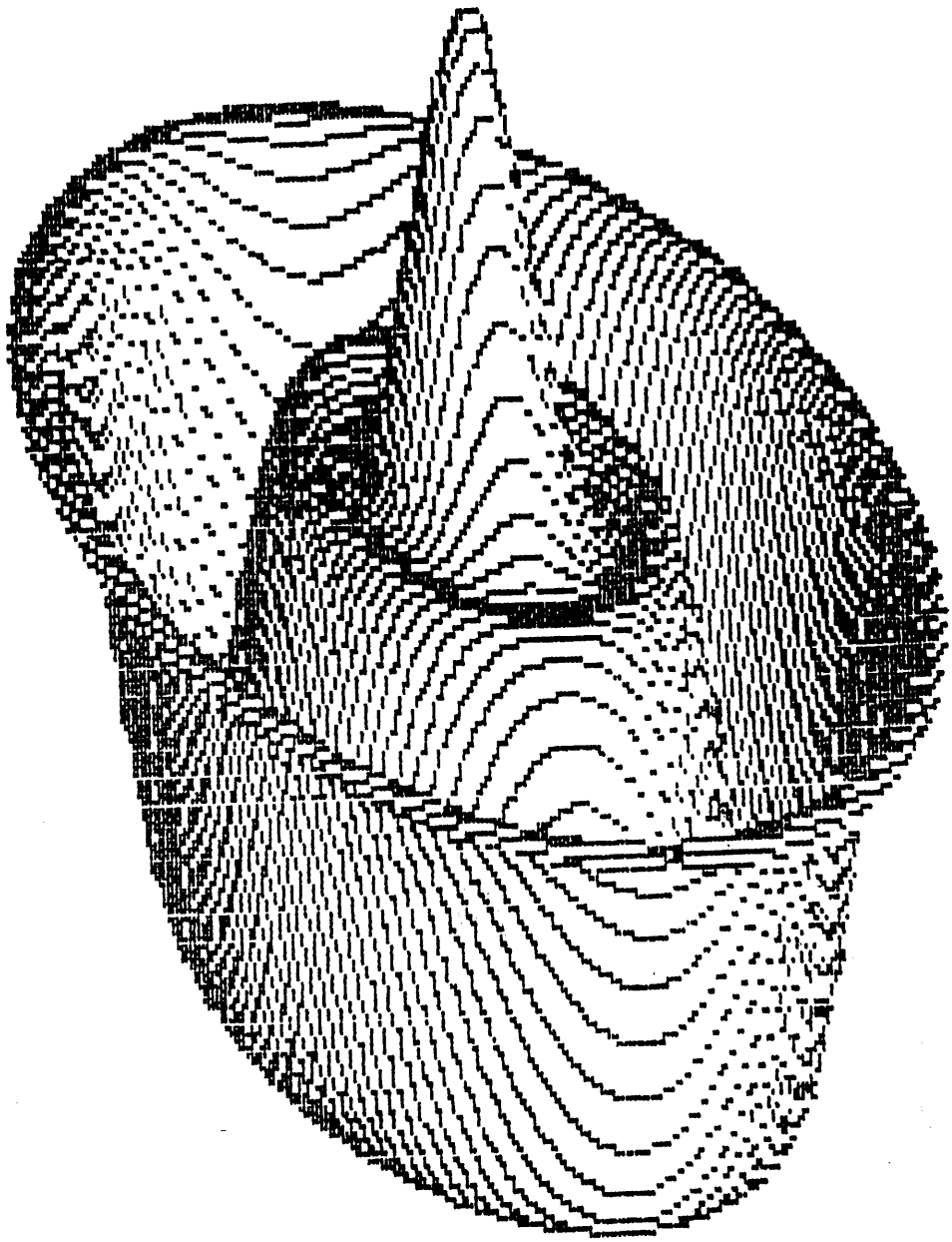
LNWBASIC



Modular Software Associates

TABLE OF CONTENTS

LNWBASIC INTRODUCTION

LNWBASIC Version 4.0 adds a set of commands to disk BASIC for all models of the LNW80 computer. Many powerful yet easy to use graphics commands have been added, allowing full use of the LNW80's graphics capabilities from BASIC. LNWBASIC also includes many non-graphics BASIC enhancements, such as a spooler/despooler, an improved tracer, RS232-C commands, defineable and pre-defined keys, and many more.

LNWBASIC is produced by the CREATOR program. The CREATOR allows you to customize LNWBASIC for your needs. Only those commands that you wish to use are included in the LNWBASIC program you create, thereby saving memory. You can make another LNWBASIC program with a different set of commands whenever you wish, simply by executing CREATOR from DOS. After an LNWBASIC module has been created, you execute it as you would BASIC. LNWBASIC will automatically run the version of BASIC you are using, and then initialize itself.

The accompanying 35 track, single-density diskette contains the programs: LNWBASIC/CMD, CREATOR/CMD, RECOVER/CMD, DEMO/LNW and several other demonstration programs (each ending with the extension /LNW). The diskette also contains the programs C80X16/CMD and C80X24/CMD which are 80 column driver programs. These programs are supplied on an as-is basis and hold no warranty. They may or may not allow full compatibility with DOSPLUS or other Disk operating system and all risk for use is left entirely with the user. Refer to the last page of this manual for the END-USER SUB-LICENSE and warranty disclaimer.

NEW LNW80's:

All the commands available with LNWBASIC will work on the latest version of the LNW80 Model II computer. The BLINK, LCASE, and REPEAT commands, however, should not be used when operating under a disk operating system that contains similar functions (such as DOSPLUS).

LNW80 Model I/II:

The JOY and PAGE commands were specifically written for the latest version of the LNW80 Model II computer, and will not work on other models. They should not be included when you create an LNWBASIC module using CREATOR. Certain other functions may or may not work, depending upon how recent your computer is. These include gray scales on your black-and-white monitor in MODE 2 graphics, the backdrop color in MODE 2 graphics, expanded MODE 2 graphics (160 X 192 as opposed to 128 X 192), and RS232-C initialization.

LNWBASIC CREATOR

This program produces an LNWBASIC program on disk, composed of the commands you select. Once an LNWBASIC program has been created, it can be executed as described in EXECUTING LNWBASIC. You need run CREATOR only when you desire a different version of LNWBASIC.

When you execute CREATOR, you will be asked what commands you wish to include in the LNWBASIC program you are creating. You will be shown every keyword in alphabetical order. A few keywords, such as DO (UNTIL) and PON (POFF), are shown in pairs (the second keyword will be displayed in parentheses). After deciding what keywords to include, the LNWBASIC program you have just created will be written to disk. You will then be ready to execute LNWBASIC.

To execute the LNWBASIC CREATOR, type "CREATOR" when you see "DOS READY". (CREATOR occupies memory from 5200H to DFFFH, and therefore will overwrite any custom driver routines, etc, that occupy this region.) The screen will clear, and the following will appear:

```
MODULAR SOFTWARE ASSOCIATES - LNWBASIC CREATOR - VERSION 4.0

KEYWORD                    DESCRIPTION                    SIZE     USE?
BLINK       SWITCH BLINKING CURSOR ON/OFF                 148      ?
```

Notice the information that is displayed for BLINK--this is the same type of data that will appear for every LNWBASIC keyword. First, there is the keyword BLINK, followed by a short description of what BLINK does. Next is the number of bytes (in decimal) that would be added to LNWBASIC if you decide to include BLINK. The size for a particular keyword can vary greatly from one use of CREATOR to another, depending on what other keywords have (or have not) been included at the point when the size is displayed. The size is displayed to give an approximate idea of the size of code associated with a keyword. It is not intended to be an aid in achieving optimal memory usage, since the figure shown can be misleading.

You must now decide whether or not to include BLINK in the LNWBASIC program you are creating. If you wish to include it, you must press the "Y" key; to exclude it press the "N" key. The same type of information for the next keyword will now be shown, and you must decide whether or not to use it. When you have answered "Y" or "N" for every LNWBASIC keyword, the following will appear:

```
TOTAL SIZE OF COMMANDS INCLUDED: XXXXX

FILESPEC TO SAVE TO (PRESS ENTER FOR "LNWBASIC/CMD" DEFAULT)?
```

The total size given is the total byte count which will be taken up by the LNWBASIC program being created once it is executed. The size on disk will be considerably larger due to initialization requirements.

The LNWBASIC program just created will now be saved to disk. If you press <ENTER> in response to the "FILESPEC ..." question, it will be saved under the name LNWBASIC/CMD. If you desire to have it saved under a different name, you may type in a standard DOS filespec and press <ENTER>. (The filespec may not include a password, however.) A short delay will occur in which the relocatable LNWBASIC program is created. After the delay, the disk drive should turn on and the LNWBASIC program will be saved to disk. If an error occurs at this point (such as ILLEGAL FILE NAME or DISK SPACE FULL), it will be necessary to reset the computer and run CREATOR again.

IMPORTANT NOTE: If the following message appears;

XX TOO MANY REFERENCES RELOCATED - CAN'T CREATE THIS MODULE!

please note the exact circumstances and report them to Modular Software Associates. An unexpected error has occurred that would require a modification of LNWBASIC for successful creation of the particular configuration.

EXECUTING LNWBASIC

Once you have run the LNWBASIC CREATOR program and created an LNWBASIC program on disk, it is very simple to execute LNWBASIC. The normal "BASIC/CMD" file must exist for the created LNWBASIC program to execute. You execute LNWBASIC just as you would execute whatever BASIC you are using, except that you substitute "LNWBASIC" (if the LNWBASIC program you have just created is named, "LNWBASIC/CMD") for "BASIC".

If you are using NEWDOS BASIC, for example, you may type the number of files desired, memory size, etc, all on the same line as you normally would to enter BASIC. If you are using TRSDOS, then you should only type "LNWBASIC" and press <ENTER>. Answer the "HOW MANY FILES?" and "MEMORY SIZE?" questions as usual.

The screen will then clear, Modular Software Associates' copyright notice will appear, and below that a number will be displayed ("User Memory Start Address = XXXXX"). This number gives the start address in decimal of your BASIC program text area. You are now ready to use LNWBASIC.

IMPORTANT NOTE: DEBUG must be off before executing LNWBASIC, or the computer will "hang". Once you have entered LNWBASIC, you may re-enable DEBUG by CMD"D" if desired.

EXAMPLES:

LNWBASIC
will execute LNWBASIC under all compatible DOS's.

LNWBASIC 15,50000
will execute LNWBASIC and enter BASIC with 15 files and memory size set to 50000. (NEWDOS)

LNWBASIC -F:3-M:62000
will execute LNWBASIC and enter BASIC with 3 files and memory size set to 62000. (DOSPLUS)

EXITING LNWBASIC / RECOVERY OF BASIC PROGRAM TEXT

LNWBASIC modifies numerous restart vectors, device DCB's, and other vital system data areas. Unless LNWBASIC is exited using the standard BASIC command CMD"S" (CMD without an argument will also work), or through the LNWBASIC command DRUN, these data areas will not be restored. This can lead to disasterous results, requiring at least the re-booting of DOS.

Once LNWBASIC has been exited, neither "BASIC *" nor "LNWBASIC *" will succeed in re-entering LNWBASIC to recover a BASIC program. The utility program RECOVER is provided for this purpose. From the DOS READY state, execute the correct version of RECOVER (see LNWBASIC INTRODUCTION). RECOVER will automatically execute BASIC, but will not pass any parameters to BASIC (e.g. RECOVER 65000 will not set memory size to 65000). After entering BASIC, issue a LIST command to determine that the entire program text is still in memory. (Note: If the keyboard locks up, reset the computer and try RECOVER again. This is due to improperly exiting LNWBASIC.) After verifying that the program is complete, SAVE the program and exit. Do not attempt to execute the program before exiting. RECOVER is a convenient utility program, but it will not work in all instances.

MISCELLANEOUS NOTES

LNWBASIC occupies RAM below normal BASIC. This moves BASIC programs higher in RAM than usual. Some programs which use PEEK and POKE do not allow for this fact, and may not run properly under LNWBASIC.

The CIRCLE, DRAW, and LINE commands maintain a "current graphics position". This is simply the last X,Y point drawn by any of these commands. If an initial X,Y point for one of these commands is not given, it will default to th current graphics position. This allows for easy interaction among the 3 commands, making a pie chart, for example, relatively simple to draw.

The CIRCLE, DRAW, and LINE commands can not be interrupted in the middle of execution by <BREAK>, but the PAINT command can be.

LNWBASIC-specific error messages (such as "UNTIL WITHOUT DO") can not be trapped by ON ERROR GOTO processing.

In the syntax descriptions that follow, optional arguments are surrounded by braces "{ }".

BLINK

The BLINK command toggles the blinking block cursor on/off. The first time the command is executed, the blinking cursor will appear. The next time the command is executed, the blinking cursor will be replaced with the standard cursor character. If BLINK is used again, after the standard cursor has been restored, the blinking block cursor will reappear (and so on...).

EXAMPLES:

```
10 BLINK
```
if this is the first time that BLINK is used, the blinking block cursor will be turned on.

```
10 BLINK
20 INPUT"NAME";A$
30 BLINK
40 PRINT "HELLO, ";A$
```
if the standard cursor is being used prior to running this program, the following will occur: The question "NAME?" will be displayed with a blinking block cursor after it. If the reply to the input statement was "JOHN", the line "HELLO, JOHN" will appear. Note the blinking cursor will have been replaced with the standard cursor.

CALL int-expression
CALL lsb,msb expression
CALL pos-expression

This command allows you to easily call a machine language subroutine. No DEFUSR or USR(X) statement is needed. The subroutine must be in memory at the time of the CALL. You can not directly pass an argument between the BASIC program and the machine language subroutine being called. If you wish to pass one or more arguments, this can be done by POKEing and PEEKing into specified memory locations. The subroutine being called should return to BASIC by the Z-80 RET statement. The CALL statement saves all registers used by BASIC.

EXAMPLES:

CALL &HE000
will call a subroutine located at hex E000.

20 CALL +50000
will call a subroutine located at decimal 50000.

30 CALL X + PEEK(35000)
will call a subroutine located at the sum of the current value of X and the contents of memory location 35000.

CALL 0,200
will call a subroutine at 0,200 in lsb,msb (decimal) notation.

200 CALL -100
will call a subroutine located at -100 (65436 decimal).

10 X=-100 : CALL X
will call a subroutine located at -100 (65436 decimal).

1000 X=65436 : CALL X
The intent of this program line is the same as in the preceeding examples, but will result in an OVERFLOW error. This is because "X" will be interpreted as an "int-expression". To force "pos-expression" interpretation, the correct usage is CALL +X.

10 X=&HF000+12 * 128 : CALL X
will call a subroutine located at the evaluated expression. Note that there can be no space between a hexadecimal number and next part of the expression - otherwise a SYNTAX ERROR may occur.

CIRCLE {XC},{YC},XR {,YR} {,SA} {,EA} {,RA}

The CIRCLE command may contain up to 7 parameters; XC (X-Center), YC (Y-Center), XR (X-Radius), YR (Y-Radius), SA (Start-Angle), EA (End-Angle) and RA (Rotation-Angle). It is, however, only necessary to specify 1 - XR. It is necessary that the default values for XC or YC be given a "place holder" by use of a comma. Other parameters may be assigned default values by specifying ",," (comma,comma) in place of the normal ",value,". CIRCLE, then, could contain any combination of between 1 and 7 parameters.

The default values for XC and/or YC are determined by the last point drawn by CIRCLE, LINE or DRAW. If a CIRCLE is drawn using only XC, YC and XR then the center-point becomes the new default point. The default value for the Y-Radius (YR) is .65*XR (X-Radius). The default for Start-Angle (SA) is Ø degrees; for End-Angle (EA), 360 degrees. The default Rotation Angle of the determined circle or ellipse is Ø degrees.

Angles are measured in degrees from the three o'clock position of the circle in a clock-wise rotation. Values for these angles may range from Ø-360 degrees. 360 degrees is normally associated with the same point as Ø degrees. Under the CIRCLE command this is not so. Due to rounding errors with multiplication of pi, the point associated with 360 degrees is the last point drawn before overwriting the Ø degree point. If a value is specified outside of Ø-360 degrees, an ILLEGAL FUNCTION CALL error will result. Neither the specified center point of a circle nor either the X or Y radius need fall within displayable point boundaries. For example, a center point of 1000,500 is legal. The user may wish an arc of an "imaginary" large circle to be displayed.

NOTES: When a value is not specified for the Y-Radius (YR) the default is .65*XR (X-Radius). This gives a nearly perfect circle on a BMC or LEEDEX monitor. Users with an NEC B/W ("green-screen") monitor will find the value .6*XR closer to a circle on their monitor. Circles drawn with just the X-center (XC or default ","), the Y-center (YC or default ",") and the X-radius (XR) specified; draw a circle at a much faster rate than with more parameters. Also the new default X and Y become XC and YC respectively. When a circle is drawn with more than the first three parameters, such as with a start and/or end angle, the last point drawn becomes the new default point for CIRCLE, LINE and DRAW.

EXAMPLES:

```
10 MODE 1
20 CIRCLE 240,96,100
```

will draw a circle with the center at point 240,96 (roughly the center of the screen) with an X-axis radius of 100. This will appear as a "perfect" circle on a BMC or LEEDEX monitor and slightly flattened on others.

```
10 PCLS
20 MODE 1
30 CIRCLE 240,96,100,10
```

will clear the graphics screen, set the mode to hi-res B/W and draw an ellipse centered on point 240,96 with a major axis (X-axis) radius of 100 and a minor axis (Y-axis) radius of 10 - i.e. a very "flattened" circle.

```
10 PCLS
20 MODE 1
30 CIRCLE 240,96,100,,0,180
```

will clear the graphics screen, set the mode to hi-res B/W graphics and draw an arc of a circle centered at 240,96. The Y-axis radius defaults to .65 * X-radius; this is the default condition as specified by the ",," (comma - comma). The portion of the circle from 0 degrees to 180 degrees is drawn. Note that 0 degrees is the three o'clock position of the circle and 180 is the nine o'clock position as drawn in a clock-wise direction.

```
10 PCLS
20 MODE 2
30 FLS
40 COLOR 3
50 CIRCLE 64,96,40,15,,270,45
```

will clear the graphics screen, set the mode to lo-res color, "white" the text screen (i.e. enable all graphics points), set the default color to red and draw an arc of the ellipse with a major axis radius of 40 and a minor axis radius of 15. The portion of the ellipse from 0 degrees (as defaulted to by the ",,") to 270 degrees will be drawn. Additionally, the specified arc will be rotated by 45 degrees.

```
10 PCLS
20 MODE 1
30 FOR I=20 TO 100 STEP 20
40 CIRCLE 240,96,I,65
50 NEXT I
60 FOR I=20*.65 TO 65 STEP 20*.65
70 CIRCLE 240,96,100,I
80 NEXT I
```

will clear the graphics screen, set the mode to hi-res B/W, and draw the "skeleton" of a globe. First the longtitude and then the latitude lines will be drawn.

```
10 PCLS : CLS
20 MODE 3
30 COLOR 5
40 CIRCLE 197,96,1
50 FOR I=20 TO 120 STEP 20
60 CIRCLE ,,I
70 NEXT I
```

will clear the graphics screen and then clear the text screen. The mode will be set to hi-res color (RGB) with the color being set to blue (COLOR 5). A circle with radius of 1 (i.e. a point) will be drawn in the center of the MODE 3 screen. Then, using the default

values for both the X-center and Y-center (which were defined by the first circle drawn as 197,96), multiple concentric circles will be drawn. NOTE: if the first circle drawn had used more than 3 parameters (XC,YC,XR), the default center for the FOR-NEXT circle loop would not have been the center point - it would have been the last point on the circle (or partial circle) drawn.

```
10 CLS
20 PCLS
30 MODE 1
40 DRAW"B,M240,96"
50 FOR I=60 TO 360 STEP 60
60 CIRCLE ,,120,,I-60,I
70 DRAW"M240,96"
80 NEXT I
```

will draw a "pie chart" circle of radius 120 and "slices" of 60 degress. The text screen is cleared, the grahics screen is cleared, and the MODE is set to 1 (hi-res B/W). Line 40 uses the DRAW command to do a blank (B) move (M) to the point 240,96. This sets the default point (used by CIRCLE, DRAW and LINE) to 240,96. When an arc of a circle is drawn in line 60, this point is used as the center point (defaults for XC and YC). The DRAW command in line 70 does two things - 1) it draws a line from the end of the arc drawn in line 60 to the center point (240,96) of the circle and 2) it makes the point 240,96 again the "new" default point for the next CIRCLE command in the FOR-NEXT loop. The circle and "pie pieces" are drawm by the FOR-NEXT loop for a complete circle.

COLOR int-expression
COLOR {int-expression}, int-expression

The first argument of the COLOR command sets the color for use by the next graphics command. It is used for MODE 2 and 3 graphics (low resolution and high resolution color). Once COLOR is set, it is the value used by PSET, LINE, CIRCLE and DRAW. DRAW, however, may change its current color by using the "C" option. This does not affect the color when commands other than DRAW are used. COLOR has no effect on MODE 0 or 1 commands.

The second argument of the COLOR command changes the backdrop color. It is used exclusively for MODE 2 graphics. The backdrop color will change immediately upon execution of the COLOR command. The backdrop color will be seen as the border color on the color graphics monitor when the text screen is "white". When the text screen is clear, the backdrop color will appear on the entire screen of the color graphics monitor.

NOTE: Not all LNW80 computers support gray scales or backdrop colors.

The legal values for COLOR range from 0 to 7. Using a value outside of this range will result in an ILLEGAL FUNCTION CALL error. On a black and white monitor, the gray scale increases in darkness from 0 (white) to 7 (dark grey). On a color monitor, the corresponding colors are:

COLOR 0	WHITE
COLOR 1	GREEN
COLOR 2	YELLOW
COLOR 3	RED
COLOR 4	MAGENTA
COLOR 5	BLUE
COLOR 6	BLUE-GREEN
COLOR 7	BLACK

EXAMPLES:

```
10 PCLS: FLS: MODE 2
20 COLOR 3
```

will clear the graphics screen, "white" the text screen (enabling all points), set the graphics mode to lo-res color and set the default color to red.

```
10 MODE 2
20 PCLS 3 : CLS
30 COLOR ,5
40 DO : UNTIL (INKEY$ <> "")
50 FLS
60 GOTO 60
```

demonstrates the meaning of backdrop color. Line 10 sets the current graphics mode to 2. Note that backdrop colors are applicable

only to mode 2. Line 20 sets all points of the graphics screen to red (3), and clears the text screen. Line 30 sets the backdrop color to blue (5). Line 40 waits until a key is pressed before allowing line 50 to be executed. The color graphics monitor at this point will show a blank, blue screen. Although graphics memory contains all points set to 3 (from the PCLS 3 command), the backdrop color shows through because graphics memory is not being displayed (since the text screen is clear). After a key is pressed, line 50 will execute, "whiting" the text screen, and causing graphics memory to be displayed on the color monitor. Graphics memory "overlayed" the backdrop color, displaying a red screen bordered by the blue backdrop.

```
10 MODE 3 : PCLS : CLS
20 COLOR 5
30 CIRCLE 100,50,100
```

will draw a blue (5) circle on a high resolution color (RGB) monitor.

```
10 FLS
20 MODE 2: PCLS: COLOR 3
30 LINE 0,0,127,191,SET
40 COLOR 7
50 LINE 127,0,0,191,SET
```

will "white" the text screen (enabling all points), set the mode to lo-res color, set the color to red, draw a red diagonal line from the top left of the video screen to the bottom right, set the color to black and draw a black diagonal line from the top right of the video screen to the bottom left.

```
10 FLS: MODE 2: PCLS 7
20 FOR I=1 TO 100
30 COLOR RND(8)-1
40 PSET RND(128)-1,RND(192)-1
50 NEXT I
60 GOTO 60
```

will "white" the text screen, set the mode to lo-res color, turn the graphics screen to black (PCLS 7), and draw 100 random points of random colors.

```
10 PCLS : FLS : MODE 2 : COLOR,0
20 FOR Y=0 TO 105 STEP 15
30 COLOR Y/15
40 LINE 0,4,127,Y+14,SET,BF
50 NEXT
60 GOTO 60
```

will display the color and grey scales available. The color scale will appear on the color graphics monitor; the grey scale, if available, will appear on the text monitor. Line 10 sets graphics memory to 0 (white), "whites" the text screen to enable displaying of graphics memory, sets the graphics mode to lo-res color, and changes the backdrop to white. Lines 20-50 draw a solid rectangle in each of the available colors (from 0 to 7).

CONV int-expression
CONV lsb,msb expression
CONV pos-expression
CONV "A"
CONV "AA"

This command is used to display different representations of the same number. CONV "A" means convert the single alphanumeric character A. CONV "AA" means convert the 2-character string AA, where AA can be any 2 alphanumeric characters (not necessarily the same).

The following illustrates the display format of CONV:

```
CONV +50000                                          (user entry)
&HC350    " P"   -15536    +50000    80,195        (computer reply)
 HEX       ASCII
```

The first number displayed is the hexadecimal (base 16) representation. Next follows the 2 character alphanumeric or graphic representation. The first character displayed corresponds to the most significant byte (msb) of the hexadecimal representation, while the second character corresponds to the least significant byte (lsb). The byte will be displayed as a blank if it is less than 0021H (33 decimal), or greater than 00BFH (191 decimal). An exception is that a zero byte will be considered as an ASCII NUL (""), and will not be displayed. The third representation given is the decimal integer form of the number. This is the number you would use to PEEK or POKE an address above 32767. The fourth number displayed is the positive decimal representation. The fifth (last) representation is the number in decimal lsb,msb format. This number is very useful when PEEKing or POKEing addresses in decimal.

EXAMPLES:

```
CONV "bD"
```
will display the representations of the letters "bD".

```
CONV &HBD+ 128
```
will display the representations of the sum of the hex number 00BDH and the decimal number 128. Note that there can be no space after a hexadecimal number and the next portion of an expression - otherwise a SYNTAX ERROR may occur.

```
10 X=100 : Y=0 : CONV X,Y
```
will display the representations of the lsb,msb number 100,0.

```
CONV +65000
```
will display the representations of the pos-expression +65000

DEFKEY {old character, {new character}, definition}

Where all arguments are string expressions of length 1 or more.

DEFKEY allows any 10 keys to be re-defined as strings of ASCII character codes of length 1 or more. This is accomplished by replacing 1 of the 10 current definitions with a new definition. The first argument specifies which current defined key is to be replaced. The second argument tells what key it is to be replaced with. If this argument is a null string or is missing, the first argument will be used. The third argument gives the definition for the new defined key. For example, DEFKEY "0","A","A=PI*R**2" will remove the "0" key from the defined key list and insert the "A" key in its place. The definition for the "A" key becomes the string "A=PI*R**2". Thereafter, whenever the "A" key is pressed (when the defined key output is enabled), the string "A=PI*R**2" will appear.

A simple method has been implemented to toggle the defined key output on and off. To enable (or disable if enabled) the defined key output, simply hold down the <SHIFT> & <down-arrow> keys and press the "D" key. (It is easiest to press the <SHIFT> key first, then while keeping the <SHIFT> key pressed, hold down the <down-arrow> key. With these 2 keys depressed, simply hit the "D" key to toggle the output on or off.)

A second method is provided to enable or disable the defined key output. DEFKEY without any arguments toggles the defined key output on and off. This allows the defined keys to be enabled or disabled within a BASIC program.

A total of 221 characters may be defined for the 10 key definitions. If a DEFKEY command is given which would cause the total to exceed this limit, a STRING TOO LONG error will result. The maximum length of any one string is 127 characters; a STRING TOO LONG error will be given if this limit is exceeded. If the third argument of DEFKEY is a null string, an ILLEGAL FUNCTION CALL error will be given. The first and second arguments of DEFKEY should be strings of length 1, but no error will be given if the string is longer. Only the first character of the string will be used, the others will be ignored. An ILLEGAL FUNCTION CALL error will also be returned if the first argument is not a currently defined key.

It is not necessary to have DEFKEY included in an LNWBASIC module in order to use the defined keys. The initial set of defined keys can be changed by the LOADKEY command, thus providing a method of re-defining the keys without DEFKEY. Refer to DISKEY, LOADKEY, and SAVEKEY for further commands relating to defined keys.

EXAMPLES:

The following examples were written to work with the defined keys as they are initially set with every new LNWBASIC created.

DEFKEY

will toggle the defined key output on or off. If the defined key output was on before this statement was executed, it will be disabled, otherwise it will be enabled.

```
DEFKEY"0","d","CMD"+CHR$(34)+"DIR"+CHR$(13)
```

removes the definition for "0" and defines the "d" key to be the string 'CMD"DIR <RETURN>'. When this key is pressed (with defined key output enabled), a DIRectory of drive 0 will be given (assuming your DOS supports the BASIC statement CMD"DIR").

```
DEFKEY "1",CHR$(31),"PCLS"+CHR$(13)
```

removes the definition for "1" and defines the <CLEAR> key (ASCII code 31) to be the string "PCLS <RETURN>".

```
DEFKEY CHR$(31),"","CLS:PCLS"+CHR$(13)
```

removes the above definition for <CLEAR> and replaces it with the string "CLS:PCLS <RETURN>". The null string for the second argument is used for convienence; CHR$(31) for the second argument would be functionally equivalent.

```
DEFKEY CHR$(31),,"CLS:PCLS"+CHR$(13)
```

is functionally equivalent to the preceding example.

```
DEFKEY "2","#",CHR$(183)+CHR$(187)
```

removes the definition for "2" and defines the "#" key to be the string consisting of (low res) graphics codes 183 and 187. DEFKEY can be used in this manner to aid in a programming technique known as "string packing".

```
1000 #"DODEF" : REM ROUTINE TO CHANGE DEFINED KEYS
1010 DISKEY : REM DISPLAY CURRENT DEFINITIONS
1020 LINEINPUT"WHAT KEY IS TO BE REPLACED? ";OLD$
1030 LINEINPUT"WHAT IS THE NEW KEY (<ENTER> IF SAME AS OLD KEY)?";NW$
1040 IF NW$="" THEN NW$=OLD$
1050 PRINT"TYPE THE STRING TO BE ASSIGNED TO ";NW$
1060 LINEINPUT NS$
1070 DEFKEY OLD$,NW$,NS$ : GTO #"DODEF"
```

is a short program using DEFKEY. Note that the defined key output must be disabled when this program is run or it will be impossible to type the key to be replaced.

DESPOOL "filespec" {,S}

The DESPOOL command without the trailing ",S" will despool the specified file to a parallel printer. The command with the trailing ",S" will despool the specified file to the RS-232-C serial port. The spooler is interrupt driven; normal processing can continue while a file is being printed. An occasional short pause will occur as the disk file is being accessed for more information. This command allows the operator to compose, edit, or run BASIC programs which may include disk I/O, while a previously spooled file is being printed. The despooling is automatically terminated upon reaching the end-of-file. To cancel the despooling operation, hold down the shift key, the down arrow key, and the "C" key simultaneously (this outputs a 'control C'). To pause the despooling, key a control-S (XOFF). This is done by holding down the shift key, the down arrow key and the "S" key. To continue despooling, key in a control-Q (XON). This is again done by holding down the shift key, the down arrow and the "Q" key.

Only ASCII encoded files may be despooled. BASIC programs saved without the ",A" option may not be despooled. An attempt to despool two files at the same time will result in an ILLEGAL FUNCTION CALL error.

EXAMPLES:

```
10 DESPOOL"TEXT/FIL"
```
will start the despooling of the file "TEXT/FIL" to the parallel port printer.

```
20 DESPOOL"CODEFILE/TXT.SECRET:1",S
```
will, upon execution, start despooling the file "CODEFILE/TXT" on drive 1 with the password "SECRET" to the serial port.

DISKEY

This command displays the current set of defined keys (see DEFKEY and LOADKEY). When this command is issued, the screen is cleared, and each of the 1Ø currently defined keys are displayed in the following format:

key blank string

where:

key = The ASCII representation of the defined key. Keys such as <BREAK> and <CLEAR> will therefore appear as blanks.

blank = A blank space to separate the key from its definition.

string = The string of ASCII characters that the key is defined as.

EXAMPLE:

1Ø DISKEY
when executed, will display the current set of defined keys.

DISKEY
will display the current set of defined keys.

DLOAD filespec

The DLOAD command is used to load object-code files into RAM and return control to BASIC. Command (/CMD) files may be loaded and, providing the entry point is known, executed from within BASIC.

EXAMPLES:

```
10 DLOAD"PRINTER/CMD"
```
will load the command file "PRINTER/CMD" into RAM.

```
10 DLOAD "SORT/CIM"
20 CALL &HF000
30 PRINT"SORT DONE"
```
will load the object-code file "SORT/CIM" and execute it (if the start address is hex F000). The line "SORT DONE" will then be displayed.

DO {statement} {statement} ... {statement}
UNTIL (true/false expression)

The DO and UNTIL commands form an iterative (repetitive) loop. The DO command denotes the start of the loop, while the UNTIL command terminates the loop. The body of the loop consists of 0 or more statements between DO and UNTIL. These statements will be executed repeatedly until the (true/false expression) argument following UNTIL is true. (The parentheses surrounding the "true/false expression" after UNTIL are mandatory.) DO/UNTIL loops can be nested up to 10 levels; nesting past a level of 10 will cause an OVERFLOW error.

If an UNTIL is executed without previously executing a matching DO, an UNTIL WITHOUT DO error will occur. Mismatched DOs and UNTILs can result from transferring (GOTO) out of the DO/UNTIL loop body. To immediately terminate a DO/UNTIL loop, use the following procedure--never transfer (GOTO) out of the loop.

```
10 I=0
20 DO
30   GOSUB 1000
.
.
100  IF T=2 THEN I=10 : GOTO 200
.
.
190  I=I+1
200 UNTIL (I=10)
```

In line 100, the loop terminating condition (I=10) is set to true. Control is then transferred to the line containing UNTIL, where the loop will be properly terminated.

A DO/UNTIL loop should not be imbedded within IF/THEN/ELSE statements, unless the entire loop is contained within the same program line as the IF. The following two examples illustrate this restriction. The first example shows an incorrect use of a DO/UNTIL within an IF/THEN/ELSE. The second example shows how the first example can be written correctly.

```
10 IF I=1 THEN DO
20   PRINT"I=1"
30   INPUT I
40 UNTIL (I<>1)
```

The fault in the above program is that line 20 will be executed even if I does not equal 1. This is due to the fact that the "scope" of the IF statement only applies to the line in which it appears.

```
10 IF I=1 THEN DO : PRINT "I=1" : INPUT I : UNTIL (I<>1)
```

This program line will execute the DO/UNTIL loop only if I=1. There are, of course, several other methods of coping with the IF/THEN/ELSE restriction, two of which are shown below.

```
10 IF I=1 THEN GSUB #"DO I LOOP"
.
.
999 END
1000 #"DO I LOOP"
1010 DO
1020   PRINT "I=1"
1030   INPUT I
1040 UNTIL (I<>1)
1050 RETURN

10 IF I<>1 THEN GTO #"DONE LOOP"
20 DO
30   PRINT "I=1"
40   INPUT I
50 UNTIL (I<>1)
60 #"DONE LOOP" : REM REST OF PROGRAM FOLLOWS
```

The DO/UNTIL loop construct executes slower than a FOR/NEXT loop, since it is implemented as an add-on to an existing interpreter (LEVEL II BASIC). However, it can provide structure and clarity to a BASIC program, which are often more desirable than speed of execution.

EXAMPLES:

```
10 A$="" : DO : INPUT"WHAT IS YOUR NAME";A$ : UNTIL (A$<>"")
```

will continue to INPUT A$ until something other than <ENTER> is INPUT.

```
10 TRUE=0
20 DO : CLS
30   GOSUB 200
.
.
100 UNTIL (TRUE OR A=B)
```

will execute the DO/UNTIL loop until the variable TR equals -1 or until A=B.

```
10 DO
20   GSUB #"INIT GAME"
30   DO : GSUB #"PLAY ROUND" : UNTIL (GAMEOVER)
40   GSUB #"WINNER"
50 UNTIL (0)
```

is an example of nested DO/UNTIL loops. The DO/UNTIL loop in line 30 is nested within the loop of lines 10-50. This is a nesting level of 2. (Notice that if the subroutine #"PLAY ROUND" has a DO/UNTIL loop, it too would be nested within the loop of lines 10-50.) The DO/UNTIL loop in line 30 will terminate when the variable GA equals -1. The loop of lines 10-50 will "never" terminate, since 0 can never be true (that is, can never equal -1).

DRAW string, where "string" consists of one or more of the following arguments (in any order, including repetition of the same argument):

Output Arguments --
- D - Draw line Down.
- E - Draw line at 45 degree angle.
- F - Draw line at 135 degree angle.
- G - Draw line at 225 degree angle.
- H - Draw line at 315 degree angle.
- L - Draw line Left.
- M - Draw line from current point to specified point.
- R - Draw line Right.
- U - Draw line Up.

Output Modifier Arguments --
- B - Blank - Move current graphics position, but don't draw.
- N - No update - Draw, but don't update current graphics position.
- Z - Zero - Erase line rather than draw it.

Mode Arguments --
- A - Current DRAW Angle.
- C - Current DRAW Color.
- S - Current DRAW Scale.

Miscellaneous Arguments --
- X - Execute string as one or more DRAW arguments.

The DRAW command allows complicated designs to be quickly and easily drawn on the CRT. Straight lines of any length in any of 8 directions may be drawn. Entire figures may be rotated by 9Ø degrees, and can be expanded or contracted.

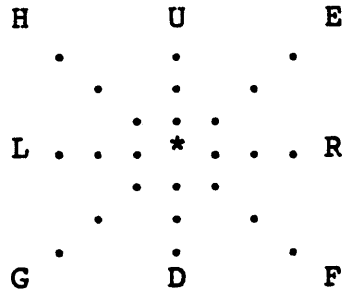
Output Arguments - D,E,F,G,H,L,M,R,U

Syntax:
```
argument {length} (except M argument);
M X-coordinate, Y-coordinate
```

X-coordinate, Y-coordinate, and length can be integer variables or constants; not expressions. If length is not given, a length of 1 will be used.

These arguments, unless preceeded by an Output Modifier argument, draw a line (of length Ø or more) from the current graphics position. The end position of the line drawn then becomes the "new" current graphics position.

The following chart shows what direction a line is drawn from the current graphics position for the corresponding argument:



When a line is drawn, the current graphics position is always included in the output. The command DRAW"D1" will set the current graphics position and the point below. A length of zero can be used to set only the point at the current graphics position.

As with all LNWBASIC graphics commands, points not visible on the screen may still be drawn. For example, the command DRAW"B,M470,30,R50" in MODE 1 graphics will result in a line of length 50 being drawn. Only the 10 points from 470,30 to 479,30 will be displayed, however. The current graphics position will be updated to 520,30. The command DRAW"D5" would then draw a line of length 5 from 520,30 to 525,30. Since this is not displayable on the screen, no visible line would be drawn, but the current graphics position would be updated to 525,30.

The **M** argument is used to move to an absolute point on the screen (X-coordinate, Y-coordinate). If it is immediately preceeded by the B argument, no line will be output. The command DRAW "B,M240,96" will make the current graphics position the approximate center of the screen (in high resolution mode) without drawing a line. Whenever the commands RUN or NEW are executed, the current graphics position is reset to 0,0. The following will cause a line to be drawn from the upper left corner of the screen to the middle:

```
10 PCLS : CLS : MODE 1
20 DRAW "M240,96"
RUN
```

If the X-coordinate, Y-coordinate pair are out of the displayable bounds of the current graphics mode, the line wil still be drawn as though the point did exist. Also, the current graphics position will be updated to this out-of-bounds point.

Output Modifier Arguments - B,N,Z

These arguments affect the operation of the next Output argument executed (whether or not the Modifier & Output arguments appear in the same DRAW command).

The **B** (blank) argument causes the next Output argument to not draw a

visible line on the screen, but to update the current graphics position as though the line had been drawn.

The **N** (no update) argument causes the next Output argument to draw the line as usual, but do not update the current graphics position.

The **Z** (zero) argument causes the next Output argument to function as usual, except that the line will be erased (reset) rather than drawn (set). The argument does not affect the updating of the current graphics position (it will be updated as usual).

There are no restrictions on the use of Output Modifiers, although most combinations are meaningless. For example, the command DRAW"B,N,M5Ø,5Ø" does absolutely nothing since the line is not drawn (B argument) and the current graphics position is not updated (N argument).

Mode Arguments - A,C,S

Syntax:
A {numeric variable or constant} (Ø-3, default = Ø);
C {numeric variable or constant} (Ø-7, default = COLOR);
S {numeric variable or constant} (1-255, default = 4).

Mode arguments stay in effect until re-issued or until "NEW" or "RUN" is executed. The default values will be used after the execution of "NEW" or "RUN". An invalid argument to a Mode argument will result in an ILLEGAL FUNCTION CALL error.

A (angle) mode determines the degree of rotation (if any) to be used when drawing all subsequent lines. The following are the possible angles of rotation:
Ø = Ø degrees rotation (no rotation);
1 = 9Ø degrees rotation (clockwise);
2 = 18Ø degrees rotation (clockwise;
3 = 27Ø degrees rotation (clockwise).
For example, the command DRAW"A1,R5Ø" is functionally equivalent to the command DRAW"AØ,D5Ø". Similarly, the command DRAW"A3,R5Ø" is equivalent to DRAW"AØ,U5Ø".

C (color) mode determines the color subsequent lines will be drawn in. This argument has no visible effect in black and white graphics modes. Also, this argument does not affect the current color as determined by the COLOR command. If C is not used, the current color will be used. If C is used without a value, the current color will also be used.

S (scale) mode determines the "scale" subsequent lines are to be drawn at. The scale is given by a number between 1 and 255, and indicates the scale in units of 1/4. Normal (1) scale is S4. Scale allows objects to be easily expanded or contracted. The command DRAW"S4,R2Ø,D2Ø,L2Ø,U2Ø" will draw a box looking exactly the same as the box drawn by DRAW"S8,R1Ø,D1Ø,L1Ø,U1Ø". If the length of an Output argument does not come out to be an integer because of the

current scale, it will be rounded to the nearest integer. For example, DRAW"S2,R15" will result in a horizontal line of length 8.

Miscellaneous arguments - X

Syntax:
X string variable

The X (execute) argument provides a method of specifying a frequently used sequence of DRAW arguments without having to explicitly repeat them. It allows a string constant (used in all examples so far) to be replaced by a string variable. The next example illustrates this process:

```
10 MODE1 : PCLS : CLS
20 BOX$="R10,D10,L10,U10"
30 DRAW"B,M240,96,XBOX$,S16,XBOX$"
```

The string variable BOX$ contains the string assigned to it in line 20 which will draw a box of size 10. Line 30 puts the current graphics position in the approximate center of the screen, and then draws the box. In effect, the statement "XBOX$" is replaced by "R10,D10,L10,U10". Finally, the box is drawn again (at 4 times its original size), by the next "XBOX$" statement. There are no restrictions on the use of the X argument, other than (of course) the string variable used must contain a valid DRAW string. A null string will result in an ILLEGAL FUNCTION CALL error.

EXAMPLES:

```
10 CLEAR 500 : CLS : PCLS : MODE 1
20 DRAW"B,M240,96,R20,D20,L20,U20"
RUN
```

will draw a rectangle in the approximate center of the screen. Line 10 reserves string space, clears the text & graphics screens, & sets the graphics mode to high-resolution black & white graphics. (This line will be assumed in all subsequent examples.) The DRAW command in line 20 positions the current graphics position near the screen's center ("M240,96") without drawing a line ("B"). A line of length 20 is then drawn to the right of this position ("R20"). At this point (before the "D20" is executed) the current graphics position is at 260,96, having been moved 20 in the positive X direction ("R20"). The "D20" is then processed, causing a line to be drawn of length 20 down from 260,96. At this point the current graphics position is 260,116. After "L20" is executed, the current graphics position will be 240,116, and it will become 240,96 again after "U20" is executed and the rectangle is completed.

```
20 DRAW "B,M240,96,E20,F20,G20,H20"
RUN
```

will draw a diamond at the center of the screen. The updating of the current graphics position is similar to the preceeding example. After execution of "E20" the current graphics position will be at 260,76, having been moved 20 in the positive X direction and 20 in the negative Y direction. The subsequent graphics positions are: 280,96 , 260,116 , and 240,96.

```
20 FOR A=0 TO 3
30 DRAW"AA,B,M240,96,R80,D60,L40,U20,H40"
40 PRINT@640,"ANGLE = ";A : FOR X=1 TO 1000 : NEXT
50 NEXT
RUN
```

demonstrates the effect of the A (angle) argument. Each figure is rotated 90 degrees from the preceeding figure.

```
20 FOR X=80 TO 330 STEP 120
30 FOR SZ=1 TO 20
40 DRAW"B,MX,86,SSZ,B,U10,B,RMV,R5,F10,D10,G10,L10,H10,U10,E10,R5"
50 NEXT
60 MV=MV+8 : NEXT
RUN
```

demonstrates the effect of the S (scale) argument. Lines 30-50 form an inner loop which draws a figure made up of 20 individual figures. Lines 20 & 60 make up the outer loop and cause 3 of the composite figures to be drawn. (The inclusion of "B,RMV" (line 40) with the changing value of 'MV' (line 60) is what causes the 3 figures to appear to be shifting perspective.)

```
15 BX$="N,U30,R40,U30,L40,E20,R40,N,G20,D30,G20"
20 FOR A=0 TO 3
30 DRAW"AA,B,M240,96,XBX$"
40 PRINT@640,"ANGLE = ";A : FOR X=1 TO 1000 : NEXT
50 NEXT
```

illustrates the usage of the X (execute) argument. This example is the same as the A (angle) example, with the exception of lines 15 & 30. Line 15 was added to define the string variable BX$. BX$ is set equal to the DRAW arguments needed to draw a simple box. Line 30 was modified to execute BX$ within the DRAW command. Using the X argument in this manner allows the same figure to be easily drawn in different DRAW commands, without the need for re-typing the arguments that draw the figure.

```
10 CLS
20 PCLS
30 MODE 1
40 DRAW"B,M240,96"
50 FOR I=60 to 360 STEP 60
60 CIRCLE ,,120,,I-60,I
70 DRAW"M240,96"
80 NEXT I
```

will draw a "pie chart" circle of radius 120 and "slices" of 60 degrees. The text screen is cleared, the graphics screen is cleared, and the mode is set to 1 (hi-res B/W). Line 40 uses the DRAW command to do a blank (B) move (M) to the point 240,96. This sets the default point (as used by CIRCLE, DRAW and LINE) to 240,96. When an arc of the circle is drawn in line 60, this point is used as the center point (defaults for XC and YC of CIRCLE). The DRAW command in line 70 does two things - 1) it draws a line from the end of the arc drawn in line 60 to te center point (240,96) of the circle and 2) it makes the point 240,96 again the "new" default point for the next CIRCLE command the FOR-NEXT loop. The circle and "pie pieces" are drawn by the FOR-NEXT loop for a complete circle. SEE THE CIRCLE COMMAND FOR THIS SAME EXAMPLE AND EXPLANATION.

DRUN filespec

The DRUN command exits LNWBASIC and executes the requested object file. This is a final exit. A command file may not be run with a return to LNWBASIC. If an error occurs during the execution of DRUN a return to DOS READY will result. The complete file name must be used including extension as no defaults are implemented.

NOTE: Anomalous results may occur if the command file being executed resides in memory occupied by LNWBASIC (i.e., below the start of BASIC program text).

EXAMPLES:

DRUN"DIRCHECK/CMD"
will exit LNWBASIC, run the program "DIRCHECK/CMD" and exit to DOS.

10 DRUN "BASIC/CMD"
will exit LNWBASIC and load and execute "BASIC/CMD". This will allow the user to enter "normal" BASIC without exiting to DOS.

FLS {character code}
FLS {"character"}

FLS fills the text screen with the specified character. This command is provided for ease of use in MODE 2 (lo-res color) and MODE 3 (hi-res color - i.e. RGB) graphics. If no argument is given, a default value of 191 (all-blocks-on, low-resolution graphics) is used.

In MODE 2 graphics, FLS 191 will "white" the text screen, thereby allowing graphics memory to be displayed on the color graphics monitor. In MODE 3 graphics, FLS can be used to "map" all enabled graphics points to a particular color. The formula for colors in MODE 3 (RGB) graphics is COLOR * 9 + 64 = character value for use by FLS. COLOR is the same as used by the COLOR command.

The first argument form requires an integer expression between Ø and 255. The ASCII character associated with the argument will be used to fill the text screen.

The second argument form requires a single character enclosed in double quotes. This character will be used to fill the screen.

EXAMPLES:

```
1Ø FLS
1Ø MODE 2 : PCLS 2
```
will fill the 1Ø24 positions of text screen with the low resolution (MODE Ø) all-blocks-on graphics block, set the graphics mode to lo-res color, and set all graphics memory to 2 (yellow).

```
FLS "*"
```
will fill the text screen with asterisks ("*").

```
1Ø X = RND(64) : FLS X+127 : GOTO 1Ø
```
will fill the text screen with a random graphics block (128 - 191) upon each execution of FLS.

```
1Ø PCLS 7
2Ø MODE 3
3Ø FLS 3*9+64
```
will white the graphics screen, set the MODE to hi-res color (RGB) and "map" all points on the graphics screen to red.

GSUB line number
GSUB #"label"

This command allows for greater flexibility in calling a BASIC subroutine. The argument "line number" may be a constant, variable, or expression. It denotes the BASIC program line number to GOSUB. If the second argument form of GSUB is used, '#"label"' indicates that the line number to GOSUB is labeled with #"label". Other than providing additional methods of referring to the subroutine being called, GSUB performs identically to GOSUB. However, GSUB may not be substituted for GOSUB in an ON n GOSUB command.

EXAMPLES:

```
10 GSUB 40000 : GSUB 100
20 GSUB SN*100
```
will cause calls to subroutines located at line numbers 40000, 100, and the product of the variable SN and the number 100.

```
10 GSUB #"INIT"
20 GSUB #"PROCESS"
.
.
1121 #"INIT"
.
.
1300 RETURN
.
.
1380 #"PROCESS"
.
.
1420 RETURN
```
will cause the subroutines located at lines 1121 & 1380 to be called.

GTO line number
GTO #"label"

This command provides additional methods of specifying a line to GOTO. The argument "line number" may be a constant, variable, or expression. It denotes the BASIC program line number to GOTO. The argument '#"label"' refers to the line which begins with the matching #"label". Other than allowing greater flexibility in referring to a line number to GOTO, GTO performs identically to GOTO. However, GTO may not be used in place of GOTO within an ON n GOTO or ON ERROR GOTO command.

EXAMPLES:

10 GTO 10
will cause an "infinite" loop upon execution.

10 #"HERE" : GTO #"HERE"
will cause an "infinite" loop similar to the previous example.

10 GTO X
will cause execution to transfer to the line specified by the current value of the variable X, assuming such a line exists.

HIMEM int-expression
HIMEM lsb,msb expression
HIMEM pos-expression

HIMEM allows you to protect high memory so that it will not be used by BASIC. The argument specifies the address of the last byte usable by BASIC. An ILLEGAL FUNCTION CALL error will result if this address is not at least 200 bytes above the start of user memory. HIMEM resets the values of all variables and does a CLEAR 50; caution should therefore be employed when using HIMEM within a BASIC program.

EXAMPLES:

```
HIMEM &HFFFF
```
will allow BASIC to use all of memory (48K machine).

```
10 HIMEM +40000
```
will protect memory above 40000 (decimal).

```
10 INPUT X : HIMEM +X
```
will protect memory above the input value of X. Note the use of the positive ("+") sign before X. This is to allow for values greater than 32767. A different way to limit the values of the argument is:

```
10 INPUT X% : HIMEM X%
```
will allow X to range from -32768 to +32767.

JOY int-expression, numeric variable

The JOY command reads the position of up to 2 joysticks and returns the position value in the specified numeric variable. Int-expression in the JOY command line can assume a value from 0 to 3. If a value is specified outside this range, an ILLEGAL FUNCTION CALL will result. The value returned to the variable is ranges from 0 to approximately 400-512. The actual upper limit depends on the joystick (and LNW80!) used. Specifying a string variable or a double precision variable for the returned value will also result in an ILLEGAL FUNCTION CALL. Joysticks have both an X-axis and a Y-axis associated with them. The X-axis goes from left to right where 0 is the furthest left value and the right-most position returns the highest value. The Y-axis goes up and down with the uppermost position returning 0 and the lowest position returning the highest value.

Joysticks are set up according to the following int-expression table:

int-expression	joystick	axis
0	0	X-axis
1	1	X-axis
2	1	Y-axis
3	0	Y-axis

Associated with the two joysticks are "fire" buttons. The buttons are mapped to memory location &H37E4. Bit 2 is associated with the button on joystick 0 and bit 3 with joystick 1. The following BASIC program shows how to read the buttons:

```
10 IF PEEK(&H37E4) AND 4 = 4 THEN PRINT"JOYSTICK 0'S BUTTON IS ON"
20 IF PEEK(&H37E4) AND 8 = 8 THEN PRINT"JOYSTICK 1'S BUTTON IS ON"
30 IF PEEK(&H37E4) AND 12 = 12 THEN PRINT"BOTH BUTTONS ARE DEPRESSED"
```

NOTE: If a joystick is not connected to the LNW80, the value returned to the specified variable will be -1. A delay loop between "JOY" commands may be necessary due to hardware requirements for a "settling time" between reads. To convert the range returned by the JOY command to comparable numbers of the currently used MODE, use the following formula:

point value = (highest mode value * JOY value) / (highest JOY value)

EXAMPLES:

```
10 JOY 0,X : FOR I=1 TO 10 : NEXT
20 JOY 3,Y : FOR I=1 TO 10 : NEXT
30 PRINT X,Y
```

will read and print the position of joystick Ø. X will obtain the X-axis value and Y the Y-axis value. The values will both be -1 if the joystick is not plugged into the LNW8Ø. If the joystick is in the uppermost left-hand position, both X and Y will be Ø. If the stick is in the uppermost right-hand position, the X value will be the highest value returned by the joystick while the Y value will be Ø. If this is not the case, many joysticks have "fine-tune" controls (trim pots) that may be adjusted accordingly.

```
1Ø CLS : PCLS
2Ø MODE 1
3Ø JOY 1,X : FOR I=1 TO 5 : NEXT
4Ø JOY 2,Y
5Ø IF PEEK(&H37E4) AND 8 <> 8 THEN PSET X,Y ELSE PRESET X,Y
6Ø GOTO 3Ø
```

will act as an "etch-a-sketch" program. The text screen and then the hi-res screens are cleared. The X-axis of joystick 1 is read into the X variable and the Y-axis of joystick 1 is read into the Y variable. If the button on joystick 1 is not depressed, a point corresponding to the X and Y values is set. If the button is depressed, the specified point is reset (i.e. erased). The program then loops "indefinitely" - using the joystick as a "pencil"

Holding the joystick in the furthest right and down position, execute:

```
JOY Ø,XL : PRINT XL : JOY 3,YL : PRINT YL
```

This will return values that show the joystick's limits. The preceding example should be modified as follows to give the "best" etch-a-sketch program; Add:

```
45 X=X * 479/XL : Y=Y * 191/YL
```

#"label"

The pound sign ("#") preceding a string constant serves to identify the string as a label. If a line is to be labeled, the pound sign must be the first non-blank character in the line after the line number. The pound sign must be followed by a double quote, a string constant of 0 to 251 alpha-numeric characters (except a double quote), and the terminating double quote. If the line is not null, there must be a colon seperating the label and the BASIC statement which follows. Labels should be unique, although no check is performed to insure this. If two lines are labeled with the identical label, a reference to the label will always refer to the first (lowest numbered) line. A line should only be labeled once--any subsequent labels will be ignored.

EXAMPLES:

```
100 #"DELAY" : DO : T$=INKEY$ : UNTIL (T$<>"") : RETURN
```

labels line 100 as #"DELAY". It may now be referenced by a GSUB #"DELAY" statement.

```
10 #"START"
.
.
5000 GTO #"START"
```

labels line 10 #"START". A sample reference is shown in line 5000.

LCASE

The LCASE command toggles the lower case display enable. The first time the command is used, lower case letters may be entered and displayed by using the shift key and the appropriate letter. The next time the LCASE command is used, lower case entries are inhibited.

Lower case characters can be used anywhere in composing a BASIC program but can only be displayed by PRINT statements or in REM lines.

EXAMPLES:

```
LCASE
10 PRINT"This is a test."
20 LCASE
```

if LCASE has not been used prior to the LCASE command executed in command mode, lower case will be enabled. Line 10 may be input by using the shift-key entry for all lower case characters. Line 20 will inhibit any further lower case entry or display. When the program is RUN, the line, "This is a test.", will be displayed.

```
10 LCASE
20 STOP
```

will result, if lower case is not enabled before the program is run, in the display of the message, "Break in 20". Note that the message is displayed in both upper and lower case letters.

LINE {X1},{Y1},X2,Y2,SET {,shape}
LINE {X1},{Y1},X2,Y2,RESET {,shape}

Where shape is "B" or "BF".

The LINE command with just the SET option turns on the points of the line determined by the points X1,Y1 and X2,Y2. The LINE command with the RESET option resets the points of the determined line. The B option allows the perimeter of the box determined by the line (i.e. the diagonal) to be SET or RESET. The diagonal is not drawn or erased. The BF option allows the perimeter to be SET or RESET with the "inside" of the box either SET or RESET. That is, the Box is drawn and Filled (SET) or the box is erased (RESET). In MODE 2 graphics, the line or box, if the SET option is used, is drawn with the specified color as determined by the last COLOR command. If the RESET option is used in MODE 2 graphics, the specified line or box is drawn with the current background color (reference the PCLS command).

Either X1 or Y1 may default to the current point as set by CIRCLE, DRAW or LINE. "LINE ,Y1,X2,Y2,SET" would use the default X for X1. "LINE X1,,X2,Y2,RESET" would use the default Y for Y1 and "LINE ,,X2,Y2,SET,B" would use both the default X and Y. Note that the last point SET (or RESET) by the LINE command becomes the new default point as used by CIRCLE, DRAW and LINE.

LINEs may be drawn to and from points not displayable on the screen. That is, the command LINE Ø,Ø,1ØØØ,96,SET is an allowable statement with the result that the line from point Ø,Ø to point 1ØØØ,96 will be "drawn". Only the portion of the line that falls within the graphics screen co-ordinates will actually be displayed. This enables the user to constuct graphics figures drawn to "imaginary" view-points for perspective renderings and for "windowing" effects.

EXAMPLES:

```
1Ø MODE 1
2Ø LINE Ø,Ø,479,191,SET
```
will draw a line from the top left of the video display to the lower right.

```
1Ø MODE 3
2Ø LINE Ø,Ø,479,191,SET,B
```
will draw a box around the outside of the graphics portion of the RGB color monitor.

```
1Ø MODE 3
2Ø LINE 479,191,Ø,Ø,RESET,B
```
will erase the box as drawn in the previous example.

```
100 MODE 0
110 LINE 0,0,20,47,SET,BF
```
will draw and fill-in the low-res box determined by the designated line.

```
5  MODE 1
10 LINE 100,100/3,100*1.6,(100*2+5)/3),RESET,BF
```
will erase the box as determined by the diagonal line specified by the evaluated arguments.

```
5  FLS
10 MODE 2
20 COLOR 3
30 X1=0: Y1=0: X2=127: Y2=191
40 LINE X1,Y1,X2,Y2,SET
```
will draw a red line from the top left of the color monitor to the bottom right.

```
10 FLS
20 MODE 2
30 LINE 127,191,0,0,RESET
```
will set the line from the preceding example to white (COLOR 0). See above text for explanation.

```
5  MODE 1: CLS: PCLS
10 FOR I=1 TO 10
20 LINE RND(480)-1,RND(192)-1,RND(480)-1,RND(192)-1,SET,BF
30 NEXT I
```
will set graphics mode to hi-res B/W, clear the text screen, clear the graphics screen and draw and fill-in 10 random boxes.

```
10 CLS : PCLS
20 MODE 1
30 LINE 0,0,479,191,SET
40 LINE ,,479,0,SET
50 LINE ,,0,191,SET
60 LINE ,,0,0,SET
```
will clear the text screen, clear the graphics screen and enter the hi-res B/W mode. A line will be drawn from the upper left corner f the screen to the lower right. In line 40 the last point drawn befor this line (i.e. 479,191) becomes the origin point. A line will be drawn from the lower right corner to the upper right. In line 50, a line from the upper right corner will be drawn to the lower left corner and in line 60, a line will be drawn from the lower left to the upper left corner.

LOADKEY filespec

This command allows a previously saved file (see SAVEKEY) of defined keys (see DEFKEY) to be loaded from disk. This list of defined keys from disk replaces the current list of defined keys in memory. If the file to be loaded is not a list of defined keys (that is, if it was not written by SAVEKEY), an ILLEGAL FUNCTION CALL error will result.

EXAMPLES:

```
LOADKEY "GRAPHICS/KEY"
```
will load the defined keys from the file "GRAPHICS/KEY" if the file was saved by the SAVEKEY command.

```
.
.
.
2000 LOADKEY "PLAYER1.PASSWORD/KEY" : REM GET 1ST PLAYER'S KEYS
.
.
.
```
will, upon execution, load the specified defined keys file.

LOC. {"string"}
LOC. {string}

This command allows you to locate instances of a string within the BASIC program currently in memory. The LOC. command with an argument locates the first occurrence of the string within the program. LOC. without an argument locates the next occurrence of the last string searched for. The maximum argument string length is 20 characters.

The first argument form of LOC. ("string") should be used if the string to be searched for occurs in a quoted string (PRINT "string" or INPUT "string", for example) or a REMark. Double quotes may not be imbedded within the argument string since they serve as delimiters of the string. The second argument form of LOC. (string) should be used to locate a string that is not within a quoted string or REMark. The two forms of LOC. allow you to distinguish between a BASIC keyword and the corresponding character string, as illustrated in the examples below. Both forms use a string constant as an argument, not a string expression. That is, the commands 'A$="HELLO":LOC.A$' will locate the variable A$, it will not locate "HELLO".

The LOC. command maintains its own current line pointer, but will adjust BASIC's LIST and EDIT line pointer as described below. To start a search for a string, use LOC. with an argument. LOC. will start its search from the first program line. If the string is located, the line containing the string will be LISTed on the screen. BASIC's LIST and EDIT pointer will now be changed to point to the LISTed line. LOC.'s own pointer will also have been changed so that the next LOC. (without an argument) will search beginning with the next line after the line just LISTed. (This means that two occurrences of the same string within the same line will be located only once.) Once the line has been LISTed, you have several choices:

1). Press <BREAK> to end LOC. and return to Command mode or the executing BASIC program (should LOC. have been executed within a program);

2). Press the "E" key to enter the EDIT mode in order to EDIT the line just LISTed. After ending EDIT, you will return to the Command mode as usual;

3). Press <RETURN> to do an automatic LOC. (LOC. without an argument). This will find the next occurrence of the string just located.

If a search for a string fails, no line will be LISTed to the screen, and BASIC's LIST and EDIT pointer will be reset.

EXAMPLES:

The examples below assume the following program is in memory:

```
10 GSUB #"NAME?"
20 PRINT : PRINT "THANK YOU ";
30 PRINT "VERY MUCH ";NM$
40 GOSUB 100
50 END
60 #"NAME?" : INPUT"WHAT IS YOUR NAME";NM$ : RETURN
```

```
LOC.?                   (Locate the first occurrence of the BASIC
                         token PRINT)
                  Line 20 is LISTed to the screen
<RETURN>                (Locate the next occurrence of PRINT)
                  Line 30 is LISTed to the screen
<BREAK>                 (Exit LOC.)
                  Command mode entered
LOC."?"                 (Locate the first use of a question mark)
                  Line 10 is LISTed
<RETURN>                (Locate the next occurrence of "?")
                  Line 60 is LISTed
E key pressed           (EDIT line 60)
                  EDIT mode is entered
     .                          .
     .                          .
                  EDIT mode is exited
LOC.                    (Locate the next occurrence of "?")
                  No line is LISTed and Command mode is entered
LOC.GOSUB               (Locate the first use of the GOSUB token)
                  Line 40 is LISTed
<BREAK>                 (Exit LOC.)
                  Command mode is entered
LOC.NM$                 (Locate the first use of the variable NM$)
                  Line 30 is LISTed
<BREAK>                 (Exit LOC.)
                  Command mode is entered
```

MODE int-expression

The MODE command "turns-on" the graphics screen specified. There are 4 modes or types of graphics supported by the LNW-80. MODE 0 is the "normal" lo-res graphics. MODE 1 is the hi-res B/W graphics display. MODE 2 is the lo-res color mode while MODE 3 is the hi-res RGB mode. The graphics display resolution in MODE 0 is 128 X 48. MODE 1 can display points in a 480 X 192 screen. 160 X 192 is the resolution displayable in MODE 2, although older models of LNW computers are only capable of displaying 128 X 192. MODE 3 resolution is 384 X 192. If a MODE less than 0 or greater than 3 is specified, an ILLEGAL FUNCTION CALL error will result.

NOTE: Inverse video, as described in the LNW-80 literature, is not supported by the MODE command. Inverse video may be obtained in MODEs 0 or 1 by the following; X=INP(254) : OUT 254,X+1.

EXAMPLES:

```
10 CLS
20 MODE 1
```
will clear the text screen and enter the hi-res B/W display mode.

```
10 FLS
20 PCLS
30 MODE 2
```
will "white" the text screen (i.e. enable all points), clear the graphics screen and enter lo-res color mode.

```
10 FLS 9+64
20 PCLS 7
30 MODE 3
```
sets the RGB color to green, fills the graphics screen and then "flips" to the green filled screen. NOTE: line 20 does <u>not</u> enable the graphics display but, fills the graphics display "invisibly".

```
10 I=0 : MODE I
```
will turn off any current graphics display and return to normal text mode/display.

MOVE destination address, start address, byte count

The MOVE command allows a block of memory 1 - 65,536 bytes long to be copied to another place in memory. The 3 arguments of MOVE are all mandatory, and all are int-expressions. There is no restriction placed on the value of any of the 3 arguments. **IMPORTANT NOTE: It is very easy to destroy your program using MOVE!!**

The MOVE command is useful for moving text & graphics (MODE 0) data on the text screen, as well as large array transfers. The MOVE command will determine the direction of the transfer from the destination and start address. This is of importance only when the two areas overlap, to insure that the data being moved does not destroy the data to be moved. When the destination address is lesser than the start address, the transfer will progress in a "positive" direction. That is, after each byte is transferred, the destination & start address will be incremented by one. For example, the command MOVE &HA000,&HA001,3 will cause A001H to be copied into A000H, A002H to be copied into A001H, and A003H to be copied into A002H. Similarly, if the destination address is greater than the start address, the addresses will be decremented by one after each byte is transferred.

EXAMPLES:

```
10 FOR X=1 TO 10
20   FOR Y=1 TO 63
30     MOVE &H3C00,&H3C01,1023
40   NEXT
50   FOR Y=1 TO 63
60     MOVE &H3FFF,&H3FFE,1023
70   NEXT
80 NEXT
```

will scroll the text screen left and right 10 times. Line 30 scrolls the screen to the left by 1 character position by moving all of text screen memory down one byte. That is, memory from 3C01H - 3FFFH is copied into 3C00 - 3FFE. Line 60 scrolls the screen to the right by 1 character position. This is done by copying video memory from 3FFEH - 3C00H into 3FFFH - 3C01H.

```
10 DEFINT A,B
20 DIM A(999), B(999)
30 PRINT "START FOR/NEXT LOOP:"
40 FOR X=0 TO 999 : A(X)=B(X) : NEXT
50 PRINT "STOP FOR/NEXT LOOP, START MOVE:"
60 MOVE VARPTR (A(0)), VARPTR (B(0)), 1000*2
70 PRINT "STOP"
```

demonstrates the difference in speed between a FOR/NEXT loop and MOVE when used to copy the contents of one large array to another.

NTROFF
NTRON {line range,} {expression} {,expression} ... {,expression}

The NTRON command enables a trace facility for debugging BASIC programs. It allows you to specify what lines of your BASIC program are to be traced, as well as providing a unique expression-trace capability. A BASIC program must be in memory at the time the NTRON command is executed, or an ILLEGAL FUNCTION CALL error will occur. The NTROFF command disables this trace facility.

The "line range" argument determines what lines of the BASIC program currently in memory will be traced. (See below for a detailed description of "line range".) If missing, all lines of the BASIC program will be traced. It is important to remember that the line range entered remains fixed, even if lines are added to the BASIC program (until you execute another NTRON). For example, suppose the BASIC program in memory at the time an "NTRON" (no line range) is executed has lines ranging from 10 to 50. If you later add lines between lines 10 - 50, these will be traced without doing another NTRON. However, if you add lines outside this range (before line 10, or after line 50), these lines will not be traced unless you execute another NTRON. The above example would still apply even if the command entered had been "NTRON (0:65529)".

The remaining arguments to NTRON constitute the expressions (if any) whose values are to be traced. There may be 0 or more of these expressions. Too many expressions (the limit depends on the sum of the length of the expressions) will result in an OUT OF STRING SPACE error. (You can not increase this limit by CLEARing more string space.) If 2 or more expressions are given, they must be seperated by commas. (Also note that if the first argument (line range) is given, a comma must seperate it from an expression.) The expressions used may be any valid BASIC expression which could properly be put in a single PRINT statement. However, NTRON does virtually no syntax checking on these expression arguments. If you enter an invalid expression, NTRON will more than likely accept it. An error will not occur until the program is run and the first line is about to be traced! At this point you will receive a syntax (or another) error, in what might appear to be a correct line. If you execute an NTROFF and the program line subsequently runs without an error, the cause of the error was probably an improper NTRON expression argument.

NTRON remains active with the arguments given (if any) until an NTROFF is executed. Before a BASIC program line being traced is about to be executed, NTRON's output will be displayed on the video screen. This consists of zero or more lines (depending on the number of arguments given with the NTRON command) of "expression=" expression value, followed by a LIST of the line to be executed. It is important to remember that the values of the expressions (if any) are those that exist before the line is executed. (This is particularly important when the expression contains a division operation. If the divisor has not yet been assigned a value, a DIVISION BY ZERO error will occur.)

NTRON can be of great value in determining BASIC program bugs. Used within a BASIC program, it can provide for the tracing of several disjointed line ranges. This can be accomplished by simply having numerous NTRON commands with different line ranges throughout the program.

Although most output formatting commands (TAB, ;, USING, etc) can be included within "expression", this is not recommended. In particular, it is very difficult to use USING properly. Any "expression" which appears after USING will be formatted with USING (or cause an error if this can not be done). (Remember that the expression arguments must be expressions which may be entered into a single PRINT statement.) Also, the USING variable must be defined before the first line to be traced is executed, or an error will result. It is best to use expressions which do not alter the output format of NTRON, however if formatting commands are used, be prepared for anomalous results.

EXAMPLES:

```
NTRON
```
will cause a trace of all BASIC program lines currently in memory. No expressions will be traced.

```
NTRON(500:1000),X,Y
```
will cause a trace of all BASIC lines between 500 and 1000 (inclusive). The current values of the variables X and Y will be displayed preceeding the line trace (LIST).

```
NTRON INT(X)+A*B,A$
```
will cause a trace of all BASIC program lines. Before each line is traced, the values of the expression INT(X)+A*B and of the variable A$ will be displayed.

```
10 NTRON(:50),RB,TE$,LEN(A$),MID$(A$,LEN(A$)-1,2)
.
.
50 REM
.
.
1000 NTRON(1000:)
```

will cause lines 10 through 50 to be traced after line 10 is executed. Additionally, the values of the expressions shown will be displayed prior to the line trace display. After line 1000 is executed, only line 1000 and any lines that follow will be traced.

line range

Line range specifies a line or a range of lines of a BASIC program. Whenever used it must be enclosed by parentheses. A line range must be followed by a comma or a colon (whichever is appropriate) if it does not terminate a line. A line range can be given in several forms, as detailed below. Several examples illustrate the usage of each form. The examples use constants for "line number", but it should be understood that variables or numeric expressions may also be used. All examples assume the following program is in memory:

```
10 REM
20 PRINT "THIS IS A SHORT PROGRAM."
30 PRINT "ITS ONLY PURPOSE IS TO"
40 PRINT "OCCUPY LINES 10 - 50"
50 REM
```

1). (line number 1 : line number 2)
specifies the range of lines beginning from "line number 1" (inclusive) through "line number 2" (inclusive). If "line number 1" does not exist, the first line number that does exist which is greater than "line number 1" will be used as the lower limit of the range. If "line number 2" does not exist, the first line number that does exist which is less than "line number 2" will be used as the upper limit of the range.

EXAMPLES:

(20 : 40)
includes lines 20, 30 and 40.

(5:25)
includes lines 10 and 20.

(18:100)
includes lines 20, 30, 40 and 50.

2). (: line number)
specifies the range of lines beginning from the first program line through "line number" (inclusive). If "line number" does not exist, the first line number that does exist which is less than "line number" will be used as the upper limit of the range.

EXAMPLES:

(:100)
includes lines 10, 20, 30, 40 and 50.

(: 28)
includes lines 10 and 20.

(:5)
does not include any line and will result in an error.

3). (line number :)
specifies the range of lines beginning from "line number" (inclusive) through the last program line. If "line number" does not exist, the first line that does exist which is greater than "line number" will be used as the lower limit of the line range.

EXAMPLES:

(Ø :)
includes lines 1Ø, 2Ø, 3Ø, 4Ø and 5Ø.

(45 :)
includes line 5Ø.

4). (line number)
specifies a particular line which must exist. If "line number" does not exist, an error will result.

EXAMPLES:

(1Ø)
specifies line 1Ø as the "line range".

(18)
will result in an error since line 18 does not exist.

PAGE display-page {,read/write-page}

The PAGE command allows any one of the 4 available graphics memory "pages" to become the current display and/or read/write page. The arguments of PAGE are int-expressions, and must be in the range of Ø to 3.

The graphics memory pages are used in MODEs 1,2, and 3 graphics. The current display page and the current read/write page do not have to be identical, although they normally are. At power-up, the current display and read/write pages are both page Ø.

NOTE: Older LNW8Ø models only have 1 graphics memory page. The PAGE command will not work properly on these models.

EXAMPLES:

```
PAGE 2
```
will change the current graphics display page to 2, regardless of the current MODE. The graphics read/write page will not be changed.

```
1Ø MODE 2
2Ø PAGE Ø,3 : PCLS
3Ø CIRCLE 5Ø,5Ø,2Ø : PAINT 5Ø,5Ø,5
4Ø FLS : PAGE 3
5Ø GOTO 5Ø
```
will change the graphics MODE to lo-res color in line 1Ø. Line 2Ø changes the current graphics display page to Ø, the current read/write page to 3, and clears this page. Note that the contents of graphics memory page Ø are not affected by the PCLS. Line 3Ø draws a circle and fills it in, but it is not seen until after the display page is changed to 3 in line 4Ø.

```
1Ø PAGE Ø,Ø : MODE 1
2Ø PSET 5Ø,5Ø
3Ø PAGE Ø,1
4Ø PRINT POINT (5Ø,5Ø)
```
will print a Ø (reset) since the point 5Ø,5Ø is on graphics memory page Ø, but page 1 is being read. (This assumes that the point 5Ø,5Ø isn't set on page 1, of course.) Note that this may cause some confusion, since the point 5Ø,5Ø is set on the graphics memory page being displayed.

PAINT X,Y {,RESET}
PAINT X,Y {,paint color} {,border color}

In black & white graphics MODEs (Ø & 1), PAINT "paints" (fills in) an area by either setting or resetting all graphics dots. In color graphics MODEs (2 & 3), PAINT "paints" in the paint color specified, up to the border color specified. The first argument form is used for black & white graphics, the second form is for color graphics.

The first 2 arguments give the X,Y coordinates of a point (any point) in the region to be painted. The arguments are mandatory, and must be int-expressions. The point X,Y must be a displayable point within the current graphics MODE or an ILLEGAL FUNCTION CALL error will result.

In black & white graphics, the third argument (optional) specifies the "color" to paint in. If it is present, it must be the BASIC keyword RESET. If this argument is missing, it defaults to SET. PAINT with the SET option will set all points from the given starting point within an enclosed area. An area is bounded by the sides of the display and points which are set. PAINT with the RESET option will reset all points in an area bounded by the sides of the display and points which are reset.

In color graphics, the third & fourth arguments are optional. The third argument specifies the color to paint in. If it is present, it must be an int-expression in the range of Ø-7 (see COLOR). If it is missing, it defaults to the current COLOR value. The fourth argument specifies the border color which bounds the area to paint. If it is present, it be an int-expression between Ø-7. If it is missing, it defaults to the current COLOR value.

Unlike other commands, PAINT can be interrupted in the middle of its execution by holding down the <BREAK> key. This will terminate the execution of the command; it cannot be CONTinued. (If issued, a CONT command will continue with the execution of the statement immediately following the interrupted PAINT command.)

The PAINT command cannot be used successfully with the PLOT command to paint patterns. If the PLOT command is being used to generate patterns with other graphics commands, a command must be issued ('PLOT') to force graphics to "normal" before PAINTing.

NOTE: Due to the nature of the color mapping in MODE 3 (hi-res color) graphics, it is very difficult to use PAINT successfully in MODE 3. Problems will be encountered whenever the paint color and the border color differ.

EXAMPLES:

```
10 CLS : MODE 1 : PCLS
20 LINE 0,0,150,10,SET,B
30 PAINT 1,1
40 GOTO 40
```

will draw a box (line 20) and fill it in (line 30). (This could also have been done using the F option of LINE, of course.) Any point within the box (but not on it) would have worked; there is nothing special to the point 1,1. Replace line 30 with:

```
30 PAINT 150,10
```

and RUN. (Remember that holding down the <BREAK> key will stop PAINT from painting.) This example demonstrates why the point given must not be on the perimeter of the area to be painted.

```
10 MODE 0 : CLS
20 LINE 0,0,50,20,SET,B
30 PLOT 2,2
40 PAINT 1,1
50 GOTO 50
```

demonstrates that PLOT cannot be used effectively with PAINT. Changing line 30 to:

```
30 PLOT
```

will properly set the graphics to "normal" to insure that PAINT works as it should. This is necessary only when PLOT has been used prior to PAINT (since the last RUN or NEW) to set a graphics pattern.

```
10 CLS : MODE 1 : PCLS
20 DRAW "B,M240,96,S8,R5,U5,R5,D15,E10,R5,G20,H25,E20,D25"
30 FOR X=1 TO 1000 : NEXT
40 PAINT 241,97
50 FOR X=1 TO 1000 : NEXT
60 PAINT 241,97,RESET
```

will draw a figure, paint it, and then "erase" it. Line 20 draws a complex figure, and line 30 causes a short delay to allow you to see the figure before it is painted. Line 40 paints the figure, and line 50 causes another short delay. Finally, line 60 paints the figure using the RESET option, which resets all set points in an area bounded by reset points. In the example, this causes the entire figure to be reset.

```
10 MODE 2 : PCLS : FLS
20 COLOR 1 : CIRCLE 50,50,20
30 COLOR 2 : CIRCLE 50,50,10
40 PAINT 50,50
50 PAINT 50,50,3,1
60 PAINT 50,50,,1
70 PAINT 50,50,0,0
```

demonstrates the PAINT command in lo-res color graphics (MODE 2). In line 10 the MODE is set to lo-res color, the background color is set to white (0), and the lo-res color graphics sreen is enabled by whiting the text screen. Line 20 sets the current default COLOR to green (1), and draws a circle of radius 20. Line 30 sets COLOR to yellow and draws a concentric circle of radius 10. Before line 40 is executed, there is a small yellow circle inside a larger green circle. Line 40 paints in yellow, up to a yellow border (due to the defaults of the missing arguments). This results in a solid yellow circle inside a larger green (not solid) circle. Line 50 paints in red (3), up to a green (1) border, resulting in a large solid red circle with a green circumference. Line 60 paints in yellow (the last COLOR value) up to a green border, changing the solid red circle to solid green. Line 70 resets the solid green circle by painting in white (the background color), up to a white border.

PCLS {int-expression}

This command is used to set the graphics screen to a specified value. PCLS used without a following expression sets all graphics points off. This is equivalent to a PCLS 0 command. PCLS with an optional argument sets graphics points as specified by the argument's value. The value specified must be in the range 0 to 7 inclusive. If the evaluated expression (i.e. value) falls outside of this range, an ILLEGAL FUNCTION CALL error will occur.

The PCLS command is useful in clearing (PCLS 0) or "filling" (PCLS 7) the hi-res B/W (MODE 1) or hi-res color (MODE 3) graphics screen. In MODE 2 (lo-res color) graphics, the PCLS command is used to set the background color. The argument specifies the color (see COLOR command) to set the background color to. All subsequent MODE 2 graphics involving RESETting points will set points to this background color.

EXAMPLES:

```
10 MODE 1
20 PCLS
```
will turn on the hi-res B/W graphics screen and "clear" the graphics screen.

```
10 MODE 1
20 PCLS 0
```
is equivalent to the above example.

```
10 MODE 1
20 PCLS 7
```
will turn on the hi-res B/W graphics screen and fill in the screen - i.e. turn the entire screen white.

```
10 MODE 2
20 FLS
30 PCLS 3
```
will turn on the lo-res color graphics screen, map all points "on" (FLS whites the text screen), and turn the screen red (COLOR 3 = red).

```
10 MODE 2
20 FLS
30 FOR I=0 TO 7
40 PCLS I
50 FOR J=1 TO 500
60 NEXT J,I
```
will turn on the lo-res color graphics screen, map all points "on", and alternate between all 8 colors - delaying between each.

```
10 MODE 2 : FLS
20 PCLS 4 : COLOR 7
30 LINE 0,0,50,50,SET,BF
40 FOR X=1 TO 1000 : NEXT
50 LINE 0,0,50,50,RESET,BF
```

will set the MODE to lo-res color, "white" the text screen (enabling color graphics), set the background color to magenta (4), set the current (foreground) COLOR to black (7), draw a solid, black rectangle, and then (after a short delay), reset the rectangle to the background color of magenta.

```
10 MODE 3
20 PCLS 7
30 FLS 2*9+64
```

will set the MODE to hi-res color, set all hi-res graphics points on, and "map" the color yellow (2) onto all graphics points (thereby turning the hi-res graphics screen yellow).

```
10 MODE 3
20 FLS (RND(8)-1)*9+64
30 PCLS RND(8)-1
40 FOR I=1 TO 500 : NEXT
50 GOTO 20
```

will turn on the hi-res color mode, map all graphics points to a random color, "fill" the graphics screen with random vertical line patterns and loop.

PGET array-name,X1,Y1,X2,Y2
PGET array-name,old-color TO new-color

The first argument form of the PGET command saves the graphics data from the specified rectangle into the specified array. (The array should be dimensioned large enough to hold the desired data prior to the PGET, or a SUBSCRIPT OUT OF RANGE error will occur.) PGET is to be used in conjunction with the PPUT command to speed up the displaying of complex graphics images. The second argument form of PGET changes the data already in an array.

The first argument, array-name, must be a string constant. It denotes the name of a dimensioned array. The array must not be a string array or a TYPE MISMATCH error will occur. It is recommended for clarity that an integer array be used with PGET and PPUT.

The arguments X1,Y1,X2,Y2 are int-expressions, and give the coordinates of the two corners of the desired rectangle. The point X1,Y1 should be the upper-left corner point of the rectangle, and the point X2,Y2 should be the lower-right corner point. The rectangle is limited to a size of 256 X 256.

The second argument of the second argument form is similar in syntax to the "initial-value TO final-value" portion of a "FOR variable-name = initial-value TO final-value" statement. That is, "int-expression TO int-expression", where the evaluated expressions in this case represent a color, and must be in the range of Ø-7. If the data in the specified array represents MODE 2 or 3 (color) graphics data, then this argument form allows all points of a particular color to be changed to another color. If the data in the array did not come from a PGET of MODE 2 or 3 graphics data, the result will be unpredictable.

The following formulas will be useful in calculating the minimum dimension of an integer array to be used with PGET:

```
# of points = (X2 - X1 + 1) * (Y2 - Y1 + 1)
Minimum DIM = (# of points / 16) + 2     (MODES Ø & 1)
Minimum DIM = (# of points / 4) + 2      (MODES 2 & 3)
```

EXAMPLES:

```
1Ø DIM A%(3Ø)
2Ø MODE Ø : CLS
3Ø LINE Ø,Ø,2Ø,2Ø,SET,B
4Ø PGET A%,Ø,Ø,2Ø,2Ø
5Ø FOR X=Ø TO 1ØØ STEP 25
6Ø PPUT A%,X,25
7Ø NEXT
8Ø GOTO 8Ø
```

demonstrates PGET in MODE 0. The rectangle drawn in line 30 is read into the array A% by line 40. The loop in lines 50-70 causes 5 images of the data in A% to be displayed.

```
10 DEFINT A : DIM A(250)
20 MODE 1 : PCLS : CLS
30 CIRCLE 240,96,30,,,,90
40 FOR X=1 TO 10
50 LINE 210,64,270,127,SET,B : LINE 210,64,270,127,RESET,B
60 NEXT
70 PGET A,210,64,270,127
80 PPUT A,150,64 : PPUT A,270,64,MERGE
```

is an example of PGET & PPUT in MODE 1. The flashing rectangle created by the loop in lines 40-60 shows the exact area that is transferred to the array A in line 70. In line 80, 2 images of the data in A are displayed. (Note how much faster the data is displayed with the MERGE option of the PPUT command.)

```
10 DEFINT C : DIM CI(375)
20 MODE 2 : FLS : PCLS 5
30 COLOR 2 : CIRCLE 50,50,10 : LINE 40,40,60,60,SET
40 PAINT 50,45,4 : PAINT 50,60,1
50 PGET CI,35,30,70,70
60 PGET CI, 1 TO 6 : PGET CI,4TO7
70 PPUT CI,100,100
80 PGET CI,5 TO 1 : PPUT CI,100,50
90 DO : UNTIL (INKEY$<>"")
100 PCLS 1 : PPUT CI,50,50
```

demonstrates PGET & PPUT in MODE 2 graphics. Line 20 sets the background color to blue (5). A yellow (2) circle with a line through it is drawn by line 30. Line 40 paints half the circle magenta (4), and half green (1). Line 50 reads the area containing this circle into the array CI. Line 60 changes the circle to half blue-green (6), and half black. Line 70 displays the new circle. Line 80 changes the background portion of the circle from blue to green, and displays it. Line 90 delays until a key is pressed. Line 100 clears the graphics screen and sets the background color to green. The circle is then displayed again, looking much better now that the background color has been changed to match the data in CI.

PLOAD filespec {,MERGE}

The PLOAD command loads a disk file into graphics memory. Normally, the disk file to be loaded is a graphics memory-image created by the PSAVE command. However, PLOAD does not restrict the loading of any kind of file.

The first argument indicates the disk file to be loaded. When the second argument is not used, the contents of this disk file will replace the current contents of graphics memory (regardless of the current graphics mode). If the second argument **is** used, it indicates that the contents of the file are to be merged (OR'd) with graphics memory. The MERGE option provides a way of loading various portions of the screen at different times. This can be accomplished by loading several PSAVE files, each of which were created with only parts of the screen filled. When a file is MERGEd with graphics memory, only the area which contained an image when the file was created will be affected; the rest of graphics memory will be unaltered.

EXAMPLES:

```
10 CLS : PCLS : MODE 1
20 PLOAD "PAGE1/GRF" : REM GET 1ST PAGE OF DISPLAY
   .
   .
   .
```

when executed, will load the file "PAGE1/GRF" from disk into graphics memory.

```
PCLS : MODE 1 : PLOAD"TEST/GRF"
```

will load the file "TEST/GRF" into graphics memory.

```
10 DATA "FIG1/GRF", "FIG2/GRF", "FIG3/GRF", "FIG4/GRF"
20 FOR X=1 TO 4 : READ FIG$(X) : NEXT
   .
   .
   .
10000 PCLS : FOR I=1 TO 4 : PLOAD FIG$(I), MERGE : NEXT : RETURN
```

lines 10-20 read in 4 filenames into the array FIG$. When the subroutine at line 10000 is called, the graphics screen is cleared, and each of the 4 files is loaded into graphics memory. Since the "MERGE" option is used, the contents of the files will be merged into graphics memory.

PLOT {SET-count {,RESET-count}}

The PLOT command initializes the SET (on) and RESET (off) counters used by all LNWBASIC graphics commands. These counters determine the "pattern of dots" which make up a line, circle, DRAW-figure, etc. Only SET graphics are affected by PLOT; there is no way to affect RESET graphics. The command PLOT 1,1 will cause the line drawn by LINE 0,0,20,0,SET to consist of alternating SET and RESET dots; it has no affect on LINE 0,0,20,0,RESET.

Both arguments of PLOT must be integer expressions in the range of 0-255. The first argument determines the number of consecutive points that will be SET in all subsequent graphics drawn. The second argument determines the number of consecutive RESET points that will be drawn after all the consecutive SET points have been drawn. For example, PLOT 5,5 will cause a pattern of 5 dots on (SET), followed by 5 dots off (RESET). This pattern repeats until another PLOT command is issued. The pattern does not start over for each new figure drawn; instead, the pattern continues its sequence uninterrupted. If the second argument is missing, it defaults to the value of the first argument. For example, PLOT 1,1 is identical in meaning to PLOT 1.

PLOT with no arguments will force graphics to the "normal" pattern of all dots on (SET). This can also be accomplished by PLOT n,0 (n any value between 1 and 255). (This is because the RESET count is 0.) PLOT 0, as well as PLOT 0,n (n any value between 0 and 255), will force the graphics to a pattern of all dots off (RESET). (This is because the SET count is 0.) PLOT 0 can be used to erase various graphics such as circles, DRAW-figures, etc., which cannot otherwise be easily erased.

The PLOT command affects all LNWBASIC graphics commands, including PAINT. In order for PAINT to work properly, graphics must be set to "normal" prior to PAINT by issuing a PLOT (no arguments) command.

EXAMPLES:

```
10 MODE 1 : FLS
20 PLOT 3,2
30 LINE 100,70,200,130,SET,B
40 PLOT
50 CIRCLE 150,100,30
60 PAINT 150,100
70 GOTO 70
```

draws a box using the PLOT command. Line 20 causes all subsequent LNWBASIC SET graphics to use a pattern of 3 dots "on" (SET) and 2 dots "off" (RESET). Line 30 draws a box using this pattern. Line 40 returns graphics to normal (all dots SET). This allows a solid circle to be drawn and painted by lines 50-60.

```
10 MODE 2 : PCLS 5 : FLS : COLOR 4
20 PLOT
30 CIRCLE 64,30,20,,,,45
40 PLOT 0
50 CIRCLE 64,30,20,,,,45
60 GOTO 20
```

is an example of how to erase (RESET) a circle. The circle is drawn in line 30, after graphics was set to all dots "on" in line 20. Line 40 sets graphics to all dots "off", causing the CIRCLE command in line 50 to erase the circle.

```
10 CLS : MODE 0
20 FOR Y=0 TO 47
30 PLOT Y+1
40 LINE 0,Y,127,Y,SET
50 NEXT
60 GOTO 60
```

demonstrates various effects of PLOT on a simple line. After running this example, replace the first Y coordinate in line 40 with 0 and run the new program. Then, change the first Y coordinate in line 40 back to Y, replace the second Y coordinate with 0, and run this program. Finally, replace both Y coordinates in line 40 with 0 and RUN.

POFF
PON

The PON command, when executed, will cause video output to be echoed to the printer. The printer must be on and ready or the system will "hang". The POFF command is used to disable the video to printer echo. The command may be used with custom printer driver routines (serial,TRS232,etc.) providing the routine has been loaded before issuing a PON. See SPOOLON for the use of it and PON simultaneously.

EXAMPLES:

```
1Ø PON
2Ø PRINT"THIS IS A TEST"
3Ø POFF
```
will display the line "THIS IS A TEST" on the video display and also output the line to the printer.

```
PON : LIST
```
this command will cause a program in memory to be listed on the video display and to also be output to the printer.

POINT(X,Y)

The POINT command is used to return the status or color of the specified graphics point. The arguments X,Y are mandatory, and may be constants, variables, or integer expressions.

In MODEs 0 or 1, POINT returns the value 0 if the point is not set (reset) or the value -1 if the point is set. This is the same as found in LEVEL II or DISK BASIC. The value -1 is used by BASIC as TRUE. Any value other than -1 is considered FALSE. If the evaluated expression, "IF POINT(X,Y) " is TRUE (i.e. = -1) then the rest of the statement line is executed.

POINT is used in MODEs 2 & 3 to return the color of the tested point. If the point 100,20 had been PSET with COLOR 3, then POINT(100,20) would be equal to 3. In MODE 2 a point is always on (set), and POINT will always return a value of 0-7 to denote its color. In MODE 3, however, a point can be off (reset), as well as being set to a color 0-7. To identify a point which is off in MODE 3, a value of -1 will be returned. Note that -1 in this case has the opposite meaning as in MODEs 0 & 1.

POINT will always return a value of 0 if the point specified falls outside of the displayable area.

EXAMPLES:

```
10 MODE 0
20 SET(10,10)
30 IF POINT(10,10) THEN PRINT"POINT 10,10 IS SET"
```

will set the graphics mode to lo-res B/W, set point 10,10 and print the statement, "POINT 10,10 IS SET" to the video screen.

```
10 MODE 0
20 CLS
30 PRINT POINT(127,47)
```

will set the mode to lo-res B/W, clear the text screen and print the value 0. This is due to the fact that in "normal" graphics mode (128 X 48 B/W), CLS clears all grahics points. If a point is <u>not</u> set, POINT will return a 0 value.

```
10 MODE 2 : PCLS : FLS : COLOR 3
20 LINE 0,0,100,100,SET
30 PRINT POINT(50,50)
```

will set the mode to lo-res color mode, clear the graphics screen, enable all graphics points (by "whiting" the text screen), set the default color to red, draw a red diagonal line and print the value 3. This is due to the fact that in MODE 2 graphics, POINT returns the color value of the tested point.

```
70 PRINT POINT (10,190)
```

will, if added to the the above example, print the value 0. A PCLS command, without an argument, sets all graphics points to 0.

PPUT array-name,X,Y {,option}

Where option = MERGE or RESET.

The PPUT command transfers graphics data stored by the PGET command from an array to graphics memory. "array-name" is a string constant, specifying the array containing the data to transfer to graphics memory. The arguments X,Y are int-expressions, specifying the upper-left corner point of the rectangular area where the data is to be transferred.

If no option is given, PPUT will simply transfer the data to graphics memory exactly as stored in the specified array. In other words, the contents of the array will replace the contents of graphics memory in the area given.

If the MERGE option is used, only those points which are set (i.e., not equal to the background color), will be transferred to graphics memory. The MERGE option is usually considerably faster, but it can produce different results from transferring the entire array.

If the RESET option is used, those points which are set (i.e., not equal to the background color), will be reset when transferred to graphics memory.

EXAMPLES:

```
10 MODE 0 : CLS : DEFINT A : DIM A(30)
20 LINE 35,10,55,20,SET,B
30 LINE 38,13,52,17,SET,BF
40 PGET A,35,10,55,20
50 PPUT A,40,15
60 PPUT A,45,20,MERGE
```

demonstrates the difference between using and not using the MERGE option.

```
10 DEFINT A : DIM A(975)
20 MODE 2 : PCLS 7 : FLS
30 COLOR 0 : CIRCLE 50,50,20
40 PGET A,25,10,75,85
50 PCLS 5 : PPUT A,0,50
60 PGET A,7 TO 5 : PGET A,0 TO 1
70 PPUT A,60,50,MERGE
80 DO : UNTIL (INKEY$<>"")
90 PPUT A,0,50,RESET
100 DO : UNTIL (INKEY$<>"")
110 PGET A,5 TO 7 : PPUT A,0,50,RESET
```

demonstrates the RESET option. Line 20 sets the mode to lo-res color, the background color to black (7), and enables the color graphics display by whiting the text screen. Line 30 sets the default COLOR to white (0), and draws a white circle. Line 40 copies an area of graphics memory around this circle to the array A. Line 50 sets all of graphics memory to blue (5), thereby changing the background color, and transfers the white circle and surrounding area to graphics memory. Since the background color was changed, the

previously "transparent" black background is now quite visible. Line 6Ø changes all black points in A to blue, and also changes the color of the circle from white to green. Line 7Ø transfers the new contents of A to graphics memory. Note that the black background is now blue, and is therefore not noticeable. Line 8Ø delays until a key is pressed. Line 9Ø transfers the data in A to graphics memory using the RESET option. Since only the circumference of the circle is "set" (i.e., contains points not equal to the background color), only those points are reset. This leaves the black area still visible. After a key is pressed, line 11Ø will reset this area, by changing the blue points in A back to black, and doing another PPUT with the RESET option.

PRESET X,Y
PSET X,Y

The PRESET command in MODEs Ø,1 or 3 will turn off the specified point. In MODE 2, the specified point will be set to the background color (see PCLS). The PSET command turns on the specified point. In MODE 2 (lo-res color) and MODE 3 (hi-res color), the point will be set to the color specified in the last COLOR command.

In MODE Ø, X may vary from Ø to 127 and Y from Ø to 47. PSET and PRESET in MODE Ø are equivalent to SET and RESET in "normal" BASIC. In MODEs 1 or 3, X may vary from Ø to 479 and Y from Ø to 191. In lo-res color mode (MODE 2), X should be in the range Ø to 159 and Y in the range Ø to 191. If the value of X or Y is outside this specified range, no operation will be performed.

The integer portion of floating point numbers will be used in determining the appropriate action - i.e. PSET 1.5,5Ø.6666 will result in the point 1,5Ø being set.

EXAMPLES:

```
1Ø CLS
2Ø PCLS
3Ø MODE 1
4Ø PSET 24Ø,96
```

will clear the text screen, clear the graphics screen, enter the hi-res B/W mode and set the point approximately in the middle of the screen.

```
1Ø MODE Ø
2Ø FOR I=Ø TO 127
3Ø PSET I,2Ø
4Ø NEXT I
```

will set the points in the horizontal line determined by Ø,2Ø and 127,2Ø. PSET I,2Ø in this example is equivalent to SET(I,2Ø) in the "regular" BASIC.

```
1Ø MODE 2
2Ø FLS
3Ø FOR I=1 TO 5Ø
4Ø COLOR RND(8)-1
5Ø PSET RND(128)-1,RND(192)-1
6Ø NEXT I
```

will enter the lo-res color graphics mode, enable all graphics points, and turn-on 5Ø random points of random color.

```
10 MODE 3 : PCLS
20 COLOR 1
30 FOR I=0 TO 191
40 PSET I,I
50 NEXT I
60 FOR I=0 TO 191
70 PRESET I,I
80 NEXT I
```

will turn on the hi-res color screen, clear graphics memory, set the color to reen, and draw a diagonal line from the top left corner of the RGB monitor to about the middle bottom of the screen. The line will then be erased.

```
10 CLS
20 PCLS
30 MODE 1
40 FOR I=0 TO 479
50 PSET I , SIN(I*2*3.14/479)*96+96
60 NEXT I
```

will clear the text screen, clear the graphics screen, set the mode to hi-res B/W and draw a sine-wave on the video display.

PSAVE filespec

The PSAVE command saves graphics memory to a disk file. The mandatory argument, filespec, indicates the disk file to use or create. All of the contents of graphics memory, regardless of the current graphics mode, are saved to the file specified. The contents are saved in a special format using space compression, to save disk space. Depending upon the contents of graphics memory, the file contents can range in size from less than 1K bytes (all Ø's in graphics memory), to 16K bytes (all 1's in graphics memory). The file may be loaded into graphics memory from disk using the PLOAD command.

NOTE: While it is not required, it is suggested that the file extension, "GRF" be used for graphics screen files.

EXAMPLES:

```
PCLS : PSAVE "BLANK/GRF"
```
will save a file which will clear the graphics screen when PLOADed from disk.

```
.
.
.
5ØØ PSAVE GR$(X)+"/GRF"
```
will save the current contents of graphics memory to the disk file determined by the specified string expression.

QUICKEY

This command toggles the quick key entry method on and off. The first time the command is given, quick key entry is enabled; the next time it is given, quick key entry will be disabled. Quick keys have been pre-defined and can not be modified; they are totally distinct from defined keys (see DEFKEY).

Quick key entry allows you to enter entire BASIC keywords with just one keystroke. While holding down the <CONTROL> key, press the desired key. If the key you press is a defined quick key, then the associated keyword will immediately be displayed on the video screen. If the key is not defined as a quick key, the <CONTROL> key will be ignored, and the key will be displayed as usual.

The table below lists BASIC keywords and associated keys, organized alphabetically by the BASIC keyword. An attempt has been made to make quick key entries easily recallable. Many of the keywords are mnemonic (that is, the first letter of the keyword matches the associated key). Exceptions are the BASIC string keywords (that is, keywords with "$" in them) which have been assigned to numeric keys. Other keywords also use mnemonics different from the BASIC keyword (Xfer (X) for GOTO or Bring (B) for LOAD, for example). Some keywords use keyboard position to aid in recalling their assigned quick keys (RETURN (H), for example, is next to GOSUB (G)). Of course, some keywords had to be assigned at random (you can't win them all!).

AUTO	— A	KILL"	— K	READ	— Y
CHR$(	— 4	LEFT$(	— 1	RESET	— Ø
CLEAR	— Z	LIST	— L	RETURN	— H
CLOSE	— J	LOAD"	— B	RIGHT$(	— 3
CMD"	— C	MEM	— M	RUN <ENTER>	— R
DATA	— D	MID$(	— 2	SAVE"	— S
DIM	— U	MKD$	— 8	SET(	— W
ELSE	— E	MKI$	— 9	STR$(	— 6
FOR	— F	MKS$	— Ø	STRING$(	— 7
GOSUB	— G	NEXT	— N	THEN	— T
GOTO	— X	OPEN"	— O	VARPTR(	— V
INKEY$	— 5	PEEK(	— P	blank:blank	— :
INPUT	— I	POKE	— @		

(The above table may be photocopied and placed on your LNW8Ø for easy reference.)

REPEAT

The REPEAT command implements the keyboard auto repeat and "beep" toggle. The first time this command is executed, the routine takes effect. A key depressed for about half a second will repeat. If a speaker/amplifier is hooked to the cassette-out line of the LNW 8Ø (the line that goes to the AUX on the cassette recorder), a beep will be heard upon depressing a key. The second time this command is executed, the auto repeat and "beep" are turned off. The REPEAT command should not be used with Radio Shack's lower case software as lower case characters cannot be input with REPEAT in use.

NOTE: The use of REPEAT in double width character mode causes keyboard bounce while its use with the lower CPU speed causes character loss.

EXAMPLES:

REPEAT
will enable key repeat and "beep" if this is the first time the command is issued.

```
2Ø REPEAT
3Ø INPUT"NAME?",A$
4Ø REPEAT
```

if a REPEAT command has already been issued, as in the preceding example, line 2Ø will turn off the auto repeat. Line 3Ø will input A$ without allowing auto-repeat. Line 4Ø will again enable repeat/"beep".

REST line number
REST #"label"

This command allows for selected reading of DATA statements. The argument specifies a BASIC program line number containing a DATA statement. After the REST command is executed, the next READ will start with the first item in the DATA statement within the line just RESTored.

The "line number" argument form may be given as a constant, variable, or expression. The second REST argument form (#"label") refers to a line which has been so labeled. With either argument form, if the line referenced does not contain a DATA statement, an ILLEGAL FUNCTION CALL error will result.

EXAMPLES:

10 REST 40000
will reset the DATA pointer so that the next READ will start with the first item in the DATA statement in line 40000.

10 REST A(X)*100
will cause the next READ to start with the first item in the DATA statement in the line determined by the product of A(X) and 100.

RS232

This command initiates dialog that will result in the initialization of the RS-232-C interface. Current status of the UART sense switches will be displayed within vertical bars. If the baud setting is not within the range possible to set with RS232, the word "OTHER" will be displayed. To use the parameters that were switch set, hit "ENTER" in reply to each question. To change settings, enter the appropriate number. Recommended settings are denoted by an asterisk.

EXAMPLE:

RS232
will result in the following dialog:

```
            RS-232-C INITIALIZATION

BAUD RATE (Ø=11Ø, 1=134.5, 2=3ØØ*) !2!? Ø
PARITY (Ø=ENABLE*, 1=DISABLE) !Ø!? Ø
STOP BITS (1*, 2) !1!? 1
WORD LENGTH (5, 6, 7*, 8) !7!? 7
PARITY (Ø=ODD, 1=EVEN*) !1!?
```

the numbers after the "?" are user entered. The numbers inside the vertical bars are the current UART switch settings. This dialog has resulted in a baud rate of 11Ø, parity enabled, 1 stop bit, 7 bit word length excluding parity bits, and even parity.

RSIN {,R}

The RSIN command without the trailing ",R" will enable the input of characters from either the keyboard or the RS-232-C interface. Provided processing is not too lengthy between input of characters, rates up to and including 300 baud can be supported. The RSIN command with the trailing ",R" (i.e. RSIN,R) will inhibit input from the serial port. Further input will only be from the keyboard.

This command used in conjunction with the RSOUT command makes possible the implementation of terminal programs, automatic logon to time share networks, and remote terminal input and output in BASIC!

NOTE: Do not implement the RSIN command if a RS-232-C board or similar serial I/O device is not installed. An infinite loop will occur and a system reset will be necessary.

EXAMPLES:

```
10 RSIN : RSOUT
20 LPRINT : LPRINT
30 LINEINPUT A$ : IF A$<>"USER ID?" THEN 30
40 LPRINT"123,45,6789" : REM USER LOGON ID
```

this example shows how simple an automatic logon procedure for a time share system is to implement. In line 10, the RSIN command enables input from the serial port. "RSOUT" routes printer output to the serial port. Line 20 outputs two carriage returns. This is usually necessary to establish baud rate synchronization with a time share network. In line 30, a character stream, up to a carriage return, is stored in the string variable A$. If A$ is not equal to "USER ID?" the program loops at line 30. Once the IF/THEN clause is true, the string "123,45,6789" is output to the serial port in line 40. Further processing could include an interactive terminal program - etc.

```
100 RSIN,R
```

will inhibit any further input from the serial port.

RSOUT {,R}

The RSOUT command without the trailing ",R" will route printer output to the RS-232-C interface. The serial port should be initialized before implementing this command. While any baud rate possible with RS-232-C is supported, no provision has been made for outputting nulls after a carriage return. Nulls are required by some serial printers and other devices. The RSOUT command with the trailing ",R" (i.e. RSOUT,R) will reset printer output to the condition existing before the RSOUT command. Refer to the RSIN command.

EXAMPLES:

```
10 RSOUT
20 FOR I=1 TO 10 : LPRINT"HELLO" : NEXT I
30 RSOUT,R
```

this example uses the RSOUT command in line 10. The 10 "HELLO"'s output to the printer in line 20 will go to the serial output port. Line 30 resets to conditions existing before line 10 was executed.

```
10 RSOUT : PON
20 PRINT"THIS IS A TEST"
30 POFF : RSOUT,R
```

line 10 first routes printer output to the RS-232-C interface and then echoes all video output to the printer (i.e. the serial port). Line 20 outputs the line "THIS IS A TEST" to the video display and the serial port. Line 30 turns off the video to printer echo and reroutes further printer output as per the conditions existing before line 10 was executed.

SAVEKEY filespec

The SAVEKEY command saves the current defined key list (see DEFKEY) to the specified disk file. The file may be an LNWBASIC module, or a file containing only defined keys. If the file is an LNWBASIC module, the defined key list within the module on disk will be replaced by the current defined key list. Whenever the LNWBASIC module is subsequently executed, the newly saved key list will appear as the current defined key list. If the file is not an LNWBASIC module, then LOADKEY must be used to load the key list from the file.

NOTE: While it is not required, it is suggested the extension, "KEY" be used for defined key files.

EXAMPLES:

SAVEKEY "LNWBASIC/CMD:Ø"
will save the current defined key list to the file specified. Whenever this LNWBASIC module is executed, the defined keys will be those defined keys just saved.

2ØØ SAVEKEY "F1/KEY"
when executed, will save the current defined key list to the file "F1/KEY". The defined key list from this file can be loaded at any time by the command 'LOADKEY "F1/KEY" '.

SOUND SET value
SOUND freq1 {,freq2} {,step} {,duration} {,repeat}

The SOUND command produces many varied sound effects. It's versatility ranges from making music to imitating phaser-gun sound effects.

The first form of the SOUND command sets a constant duration value which is used for all subsequent SOUND commands. This value ranges from 1-255, and is initially set at 128. However, it is not reset by NEW or RUN. It is up to you to keep track of the current value.

The second form of the SOUND command plays the desired tone(s). All arguments range from 1 to 255. 0 and values greater than 255 will result in an ILLEGAL FUNCTION CALL error. Only the 1st argument of SOUND is mandatory; all other arguments will use a default value if missing. You must use ",," (comma, comma) as a place-holder for missing arguments if you desire to specify a later argument. For example, SOUND 50,,,100 will use default values for the missing 2nd & 3rd arguments, and the specified values (50 & 100) for the 1st and 4th arguments. The 5th argument is missing entirely and will also default.

The 1st & 2nd arguments of SOUND (freq1 & freq2) specify a range of frequencies to play. If the 2nd argument is missing, it will default to freq1. Thus, SOUND 100 and SOUND 100,100 are functionally identical. The SOUND command will "step" up or down from freq1 to freq2, depending upon their values. For example, SOUND 100,150 will step from 100 to 150, while SOUND 150,100 will step from 150 to 100.

The 3rd argument of SOUND specifies the step count to use while stepping from freq1 to freq2. If this argument is missing, a step count of 1 will be used. The step count is analogous to STEP in a FOR/NEXT loop, except that a negative count is not allowed. As explained above, SOUND will automatically step down if freq1 is greater than freq2.

The 4th argument of SOUND specifies the duration for each frequency in the range. If missing, the longest duration (255) will be used. The SOUND SET command alters the effect of this argument, making it possible to play a song in 2 different tempos just by changing the SET value. A little experimenting will help clarify the interaction between this argument and the SET value.

The 5th (last) argument of SOUND specifies the number of times to repeat the entire frequency range determined by the previous 4 arguments. For example, SOUND 100,150 will step from 100 to 150 one time, while SOUND 100,150,,,5 will repeat the sequence five times. If this argument is missing, a repeat count of 1 will be used. The repeat count allows several minutes of SOUND to be played from a single command. If desired, you may stop SOUND by holding down the <BREAK> key.

EXAMPLES:

```
10 SOUND SET 160 : GOSUB 20 : SOUND SET 95 : GOSUB 20 : STOP
20 SOUND 30 : SOUND 27 : SOUND 34 : SOUND 69
30 SOUND SET 255 : SOUND 45 : RETURN
```

demonstrates a use of the SOUND SET value. (The notes shown are for the Model I. Model III users should find the matching values to use, and should also increase the 2 SET values in line 10 to 185 and 110 respectively.) The 2 SET values in line 10 are used to play the notes in line 20 in different tempos. The SET in line 30 is used to achieve the longest possible single tone.

The following examples all assume a SOUND SET value of 128:

```
SOUND 50,100,5,80
```

will play every 5th frequency from 50 to 100. A duration of 80 is used.

```
SOUND 100,50,5,80
```

will play the same frequencies as above in reverse order.

```
SOUND 50,100,5,80,2
```

will play the entire sequence 2 times.

```
FOR X=1 TO 8 : SOUND 50,100,5,X,10 : NEXT
```

demonstrates the effect of changing the duration argument.

SPOOLOFF
SPOOLON filespec

The SPOOLON command directs printer output to the specified disk file. The command is disabled and the disk file closed by the SPOOLOFF command. Spooling, used in conjunction with the DESPOOL command, allows for more efficient hardcopy output. Printouts that normally would require a large amount of time to output may be quickly spooled. Later the created file may be despooled while other computer operations are being performed.

SPOOLON may be used before a PON command. This allows all output to the video display to also be output to a disk file. This capability is useful for keeping track of programs run, or for debugging purposes. A POFF command should be issued before SPOOLOFF if PON was previously used and a printer is not "ready". If this is not done, and the printer is not powered up and ready, the system may "hang".

Only one file at a time may be spooled to. If there is a problem initializing the requested file, or SPOOLON has already been issued, an ILLEGAL FUNCTION CALL will occur. If an error such as DISK SPACE FULL occurs during spooling, an error message will be displayed and the spooled-to file will be closed. If PON was used, a POFF command will be issued.

NOTE: While it is not required, it is suggested that the file extension, "SPL" be used for SPOOL files.

EXAMPLES:

```
10 SPOOLON"HOLD/TXT"
```
this creates a file "HOLD/TXT" and future output directed to the printer will be placed in the file.

```
100 SPOOLON"JUNK/PRT.PASSWORD:1"
```
will create a file "JUNK/PRT" on drive 1 with the password, "PASSWORD". Printer output will be directed to this file.

```
10 SPOOLON"TEST/HLD"
20 PON
30 PRINT"TESTING ... 1,2,3"
40 POFF : SPOOLOFF
```
this program will create the file, "TEST/HLD", output the line "TESTING ... 1,2,3" to both the video display and the file, stop video-to-printer echo, and close the file.

XSTR$(string)

This command allows a string to be executed as though it were a BASIC statement. Any statement or statements which can appear within a BASIC program line or in command mode may be used. The only exception to this is the XSTR$ command itself; the XSTR$ command may not be nested. That is, the command XSTR$("XSTR$(A$)") is not allowed, and will result in an OVERFLOW error if used. Similarly, the statements:

```
10 A$="GOSUB 20" : GOTO 30
20 XSTR$("INPUT B$") : RETURN
30 XSTR$(A$) : PRINT "I'M BACK"
```

will cause an OVERFLOW error. Line 10 sets the string variable A$ to the executable BASIC statement "GOSUB 20", and then transfers execution to line 30. Line 30 executes the string assigned to A$, "GOSUB 20". Line 20 causes the OVERFLOW error by attempting to execute an XSTR$ command within an XSTR$ command (line 30).

The syntax of the executable statement(s) used in an XSTR$ command is identical to that required by BASIC. The only exception is that strings must always be delimited by a closing quote. For example,

```
10 PRINT "HELLO
20 A$="YES
```

are valid BASIC statements, but must not be used in an XSTR$ command without a closing quote.

```
10 XSTR$("PRINT"+CHR$(34)+"HELLO"+CHR$(34))
20 XSTR$("A$="+CHR$(34)+"YES")
```

demonstrates both the correct & incorrect methods of using the above BASIC statements within an XSTR$ command. (In both cases it is necessary to use "CHR$(34)" to insert a double quote (") within the string being used.) Line 10 correctly delimits the string with a closing quote and will work properly. The command when executed will be 'PRINT"HELLO"'. Line 20 does not delimit the string, and will cause improper results. This is because the string to be executed is 'A$="YES'; it does not have the required closing quote.

EXAMPLES:

```
10 XSTR$("PRINT A")
```

will print the value of the variable, A.

```
10 LINEINPUT"ENTER THE EQUATION AS 'X=f(Y)' (e.g. X=Y*3) ";A$
20 INPUT "ENTER THE VALUE OF Y";Y
30 XSTR$(A$) : PRINT"THE RESULT IS X=";X
40 GOTO 10
```

will allow an equation to be entered from the keyboard and executed. Line 10 inputs the equation into the variable A$. Line 30 executes the equation and prints the result.

ZGET array-name,X1,Y1,X2,Y2

The ZGET command saves the graphics data from the specified rectangle into the specified array. ZGET differs from PGET in that PGET is based upon single points while ZGET works directly on bytes. This means that ZGET is **much** faster than PGET and when used in conjunction with the ZPUT command is suitable for graphic animation.

The first argument of the command, array-name, must be of the string constant **type**. It denotes the name of the referenced, dimensioned, array. The array must not be a string array (A$, for example) or a TYPE MISMATCH error will occur. For clarity, an integer array is recommended.

The arguments X1,Y1 and X2,Y2 are int-expressions, and give the coordinates of diagonally oposite corners of the desired rectangle. As ZGET works on contiguous **bytes** of graphics memory, the values of X1 and X2 may range from 0 to 63. In MODEs 1 and 3, this corresponds to 0 - 383; in MODE 2 from 0 - 127 in point (bit) notation. This means that in MODEs 1 and 3, there are 6 points per byte and in MODE 2 there are 2 points per byte. The values of Y1 and Y2 may range from 0 - 191 in all modes. This directly corresponds to Y-axis point valus. The designated area is limited to a size of 63 X 191. Rectangles outside this range will result in an ILLEGAL FUNCTION CALL error.

The following formula will be useful in calculating the minimum dimension of an integer array to be used by ZGET:

Minimum DIM = (X2 - X1 + 1) * (Y2 - Y1 + 1) / 2 + 2

Where X1 and X2 are in the range 0-63 NOT in the range as specified by the MODE, and Y1 and Y2 are in the range 0-191.

NOTE: ZGET and ZPUT are very fast. Thay are the most effective commands for doing real-time animation under LNWBASIC. PGET and PPUT are useful for manipulating shapes and moving them on a point by point basis.

EXAMPLES:

```
10 DIM A%(29)
20 CLS : PCLS
30 MODE 1
40 LINE 0,0,17,17,SET,B
50 ZGET A%,0,0,2,17
60 PCLS
70 ZPUT A%,32,96
```

will draw a box in the upper left corner of the screen, erase the screen and then make the box appear "instantly" in the center of the screen. In line 10, A% is dimensioned to 30, the formula described in the text is as follows; 0-17 is 18 points or bits, there are 6 points per graphics byte in MODEs 1 and 3 so, 18/6 = 3 bytes, the X

axis in ZGET starts at 0 so 0-2 gives us the X-axis value - (2 - 0 +1) * (17 - 0 + 1) / 2 gives us 3*18/2 or 27, 27 + 2 = 29. Line 20 and 30, clear the text screen, clear the graphics screen and set the MODE to 1 (hi-res B/W). Line 40 draws a rectangle in the upper left corner of the display. Line 50 ZGETs the rectangle into the array, A%. Line 60 clears the screen and line 70 ZPUTs the rectangle in approximately the center of the screen. **NOTE:** the calculations used for DIMming the array as well as the fact that there are only 6 points to a byte (i.e. points 0,0 to 17,0 in MODE 1 and 3 are in the range 0,0 to 2,0 for ZGET) in hi-res graphics. In MODE 2, there are 2 points to the byte (i.e. points 0,0 to 5,0 are in the range 0,0 to 2,0 for ZGET).

```
10 DIM A%(30),B%(30)
20 CLS : PCLS
30 MODE 1
40 LINE 0,0,17,17,SET,B
50 ZGET A%,0,0,2,17
60 ZGET B%,0,18,2,35
70 PCLS
80 FOR I=0 TO 63
90 ZPUT A%,I,90
100 FOR J=1 TO 20 : NEXT J
110 ZPUT B%,I,90
120 NEXT I
130 GOTO 70
```

will form a rectangle in the upper left of the screen, erase the screen, and make the box "move" from left to right centered vertically on the display. Line 10 DIMmensions 2 arrays - the A% array will hold the rectangle figure as defined in line 40 and the B% array will hold a "blank" area of the screen to erase the rectangle before it is moved to the next position across the screen. The program is the same as the preceding example up to line 50. Line 60 ZGETs a blank area under the rectangle to the array B%. Line 70 erases the graphics screen. Lines 80-120 draw the box, delay so we can see the box, erase the box (in line 110) by ZPUTting a blank array over the rectangle, and loop so the rectangle "moves" across the screen from left to right. Line 130 loops to the beginning of the movement.

ZPUT array-name,X,Y

The ZPUT command transfers graphics data stored by the ZGET command from an array to graphics memory. X and Y are int-expressions which denote the upper left coordinate of the area to be displayed to. The array-name is of the string constant **type**. This specifies the numeric array from which the graphics data is to be gotten. If a string array (i.e. A$) is specified, an ILLEGAL FUNCTION CALL error will occur.

The value of X may range from 0-63 and the value of Y from 0-191. There are 6 points per X coordinate in MODEs 1 and 3 and 2 points per X value in MODE 2. In other words, the X value specified is NOT the point boundry for an area to be displayed to, but is a byte boundry.

NOTE: the use of ZPUT and ZGET are the fastest method of manipulating graphics shapes under LNWBASIC.

EXAMPLES:

```
10 DIM A%(194) : REM" (7-0+1)*(47-0+1)/2 + 2"
20 CLS : PCLS
30 MODE 1
40 CIRCLE 24,24,23
50 ZGET A%,0,0,7,47
60 PCLS
70 FOR I = 1 TO 50
80 ZPUT A%,RND(64)-1,RND(192)-1
90 NEXT I
```

will clear the screen, draw a circle in the upper left corner, erase the screen and then draw 20 "copies" of the original circle at random locations. Line 10 DIMmensions the array to the minimum size necessary for the holding the imaginary rectangle that will hold the circle to be drawn in line 40 (see the ZGET command). Line 20 clears the text screen and clears the graphics screen. Line 30 sets the MODE to 1 (hi-res B/W) and line 40 draws a circle in the upper left cornet of the display. Line 50 ZGETs the rectangle from 0,0 to 7,47 (crresponding to the point coordinates 0,0 to 47,47 in MODEs 1 and 3). See the ZPUT and ZGET command explanations for byte vs. point coordinates. Line 60 clears the graphics screen and the FOR-NEXT loop in lines 70-90 draw the original circle 20 times at random locations.

```
10 MODE 1 : CLEAR 500
20 DEFINT A,I
25 REM" (36-26+1) * (126-66+1) / 2 + 2 = 337
30 DIM A0(337),A1(337),A2(337),A3(337),A4(337),A5(373),A6(373)
40 DIM A7(373),A8(373),A9(373)
50 FOR I = 0 TO 9
60 CLS : PCLS
70 CIRCLE 192,96,30,30*.65*I/9+1,,,30
80 N$=RIGHT$(STR$(I),1)
90 XSTR$("ZGET A"+N$+",26,66,36,126")
100 NEXT I
```

```
110 PCLS
120 FOR I = 0 TO 9
130 GOSUB 190
140 NEXT I
150 FOR I = 9 TO 0 STEP -1
160 GOSUB 190
170 NEXT I
180 GOTO 120
190 N$=RIGHT$(STR$(I),1)
200 XSTR$("ZPUT A"+N$+",26,66")
210 RETURN
```

will make an ellipse in the center of the screen, erase the screen, make a smaller and smaller ellipse (for nine times), erase the screen and then continuously "flip" (or rotate) the original ellipse. THIS IS AN EXAMPLE OF ANIMATION USING LNWBASIC.

Line 10 sets the MODE to 1 (hi-res B/W) and CLEARS enough string space for use by the XSTR$ commands in lines 90 and 200. Line 20 defines the variables A and I to be integers. A is made an integer variable in order to make the calculations for the ZPUT commands simpler. Line 25 shows the calculations necessary for use with ZGET in line 90. See the ZGET command for an explanation. Lines 30 and 40 DIMmension the arrays that will hold the different ellipse aspects. The FOR-NEXT loop from line 50 to line 100 does the following; clears the text screen, clears the graphics screen (and erases any left-over images), draws an increasingly wider ellipse in about the center of the screen, converts the FOR-NEXT variable (I) to a string variable (N$) for use in the XSTR$ function in line 90, and obtains (ZGETs) the current ellipse from the screen and places it in the appropriate integer array (A0 to A9). The FOR-NEXT loop from line 120 to line 140 puts (ZPUTs) to the screen 10 images very quickly. The images are of the wider and wider ellipse until the ellipse is at its widest. Lines 150 to 170 work in a similar fashion bu make the ellipse "skinnier". Do to the speed at which the images are placed onto the graphics screen, it appears that the ellipse is "flipping" or "rotating". Line 180 causes the rotation to coninue be looping to line 120. Lines 190 to 210 are a subroutine used by the preceding FOR-NEXT loops. Line 190 converts the FOR-NEXT index (I) to be converted to a string variable (N$). Line 200 obtains the various ellipse images to be obtained from the appropriate array (A0 to A9) and places then on the graphics screen. Line 210 returns to the calling routine.

GLOSSARY FOR LNWBASIC

ASCII

American Standard Code for Information Interchange. This method of coding is used for textual data.

baud

Signalling speed in bits per second. The term is usually used in conjunction with serial input and output. See RS-232-C.

command file

A DOS file with the extension /CMD. The file consists of Z-80 object code (machine language).

DCB (data/device control block)

An area in RAM associated with an Input/Output device.

expression

A meaningful sequence of one or more constants or variables possibly used with operators and functions.

filespec

A string constant or expression which specifies a particular disk file. It consists of a mandatory filename, of up to eight characters, followed by an optional extension, password, and drivespec. The following file extensions are recommended:

/GRF--For graphics image files used by PLOAD & PSAVE.
/KEY--For defined key files used by LOADKEY & SAVEKEY.
/LNW--For BASIC program files written using LNWBASIC.
/SPL--For spool files as used by SPOOLON.

int-expression

An expression that when evaluated lies within the BASIC integer range (-32768 to +32767). If the expression value is not an integer, it will be rounded to the closest integer. Int-expressions may contain hexadecimal constants but the hex number may not have a space between it and the next part of the expression. Example: &HF000+ 10 * 5.

#"label"

A string constant of up to 251 alphanumeric characters preceded by a pound sign and enclosed in double quotes (#"ALABEL"). Any characters other than a double quote may appear within the label defined by the begin and end quotes.

line number

A constant, variable, or numeric expression which specifies a BASIC program line number. Usually this line number must exist within the BASIC program currently in memory, or an error will result. When used within a "line range", however, "line number" need not exist.

line range

A line or a range of lines of a BASIC program. Whenever used it must be enclosed by parentheses. See NTRON.

lsb,msb expression

lsb (least significant byte) is the remainder of a int-expression MOD 256 while msb (most significant byte) is the evaluation of an int-expression MOD 256.

pos-expression

An expression preceded by a plus sign ("+") indicating that the value of the expression should be evaluated as an integer between Ø and 65529 inclusive.

RS-232-C

Refers to a specific EIA (Electronics Industries Association) standard which defines a widely accepted method for interfacing data communications equipment.

string

A string variable, expression, or constant of length <= 255 characters.