   are in PYR(3), PYR(7), PYR(11),... ,PYR(39)
```

that is, for values of N from 0 to 9,

```
        X's are in PYR(N*4)
        Y's are in PYR(N*4+1)
        Z's are in PYR(N*4+2)
        Colors are in PYR(N*4+3)
        Of course, we have to load up PYR:
        ENTER=[PRINT "TYPE ENDPOINT NUMBER, X,Y,Z,COLOR
        VALUES:"
        INPUT N,X,Y,Z,K
        PYR(N*4)=X
        PYR(N*4+1)=Y
        PYR(N*4+2)=Z
        PYR(N*4+3)=K
        SKIP -6]
```

Run ENTER and type these values:

```
        0,50,0,0,4
        1,0,0,50,3
        2,-50,0,0,3
        3,0,0,-50,3
        4,50,0,0,3
        5,0,50,0,1
        6,0,0,50,1
        7,0,0,-50,4
        8,0,50,0,2
        9,-50,0,0,2
```

If you make a mistake, just retype the number, x,y,z and color. Use CTRL+C to get out.

Now let's print the values out in a table:

```
        PRPYR=[N=0
        L=0
        PROMPT "PYR("&(4*N+L)&")=",PYR(4*N+L)
        IF (L=L+1)<4,SKIP -1
        PRINT
        IF (N=N+1)<10,SKIP -4]
```

PRINT with no arguments prints just a NEXTLINE. Be careful of the punctuation! The '&'s are used to eliminate the spaces that commas in the same place would cause. If any of the elements is incorrect when you run PRPYR, change it with ENTER.

Store PYR on tape:

```
        PUTTAPE 2,PYR,[PYRAMID ENDPOINT ARRAY FOR LESSON8]
```

Rewind the tape and type CTRL+N then:

```
        GETTAPE XXXXXX
```

to verify that PYR is stored without errors. If it doesn't say BAD DATA, PYR is stored properly. If you have a disk, DPUT PYR instead.

Now let's draw the contents of PYR. To see a 3-D object on a 2-D screen, you have to do a projection. The easiest projection is done by throwing away the z-axis coordinates. Just use the x, y and color values as arguments to the LINE command:

```
          DRAWPYR=[CLEAR
          N=0
          LINE PYR(N*4),PYR(N*4+1),PYR(N*4+3)
          IF (N=N+1)<10,SKIP -1]
```

A straight-on projection like this is not very interesting, of course. So, we'll rotate the image. If you don't understand the SINE/COSINE math below, take it on faith.

To rotate PYR around the center of the screen (z-axis), you change the endpoints by the following formula:

```
        XNEW=X*COS(ANGLE)+Y*SIN(ANGLE)
        YNEW=-X*SIN(ANGLE)+Y*COS(ANGLE)
        ZNEW=Z
```

So our macro for z-axis rotation is:

```
        ZROT=[A=0
        BOX 0,0,110,110,0
        S=SIN(A)
        C=COS(A)
        N=0
        L=N*4
        LIN PY(L)*C+PY(L+1)*S,-PY(L)*S+PY(L+1)*C,PY(L+3)
        IF (N=N+1)<10,SKIP -2
        WAIT 1
        A=A+6;SKIP -8]
```

Note that PY is being used as an abbreviation for PYR. If you change the -8 to a -7 in the last line, you'll get a built-up image. Compile ZROT to make it go faster:

```
        COM ZROT,CZROT
        CZROT
```

We still can't see the z-axis information because, obviously, we haven't used any PYR(L+2)'s yet. We have to rotate around different axes. Rotation around the x-axis is given by:

```
        XNEW=X
        YNEW=Y*COS(ANGLE)-Z*SIN(ANGLE)
        ZNEW=Y*SIN(ANGLE)+Z*COS(ANGLE)
```

Of course, we aren't using ZNEW. But we will. Try:

```
        XROT=[A=0
        BOX 0,0,110,110,0
        S=SIN(A);C=COS(A)
        N=0
        L=N*4
        LINE PYR(L),PYR(L+1)*C-PYR(L+2)*S,PYR(L+3)
        IF (N=N+1)<10,SKIP -2
        A=A+6;SKIP -6]
```

Similarly, y-axis rotation is given by:

```
          XNEW=X*COS(ANGLE)+Z*SIN(ANGLE)
          YNEW=Y
          ZNEW=-X*SIN(ANGLE)+Z*COS(ANGLE)
```

Rotating around two axes at once is more visually interesting. First we compute the rotation around x then apply the rotation around y. We need to use the intermediate value of Z in the computation of the y rotation:

```
        DOUBLEROT=[A=0;B=0
        10CLEAR IF $T1==1,BOX 0,0,110,110,0
        SA=SIN(A);SB=SIN(B);CA=COS(A);CB=COS(B)
        N=0
         .COMPUTE X ROTATION NEW X,Y,Z (THIS IS A COMMENT)
        1MOVE XXROT=PYR(N*4)
        YXROT=PYR(N*4+1)*CA-PYR(N*4+2)*SA
        ZXROT=PYR(N*4+1)*SA+PYR(N*4+2)*CA
         .COMPUTE Y ROTATION WITH NEW X,Y,Z
        X=XXROT*CB+ZXROT*SB
        Y=YXROT
         .DRAW THE LINE
        LINE X,Y,PYR(N*4+3)
        IF (N=N+1)<10,GOTO 1MOVE
        A=A+6;B=B+12;GOTO 10CLEAR]
```
Note the addition of trigger control of image clearing. You can tighten up DOUBLEROT by combining expressions, at the expense of clarity. It will go faster compiled, of course.

Comments may be interspersed with code if the first character is a period. The entire line is taken as a comment and skipped, even if there is a ';' in the line.

Zgrass allows multi-dimensional arrays. We could have defined PYR by:
```
        ARRAY PYR 10,4
```
giving 40 elements:
```
        PYR(0,0), PYR(0,1), PYR(0,2), PYR(0,3)
        PYR(1,0), PYR(1,1), PYR(1,2), PYR(1,3)
         ...
        PYR(9,0), PYR(9,1), PYR(9,2), PYR(9,3)
```
This would eliminate the multiplication of N*4 each time, so PRPYR would look like:
```
        PRPYR=[N=0
        L=0
        PROMPT "PYR("&N&','&L&")=",PYR(N,L)
        IF (L=L+1)<4,SKIP -1
        PRINT
        IF (N=N+1),10,SKIP -4]
```
You may find this conceptually clearer, but maybe not. The following example uses a 2-D array well. Let's make an array to hold the positions of pieces in a checkerboard. Each array element will correspond to a square on the board. A 0 means no piece, 1 means red, 2 means red king, -1 means black piece, -2 means black king. The initial board setup can be done by:
```
        ARRAY CHECKERBOARD,8,8
        DATA=[Y=0
        INPUT CHECKERBOARD(X,Y)
        IF (Y=Y+1)<8,SKIP -1]
        CHECKFILL=[X=0
        DATA 1,0,1,0,0,0,-1,0
        X=X+1
        DATA 0,1,0,0,0,-1,0,-1
        IF (X=X+1)<8,SKIP -3]
```
Note the mimicking of BASIC's DATA statement. The rest of the checkers game is up to you!

End of Lesson 8.

LESSON 9 MORE ON GRAPHICS

Zgrass is a graphics language by design. It provides high-level commands for creating and manipulating visual information. The highest level commands in Zgrass are those operating on arrays called "snaps" (after "snapshots") which are saved parts of the tv screen, in essence. They are defined by the SNAP command and drawn by the DISPLAY command:

```
        STEST=[CLEAR
        BOX 0,0,20,20,3
        BOX 5,0,10,15,2
        BOX -5,0,10,15,1
        SNAP IT,0,0,20,20
        DISPLAY IT,-160%159,-100%100,0;SKIP 0]
```

The last argument of DISPLAY is the DISPLAYMODE. 0 means plop the snap on the screen erasing whatever was there before. DISPLAYMODE 0 works just like COLORMODE 0-3. Now, EDIT STEST so its last line reads:

```
    DISPLAY IT,X=X+$X1,Y=Y+$Y1,0;SKIP 0
```

Plug in a joystick into the leftmost joystick socket and run STEST. You'll notice that you "drag" the edge around. One way to avoid this is to make the snap larger to include some "white space" around it. Change STEST as follows and run it:

```
    SNAP IT,0,0,24,24
```

You can have any pattern stored in a snap as long as it fits in the largest memory segment left. The CORE command will give you a list of the memory segments. The largest snap you can store is about a quarter of the screen. Larger snaps display slower than smaller snaps, of course.

In the drawing program in Lesson 7, we used COLORMODE 5 to flash a crosshair cursor. We can do the same trick with DISPLAYMODE 1:

```
        DRAW=[CLEAR
        LINE 10,0,4
        LINE -10,0,1
        LINE 0,-10,4
        LINE 0,10,1
        SNAP CURS,0,0,20,20
        X=X+$X1;Y=Y+$Y1
        DISP CURS,X,Y,1;DISP CURS,X,Y,1
        IF $T1==1,POINT X,Y,$K1/64+2
        SKIP -3]
```

Two successive DISPLAYMODE 1's will draw and erase the crosshair cursor without affecting the points you have drawn by pulling the trigger.

WHAT'S ACTUALLY GOING ON:

You can, in fact, construct any image possible on the 320x201 pixel TV screen with the colormodes you've learned so far. However, to do the animated graphics required in video games, several features were designed into the custom Bally hardware. Zgrass uses these and adds software to give you a rather complete set of graphic capabilities for what you can do with two bits per pixel.

Plop (colormodes 0-3 and displaymode 0) works just like assignment. The old value is replaced with the new one. All other color and display modes are binary functions, that is, they take two values and return a new value which is placed on the screen. The first value taken is what is on the screen at the given pixel, the second is given or implied by the color/display mode, and the function itself is also given or implied by the color/display mode.

You might wonder just how many unique results you can get from the functions of two two-pixel values. A lot. We've tried to provide the most useful ones: 21 colormodes and 15 displaymodes with 9 options each.

Rather than do a paragraph on each mode, we will show you how to figure them out. Some are quite clear from a verbal description (for example, displaymode 60 is "plop only the green and blue portions of this snap"). Unless you've tinkered with Boolean Algebra, some functions will be new to you and unless you've done extensive graphics already, the visual ramifications will be surprising.

We will explain several of the functions with TRUTH TABLEs. Truth tables are like multiplication tables except they can show other things than show results of multiplication.
The truth table for multiplication of numbers up to 4 is:

*	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	6	8
3	0	3	6	9	12
4	0	4	8	12	16

You have no trouble looking up the value of 3*4, presumably.

Multiplication tables usually stop at 12*12, but could go on forever. The truth tables we use are limited to two bit results and thus have
only four values like the following truth table for addition modulo 4:

\4	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

The functions we will describe are commutative, that is, the order of the values can be reversed without affecting the result. (Addition and multiplication are commutative; subtraction and division are not, for example.) If you wish, you can adopt the convention that what's on the screen is looked up along the left edge of the table and what is indicated by the color/display mode is along the top edge.

Colormodes 4-7 and displaymode 1 are the hardest to understand so we'll skip them and come back later. For now, let's do logical OR and AND.

Colormodes 8-11 and display mode 2 do a logical OR function between the value on the screen and the value indicated by the colormode (or the value in the snap at that point for DISPLAY). Try the following:

```
        BOX 0,0,100,100,2
        BOX 0,-25,50,50,1
        BOX 0,25,50,50,3
        BOX -25,0,50,50,9
```

Colormode 9 is "OR with red" or, more precisely: "OR with 01." Let's explain that better. We have two bits at each pixel, giving four possible values: 0, 1, 2, and 3. In binary, they are 00, 01, 10 and 11 (pronounced zero-zero, zero-one, one-zero and one-one, respectively). You may have noticed that:

$L0 corresponds to a pixel value of 00
$L1 corresponds to a pixel value of 01
$L1 corresponds to a pixel value of 10
$L2 corresponds to a pixel value of 11

When Zgrass comes up, the default colors are white, red, green and blue. So when we refer to a pixel as green, we mean it has a 10 binary value. You can confuse yourself horribly if, for example, you switch the values of $L2 and $L3 like this:

```
      A=$L2
      $L2=$L3
      $L3=A
```
so don't do it, at least in this part of the lesson. Whenever we refer to white, red, green or blue, it assumes the default colors are in $L0-$L4.

Look at the boxes on the screen again. Now look up the truth table for OR (it's under OR!) You can see why the green box OR'ed with red turns blue by reading the table: 10 OR 01 is 11. You can also see that OR'ing anything with 00 doesn't change it and OR'ing anything with 11 changes it to 11. (Observation reveals that some of the colormodes are redundant, by the way.) In words, OR produces a zero only if both corresponding bits were zero. Otherwize it puts a one there.

Let's try the AND modes (12-15):
```
      BOX 0,0,100,100,1
      BOX 0,-25,50,50,2
      BOX 0,25,50,50,14
      BOX -25,0,50,50,14
```
14 is AND with green (10). You should be able figure out what's going on by looking at the truth table for AND. In words, AND leaves a 1 in the bit only if both values had a 1 in that bit, otherwize it's zeroed.

XOR (exclusive-or) is the trickiest but most useful color/display mode function besides plop. Look up the XOR truth table. XOR yields a one when the corresponding bits are different and a zero when they are the same:
```
      10 XOR 10 IS 00
      10 XOR 00 IS 10
      10 XOR 01 IS 11
      etc.
```
You can see that XOR on a blank screen is the same as plop or OR on a blank screen. But try:
```
      NB
      NB
```
To see why the second NB erased the first, print NB:
```
      PRINT NB
```
(NB is a system macro, by the way). Note that it assures the colormode is between 5 and 7 by using the modulus operator. The lowercase letters are local variables which we'll explain in Lesson 11. Drawing anything twice with XOR will undo it.

You ought to take some time out to experiment with the colormodes and displaymodes. Here's a simple program by Jane Veeder to do some drawing:
```
      JANEDRAW=[PROMPT "WHAT'S THE XSIZE,YSIZE OF THE
```

```
DRAWING BOX?"
INPUT WIZE,HIZE
PROMPT "WHAT'S THE SPACING FACTOR OF BOX CENTERS? (0
TO 10)"
INPUT SPACING
PROMPT "WHAT COLORMODE? (1 TO 15)"
INPUT KOLOR
PROMPT "CLEAR THE TV? (Y OR N)"
INPUT.STR ANSWER;IF ANSWER=='Y',CLEAR
PRINT "MOVE JOYSTICK KNOB TO POSITION, HOLD TRIGGER
TO DRAW"
X=0;Y=0
X=X+$X1*SPACING
Y=Y+$Y1*SPACING
BOX X,Y,WIZE,HIZE,7
BOX X,Y,WIZE,HIZE,7
IF $T1==0,SKIP -4
BOX X,Y,WIZE,HIZE,KOLOR;SKIP -5]
```

Now that you've experimented a while with XOR, clear the screen and we'll do something tricky. First type in:

```
BEHIND=[X=-100
Y=50
LINE X,Y,4
LINE X,-Y,2
IF (X=X+5)<51,SKIP -2
X=-100
BOX X,0,20,20,5
BOX X,0,20,20,5
IF (X=X+1)<51,SKIP -2]
```

When you run BEHIND, you will note that the green lines are turned blue where the box is but are restored when the box is erased. Type the following:

```
$L2=$L3
```

The lines turn blue. Run BEHIND again. The red box appears to be traveling behind the blue lines because we've made both $L2 and $L3 the same color. When the red box is behind the blue lines, the value of the blue pixels is 11; when it is not there, the value is 10. So, by giving up a color, you can make one color appear to pass behind another. Now type CTRL+B to restore the colors to default. Set $L1=$L3 and run BEHIND again. The box is now "in front of" the lines.

You now have the skills to decipher the rest of the color and display modes.

End of Lesson 9.

LESSON 10 SWAP MODULES

Zgrass is rather tightly crammed into 32K of read-only memory (ROM). Some commands and functions we wanted to have in ROM just wouldn't fit so we distribute them on tape to be read in and executed just like macros. Such commands and functions are called SWAP MDOULES because you sort of swap some of your random-access memory (RAM) for the privilege of using them.

Besides allowing for more commands and functions than can fit in 32K ROM, swap modules have two other benefits: they can be changed, updated, and added to by us without having to send you new ROM's and, with the Zgrass assembler (separate package and documentation), you or your friendly neighborhood Z-80 wizard can add your own commands written in Z-80 assembler. Replacing a macro with a well-coded swap module can result in speed increases of two to maybe a thousand times depending on what you're doing. A skilled person can even do things that Zgrass won't let you, like put some of the information USEMAP gives you into variables, for instance.

If you do not have disks, the swap modules are distributed to you in alphabetical order on audio tape. You GETTAPE them just like anything else. You should copy your swap module tape early on because repeated use of any revolving mechanical recording medium will eventually result in its failure. If you have disks, the swap modules are on the disk we send you and you use DGET to bring them in.

Read about the TXT command in the Glossary and try the following (substitute DGET for GETTAPE if you have disks):

```
GETTAPE TXT
XYAXES=[LINE 160,-80,4
LINE -140,-80,1
LINE -140,100,1
X=-146;Y=-88
N=0
TXT X,Y,1,1,3,0,0,N
X=X+20
IF (N=N+1)<15,SKIP -2
N=0;X=-146
TXT X,Y,1,1,2,0,0,N
Y=Y+20
IF (N=N+1)<10, SKIP -2]
```

Zgrass also has a rather complete string manipulation package. Besides the concatenation operator (&), the following are available: ASCII, STRING, BUMP, FORMAT, LEN, LPAD, MATCH, REPLACE, SUBSTR. Some of these are swaps and you should try out these and other swap modules once you find a use for them.

Many of the disk utilities are swap modules. You will find them under the D's in the Glossary.

End of Lesson 10.

LESSON 11 ADVANCED CONCEPTS

If you print out the system test macro NB:

```
   PRINT NB
```

you will notice the lowercase variables a and b. Variables that begin with lowercase letters (a-z) are LOCAL VARIABLES, that is, they are known only to the macro they're in, just like labels for GOTOs. These variables are stored in a list attached to the memory automatically allocated to keep track of each macro call and are deleted whenever the macro returns. CTRL+C will also automatically delete all local variables.

Local variables have the following benefits:

1. They are zeroed whenever the macro is called.

2. They go away when the macro returns.

3. Recursion is possible.

4. You can create software tools without having to worry about names conflicting with other macros.

5. They never conflict with system command names.

6. Local strings and arrays are allowed.

Local variables have the following difficulties, though:

1. They don't show up in USEMAP and cannot be interrogated by typing CTRL+Z and printing the values. You have to put the PRINT right in the macro.

2. They are not known to called macros so you have to pass the values (this is actually good programming practice, anyway). In order to pass local variables that are not part of expressions, use the ? operator to force evaluation in the current macro context:

```
   SAM=[a=10
   b=20
   PRINT SQUAREM(?a,?b)]
```

```
        SQUAREM=[INPUT a,b
        RETURN a*a+b*b]
```
The a,b in each case are different variables because they are in different macros.

PARALLELISM:

Once you start communicating using animation, you realize how important timing is. Zgrass has several advanced features for controlling timing and sequencing of execution.

You are, by now, quite familiar with typing commands and running macros. When you are running a macro, you cannot type a command and expect it to execute without typing CTRL+Z. Zgrass has two modes of operation that allow macros to run and accept commands from attention mode at the same time. These modes give you the capability of foreground and background parallelism. So far, you've only used the middleground!

You can run a macro in the background by using the .B switch on it. Try:
```
        CHANGE=[$L0=$L0+$X1*8
        $L1=$L1+$Y1*8
        $L2=$K1]
        CHANGE.B
        NB.B
```
Notice that even though NB is drawing, you can change $L0, $L1 and $L2 with the joystick. You can also clear the screen at will or type other commands, start other .B macros, etc. (Note that variable b in NB is not re-initialized so it continues to increment until ERROR#23 eventually happens.)

Zgrass interleaves the command lines in CHANGE with those in NB and also slips in anything you type at the keyboard. Press CTRL+X and see the interleaving (press CTRL+Q to stop/start the printing).

If you run a macro at normal (middleground) level, it will suspend the .B macros until it is done. Thus, regular macros have precedence over .B's. You can stop all .B macros with CTRL+C or stop them selectively with the STOP command. If you press CTRL+A, .B macros will be interleaved with regular macros. You can set CTRL+A with the CONTROL command, of course.

You can also run macros in the foreground with .F. A .F macro has precedence over .B and regular macros. .F macros are assumed to be short and not contain infinite loops. They are restarted every 1/60 second or as fast as possible. It is a good idea to compile .F macros for speed.

The TIMEOUT command allows you to have the .F macro execute at multiples of 1/60 second. Say you want to draw an XOR box every five seconds. Five times sixty is 300. Try the following:

```
    TIMEDBOX=[TIMEOUT 300
    BOX 0,0,200,200,7]
    COMPILE TIMEDBOX,CTIME
    CTIME.F
```

You can have regular or .B macros running at the same time (try NB!).
Some further notes:

1. .B macros start over from the beginning automatically unless STOPped. No SKIP or GOTO is needed.

2. The interleaving of .B macros is on a line-by-line basis. Semi-colons don't count, blank lines do, so you can fine-tune the interleaving.

3. .F macros don't interleave. They are assumed to be short.

4. Macros run from within a .B macro are not interleaved unless run as .B. If called without a .B, they are interleaved as if they were a single line. If run with a .B, they run in parallel with the macro they're run from. You can setup variables to cause one .B macro to wait for another to continue, of course.

5. You can .B or .F the same macro multiple times, up to 128 times. You can have any number of .B and .F macros running at once. Obviously, things get pretty slow after a while.

6. It's a good idea to use local variables in .B and .F macros, especially if you .B a macro twice at the same time.

7. When executing in .B or .F mode, the local variables are not re-initialized to zero when the macro restarts at the beginning.

8. If your .B and .F macros have a lot of time-consuming graphics, the interleaving will not

appear to be particularly smooth.

ERROR TRAPPING:

If you wish, you can trap error messages and process them yourself, a useful feature for bulletproofing software for naive users.

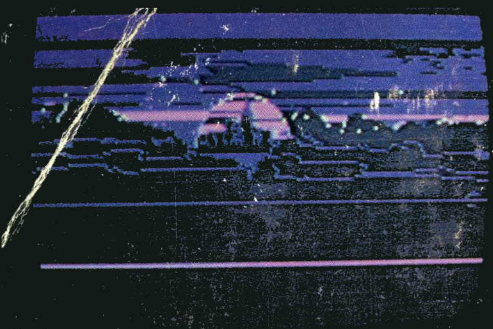
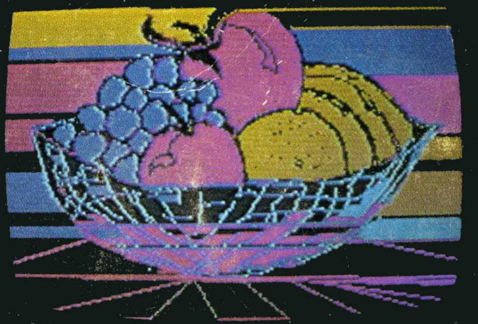
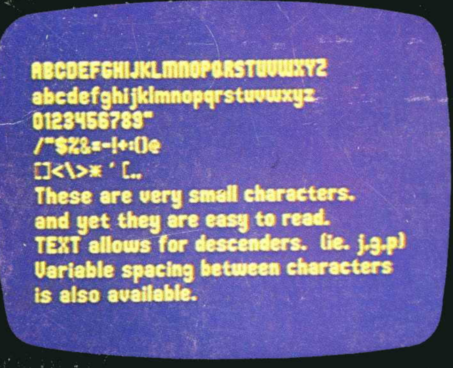
The ONERROR command takes a label like GOTO but the label is only branched to when an error occurs. For instance:

```
      UGH=[ARRAY SAM,10
      A=-1
      ONERROR 1OUT
      PR SAM(A=A+1);SKIP 0
      1OUT PRINT "SAM OUT OF BOUNDS"]
```

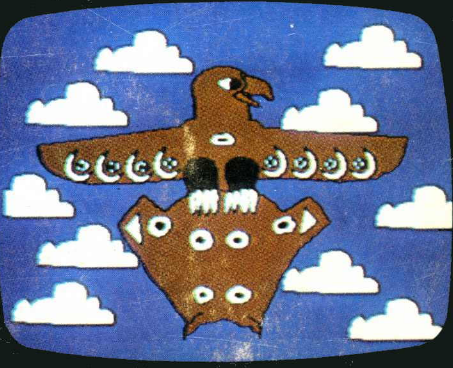
You can get the error number and command line in error into variables with the GETERROR swap command. LOOPMAX is a command used to catch infinite loops.

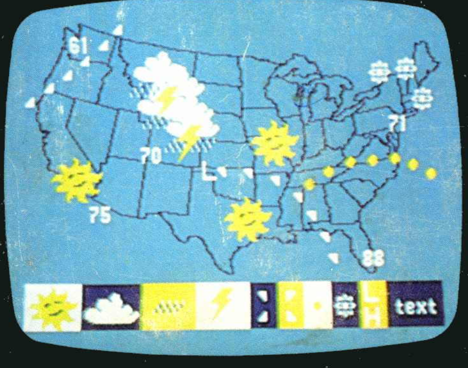
LOOPMAX and ONERROR do not work when compiled.

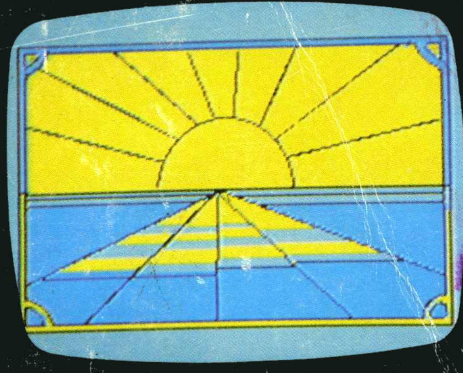
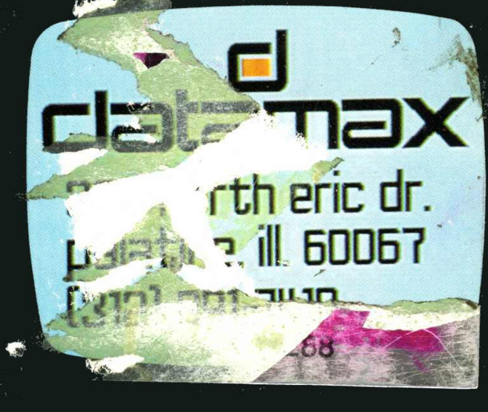
End of Lesson 11 and End of Lessons.




ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
0123456789"
/"$%&=-|+:()e
[]<\>* ' [..
These are very small characters.
and yet they are easy to read.
TEXT allows for descenders. (ie. j,g,p)
Variable spacing between characters
is also available.

AMERICAN
DANCE
THEATRE







datamax
[illegible]rth eric dr.
[illegible], ill. 60067
([illegible]) [illegible]
[illegible]68