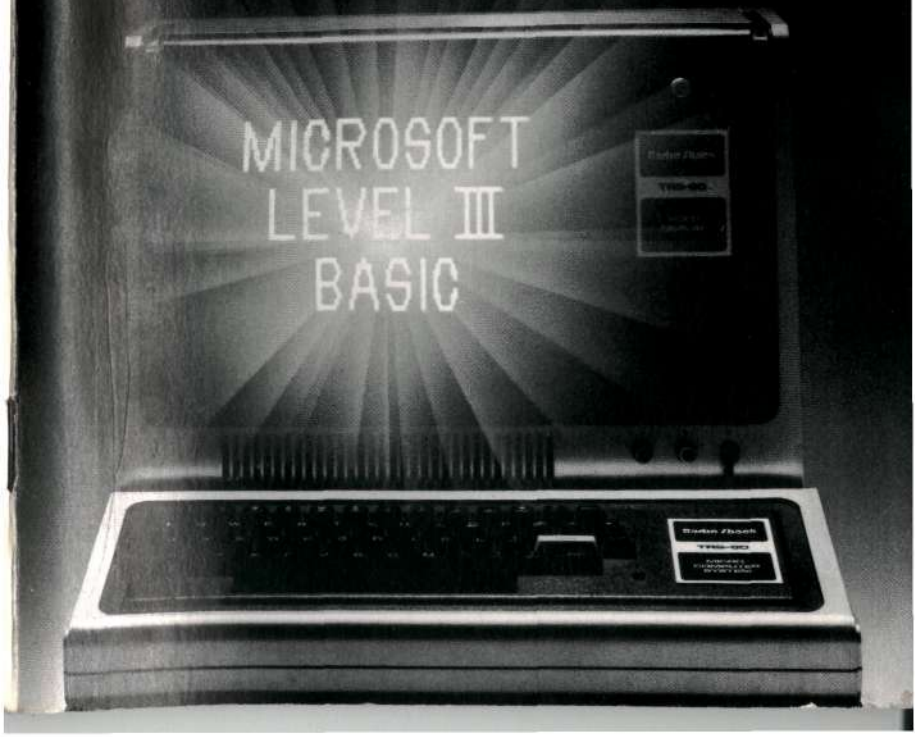
Program Instructions For

MICROSOFT

LEVEL III BASIC



LEVEL III BASIC

Produced by Microsoft
Written by Bill Gates
Documented by Andrea Lewis
Instruction Booklet by David Bunnell

Microsoft Consumer Products
10800 NE Eighth, Suite 819, Bellevue, WA 98004

COPYRIGHT NOTICE

Microsoft LEVEL III BASIC is copyrighted under United States Copyright Laws by Microsoft.

It is against the law to copy LEVEL III BASIC on cassette tape, disk, or any other medium for any purpose other than personal convenience.

It is against the law to give away or resell copies of Microsoft LEVEL III BASIC. Any unauthorized distribution of this product deprives the authors of their deserved royalties. Microsoft will take full legal recourse against violators.

If you have questions on this copyright, please contact:

Microsoft Consumer Products
10800 NE Eighth, Suite 819
Bellevue, WA 98004

Table of Contents

Chapter ONE:

Getting Started with LEVEL III

Chapter TWO:

LEVEL III Programming Convenience

Chapter THREE:

LEVEL III Computer Graphics

Chapter FOUR:

LEVEL III Features From DiskBASIC

Chapter FIVE:

LEVEL III Expansion Interface Features

Chapter ONE:

Getting Started with LEVEL III

- LEVEL III BASIC Explained
- A Word About Microsoft
- The Right Hardware
- LEVEL III Cassette
- How to Load LEVEL III
- What to Do About Loading Problems
- LEVEL III Notation Format Rules
- If You Have a TRS-80 Screen Printer

LEVEL III BASIC Explained

LEVEL III BASIC is a software package, supplied on cassette tape, that enhances Radio Shack's TRS-80 Level II BASIC.

If you have a TRS-80 Computer with Level II BASIC and 16K or more RAM memory, LEVEL III BASIC gives you a new dimension of computer programming. In only 5.5K RAM memory, it provides your TRS-80 with all the non-disk features that are currently only available with TRS-80 Disk BASIC. These include a new string function (INSTR), enhancements to the MID$ and USR functions, user-defined functions and the DEFUSR statement, hexadecimal and octal constants, LINE input, and long error messages. And this is only the beginning.

LEVEL III BASIC includes dynamic features never before available to TRS-80 users. These include advanced computer graphics commands, automatic program re-numbering, abbreviated entries, a timed INPUT statement called INPUT #LEN, and more.

With LEVEL III BASIC, you'll also find that keyboard bounce has been corrected, tape operations are more reliable, and output to an RS-232 device has been simplified.

LEVEL III BASIC was written by Microsoft to give you the tools for writing better programs. Not only does it increase the programming power of your TRS-80, it gives you features that make programming easier and faster.

The people at Microsoft hope you will be pleased with this addition to your TRS-80.

A Word About Microsoft

Microsoft produces high-quality, concise software for today's microprocessors.

Microsoft's BASIC Interpreter, in its several versions, has become the standard high-level programming language used in microcomputers. In addition to Radio Shack TRS-80 Level II BASIC and TRS-80 Disk BASIC, Microsoft has supplied BASIC Interpreters for the Commodore PET, the Apple II Computer, NCR 7200, Compucolor II, OSI, Pertec Altair, and many others.

Microsoft's careful approach to the development of microprocessor software has allowed the production of large amounts of bug-free, well-designed code in a minimum amount of time. Currently available: BASIC interpreters for the 8080, 6800, and 6502 microprocessors, a FORTRAN compiler, assembler, loader and runtime library package for the 8080 and Z-80 microprocessors and an ANS-74 COBOL compiler for the 8080 and Z-80, and a complete offering of systems software for the new 16-bit microprocessors.

Microsoft Consumer Products was founded as a division of Microsoft in the summer of 1979 to provide microcomputer users with high quality system and utility software as well as application software.

LEVEL III BASIC is just the first of many Microsoft products being planned for the end-user or consumer market. All of these software packages will be marketed by Microsoft Consumer Products.

Microsoft Consumer Products is dedicated to providing only the best, most reliable microcomputer software.

For more information on Microsoft or Microsoft products, please write to: Microsoft Consumer Products
10800 NE Eighth, Suite 819
Bellevue, WA 98004

The Right Hardware

LEVEL III BASIC can be used with the Radio Shack TRS-80 Microcomputer with Level II BASIC and 16K RAM minimum memory.

While LEVEL III is supplied on cassette tape and is primarily intended for use with cassette storage, it can be used on a Disk-based TRS-80 system. LEVEL III itself can be stored and retrieved from disk, however, LEVEL III generated programs subsequently have to be stored and retrieved from cassette tape. You do not have access to disk storage commands while in LEVEL III BASIC.

LEVEL III BASIC has three features that are exclusively for TRS-80 Computers with the Expansion Interface add-on. These include a built-in digital clock-calendar, output to RS-232 port, and a command for turning on and off the System Clock. If you have a TRS-80 with Expansion Interface, you can read more about these features in the last chapter.

LEVEL III Cassette

The LEVEL III Cassette that comes in your LEVEL III package is a high-quality recording from Microsoft.

You will notice that all the recordings are on "SIDE ONE" of the cassette. There are two consecutive recordings of LEVEL III BASIC (Cassette File), followed by two consecutive recordings of LEVEL III BASIC (Disk File).

The "Cassette File" recordings are for TRS-80 Computers with cassette tape storage.

The "Disk File" recordings are provided for the convenience of TRS-80 users with Disk storage who want to store LEVEL III BASIC as a file on diskette.

If you listen to the tapes, you will note that there are five seconds between the two cassette files; five seconds between the two disk files; and ten seconds between the cassette and the disk recordings.

How to Load LEVEL III*

Cassette File. The first two recordings on SIDE ONE of your LEVEL III cassette are "Cassette File" recordings. These are identical recordings of LEVEL III BASIC to be used with TRS-80 Computers with cassette mass storage. (The Second recording is a backup recording.)

The "Disk File" recordings that follow are for storing and loading LEVEL III from disk. Unless your TRS-80 is equipped with Expansion Interface and disk drive, you needn't concern yourself with "Disk File." For those who are interested, the instructions for using these recordings are on the following page.

To load LEVEL III BASIC from cassette, use the following instructions:

1. Put the LEVEL III cassette into the TRS-80 recorder so that SIDE ONE is up.
2. Rewind the tape to its beginning.
3. Enter the command, SYSTEM, and press the ENTER key.
4. In response to the * ? prompt, enter: LEV 3 ENTER
5. Press the PLAY button on the TRS-80 recorder. There should soon be two asterisks in the upper-right corner of the TRS-80 screen (the one on the right "blinks"). These two asterisks signify that LEVEL III is loading.
6. A successful load will result in another * ? prompt. In response to this prompt, enter: / ENTER
7. Assuming that all goes well, the screen will display a MICROSOFT COPYRIGHT notice, followed by the same prompt you get with Level II BASIC:

```
READY
>
```

*See WARNING, end of this section.

Disk File. The Disk File is for TRS-80 Computers equipped with an Expansion Interface, Disk Drive, and any one of the following Disk Operating Systems (DOS): Radio Shack 2.1, Radio Shack 2.2, NEW DOS, and 3.0. Also required is the Radio Shack TAPEDISK utility software, which is included with most DOS's. If you do not have TAPEDISK, ask your Radio Shack dealer about it.

Disk File lets you store LEVEL III on diskette so that it can subsequently be loaded from disk instead of tape.

Use the following instructions to save LEVEL III on diskette:

1. Put the LEVEL III cassette into the TRS-80 recorder so that SIDE ONE is up.
2. Turn on the TRS-80. In response to the DOS READY prompt, enter:

   TAPEDISK ENTER
3. This should result in a ? prompt. In reponse to ?, enter:

   C DLEV3 ENTER
4. Press the PLAY button on the TRS-80 recorder. A single blinking * should soon be displayed in the upper-right corner of the TRS-80 screen.
5. A successful load will result in another ? prompt. In response to this second ?, enter:

   F DLEV3/CMD:n 5500 6A00 5500 ENTER

   Drive number

   NOTE: LEVEL III can be stored on any Drive, but the corresponding Drive number has to be entered at "n" above. Don't forget that the diskette at Drive "n" must be formatted.

6. As LEVEL III is saved, you will hear the familiar sound of the disk. A successful save will result in another ? prompt. In response to this third ? prompt, enter:

   E ENTER

   This should result in an exit from the TAPEDISK utility and return you to DOS READY.

How to Load LEVEL III from diskette. Once LEVEL III has been successfully saved on a diskette, follow these instructions to load it into the TRS-80:

1. Correctly put the diskette containing the LEVEL III BASIC file into the appropriate disk drive.
2. Turn on the TRS-80.
3. In response to DOS READY, enter:

   BASIC ENTER

4. Respond to the prompt, FILES?, by pressing the ENTER key. Respond likewise to the next prompt, SIZE OF MEMORY?
5. Now that BASIC is up, go back to DOS by entering:

   CMD"S" ENTER

6. Load the LEVEL III file by entering:

   DLEV3 ENTER

These steps should result in a successful load. You are now ready to program in LEVEL III BASIC.

WARNING:

Before loading LEVEL III or any other recordings into the TRS-80 microcomputer, we strongly urge you to disconnect the smallest grey plug that is normally inserted into the "MIC" jack of the tape recorder.

If for any reason during the actual reading of a tape the TRS-80 turns off the recorder (via the smallest grey plug), a "spike" may be recorded on the tape. Should this happen, the recording you are entering will be **permanently** damaged.

Our experience shows that this is most likely to occur when using the Radio Shack CTR-80 recorder, but we recommend that you still disconnect the smallest grey plug no matter what recorder you are using.

What to Do About Loading Problems

The TRS-80 is known to be "volume sensitive" when it comes to loading programs from cassette.

Once LEVEL III is in your TRS-80, you can use its LOAD and SAVE commands for storing programs. These commands correct the volume sensitivity problem. However, you still have to load LEVEL III itself under LEVEL II's SYSTEM command.

The most common loading problem is finding the correct settings for the TONE and VOLUME controls on the TRS-80 recorder. We suggest that you start with a low VOLUME setting and adjust it up one half level each time you attempt a load. The TONE control is not as important, but change it also.

Since the sensitivity of individual cassette recorders varies significantly, there is no way to determine specific settings. Once a tape loads, it is a good idea to write down the settings on the cassette label for future reference.

If you still cannot load a tape, you might try cleaning and demagnetizing the head of your TRS-80 recorder. Use a high-quality head cleaner and an inexpensive head demagnetizer for these tasks. Both can be purchased at Radio Shack or many other electronics outlets for under $10 (at the time of this writing). We don't recommend so-called "cleaner-tapes" as they are often abrasive and will damage the head of your recorder.

Other suggestions to try before taking the matter up with your Radio Shack dealer include the following:

- Try loading the second recording on your LEVEL III cassette.
- Try loading the tape with a different cassette recorder.

- Dust and other praticles can sometimes prevent a load. To remove particles, run the tape through REWIND and FAST FORWARD a few times.
- Don't try loading the TRS-80 when you first turn it on. Let it warm up a few minutes instead.
- Remove the earphone jack and run the tape to listen for the leader tones and digital signals of the files. If you don't hear these sounds, try a different recorder. If you still don't hear them, chances are you have blank tape.
- Ask your Radio Shack dealer about Radio Shack's "cassette modification" fix. This hardware correction should make your Level II BASIC less dependent upon exact VOLUME settings.

LEVEL III Notation Format Rules

The LEVEL III Notation Format was devised to help you understand how to correctly structure LEVEL III instructions. An example of this notation follows:

```
MID$(string1, n[,m]) = string2
```

The following rules apply to notation:

1. Items in capital letters must be input **exactly** as shown. "MID$" in the above example has to be entered as: MID$
2. Items in lower case letters are to be supplied by the user. "string1" and "string2" above are examples.
3. Items in square brackets [] are optional. The ",m" inside the square brackets above may be included in some MID$ statements, but not in others.
4. All blank spaces are optional, unless otherwise noted. As you probably know, BASIC doesn't mind if you run words and numbers together in program statements. Often this is done to conserve memory space.

If You Have a TRS-80 Screen Printer

The following applies **only** to systems with Radio Shack's TRS-80 Screen Printer. It **does not apply** to systems with TRS-80 Line Printers I and III, or with Line Printer II (the Radio Shack thermal printer), or with any other printers.

If your TRS-80 system has a TRS-80 Screen Printer and you want to print out LEVEL III generated screen-contents, you must do the following:

1. Before entering the print instructions as you normally would with Level II BASIC, enter:

   ```
   POKE 5657, E3
   POKE 5658, O3
   ```

2. Execute an INPUT statement, such as INPUT X$.

3. Once you've finished printing out the material you want, enter:

   ```
   POKE 5657, 92
   POKE 5658, 51
   ```

4. Again, execute an INPUT statement. In response to the prompt (?), press the ENTER key.

Chapter TWO:

LEVEL III Programming Convenience

- How to Use Abbreviated Entries
- Create Your Own Abbreviated Entries
- How to Renumber Program Lines
- Saving and Loading LEVEL III Programs
- No More Coded Error Messages

How to Use Abbreviated Entries

Abbreviated Entries are very handy for entering frequently used BASIC instructions and string expressions, such as a common response to an INPUT statement.

To enter an Abbreviated Entry, all you have to do is press [SHIFT] and *one of the letter keys ([A] through [Z]). For example, to RUN a program,* you could enter [SHIFT] [R] instead of: [R] [U] [N] [ENTER] .

LSET LIST. LEVEL III maintains a list of 26 Abbreviated Entries. You can display this list on your TRS-80 screen by entering:

LSET LIST [ENTER]

If you change any of the Abbreviated Entries (see next section), the list is automatically updated.

Turning Abbreviated Entries "Off" and "On". You can "turn off" the whole list of Abbreviated Entries with the command:

LSET RESET [ENTER]

And back on again with:

LSET SET [ENTER]

Abbreviated Entries

SHIFT Key	Enter
A	AUTO
B	GET@(
C	ELSE
D	EDIT.↓
E	EDIT
F	GOTO
G	GOSUB
H	INKEY$
I	INPUT
J	LINE INPUT
K	LINE(
L	LIST
M	LSET
N	NEXT
O	PRINT USING
P	PUT@(
Q	RETURN
R	RUN↓
S	SAVE"
T	THEN
U	TIME$
V	LOAD"
W	LEFT$(
X	MID$(
Y	RIGHT$(
Z	STRING$(

NOTE: The downward arrow (↓) is the symbol in this chart for ENTER.

Create Your Own Abbreviated Entries

The following command lets you create your own Abbreviated Entries:

LSET letter = string expression

where "letter" is any letter A-Z, and "string expression" is any string up to 15 characters.

LSET may be used as a command or a program statement. Abbreviated Entries work during program execution as well as program editing. For example, if you're writing a program with several graphics statements, you may want to execute the command:

```
LSET A="(X1,Y1)-(X2,Y2)"
```

Then, instead of typing (X1,Y1)-(X2,Y2) every time it comes up (which can be frequent), all you do is enter:

SHIFT A

You can also use Abbreviated Entries to anticipate common INPUT responses. For example,

```
510 LSET R="REPEAT"
520 LSET C="CONTINUE"
```

Suppose a program asks you to enter "REPEAT" or "CONTINUE" at the end of each of several routines. By tacking the above LSET statements on the end of the program, you can respond to this INPUT with a SHIFT R or a SHIFT C, instead of spelling out the complete answer.

Adding ENTER. To make ENTER a part of an Abbreviated Entry, use the CHR$ string with the ASCII value for ENTER (13)

```
10 LSET L="LIST"+CHR$(13)
```

This will change LSET L so that when you enter SHIFT L the result will be LIST ENTER, instead of just LIST.

How to Renumber Program Lines

LEVEL III BASIC'S NAME command is an editing tool that helps you organize your programs better and write them faster.

You use the NAME command to renumber program lines as you actually write a program or you use NAME to "clean-up" a finished program by giving it sequenced line numbers (such as 10, 20, 30, 40, etc.)

NAME Format. The format of NAME is:

NAME [[new number][,[old number][,increment]]]

With NAME, you can renumber the entire program from the first line number or you can renumber a sequence of a program from a designated line number ("old number" in above format) to the end of the program.

The "new number" can be any legitimate line number. It becomes the first line number of the new sequence.

For example, you would use "new number" if you were renumbering a sequence from line 50, and you wanted line 50 to become line 100. The "new number" would be 100.

"New number," as well as "old number" and "increment" are optional. When they are not included in a NAME command, each has a "default number" which LEVEL III refers to.

The default of the "new number" is 10. Whenever a "new number" is not included in a NAME command, the first line number of the new sequence becomes Line 10.

Note: If "new number" is omitted, you cannot renumber a program sequence with a prior Line number of 10 or greater. Line numbers always have to be in numerical sequence, beginning with the smallest number and moving to the greatest. You cannot use the NAME command to renumber programs in a way that puts its statements out of sequence. The result will be an ILLEGAL FUNCTION error.

"Old Number" is the line number from which the Renumbering sequence is to begin. It is optional, and the default is the first line of the program.

"Increment" *is the increment to be used in the new sequence. It is optional, and the default is 10.*

*NAME also changes all line number references following GOTO, GOSUB, THEN, ELSE, RESUME, INPUT#*LEN, ON . . . GOTO and ON . . . GOSUB statements to reflect the new line numbers. If a nonexistent line number appears after one of these statements, the error message, the error message UNDEFINED LINE xxxxx IN yyyyy is printed. The incorrect line number reference (xxxxx) is changed to all spaces. Line number yyyyy may be changed.

NAME cannot be used to change the order of program lines (for example, NAME 15,30 when the program has three lines numbered 10, 20 and 30) or to create line numbers greater than 65529. An ILLEGAL FUNCTION CALL error will result.

The first time you use the NAME command, BASIC adds spaces to the end of each line number that has fewer than five characters. These spaces will not accumulate, however, when subsequent NAME commands are executed.

A NAME command with no arguments will renumber the entire program, using 10 as the first line number and incrementing by 10 for each successive line.

Other examples:

NAME 1000,900,20	Renumbers the lines from 900 up so they start with line number 1000 and increment by 20.
NAME 100,,50	Renumbers the entire program. The first new line number will be 100. Lines will increment by 50.
NAME,,100	Renumbers the entire program. The first new line number will be 10. Lines will increment by 100.

Saving and Loading LEVEL III Programs

LEVEL III BASIC replaces the Level II CSAVE and CLOAD commands with SAVE and LOAD. When using SAVE and LOAD, exact volume settings on your recorder are less critical than with CSAVE and CLOAD.

To save a LEVEL III program you enter the SAVE command exactly as you would enter CSAVE. The format is:

```
SAVE "file name"
```

The file name can be any single alpha-numeric character other than double-quotes (''). The program stored on tape will then be labled by the specified file name, so that it can be loaded with a LOAD command that asks for that particular file.

Once you save a program under LEVEL III you can load it back into the machine with the LOAD command. The format for this command is similar to CLOAD:

```
LOAD ["file name"]
```

LOAD and CLOAD are interchangeable in that a program that has been stored with CSAVE can be loaded with LOAD as well as CLOAD; while a program that has been stored with SAVE can likewise be loaded with either LOAD or CLOAD. However, when you mix LEVEL III cassette tape commands with Level II commands, you don't get the improvement in cassette VOLUME sensitivity that you get by using both SAVE and LOAD.

As with CLOAD?, there is a LOAD? command which you can use after saving a program to test whether the program in memory matches the one on cassette, and thus has been saved correctly.

You can abort a LOAD or SAVE by pressing the BREAK key. Just remember that doing this leaves you with an incomplete program on cassette or a partial program in memory. Type NEW ENTER to clear your computer's RAM memory whenever this occurs.

No More Coded Error Messages

When Level III BASIC encounters an error, it prints the complete error message, not just an abbreviation.

All the error messages, along with their error codes and Level II abbreviations, are listed below.

Code	Abbreviation	Error Message
1	NF	NEXT without FOR
2	SN	Syntax error
3	RG	Return without GOSUB
4	OD	Out of data
5	FC	Illegal function call
6	OV	Overflow
7	OM	Out of memory
8	UL	Undefined line
9	BS	Subscript out of range
10	DD	Redimensioned array
11	/0	Division by zero
12	ID	Illegal direct
13	TM	Type mismatch
14	OS	Out of string space
15	LS	String too long
16	ST	String formula too complex
17	CN	Can't continue
18	NR	No RESUME
19	RW	RESUME without error
20	UE	Unprintable error
21	MO	Missing operand
22	FD	Bad file data
23	L3	Disk BASIC feature
24		Undefined USER function

Chapter THREE:

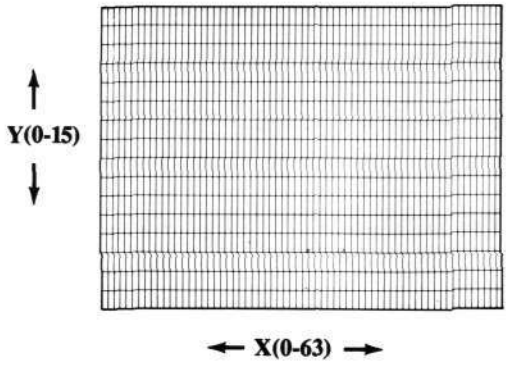
LEVEL III Computer Graphics

- ✓ Two Modes of Presentation
- ✓ How to Draw Lines and Rectangles
- ✓ GETting and PUTting Graphic Arrays
- ✓ Some Examples of Graphics Programs
- ✓ Advanced Graphics Programs

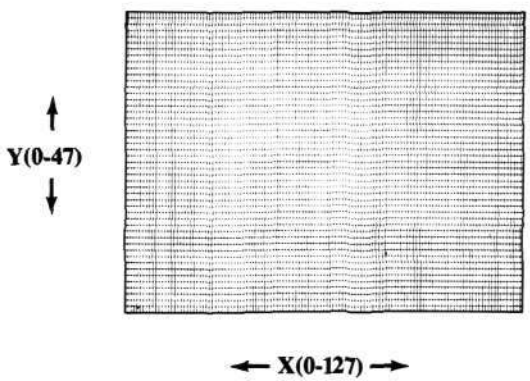
Two Modes of Presentation

With LEVEL III Graphics, you can refer to locations on the screen in one of two ways, **Character Mode** and **Graphics Mode.**

In Character Mode, the screen is divided into a grid that is 64 columns wide and 16 lines deep:



In Graphics Mode, the screen is also divided into a similar grid, only it is much finer, measuring 128 columns wide by 48 lines deep.



LEVEL III generates lines and graphics shapes in Graphics Mode by lighting or not lighting specified coordinate locations on the grid. For example, to draw a line on the screen from coordinates 32, 12 to 32, 89; LEVEL III lights these locations and all the locations between.

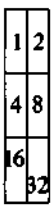
In Character Mode, graphics are generated somewhat differently. Instead of lighting or not lighting a location, you can enter an alphanumeric symbol or a "graphics symbol."

The grid locations in Character Mode actually consist of six Graphics Mode locations, like this:

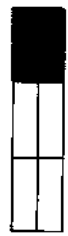
Graphic symbols are generated by filling in different combinations of these six grid locations. Since there are 64 different possibilities, there are 64 graphics symbols.

Each graphics symbol is further identified by an ASCII value from 128 to 191.

This ASCII value is determined by first assigning the following values to each of the grid locations:



The ASCII value of each graphic symbol is equal to 128 plus the total value of the locations that are lighted. For example, the following graphic symbol has an ASCII value of 131:



(128 + 1 + 2 = 131)

Some more examples:

(128 + 1 + 8 = 137) **(128 + 2 + 4 + 32 = 166)**

Alpha-numeric symbols also have ASCII values, numbering 32 through 127. A listing of these alpha-numeric symbols and their corresponding ASCII values can be found in Radio Shack's LEVEL II BASIC Reference Manual, page C/2.

You can display the graphics symbols along with their corresponding ASCII values on your TRS-80 screen with the following program:

```
5 FOR S = 129 TO 191
10 PRINT CHR$(S);:PRINT S
15 NEXT
```

Or to display the alpha-numeric symbols with their ASCII values:

```
5 FOR S = 32 TO 128
10 PRINT CHR$(S);:PRINT S
15 NEXT
```

CHR$. This string expression is used to return alpha-numeric symbols and graphic characters. It is formatted as follows:

```
CHR$(number)
```

where number is an ASCII value or value reference.

For example, the command PRINT CHR$(68) would return a "D", while PRINT CHR$(185) returns a graphics symbol.

Screen Coordinates. In both Graphics Mode and Character Mode, locations on the screen are referenced by their column coordinate (X), followed by their line coordinate (Y).

This differs from the way the PRINT@ Statement in Level III BASIC references the screen. PRINT@ references the screen in Character Mode where each location is assigned a unique number starting from the upper-left corner, at location 0, and moving across the lines until the final location is reached in the lower-right corner, location 1023.

The way PRINT@ references the screen is shown on a graph on page E/1 of the Radio Shack Level II BASIC reference manual.

To convert a PRINT@ location to Character Mode "X,Y" coordinates, use the following formula:

```
X = number - (64*Y)
Y = INT(number/64)
```

To convert from coordinates to location number:

```
number = (Y*64 + X)
```

How to Draw Lines and Rectangles

With LEVEL IIIBASIC, you can draw a line between any two locations on the screen in both Character Mode and Graphics Mode. You can also draw a rectangle between any two locations (assuming that the locations are opposite corners).

The statement for drawing lines and rectangles is not too surprisingly called the "LINE statement."

Graphics Mode. Remember, in Graphics Mode the screen is divided into an X, Y grid where X is 0 to 127 and Y is 0 to 47.

The following statement will draw a line between coordinates 34, 15, and 110, 38:

```
10 LINE(34,15) – (110,38),SET
```

To erase the line, change SET to RESET:

```
20 LINE(34,15) – (110,38), RESET
```

Now, to draw a box (rectangle) between these two points, simply add a ",B" to the first LINE statement above:

```
10 LINE(34,15) – (110,38),SET,B
```

To erase the rectangle:

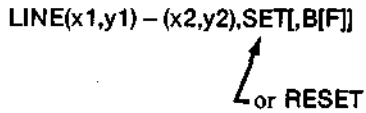
```
10 LINE(34,15) – (110,38), RESET,B
```

Finally, there is an option that will draw a "filled-in" or solidly-lighted box:

```
10 LINE(34,15) – (110,38),SET,BF
```

adding an "F" fills in the box

Graphics Mode Format. The format for the LINE statement in Graphics Mode is as follows:



The LINE statement in Graphics Mode generates a line or rectangle between coordinates (x1,y1) and (x1,y2), by lighting or "SETting" the appropriate screen locations.

RESET can be substituted for SET to erase or "unlight"a line or rectangle.

The optional "B" tells the LINE statement to generate a rectangle instead of a line.

The optional "F" tells the LINE statement to "fill-in" the rectangle by lighting all the screen locations within the rectangle. Obviously, you cannot have an "F" in a LINE statement if you don't have a "B".

Character Mode. In Character Mode the screen is divided into a more open grid, where X is 0 to 63 and Y is 0 to 15:

When you generate graphics in Character Mode, remember, you use alpha-numeric or graphic symbols.

The following LINE statement will draw a line of "X's" between Character Mode coordinates (12,8) and (55,8):

```
10 LINE (12,8) – (55,8),"X"
```

LINE statements in Character Mode do not use SET and RESET. To erase the above line of X's, replace the "X" string with a string containing a blank character, " ", as below:

```
20 LINE (12,8) – (55,8), " "
```

To generate a graphics symbol, such as the one with ASCII code 132, the CHR$ string is used:

```
10 LINE(12,8) – (55,8),CHR$(132)
```

LINE statements in Character Mode can also use string variables such as D$ in the following:

```
10 D$ = "X"
15 LINE(12,8) – (55,8),D$
```

Should the string expression in a LINE statement in Character Mode consist of more than one alpha-numeric symbol, such as "XEQ", the resulting line or rectangle will be generated with the first symbol only ("X"). The LINE statement:

```
10 LINE(12,8) – ( 55,8),"XEQ"
```

results in a line of "X's".

As with Graphics Mode, you can instruct the LINE statement in Character Mode to generate a rectangle by adding ",B" to the end of the statement or a "filled-in" rectangle by adding ",BF".

The following LINE statement results in a filled-in rectangle of X's between Character Mode coordinates (15,4) and (35,12):

```
10 LINE (15,4) – (35,12),"X",BF
```

Character Mode Format. The Format of LINE statements in Character Mode is the following:

```
LINE(x1,y1) – (x2,y2),"string expression"[,B[F]]
```

The LINE statement in Character Mode is used to generate a line or rectangle of alpha-numeric symbols or graphics symbols between any two Character Mode coordinates.

The "string expression" in the LINE statement can be any alpha-numeric symbol (such as "X", "8", "=") **or** it can be a string variable (such as D$); **or** it can be the CHR$ function, which is generally used to return a graphics symbol.

Adding the optional ",B" to a LINE statement in Character Mode causes the LINE statement to generate a rectangle between the two coordinates.

Adding the optional "F" tells the LINE statement to "fill-in" the rectangle.

GETting and PUTting Graphic Arrays

LEVEL III has two more Graphics statements in addition to the LINE statements. These statements are GET and PUT.

The GET statement is used to "get" or store in an array the contents of a specified section of the screen. This section of the screen may be defined by either Character Mode coordinates or Graphic Mode coordinates. The array can be an integer array (such as A%), a single precision array (such as A!), or a double precision array (such as A#).

GETting a graphics array does not cause the contents of the specified section of the screen to be erased. The content is now both saved in the array and on the screen.

To erase graphics, you can use one of the following methods:

1. If the graphics were created in Graphics Mode, RESET all the lines and rectangles.
2. If the graphics were created in Character Mode, replace all strings with strings containing the blank character, " ".
3. Use the PUT statement to PUT a blank array or the **part of the array that is blank** on top of the part of the screen you wish to erase. When doing animation, it is sometimes useful to GET an array that is larger than the area of the screen containing the graphics. This way, part of the array is blank and can be PUT back on the screen to erase the existing graphic at the same time the graphic part of the array is PUT back on the screen.

Once a GET statement has been executed, the resulting array can be returned to the screen at any specified section with ***the PUT statement.*** In Character Mode, you can only PUT the array back on the screen as it was saved. However, in Graphics Mode you can also specify an "action" that will change the array.

GET Example. Assuming that the following program is used to draw a rectangle on the screen,

```
10 CLS
15 LINE(22,9)-(29,20),SET,BF
```

You could GET this rectangle in **Graphics Mode** with:

```
20 DIM A%(20)
25 GET@(22,9)-(29,20),G,A%
```

Or you could GET it in **Character Mode** with:

```
20 DIM A%(20)
25 GET@(11,3)-(14,6),A%
```

GET Format. The format of a GET statement is as follows:

```
GET@(x1,y1)-(x2,y2)[,G],array name
```

The GET statement is used to GET a portion of the screen **into an array** defined by either Character Mode coordinates or Graphics **Mode coordinates.**

When the optional ",G" is included, the GET statement assumes Graphics Mode. If ",G" is not included, GET assumes Character Mode.

The "array name" can be any legal integer (%), single precision (!), or double precision (#) array. (If arrays are bigger than their default size —11—, they must be dimensioned before they are used.)

The "@" sign in a GET statement is a mandatory part of the statement.

PUT example. Assuming that you used Character mode to GET the array in the above example, you could put it back on the screen at a different location with a PUT statement:

```
30 PUT@(44,3)-(47,6),A%
```

The above PUT statement is in Character Mode. Whenever you GET an array in Character Mode, you PUT it back on the screen in Character Mode. Likewise, whenever you GET an array in Graphics Mode, you PUT it back in Graphics Mode.

Assuming that you saved the above array in Graphics Mode, you could PUT it back on the screen with the following:

```
30 PUT@(88,9)-(95,20),SET,A%
```

Notice that this PUT statement is different from the one above in that it has the word "SET" included. In Graphics Mode, the PUT statement requires an "action indicator" to tell it "how" to return the array. The action indicator, "SET", tells the PUT statement to return the contents of the array exactly as they were saved.

If you do not include an action indicator in a PUT statement, LEVEL III assumes you are PUTting the array in Character Mode.

PUT Format. The format of the PUT statement is as follows:

```
PUT@(x1,y1)-(x2,y2)[,action],array name
```

The PUT statement in Graphics Mode can have one of five action indicators, including:

SET	Puts the array on the screen exactly as it was saved, i.e., all the "on" positions are turned on and all the "off" positions are turned off.
RESET	Puts the complement of the array on the screen, i.e., all the "on" positions are turned off and all the "off" positions are turned on.
AND	Each position in the array is ANDed with the current status of that position on the screen, i.e., a position will be turned on only if it is "on" in the array and "on" on the screen.
OR	Each position in the array is ORed with the current status of that position on the screen, i.e., a position is turned on if it is "on" in the array or if it is "on" on the screen or both.
XOR	Each position in the array is XORed with the current status of that position on the screen, i.e., a position is turned on only if its status in the array is the opposite of its status on the screen.

These options give the PUT statement a great deal of flexibility. For example, a figure can appear to blink by PUTing it on the screen with two PUT statements that alternate SET and RESET, or by XORing it with the surrounding area. Or a figure can appear to move by PUTing it on the screen with gradually changing coordinates.

Dimensioning Graphic Arrays. Unless you dimension arrays in GET and PUT statements with enough array "elements" the result will be an ILLEGAL FUNCTION CALL. All arrays with more than (11) elements must be dimensioned.

Usually, you dimension arrays by simply making sure they are plenty big enough, however, if you want to make sure, you can use the following formulas:

Array	**Elements**
Integer (%)	bytes/2
Single Precision (!)	bytes/4
Double Precision (#)	bytes/8

Bytes? Each location in ***Character Mode*** uses a single byte. The rectangle defined by Character Mode coordinates (44,3)-(47,6) contains 16 character locations, and thus requires 16 bytes. An Integer array (%) that GETs or PUTs this rectangle would require a minimum of 8 array elements (16 bytes/2 = 8 elements). This array wouldn't need to be dimensioned in this program unless you want to save (3) elements of memory.

Determining the number of bytes in Graphics Mode is a bit more complicated. The number of bytes equals the total number of screen locations **divided** by 8 **plus** 2. The rectangle defined by Graphics Mode coordinates (22,9)-(29,20), contains 84 locations, and thus requires 84/8 + 2 or 13 bytes. An integer array (%) that GETs or PUTs this rectangle requires 13/2 or 7 array elements.

Some Examples of Graphics Programs

Graphics Symbols. Fills screen with the graphic symbols.

Program	*Notes*
10 CLS	clears screen
20 FOR I = 129 TO 191	assigns values
30 LINE (0,0)-(63,15),CHR$(I),BF	fills screen
40 FOR J = 1 TO 100:NEXT J	timing loop
50 NEXT I	loops back to **20**

Jagged Lines.

Program	**Notes**
10 FOR N = 1 TO 10	assigns values (10 patterns)
20 CLS	clears screen
30 Y1 = 0:X1 = 0	defines point (upper-left corner of screen)
40 FOR I = 1 TO 30	assigns values (30 lines)
50 X2 = RND(127):Y2 = RND(47)	defines second point as any random point on screen
60 LINE (X1,Y1)-(X2,Y2),SET	draws line
70 X1 = X2:Y1 = Y2	redefines first point as second *point*
80 NEXT I:NEXT N	loops back to *10*

Variation: To change this program to create its pattern of lines on a white background, add line 25 LINE (0,0)-(127,47),SET,BF and change SET to RESET in line 60.

Program

```
10 'SPACE SHIP IN CHARACTER MODE
20 CLS
30 LINE (3,1)-(3,2), SET
40 LINE (2,3)-(4,4), SET, BF
50 LINE (1,4)-(1,5), SET
60 LINE (5,4)-(5,5), SET
70 DIM A%(7)
80 GET@(0,0)-(3,3),A%
90 CLS
100 FOR Y = 12 TO 1 STEP -1
110  PUT@(0,Y)-(3,Y + 3),A%
120 NEXT Y
130 CLS
140 GOTO 100
```

Disco Ship. Image of ship flashes on and off. In Graphics Mode, this program GETs the image in A%. After clearing the screen, it repeatedly PUTs the image back on the screen. Take note of the use of "XOR" as the action indicator; this technique is what causes the flashing.

Program

```
5 CLS
10 DIM A%(50)
20 LINE (3,1)-(3,2), SET: LINE (2,3)-(4,4), SET, BF: LINE
   (1,4)-(1,5), SET: LINE (5,4)-(5,5), SET
30 GET@ (1,1)-(5,6), G, A%
40 CLS
50 PUT@ (20,20)-(24,25),XOR,A%
60 FOR T = 1 TO 50: NEXT
70 GOTO 50
```

Advanced Graphics Programs

Try running some of these programs to see if they don't give you some insights into the possibilities of LEVEL III Computer Graphics.

Airplane.

```
10 CLS:DIMA%(50), B%(50)
20 LINE(91,11)-(124,14),SET,BF:LINE (117,8)-(121,8),SET:
   LINE (116,9)-(122,9),SET
30 LINE(115,10)-(123,10),SET:LINE(97,11)-(104,11),RESET:
   LINE (86,13)-(90,13),SET
40 LINE (100,13)-(110,13),SET
50 LINE(85,11)-(85,15),SET:GET@(42,2)-(63,5),A%
60 LINE(85,11)-(85,15),RESET:GET@(42,2)-(63,5),B%
70 X1 = 63:Y = 2:Y1 = 5:CLS
80 FOR X = 42 TO 0 STEP -1
90 PUT@ (X,Y)-(X1,Y1),A%
100 FOR T = 0 TO 5:NEXT T
110 PUT@(X,Y)-(X1,Y1),B%
120 FOR T = 0 TO 15:NEXT T
130 X1 = X1 - 1
140 NEXT X
150 GOTO 70
```

Flying Duck.

```
10 CLS:DIM A%(15),B% (15)
20 LINE (12,2)-(15,2),SET:LINE(6,3)-(9,3),SET:LINE(13,3) –
   (17,3),SET:SET(5,4):SET(10,4)
30 LINE (12,4)-(13,4),SET:LINE(3,5)-(12,5),SET:LINE(4,6)-(5,6),
   SET:LINE(10,6)-(11,6),SET
40 LINE(5,7)-(10,7),SET
50 GET@(0,0)-(8,2),A%
60 LINE(5,3)-(10,4),RESET,BF:LINE(6,6)-(9,6),SET
70 GET@ (0,0)-(8,2),B%
80 CLS:X1 = 8:Y = 0:Y1 = 2
90 FOR X = 0 TO 55
100 PUT@(X,Y)-(X1,Y1),A%
110 FOR T = 0 TO 2:NEXT T
120 PUT@(X,Y)-(X1,Y1),B%
130 X1 = X1 + 1
140 FOR T = 0 TO 10:NEXT T
150 NEXT X
160 GOTO 80
```

Boxes.

```
10 DIM A%(600), B%(600)
20 CLS: FOR T = 1 TO 4: LINE (RND(127),RND(47))-(127)
   RND(47)),SET,B:NEXT
30 F$ = "*":X = RND(63):IF X > 32 THEN M = 64 – X ELSE
   M = X
40 FOR I = 1 TO M: GET@(1,0)-(X,15),A%:GET@(X,0)-(62,15),B%
50 LINE (X,0)-(X + 1,15),F$,B:F$ = " "
60 PUT@(0,0)-(X – 1,15),A%:PUT@(X + 1,0)-(63,15),B%:NEXT:
   GOTO 20
```

Radar.

```
10 FOR R = 15 TO 1 STEP – 1
20 CLS
30 FOR Y = 0 TO 47 STEP 47
40 FOR X = 0 TO 127 STEP R
50 LINE (64,24)-(X,Y),SET
60 NEXT X,Y
70 FOR X = 0 TO 127 STEP 127
80 FOR Y = 0 TO 47 STEP R
90 LINE (64,24)-(X,Y),SET
100 NEXT Y,X
110 NEXT R
```

More Ships.

```
10 CLS
20 LINE (3,1)-(3,2),SET:LINE(2,3)-(4,4),SET,BF:
   LINE (1,4)-(1,5),SET:LINE (5,4)-(5,5),SET
50 DIM A%(2):GET@(1,1)-(5,5),G,A%
60 CLS:DIMB%(2)
70 LINE (0,2)-(1,2), SET:LINE (2,1)-(4,3), SET, BF:
   LINE (4,0)-(5,0), SET:LINE (4,4)-(5,4), SET
110 GET@ (0,0)-(5,4),G,B%
120 CLS:X = 120:Y = 41
130 IF Y = 0 OR X = 0 THEN 120
140 D = RND(2):IF D = 1 THEN 190
150 S = RND(15):IF X - S < 0 THEN S = X
160 FOR X = X TO X - S STEP -1
170 PUT@(X,Y)-(X + 5,Y + 4),SET,B%
180 NEXT X:X = X + 1:GOTO 130
190 S = RND(5):if Y - S <0 THEN S = Y
200 FOR Y = Y TO Y - S STEP -1
210 PUT@(X,Y)-(X + 4,Y + 4),SET,A%
220 NEXT Y:Y = Y + 1:GOTO 130
```

Chapter FOUR:

LEVEL III Features From Disk BASIC

- INPUTting String Literals with LINE INPUT
- Adding a Time Limit with #LEN
- Replacing a Portion of One String with Another String
- You Can Search a String for a Substrin
- How to Define Functions
- Up to 10 Machine Language User Routines
- How to Convert Hex and Octal to Decimal
- SYSTEM Command Caution

INPUTting String Literals with LINE INPUT

LEVEL III has a LINE INPUT statement that is frequently used for inputting string literals. The format of LINE INPUT is as follows:

```
LINE INPUT ["prompt string";] string variable name
                                 such as A$, B$, etc.
```

Unlike Level II's INPUT statement, a LINE INPUT statement assigns a string variable name to the entire line of input. Every character typed up to ENTER is part of the string, including punctuation and leading spaces.

LINE INPUT is also different from INPUT in that it doesn't automatically display a question mark (?) when it is executed. If you want a question-mark prompt, you have to make it part of the prompt string. An example of LINE INPUT where ? is part of the prompt follows:

```
5 LINE INPUT "CITY?,STATE?" ;CS$
```

When this statement is executed, the screen will display the prompt: CITY?,STATE? The answer (user input) is assigned as a string literal to CS$.

Example program. You could use the above LINE INPUT statement in a program like the following:

```
10 LINE INPUT "CITY?,STATE?";CS$
20 PRINT CS$
RUN
CITY?,STATE? NEW YORK, NEW YORK
NEW YORK, NEW YORK
```

This program prints out the entire user input NEW YORK, NEW YORK, including the comma (,).

Adding a Time Limit with #LEN

INPUT and LINE INPUT statements have an optional feature called #LEN. #LEN lets you impose a limit on the length of time allowed before a response is given to the INPUT statement. It is particularly useful for game programs and computer assisted instructions. The format of the statement is:

```
[LINE]INPUT#LEN n,m;["prompt string";]variable
  name(s)
```

where n is the time limit in seconds and m is the line number to branch to if the time limit is reached.

The remainder of the statement is the same as the INPUT statement. The prompt string (if given) and a question mark are printed, and the items typed in at the terminal are assigned to the variables in the list.

Example program:

```
5 S = 4
10 X = RND(100)
20 Y = RND(100)
30 PRINT X;" + ";Y;" = ";
40 INPUT#LEN S, 100;Q
50 IF Q = X + Y THEN PRINT "SMART" ELSE PRINT "DUMB"
60 GOTO 10
100 PRINT "SLOW!"
110 GOTO 10
```

Replacing a Portion of One String with Another String

In Level II BASIC, the MID$ function is used to return a substring of a given string. MID$ has an additional capability in LEVEL III BASIC. It may be used on the left side of an equation to replace a portion of one string with another string. The general format is:

MID$(string1,n[,m]) = string2

where n and m are integer expressions.

The characters in string1, beginning at position n, are replaced by the characters in string2. m is optional; it refers to the number of characters from string2 that will be used in the replacement. If m is omitted, all of string2 will be used. However, regardless of whether m is omitted or included, the replacement of characters will never go beyond the original length of string1.

Example:

```
10 A$ = "KANSAS CITY, MO"
20 MID$(A$,14) = "KS"
30 PRINT A$
RUN
KANSAS CITY, KS
```

You Can Search a String for a Substring

The INSTR function in LEVEL III BASIC eliminates the need for an "instring subroutine," as described in the Level II BASIC manual. INSTR provides the same capability, and it's much easier to use. The format of the INSTR function is:

```
INSTR([n,]string1,string2)
```

INSTR searches string1 for a substring that matches string2. When a match is found, INSTR returns the starting position of the match. n is an optional integer offset which designates the starting position for the search. If n is omitted, the search starts with the first character in string1.

If no match is found, or if n is greater than the length of string1, or if string1 is null, INSTR returns zero. If string2 is null, INSTR returns n (if specified) or one.

Example:

```
10 A$ = "ABCDEABCDE"
20 B$ = "BC"
30 PRINT INSTR(A$,B$)
40 PRINT INSTR(3,A$,B$)
RUN
2
7
```

How to Define Functions

Often a program will contain a particular operation that is repeated several times. When this happens, you can save time and memory by defining your own function to perform that operation. Then, instead of writing out the entire operation each time, it is only necessary to do a function call.

The DEF FN statement is used to name and define user functions. The format of the statement is:

```
DEF FNvariable name (parameter list) = function
  definition
```

The variable name is any legal variable name. This name, prefixed by FN, becomes the name of the function. The parameter list is comprised of those variable names in the function definition that are to be replaced when the function is called. The parameter list is enclosed in parentheses and the items in the list are separated by commas. The function definition is an expression that performs the necessary operation. Variable names that appear in this expression serve only to define the function; they do not affect program variables that have the same name. A variable name used in a function definition may or may not appear in the parameter list. If it does, the value of the parameter is supplied when the function is called. Otherwise, the current value of the variable is used.

Example. In the following example, a function is defined that adds the second power of one number to the third power of another.

```
10 DEF FNA (L,M) = L↑2 + M↑3
20 C = 1:D = 2
30 PRINT FNA (C,D),FNA(5,2),FNA(4,3),FNA(2,3)
RUN
9         33          43          31
```

A function is defined called DEF FNA that adds the second power of one number to the third power of another.

The DEF FN statement can also be used to define a string function, as in the following program:

```
5 CLEAR 500
10 DEF FNST$(A$,B$) = A$ + "," + B$
20 INPUT "FIRST NAME";FN$
30 INPUT "LAST NAME";LN$
40 X$ = FNST$(FN$,LN$)
50 PRINT X$
RUN
FIRST NAME? ALEX
LAST NAME? HAMBONE
ALEX HAMBONE
```

NOTE: The variable name in the second example ends with a dollar sign ($). Function name variables, like other kinds of variables, must indicate the type of value to be returned. Thus, a function name variable may end with a $ (string), # (double precision), % (integer), or ! (single precision). The default is single precision (!).

Up to 10 Machine Language User Routines

In LEVEL III BASIC, the USR function has been expanded so that 10 different machine language user routines may exist in memory at the same time. As with Level II BASIC, the routines may be assembled with the TRS-80 Editor/Assembler and loaded with the SYSTEM command, or they may be POKEd into memory. However, it is no longer necessary to POKE the starting address of a user routine into memory. The DEFUSR statement is provided for this purpose.

USR Function. The new format for the USR function is:

USR[n](argument)

where n is an integer from 0 to 9 and argument is the value to be passed to the user routine. A USR function with no n is the same as USR0. The USR function calls a specific user routine in memory, namely the one beginning at the address specified in the corresponding DEFUSR statement.

DEFUSR Statement. The DEFUSR statement tells BASIC the starting address of a USR routine. The format of the statement is:

```
DEFUSR[n] = address
```

where n is an integer from 0 to 9, corresponding to the number of the user routine located at "address." (If n is omitted, DEFUSR0 is assumed.)

For example: A user routine called USR3 has been POKEd into memory beginning at address 28000. Before calling the user routine, the program must execute the statement:

```
DEFUSR3 = 28000
```

A calling statement for this routine might look like:

```
A = USR3(B)
```

If a user routine is called before the corresponding DEFUSR statement has been executed, an ILLEGAL FUNCTION CALL error results.

It is still possible to POKE the starting address of USR0 into memory, as described in Chapter 8 of the Level II BASIC Manual.

How to Convert Hex and Octal to Decimal

To convert hexadecimal and octal numbers to **decimal numbers, prefix** the hexidecimal number with:

&H

and the octal number with:

&

This will work most all LEVEL III instructions. The exceptions are DATA statements and in response to INPUT statements. Also, hex conversion does not work in FOR...NEXT loops, though octal conversion does work.

When combined with the PRINT command, this feature can be used as an octal/hexadecimal to decimal conversion calculator. For example,

PRINT &H4A5F ENTER

will return the correct decimal conversion: 19039

POKE Example. To POKE the hex value FF (255 decimal) into location 4A5F, use the statement:

POKE &H4A5F, &HFF

A subsequent PEEK at location 19039 will return: 255

SYSTEM Command Caution

If you enter a SYSTEM command while in LEVEL III BASIC, you will find that upon returning to LEVEL III, the ERROR messages are scrambled.

To avoid this problem, we recommend that you enter the SYSTEM command from Level II BASIC.

Chapter FIVE:

LEVEL III Expansion Interface Features

- ✓ **LEVEL III's Clock and Calendar**
- ✓ **How to Turn Off the System Clock**
- ✓ **How to Output to an RS-232 Port**
- ✓ **Lockout Recovery**

LEVEL III's Clock and Calendar

TIME$ is a string that keeps track of the date and time. The format of the string is:

```
MM/DD/YY HH:MM:SS
```

When you load LEVEL III BASIC, TIME$ contains all zeros but immediately starts keeping track of the seconds, minutes, and hours elapsed.

To set the date and the actual time, type the CMD"R" command with a 17-character argument that represents the month, date, year, hour, minute, and second. For example, to set the date at November 21, 1979 and the time at one-thirty a.m. exactly, type:

```
CMD"R","11 21 79 01 30 00"
```

BASIC will supply the punctuation, you only need to type spaces between the numbers. Note that you must include leading "0".

After executing this command, the time and date will increment properly for as long as the TRS-80 is turned on.

TIME$ may be used in any program that requires a timer or reference to a specific day, hour, etc.

How to Turn Off the System Clock

Note: If your TRS-80 includes an Expansion Interface, you need to be aware of the following:

Tape operations **other than SAVE and LOAD*** on a TRS-80 with Expansion Interface are vulnerable to interruptions from the system clock. Therefore, before doing a SYSTEM, INPUT# – 1, or PRINT# – 1 command, you must turn off the system clock. And once the command is executed, the system clock should be turned back on.

The command that turns the clock off is CMD"T" and the command that turns it back on is CMD"R." These commands may also be used as program statements.

Example. Here we turn the clock off, read values from tape, and turn the clock back on.

```
100 CMD"T"
120 INPUT# – 1,X,Y,Z
130 CMD"R"
```

***SAVE and LOAD turn the clock off and back on again automatically.**

How to Output to an RS-232 Port*

LEVEL III's PRINT# – 3 statement is used just like the PRINT statement, except data is output to the RS-232 port. The format of the statement is:

PRINT# – 3, list of items

Now it's easy to do output to a line printer (or other device) that is hooked up to your RS-232 port. Input from an RS-232 port still requires the use of machine language routines.

The first character sent by BASIC to the RS-232 causes the RS-232's UART to initialize using the switches set on the RS-232. In order to override this default initialization, send a dummy character to the RS-232 port and then re-initialize using a machine language subroutine.

*Requires RS-232 port.

Lockout Recovery

Two common occurrences that cause system lock-out are:

1. An attempt to execute an LLIST or LPRINT while the line printer is off line, or
2. An attempt to execute a LOAD while the recorder is off line.

With Level II BASIC, reset must be used to recover from the lock-out and, if your system includes an expansion interface, this results in loss of the current program.

LEVEL III BASIC monitors the break key during printer I/O and LOAD lock-outs. So you can use the break key to recover from a system lock-out without losing your program.

MICROSOFT
CONSUMER PRODUCTS
10800 Northeast Eighth, Suite 819
Bellevue, WA 98004

Catalog No. 1011
Part No. 10F01

Printed in U.S.A.