RCA

VIP TINY BASIC

INSTRUCTION MANUAL

VP 700

RCA COSMAC VIP MARKETING
New Holland Avenue
Lancaster, PA 17604

71L245

CONTENTS

REFERENCES

RCA COSMAC VIP Instruction Manual VIP-311
RCA COSMAC VIP User's Guide VIP-320

SECTION I

INTRODUCTION

When writing a language text, an author is confronted with an almost insurmountable task; to provide not only a dictionary containing the vocabulary, but to provide syntax rules, pronunciation guides, and dialectic "flavor". Regardless of the language being studied, most textbooks fail to provide sufficient information to permit the student to become truly fluent in the new language.

Fortunately, this is a condition that can be easily overcome when language being studied is a "computer" language. There are no pronunciation problems in a computer language, each word is rigidly defined, and the syntax is unforgiveable. Either you say it right or you're not understood.

This particular text is divided into several sections for the convenience of the student. The section entitled "Learning the Vocabulary" examines each of the available BASIC words in detail and offers examples for practice and reference. Section IV, "Learning the Language" consists of instructions regarding the "how to do it" of using the language. Once you've mastered the vocabulary and the syntax, you're led through the construction of a complete BASIC program, one step at a time. You'll then want to read annotated programs to see just how the new words are used in actual working programs written by professional programmers. Several such programs are listed, along with editorial asides.

The several appendices to the manual provide reference material

for use by the student who has grasped the fundamental material and simply needs to refresh his memory. Information regarding the necessary equipment, as well as wiring diagrams and technical data, are presented in these appendices for use by the hardware-oriented student.

WHAT IS BASIC, ANYWAY?

BASIC is an acronym for Beginner's All-purpose Symbolic Instruction Code. It was devised by Dartmouth professors years ago and has become the language used by personal computerists and hobbyists all over the world. BASIC is popular primarily because it's so easy to learn. Another reason BASIC is popular is that it's easy to write a machine language program, called a BASIC interpreter, which interprets the BASIC source program instructions devised by the user (that's you) and follows these instructions.

Some versions of BASIC (including the original Dartmouth version) were devised to be used on big time-sharing computers by people who weren't accustomed to dealing intimately with computers. Some versions (including the VIP BASIC) are composed of an abbreviated form of the original Dartmouth version plus additional keywords designed especially for use on the specific computer being used. While not all of the "standard" BASIC keywords are available in VIP BASIC, the VIP "dialect" provides color and sound instructions, as well as reserved, special purpose variables, which give you lots of room for creating games and displays - instructions that can't be found in other dialects of BASIC.

VIP BASIC is an "integer" BASIC, which means it uses only whole numbers. The range of numbers is quite large - from -32768 to +32767. It's also a "graphics-oriented" language which permits you to display a pattern of your choice anywhere on the screen - in any of eight different colors - against a background which can be any of four different colors - while it plays simple musical selections. VIP BASIC resides

in read only memory (ROM). BASIC in ROM elminates the need to load the BASIC interpreter by means of cassette tape.

A BASIC program consists of a sequence of BASIC statements, each of which starts with a line number (in the range 1 to 32767). Each statement must contain a BASIC keyword and an operand. The operand may be a variable, a number, or a combination of numbers and variable names (called an "expression"). Each line of code (each statement) must end in a carriage return (obtained by pressing the RETURN key).

The VIP begins execution of the BASIC program when you type RUN (and press the RETURN key). Execution begins at the lowest line number and continues until an END statement is encountered, until the highest number has been executed, or until you stop the program by either pressing CONTROL and @ keys or by switching the VIP toggle switch to RESET.

Unlike most BASIC texts you'll encounter, this manual will <u>first</u> tell you how to correct your mistakes - so you won't have to hunt through the text as you're learning the language. Since you'll find yourself making many errors in the beginning, we'll explain what the error messages are <u>before</u> there's any likelihood you'll encounter them. And, again before introducing you to the vocabulary, you'll learn about the "special" keys on the keyboard as well as some of the miscellaneous information which is usually tucked away in a hard to find place in the manual.

Then, a quick tour of the keywords - a sort of dictionary. And, finally, lessons on how to use the BASIC language.

SECTION II

EDITING

Editing A Character

If you have not completed a line (that is, you haven't pressed the RETURN key), use the backspace key to erase the last character entered.

If you have already pressed the RETURN key, you must retype the entire line in order to change any character in that line.

Deleting A Line

If you wish to delete an existing line from a BASIC program, type a RETURN followed by the line number you wish to delete, and press the RETURN key.

Inserting A Line

If you wish to insert a line between existing program lines, there must be an unused line number available between those lines. For this reason, you may wish to initially enter the program statements in line number increments of ten. This technique gives you nine unused line numbers between existing program lines for later insertion of new statements.

Deleting A Program

You may delete a program from the program storage area by typing the word NEW and pressing the RETURN key, or by moving the toggle switch on the VIP to RESET and then RUN.

ERROR MESSAGES

WHAT?	The BASIC interpreter cannot recognize the statement. Usually this is caused by an error in spelling or syntax. A question mark will appear at the point in the statement which caused the first error.
HOW?	The BASIC interpreter doesn't have enough information to perform the requested operation. This is usually caused by asking the interpreter to GOTO a non-existent line number. A question mark will appear at the point in the statement where the error first occurred.
SORRY?	The BASIC interpreter cannot perform a request because its memory resources are used up. A question mark will appear in the statement which requested unavailable resources.
LD ERR	The program was not loaded properly from the tape. Try to load it again.
ERR	The data entered for an INPUT statement is not valid. Usually this is caused by trying to enter alphabet characters rather than numeric data. Only numeric characters may be used as data for an INPUT statement.

Keyword		Definition
RUN	(R.)	Command only. RUN starts program execution beginning at the lowest line number available. To terminate execution of the program, press CONTROL @. If you want the program to be saved, you must BACKSPACE while flipping to RUN otherwise your program will be gone!
SAVE	(S.)	Command only. SAVE writes the program currently stored in the program storage area to cassette tape. While the program is being saved, the TV display is turned off and the keyboard is locked out. If the hardware is present, SAVE will start and stop the cassette motor automatically otherwise use the manual tape player controls. You may wish to store only one program on each tape. A C-10 cassette tape will easily hold the longest VIP BASIC program. ex. SAVE
LOAD	(LO.)	Command or statement. Load clears the program storage area and loads the first program encountered from cassette tape. When used as a program statement, LOAD will cause the newly loaded program to begin execution automatically at the lowest numbered line. If the hardware is present, LOAD will

Keyword		Definition
LOAD	(LO.)	start and stop the cassette motor automatically. While a program is being loaded, the TV display is turned off and the keyboard is locked out. ex. LOAD NOTE: Do not attempt to load CHIP-8 programs as BASIC and vice versa.
CLS	(CL.)	Command or statement. CLS clears the TV screen and positions the cursor in the upper left corner of the screen. ex. 10 CLS
END	(E.)	Statement only. END halts program execution anywhere in the program. It is not necessary to use END to terminate a program since the program will stop after the highest available line number has been executed. ex. 21Ø END
GOTO	(G.)	Statement only. GOTO causes the BASIC interpreter to branch to the statement with the specified line number. ex. 1ØØ GOTO 2ØØ
GOSUB	(GOS.)	Statement only. GOSUB causes the BASIC interpreter to branch to a subroutine which begins at the specified line number. The interpreter must

Keyword		Definition
GOSUB	(GOS.)	encounter an associated RETURN statement at the end of the subroutine in order to get back to the statement following the line which called the subroutine. ex. 12Ø GOSUB 2ØØØ
IF THEN	(I.) (T.)	Statement only. IF evaluates a condition or expression. If the expression or condition is true (having a value of greater than zero), THEN causes a branch to a specified line number. If the condition if false (having a value of less than or equal to zero) then next line in sequence is executed. ex. 1Ø IF X=2Ø THEN 2ØØ 2Ø IF A THEN 3ØØ
INPUT	(IN.)	Statement only. INPUT permits entry of date from the keyboard during program execution. The programmer must select a variable name into which INPUT data is to be stored. ex. 1ØØ INPUT A More than one data item may be INPUT with one INPUT statement. The items must be separated by commas both in the INPUT statement itself and

Keyword		Definition
INPUT	(IN.)	in the response typed in during program execution. ex. INPUT A, B, C
LET	(LE.)	Statement or command. Assigns a value to a variable. The variable name must be separated from the value by an equals sign, and the variable name must be on the left hand side of the equals sign. The word LET is optional; if the BASIC interpreter encounters a variable name followed by an equals sign, it will automatically assign the value of the number or variable or expression on the right hand side of the equals sign to the variable on the left. ex. 4Ø LET X=45 is the same as 4Ø X=45 5Ø LET X=A+2 is the same as 5Ø X=A+2 6Ø LET X=C is the same as 6Ø X=C
PRINT	(P.) (?)	Statement or command. Prints the value of a variable or expression. PRINT will also print a string of alphabetic characters if the characters are enclosed within quotation marks. PRINT, when placed on a line by itself, will print a blank line.

Keyword		Definition
PRINT	(P.) (?)	ex. 10 PRINT A 20 PRINT A+X 30 PRINT "VIP BASIC!" 40 PRINT Numbers are left justified, that is, they begin at the left of the line. Semicolons cause the values to be printed immediately adjacent to each other, with no spaces between them. Commas cause the values to be printed on separate lines. Semicolons at the end of a PRINT statement suppress the normal carriage-return/line feed. ex. 5Ø PRINT "X is equal to"; X 6Ø PRINT X; Y; Z;
REM	(RE.)	Statement only. REM allows you to comment your program. Alphabetic characters, spaces, and special characters may be used in REM statements without enclosure within quotation marks. ex. 3ØØ REM VARIABLE C IS COLOR
RETURN	(RET.)	Statement only. RETURN signifies the end of a subroutine which was called by the GOSUB keyword. Every BASIC subroutine must end with a RETURN statement. ex. 49ØØ RETURN

THE VIP DIALECT

The following keywords are unique to VIP BASIC, and have been implemented to permit you to use the VIP Color Board, VP-590, and the VIP Simple Sound Board, VP-595, with your BASIC programs.

COLOR (C.) Statement or command. COLOR, when used with only one paramenter, increments the sequence of background colors. When the READY prompt appears on the screen after RESET, the background color is initialized to blue. Each time COLOR is implemented, the background color is changed to the next color in the sequence. The color sequence is:

COLOR is a function, and, as such, requires a parameter to be associated with it. The parameter may be any alphanumeric character at all, since it's a "dummy" parameter and is not used by the BASIC interpreter.

ex. COLOR C

```
1Ø COLOR 4
200 COLOR P
```

COLOR (C.) Statement or command. COLOR, when used with its

COLOR (C.) full complement of five parameters, sets foreground color. The available foreground colors are:

Ø black	4 green
1 red	5 yellow
2 blue	6 aqua
3 violet	7 white

The COLOR blocks are eight bits (1 byte) wide and four bytes high. The required parameters are:

a. Color code or variable name containing code.

b. X horizontal position of upper left corner of the color block.

c. Y vertical position of upper left corner of the color block.
Note that X & Y can be any variables, or any value, although it is common to use X & Y for clarity.

d. nh number of horizontal color blocks to be displayed using this color.

e. nv number of vertical color blocks to be displayed using this color.

Each parameter in the COLOR must be used, and each parameter must be separated from the rest with a comma.

COLOR (C.)

10 COLOR 2,10,15,2,3

will display color 2 (blue) beginning at the color block containing X coordinate 10 and Y coordinate 15. Two horizontal blocks will be displayed in this color, and three vertical blocks.

20 COLOR A,X,Y,H,V

Variable names can be used to store the values of all the parameters used in the COLOR statement.

30 COLOR A,X,Y,2,3

A mixture of variable names and numeric values may be used to set the foreground color.

GOKEY (GOK.)

Statement only. Performs no function when used as a command. GOKEY causes the interpreter to branch to the specified line number if any key on the keyboard is pressed. GOKEY does not wait for the user to press a key. (See KEY variable.)

10 GOKEY 200

PRINT AT (?A.) (P.A.)

Statement or command. PRINT AT prints the desired variable or value in the specified X and Y locations on the screen. Commas or semicolons separate the

PRINT AT (?A.) (P.A.)

coordinates from the variables and separate the variables from each other. Numbers are left justified and printed immediately adjacent to each other (commas are treated as semi-colons). The position of the cursor is not affected by the PRINT AT statement, so you may want to end the statement with a semi-colon to avoid unwanted erasure of characters.

ex. 210 PRINT AT X,Y,A,B;

220 PRINT AT X,Y,4;6;

230 PRINT AT X,Y,4;B:

Strings in quotes can also be used.

ex. 240 PRINT AT X,Y,"HELLO!"

SHOW (SH.)

Command or statement. SHOW displays the pattern stored in the special variable PT at the specified X and Y coordinates on the screen. You may also specify the number of times the pattern is to be repeated horizontally and vertically.

ex. 210 SHOW X,Y

220 SHOW X,Y,2,3 The pattern will be repeated twice in the X and three times in the Y direction.

SPECIAL VARIABLES

Variable Name		Function
TI		Sets the internal timer to the value of the expression. The time value is in 60th's of a second. Once the time is set, the timer counts down to zero. TI can be tested to obtain the current timer value. ex. 10 TI=300 Sets the timer to 5 seconds 20 X=TI Sets X to the current timer value 30 IF TI>Ø then 30 waits for timer to reach Ø
FQ	(F.)	Sets the frequency of the tone generator in the Simple Sound board(VP-595) to the value of the expression, from Ø to 255, for different musical notes. No tone is generated when FQ is set at zero. FQ does not turn on the tone generator, but stores the frequency of the tone to be used by the generator when it is turned on. ex. FQ=165
TO		Turns on the tone generator and sets the tone timer to the value of the expression, from Ø to 255. Once the tone timer is set, it counts to zero and the tone is turned off. The tone time value is specified in 60th's of a second. ex. TO=180 Sets the tone timer to 3 seconds

Variable Name	Function
PT	Stores a pattern for use by the SHOW keyword. The pattern is a sequence of hexadecimal digit pairs (one byte) that represent the desired image. A maximum of 16 hex-digit pairs may be used per assignment to PT. ex. PT=FFFFFFFF Prints a solid block 8 bits wide and 4 bytes high.
TVOFF	Command or statement. TVOFF turns off the TV display so programs can execute faster. The background color will still be visible but not graphics. TVOFF disables the tone and interval timers. If you implement TVOFF in your program, you can use the GOKEY keyword to break into the program so you can turn the TV display back on and see just where your program is. You can also use break to achieve the same results. ex. 41Ø TVOFF
TVON	Command or statement. TVON turns the TV display on, and enables the tone and interval timers as well as CONTROL @. ex. 45Ø TVON

FUNCTIONS

ABS — ABSOLUTE VALUE, ABS, gives the magnitude of a number without regard to the sign and requires an argument in parenthesis. In the following example, if N is a negative number, X will be positive. If N is a positive number, X will also be positive.

ex. 15Ø X=ABS(N)

RND — RND generates a random number between Ø and 255. This number is an integer and must be stored in a variable or used immediately.

ex. 1ØØ X=RND

11Ø X=2*RND+A

HIT (H.) — A reserved variable is a special variable which is set by the BASIC interpreter when certain instructions are executed. They can be interrogated from user programs to determine their status. Reserved variable containing a zero if the last SHOW statement or PRINT AT statement had no hit. A HIT occurs when a pattern is displayed over another pattern(i.e., in the same cells on the screen). HIT is set to 1 if such a hit occurs.

ex. 130 IF H. THEN 250 Will cause a branch to line number 250 if HIT contains a 1. If HIT contains a Ø, the next line (the line immediately following line 130) is executed.

ex. 140 A=HIT

KEY (K.) Reserved variable containing the ASCII value of the key pressed during execution of the GOKEY statement.

ex. 170 P. KEY

MEM (M.) Reserved variable containing the number of bytes of memory in the program storage area available for use. This is called "free" space. MEM therefore contains the number of "free" bytes of memory.

ex. P. MEM

100 PRINT M.

KEYWORD ABBREVIATIONS

Keywords may be abbreviated by using the first one or two characters of the keyword and following it with a period.

	Keyword	Abbreviation
BASIC	AT	A.
	CLS	CL.
	COLOR	C.
	END	E.
	GOTO	G.
	GOKEY	GOK.
	GOSUB	GOS.
	IF	I.
	INPUT	IN.
	LET	LE.
	LIST	L.
	LOAD	LO.
	NEW	N.
	PRINT	P.
	REM	RE.
	RETURN	RET.
	RND	R.
	RUN	R.
	SAVE	S.
	SHOW	SH.
	THEN	T.

	Keyword	Abbreviation
Functions	ABS	A.
	HIT	H.
	KEY	K.
	MEM	M.
Variables	FQ	F.
	ON	O.

SECTION IV

LEARNING THE LANGUAGE

It isn't enough to simply know the vocabulary of a new language. One must also know something about the syntax - the grammar, if you will, in order to be considered "fluent" and to be certain of being understood when attempting to communicate with a "native" of that language. This section describes each keyword in somewhat greater detail, showing you the context in which each keyword is properly used. In some instances, program segments are listed, and it is suggested that you take the time to enter these segments into VIP memory and see for yourself how the keywords function.

In some instances when the keyword can only be used as a command and not as part of a BASIC program statement, an explanation is offered without a program segment to illustrate the use of the keyword. Again, you are advised to enter the keyword and see for yourself how it functions.

LET PRINT GOTO

```
10 LET X=1
20 LET Y=1
30 LET X=X+1
40 LET Y=Y+1
50 PRINT X
60 PRINT Y
70 GOTO 30
```

This program shows you how to assign values to variables, and how to print those values on the screen. It also shows you how to use the addition operator - just as you would use it if you wrote the addition problem out on paper. Use the other arithmetic operators the same way - but remember that * stands for multiplication and / stands for division.

The GOTO keyword in line 70 causes the program to branch back to line 30 so another "1" can be added to X. This program will go on forever - until one of four things happens: A) You get tired of it and move the toggle switch on the VIP to RESET; B) X or Y reaches a value of 32768 (which is beyond the range of numbers for VIP BASIC); C) There is a sudden power failure and the VIP is powered off; or D) You press CONTROL and then "@" key while holding the CONTROL key down.

We're going to use this program to show you the advantages of using the BASIC keyword abbreviations - and of not bothering to enter all the spaces between characters in your programs.

ABBREVIATIONS + SPACES

```
10 LE. X=1
20 LE. Y=1
30 LE. X=X+1
40 LE. Y=Y+1
50 P. X
60 P. Y
70 G. 30
```

LET IS NOT REQUIRED!

```
10 X=1
20 Y=1
30 X=X+1
40 Y=Y+1
50 P. X
60 P. Y
70 G. 30
```

ABBREVIATIONS AND NO SPACES

```
10X=1
20Y=1
30X=X+1
40Y=Y+1
50?X
60?Y
70G.30
```

When you list any of these three versions of the program, it will look exactly like the first version (at the top of this page).

The VIP automatically spells the keyword in full and inserts all the spaces necessary to make your program easy to read. Obviously, you will save keystrokes (and time) by using the version captioned "ABBREVIATIONS AND NO SPACES". Let the VIP do the work for you whenever you can....that's what it's there for!

A few more words about PRINT: You don't have to use a separate PRINT statement to print each variable. You can use a "print list" in one PRINT statement, provided you separate each item in the list from all the other items. A comma can be used to separate the items; the items will each be printed on its own line. If you like, you can use a semicolon to separate the items. If you use a semicolon, however, the items are printed immediately adjacent to each other - with no intervening spaces. You must provide the spaces, like this:

```
50 PRINT X;" ";Y
```

Type this statement into your program - and delete line 60 by typing only the line number (60) and pressing the RETURN key. Now run the program - both the X and Y value will be printed on the same line.

Alphabetic (string) messages can be printed, just as variable values can be printed. The string must be preceded by a quotation mark. VIP BASIC allows you to omit the closing quotes, but never the opening quotes. And you may not omit the closing quotes if you're using a comma or a semicolon at the end of the message.

```
 10 CLS
100 PRINT"HELLO"
110 PRINT"GREETINGS!
120 ?"HI THERE
```

Speaking of "loops" brings us to the subject of the GOTO keyword. In

our example program, GOTO causes the interpreter to "loop" back to line 30 every time line 70 is encountered. GOTO is the best way to perform repeated functions such as incrementing the value of a variable, as we do in this example. When the GOTO statement is directed at itself (that is, if line 70 said GOTO 70), the effect is that the program seems to stop everything and wait for you to either switch to RESET or to press CONTROL @. It is "branching" to itself, or waiting. This can be useful when you want to examine the results of a calculation or a display.

You can GOTO any existing line in a program. If you attempt to branch to a non-existent line number, you'll get a HOW? error message from the VIP BASIC interpreter.

CLS INPUT IF/THEN REM END

This is a "real" program - not just a program segment. It allows the user to select two numbers for the VIP to add or subtract. This example is used to show you how to utilize the PRINT keyword to enable your user to "communicate" with the computer - to have the feeling that he's in control.

```
 10 CLS                          Clear the screen
 20 ?"TYPE A NUMBER ";           Print a prompting message
 30 INPUT X                      Input the first number
 40 ?"ANOTHER NUMBER ";          Print another prompt
 50 INPUT Y                      Input the second number
 60 ?"SHALL I ADD (1)"           Print still another
 70 ?"OR SUBTRACT (2)";             prompting message
 80 INPUT C                      Input the user's decision
 90 IF C=0 THEN 400              User wants to quit
100 IF C=1 THEN 200              User wants to add
110 IF C=2 THEN 300              User wants to subtract
120 CLS                          Clear the screen
130 ?"TYPE 0 TO QUIT"            User entered the wrong
140 ?"OR 1 TO ADD OR"               number - something
150 ? 2 TO SUBTRACT ";              other than 0, 1, or
160 GOTO 80                         2 - give him another chance.
200 REM * ADDITION               User wants to add
210 ?                            Print a blank line
220 ?X;"+";Y;"=";X+Y             Print the problem and the solution
230 ?                            Print another blank line
240 GOTO 20                      And let the user do it again.
300 REM * SUBTRACTION            User wants to subtract
310 ?                            Print a blank line
320 ?X;"-";"=";X-Y               Print the problem and the solution
330 ?                            Print another blank line
340 GOTO 20                      And let the user do it again
400 REM * QUIT                   User wants to quit
410 CLS                          So clear the screen
420 ?"THANKS!"                      And thank him nicely
430 END                             And quit.
```

In lines 90, 100, and 110, the IF/THEN statement evaluates the value entered by the user and stored in variable C. If the condition (C=0, in line 90, for example) is true, THEN causes a branch to the

specified line number (400).

The REM keyword in lines 200, 300, and 400 are not executable lines of the program - they're inserted to make it easier for you to remember what's going on. Five years from now, you'll find it helpful!

RND FQ TO TVON TVOFF TI

```
 10 CLS
100 FQ=RND
110 TO=RND
120 GOTO 100
```

This program selects a random frequency (a musical note) and plays it for a random time (TO sets the note's duration). It will go on playing forever - until you switch to RESET or press CONTROL @. Of course, you must have the VIP Simple Sound board connected to your VIP for this program to work.

```
 10 CLS
 20 TVOFF
 30 I=1
 40 I=I+1
 50 IF I < 10 THEN 40
 60 TVON
 70 P.I
 80 I=1
 90 I=I+1
100 IF I < 10 THEN 90
110 P. I
```

This program demonstrates the TVON/TVOFF features of the VIP BASIC "dialect". It takes about 2 seconds for I to equal 10 if the TV is turned off, and about 5 seconds for I to equal 10 when the TV is on. It will be faster, obviously, to do number manipulation when the TV is turned off. It is interesting to note that the tone timer is turned off when the TVOFF keyword is used, and that it's turned back on when the TVON keyword is used.

```
 10 CLS
 20 PT=FF
 30 TO=50
 40 SHOW 5,5
 50 TVOFF
 60 GOTO 60
```

With this program, you'll see that you can not use the keyboard when TVOFF is implemented. The pattern will blink on the screen, and when line 50 is executed, the TV is turned off. With the TV off, the program loops in line 60 forever - until you switch to RESET.

```
 10 CLS
 20 TI=1000
 30 FQ=150
 40 TO=5
 50 P. TI
 60 IF TI > 0 THEN 40
```

This program demonstrates the use of the VIP's interval timer. In line 20, the timer value is set to 1000. The timer counts down to zero (its value is tested in line 60) and the value is displayed so you can see it.

If you plan to use the Simple Sound keywords very often, it is suggested that you read the manual that comes with the board. It will tell you which values of FQ correspond to which musical notes; which values of TO correspond to which note durations (such as half-notes, quarter-notes, etc.).

SHOW PT

In the VIP Instruction Manual, you were shown how to make hexadecimal patterns to be stored in memory for later display on the screen. The same patterns can be used with VIP BASIC, and are stored in the special variable PT. PT contains one horizontal byte and up to 16 vertical bytes at a time. PT is utilized by the SHOW keyword, which displays the pattern on the screen. SHOW has a field width of one byte horizontally and one bit vertically (and if you're familiar with the patterns in the VIP Instruction Manual, you'll understand this. If you aren't familiar with those patterns, you are referred to page 29.

The SHOW keyword permits you to display your pattern repeatedly in both the horizontal and vertical directions. The program below stores one byte of pattern in PT, and displays it only once. After you've run the program once, we'll show you how to cause the pattern to be repeated.

```
 10 CLS
100 PT=FF
110 SHOW 0,0
120 SHOW 0,0
130 ? HIT
140 GOTO 140
```

Notice that each pattern is SHOWn twice. The second display erases the first. When the pattern is displayed in the same place twice, the HIT variable is set to 1 to show that a "hit" has taken place. This is useful if you're playing a game and want to know if a target has been hit.

The SHOW keyword permits up to four parameters to be used. At least two of them must be used (the X and Y coordinates) or a WHAT? error message will result. The other two parameters permit you to specify the number of times you want the pattern to be repeated horizontally and vertically. Let's look at the program again, and use the second set of parameters:

```
 10 CLS                    Clear the screen
100 PT=FF                  Set the pattern
110 SHOW 0,0,2,4           Show it at 0,0, and repeat the pattern
120 I=1                    This is to waste time while you
130 I=I+1                    look at the pattern to be sure
140 IF I=5 THEN 160          you know exactly where it's displayed
150 GOTO 130
160 SHOW 0,0,2,4           Erase the pattern
170 SHOW 8,8,2,4           Show it in a new place
180 GOTO 180               Now stop and look at the new display.
```

GOKEY KEY

```
10 GOKEY 30
20 GOTO 10
```

Lines 10 and 20 form a loop - GOKEY doesn't wait for the user, so you have to keep going back until a key is pressed.

```
30 PRINT KEY
```

Key will contain the ASCII value in decimal form of the key pressed by the user.

```
40 GOTO 10
```

Do it forever. With this program , you can look at the ASCII values for all the keys on the keyboard, just for curiousity's sake.

```
10 X=1
20 A(X)=X
30 PRINT A(X)
40 X=X+1
50 GOTO 20
```

This program just fills the Array variable A(X) with the value of X. It will run forever, until the VIP runs out of memory space. Since the array variable is stored in the space immediately following the program text, the maximum number of elements in the array is dependent on the size of your VIP RAM memory.

```
10 CLS
20 X=1
30 PT=08
40 SHOW 10,10
50 X=X+1
60 IF X=6 THEN 80
70 GOTO 40
80 END
```

This program simply shows you how the variable HIT is set. If you display the pattern in the same place twice, it will be erased the second time - but HIT will be set. The next time the pattern is displayed in the same place, HIT will be set to 0 - and the pattern will show up on the screen. This program will display the same pattern over and over again - and set HIT each time. If there's no pattern on the screen when line 30 is executed, HIT will equal 0. Try it and see!

GOSUB RETURN PRINT AT REM

```
 10 CLS
 20 P."ENTER AN 'X'
 30 P."(HORIZONTAL)
 40 P."COORDINATE ";
 50 IN. X
 60 REM
 70 REM
 80 P."ENTER A 'Y'
 90 P."(VERTICAL)
100 P."COORDINATE ";
110 IN. Y
120 REM
130 REM
140 P."ENTER THE
150 P."VALUE TO BE
160 P."PRINTED AT
170 P."X=";X;" ";"AND
180 P."Y=";Y;" ";
190 IN. A
200 REM
210 REM
220 GOSUB 500
230 I=1
240 I=I+1
250 IF I=5 THEN 10
260 GOTO 240
500 CLS
510 PRINT AT X,Y,A
520 RETURN
```

This program allows you to use a subroutine to print different values on the screen at different places. After entering X and Y coordinates, and the value to be printed at those coordinates, the user waits for the BASIC to go into the subroutine, clear the screen, and display the number at the coordinates he's chosen. The "loop" in lines 230 - 260 permit the program to simply wait while the user examines the screen. When I is equal to 5, the entire sequence starts over. Of course, in this instance, it might not be necessary to use a subroutine - it's called only once in the entire program. But we decided to do it this way in order to show you the difference between GOTO and GOSUB.

Line 260 contains the GOTO - when that line is encountered, program control branches to line 10 and continues from there with no further branching. Line 220 contains the GOSUB to line 500 - without the RETURN in line 520, the program would display the first value entered, then display READY. The RETURN permits program control to return to the line immediately following the line which called the subroutine. Since line 220 called the subroutine (with the GOSUB keyword), and since line 230 is the line immediately following line 220, the RETURN in line 520 causes program control to be returned to line 230.

Note the REM keywords in lines 60, 70, 120, 130, 200, and 210. Usually REM statements are used to contain comments for the use of the programmer (or for whoever might be trying to understand what the programmer had in mind). In this case, the REM statements were put in to leave space for additional code.

One of the more serious problems facing a programmer is making sure his programs are "idiot-proof". You don't want the user to be able to make such a bad mistake that he can't continue the program. To prevent

this, you must check for "bad" answers. How might you do this?

You know there is a limited range of valid values for X and Y coordinates. If X is greater than 57, a two digit number cannot be fully displayed without being wrapped around the screen. If Y is greater than 28, no digit can be fully displayed. And, IF X=57, Y=27, and A is a three digit number, the program will "hang" up - and the only way the user can recover is to switch to RESET.

So add the following lines of code to your program - after you've tried to run it as it is. Enter "bad" data values - and see for yourself what happens!

```
 60 REM VALIDATE X
 70 IF X>57 THEN 600
120 REM VALIDATE Y
130 IF Y>27 THEN 700
200 REM VALIDATE A
210 IF A>99 THEN 800
600 ?"X IS TOO LARGE"
610 GOTO 20
700 ?"Y IS TOO LARGE"
710 GOTO 80
800 ?"THE VALUE IS
810 ?"TOO LARGE.  TRY
820 ?"AGAIN.
830 GOTO 140
```

Notice that we only protected the user against values of X and Y which are too large. What will happen to him if he tries to use negative values for X and Y?

What will happen to him if he presses the RETURN key without entering any data?

What will happen if he tries to enter alphabetic data?

In most of these instances, an ERR? message will result - and keep

on appearing until the user enters valid numeric answers!

In line 220, GOSUB 500 causes program control to branch to the subroutine which begins at line 500. All the program lines in the subroutine are executed before the RETURN causes the program to branch back to the line immediately following the call to the subroutine in line 220 (back to line 230). A GOSUB differs from a GOTO in that once a program executes a GOTO, it doesn't branch back to the statement immediately following the call. A subroutine is useful for performing operations repeatedly - more complex operations that simple incrementing of a variable value.

The use of CLS at the beginning of the program - and at the beginning of the subroutine - is for esthetics only. It isn't necessary to clear the screen. When you run the program, try changing lines 10 and 500 to REM statements - but don't delete them altogether. In line 220, when you GOSUB 500, a line 500 must exist or you'll get a HOW? error. And, in line 250, where the IF/THEN causes a branch to line 10, line 10 must exist - or you'll get the same message: HOW?

NEW LIST SAVE LOAD RUN MEM

There isn't a program we can use to show you the way to use some of the VIP BASIC keywords, because they can't be used in programs. One of these keywords is NEW. NEW has the same effect on your program, in effect, as moving the toggle switch to RESET and back to RUN without holding down the backspace key while you switch to RUN. The program storage area is cleared, and you can enter a new program without having to worry about using exactly the same line numbers as you used in your previous program.

LIST is another keyword that has no effect when used in a program. When you type the word LIST, and press the carriage return, the very first line of your program is displayed on the screen. To get the next line of the program, press the space key. You can't get out of LIST mode without switching to RESET and then to RUN (remember to hold down the backspace key when switching to RUN) or press CONTROL @. LIST mode automatically terminates when the last line of the program has been displayed.

You can specify a line number at which the listing is to start, by typing LIST N, where N is the line number. When you type LIST 50, for example, lines 50 and the line immediately following line 50 are both displayed on the screen. Again, you must press the space key to see the next line, and must move the toggle switch to RESET and then back to RUN to get out of LIST mode.

SAVE will save your BASIC program on cassette tape. If you SAVE

your programs, you won't have to re-enter them every time you want to use them. Simply type SAVE - and press the record and play buttons on your recorder. It's a good idea to follow this sequence when you want to save a program!

1. Press the play button on the recorder and let the motor run for a few seconds - to get the transparent tape leader past the read/write mechanism (called the tape heads).
2. Type SAVE - but don't press return.
3. Press the record and play buttons on the recorder.
4. Now press the return key.

The screen will go blank while the program is being displayed, and none of the keys will work. If you decide you did something wrong and want to stop the SAVE process, you must switch to RESET. Be sure to hold the backspace key down when switching back to RUN or you'll lose the program.

LOAD will load your program from the cassette into VIP memory so you can run it. No special procedure is necessary - just type LOAD, press the play button on the recorder, and wait. The screen will be turned off and the keyboard will be locked out, just as with SAVE. When your program is loaded, the screen will display the READY prompt.

RUN causes the VIP to execute your program, beginning with the very first line of code and continuing until either the last line in the program has been exeucted or until an END statement is encountered.

You can stop the program at anytime with CONTROL @ or by moving the toggle switch to RESET than back to RUN.

MEM tells you how many bytes of memory are left in the program storage area. If you're writing a particularly long program, you will want to check on the available space now and then to be sure your program will fit into the storage area. If your program is too long, the VIP will display a SORRY? message when the time comes to execute (or RUN) the program. Some of the available memory is used to store variable values, so you must take that into consideration. Don't write your program to fill all the available space - or you won't have any space left in which to store the variable values used by your program.

COLOR

COLOR is used to set both the background and the foreground colors for your display. When you use COLOR with only a dummy parameter, the background color is set. You can't control the sequence of colors for the background, however - it's automatically set and incremented each time the COLOR keyword is encountered (provided it doesn't have its full complement of five parameters).

```
10 CLS
20 I=1
30 COLOR I
40 I=I+1
50 IF I<5 THEN 30
60 GOTO 10
```

The program simply cycles through the background colors, so you'll be able to see what each of them looks like. Remember that when the READY prompt appears, after initial RESET, the background color is initialized to blue. The first time the COLOR keyword is encountered, the color of the background changes to black; the next time COLOR is encountered, the background color changes to green. When COLOR is again encountered, the background changes to red, and if COLOR is encountered yet again, it switches the background color back to blue. Try this one:

```
10 CLS
20 I=1
30 TVOFF
40 COLOR I
50 I=I+1
60 IF I<5 THEN 40
70 TVON
80 GOTO 80
```

Are you surprised? Did you expect the screen to change color

only once? What happens is this: The screen starts out blue, the TV is turned off - but the COLOR keyword causes the screen to change color. Only the tone and interval timers are shut down when the TVOFF keyword is implemented. The keyboard is locked out, of course, and you can't display any foreground colors - but you can change the background colors. Notice, though, how much faster the colors change in the second version - when the TV is turned off.

When used with its full complement of parameters, the COLOR keyword sets the foreground colors. There are eight different colors available to you:

Ø black	4 green
1 red	5 yellow
2 blue	6 aqua
3 violet	7 white

When the READY prompt appears, the foreground color is set to color 7 (white). You must always specify the color code when using COLOR to set the foreground colors. You may store the color code in a variable, of course - but color is the first parameter necessary.

The second and third parameters are the X and Y coordinates of the upper left hand corner of the color block in which you want the color to be displayed. A color block contains eight horizontal dots and four vertical dots - and if you specify any of the X or Y dots within a color block, all the dots in that block will change color to the color you specify.

The fourth and fifth parameters are the number of color blocks

horizontally and vertically which you specify to be changed to the same color as the first block. Remember that color blocks are eight dots wide, and that changing the color of one dot in the block changes the color of all the dots in the block.

The format for using the COLOR keyword to set the foreground color is:

COLOR color X, Y, nh, nv

where X and Y are the coordinates, nh is the number of horizontal blocks, and nv is the number of vertical color blocks - not the number of dots.

Here's a program which will show you exactly where each of the foreground color blocks are:

```
 10 CLS
 20 PT=FF
 30 Y=-4
 40 X=0
 50 I=0
 60 Y=Y+4
 70 COLOR I,X,Y,1,1
 80 SHOW X,Y,1,4
 90 I=I+1
100 IF I=8 THEN 40
110 X=X+8
120 GOTO 70
```

Note that the parameters in COLOR (line 70) are not the same as the parameters for SHOW (line 80). This is because SHOW has a field eight dots wide (as does COLOR), but only <u>one</u> dot high. COLOR color blocks are four dots high. Notice that X is incremented by 8 and Y is incremented by 4 - to set the X and Y coordinates at the upper left hand corner of each color block. The color blocks begin in X=0, 8,

16, 24, 32, 40, 48, and 56. The appropriate Y coordinates are Y=0, 4, 8, 12, 16, 20, 24, and 28.

Try changing the pattern stored in PT - you'll be surprised at the results!

ABS

```
10 CLS
20 X=-10
30 Y=ABS(X)
40 PRINT X, Y
```

This program illustrates the use of the ABS function to convert a negative value to a positive one. In some applications, it might prove helpful to keep track of the sign of the value which is used as the parameter of the ABS function. One way to do this is:

```
 10 Z=1
 20 X=-10
 30 IF X<0 THEN 100
 40 Y=ABS(X)
 50 PRINT Z*Y
 60 GOTO 120
100 Z=-1
110 GOTO 40
120 END
```

SUBSCRIPTED VARIABLE A(X)

```
 10 CLS
 20 ?"HOW MANY VALUES ";
 30 IN. K
 40 J=1
 50 ?"ENTER #";J;" ";
 60 IN A(J)
 70 J=J+1
 80 IF J<=K THEN 50
 90 CLS
100 TVOFF
110 M=0
120 J=1
130 IF A(J)<=A(J+1) THEN 190
140 T=A(J)
150 A(J)=A(J+1)
160 A(J+1)=T
170 M=M+1
180 COLOR S
190 J=J+1
200 IF J<K THEN 130
210 IF M=0 THEN 230
220 GOTO 110
230 J=1
240 TVON
250 ?J;"    ";A(J)
260 J=J+1
270 IF J<=K THEN 250
280 END GOTO 28
```

This program illustrates the use of the subscripted variable A(X) to enter, sort, and print data values.

Lines 20 - 80 constitute the input routine, in which the user specifies the number of values to be entered and then enters all those values into the array variable A(J). A(J) can be thought of as a list, of which no two elements are necessarily the same. (That is, the 5th item in the list will probably not be the same as the 7th or the 3rd items on the list.)

Lines 90 and 100 simply clear the screen and turn off the TV - VIP BASIC is faster when there's no display on the screen.

Lines 110 - 200 sort the data in ascending order, beginning with the smallest value. A "flag" is set in line 170 - if M<>0, a value has been moved from its position in the list. When the program gets to line 210, if M is not equal to zero, a value has been moved during the "loop" - and the program goes back to line 110 to check again.

After all the values have been checked (in line 200), the program makes certain the "flag" isn't set. If it were set, the program would recheck all the values in the list. Finally, lines 230 - 270 print all the sorted values, and line 280 ends the program.

When you run this program, you may enter as many values as you like (up to 32767, unless you run out of memory space first), and you may use any values you like between -32768 and +32767.

The program will patiently accept all of them - and sort them -

and print them out for you. Remember that it isn't necessary to implement the TVOFF keyword in line 100 - but the VIP will do the sorting faster if you include it. And the background colors will change each time through the loop, so you'll know that something is happening. (It takes 40 seconds for the VIP to sort 10 numbers.)

The subscripted variable A(X) is not to be confused with the single variable A. You may use both in any given program, although you probably won't want to do so. It may cause a certain amount of confusion for those who try to read your code later.

DEBUGGING YOUR PROGRAM

Now there isn't any more to tell you - the best way for you to become "fluent" in BASIC is to practice writing programs. You'll discover a lot of neat features in VIP BASIC.

Very few programs are written without error. You can cut down on the number of errors in many different ways, and this section points out some of the more frequently used methods of tracking down "bugs" (errors) in a program and eliminating them. The process is called "debugging".

First of all, write your programs in a modular fashion. Write each routine as though it were a complete program. For example, the program used to illustrate the use of the subscripted variable A(X) was written in "pieces", to permit the author to check the results of each piece before going on to the next. This technique is a favorite among "professional" programmers. At the end of each section, you can print the values of each of the variables and determine whether the values are what you expected them to be.

As each section of code is added to the program, you can print out the successive "intermediate" values of the variables - and thereby isolate an error to a specific section of the program.

If the VIP hasn't delivered an error message to you, and there is something wrong (you aren't getting the results you expected), the chances are that there is something wrong with the program logic. Write the code down on a piece of paper - and then pretend to be the computer, following each step as you encounter it, writing down each

variable value as it is assigned, and branching in all the appropriate places to the appropriate line. Usually you'll find the error in this manner.

Another favorite "debugging" technique is to place "END" statements in the program at approximately five line intervals - as your program successfully gets to each END, you can delete it and rerun the program.

A variation of the "END" technique is called "walling off" the error. Put an END statement halfway through the program - if the program gets that far without errors, you know the problem is in the second half of the code. If it doesn't get that far, you know the problem is in the first half of the program. Now, having determined which half of the program contains the "bug", divide that half in half with another END statement - and you'll be able to tell which quarter of the program has the error in it. Continue to divide the problem section of the code into halves until you've isolated the error between two END statements - and then correct the errant statement.

And, if all else fails, start over. Really start over. Begin with a definition of the task the program is to perform; draw a flowchart; assign specific blocks of line numbers to specific routines. Code each section separately, with no other code in memory, and don't combine the parts into a whole until all the parts work right.

"Lazy" programmers usually start with the first technique

outlined here; successful programmers usually start with the last one. Designing a program "on the fly" is a bad habit to get into - and it leads you to spending more time debugging than in actual programming or in enjoying the fruits of your labor.

APPENDIX I

NON-BASIC DEFINITIONS

Terms you'll hear, read, and use frequently - in almost any computer-oriented environment.

DEFINITIONS

The terms defined below are not exclusively related to BASIC; they'll be useful in almost any computer environment.

Term	Definition
Array	A group of data items with something in common, using the same name. A grocery list, for example, can be considered an array (a list is known as a one-dimensional array). A train schedule can be considered an array (tables are known as two-dimensional arrays). Arrays can consist of many dimensions. Suppose, for instance, you must keep track of 500 machine parts, each of which can be used in six different machines, and each of which is available from four or five different suppliers. The form in which you maintain this array of information might be considered a three-dimensional array. Arrays are also known as matrices. VIP BASIC can handle a one-dimension array, the size of which is limited by the amount of memory available after your program is typed in.
Subscript	The notation which specifies which data item in the array is specifically being examined or used at this moment. For instance, if you are looking at a grocery list, you know that the tenth item of the list will be different than the first item on

Term	Definition
Subscript	the list - and both will be different than the fifth item. You might specify that item seven is to be purchased at a different store by writing "Buy #7 at WACO's". In BASIC, the subscript is put inside parentheses: (7).
Initialize	Start. When you first attempt to use a BASIC variable, that first use "initializes" the variable to some value. When the computer is first turned on, the variables contain random numbers, and failing to initialize the variables before performing calculations involving them can lead to interesting and amazing results, usually incorrect.
Byte	The smallest unit of memory available to your BASIC program. It takes one byte of memory to contain one BASIC character - even if that character is a blank space.
K	Abbreviation for "kilobyte", which means 1000 bytes.
RAM	Random Access Memory. This is memory space which you can write a program or store data. It should have been called Read/Write Memory, but RWM is hard to pronounce.

Term	Definitions
ROM	Read Only Memory. The computer's operating system and the BASIC interpreter are stored here, and while you can read the data stored in ROM, you cannot write data into this space. ROM is protected against the user inadvertently destroying the operating system (without which the computer couldn't function at all) and/or the BASIC (without which you couldn't write a BASIC program). The ROM has a program (the operating system or BASIC interpreter) stored in it at the time of its manufacture, and nothing else can ever be stored in it.
PROM	Programmable Read Only Memory. This is also protected memory - but the program is written into PROM memory after the PROM has been manufactured. When you buy a computer, the ROM is already on board. When you buy a PROM, you can use a PROM programmer, put a program in it, and then plug it on the board. But once a program has been written into PROM memory, it can't ever be removed.
EPROM	Erasable Programmable Read Only Memory (are you starting to get confused?) An EPROM functions much like a ROM or a PROM, in that it is read-only memory. Like A PROM, the program is written into EPROM memory after the EPROM is manufactured. But

Term	Definition
EPROM	it is possible to erase the program from an EPROM (using ultra-violet light) and write another program into the same space. You can buy EPROMs to plug into your computer, as well as EPROM programmers and erasers.

Note: The only reason, at this point in your BASIC education, for telling you about ROMs, and PROMs, and EPROMs, is that when you're at your local computer store, you'll hear the terms quite frequently. You wouldn't want to be confused forever, would you? You can buy good beginner books to tell you all about the different kinds of memory available for use with a microcomputer system - and if you want to build up your VIP to look like a "big" micro, you might want to look into it carefully.

I/O	Input/Output devices. In order to be useful, a computer has to be able to receive data and deliver results to the outside world. (By "outside world," we mean all the world outside the computer itself. This includes keyboards, cassette units, disks, printers, video screens, etc.) The keyboard is used only for input (a computer cannot write data out to a keyboard!). A printer is used only for output (a computer cannot receive information from a printer). Some

Term	Definition
I/O	devices can be used for either Input or Output - like the cassette tape.
Keyword	BASIC is composed of keywords which function as either direct commands, without line-numbers, or as part of BASIC statements, which include line-numbers. Some keywords are "reserved" variables or function names (like TI, which is a variable in which the time can be stored, or TO, which is a function which sets the duration of the tone).
Command	A BASIC Command requires no line number. It is executed immediately, as soon as the RETURN key is pressed on the keyboard. Some BASIC keywords are used strictly as commands; they can wreak havoc on a program if used in a BASIC statement. A command begins with a BASIC keyword and can contain one or more operands, and ends with a carriage return.
Statement	A BASIC Statement requires a line-number. It is executed during the running of a program, rather than immediately when the RETURN key is pressed. Some BASIC keywords are used strictly as statements; they perform no useful function as direct

Term	Definition
Statement	commands. A Statement begins with a line number, contains one or more BASIC keywords, one or more operands, and ends with a carriage return.
Line Number	Integer (whole) numbers telling BASIC the order in which you want the instructions in a BASIC program to be executed. The line numbers must be in the range of from 1 to 32767.
Variables	Labels used to identify data. When you're writing a program, you'll want to use more than one numeric value (to do arithmetic, for instance). Each value can be stored in RAM memory and labeled with a single letter of the alphabet (the variable "name") for later reference and use.
Strings	Non-numeric characters. Strings may only be used in PRINT statements (when enclosed within quotation marks) and in REM statements (when used as comments to explain parts of a BASIC program).
Expressions	Expressions are groups of variables and/or numbers, which, when evaluated, are equal to some numeric value.
Data	Information used by the BASIC program to obtain the results for which the program was written.

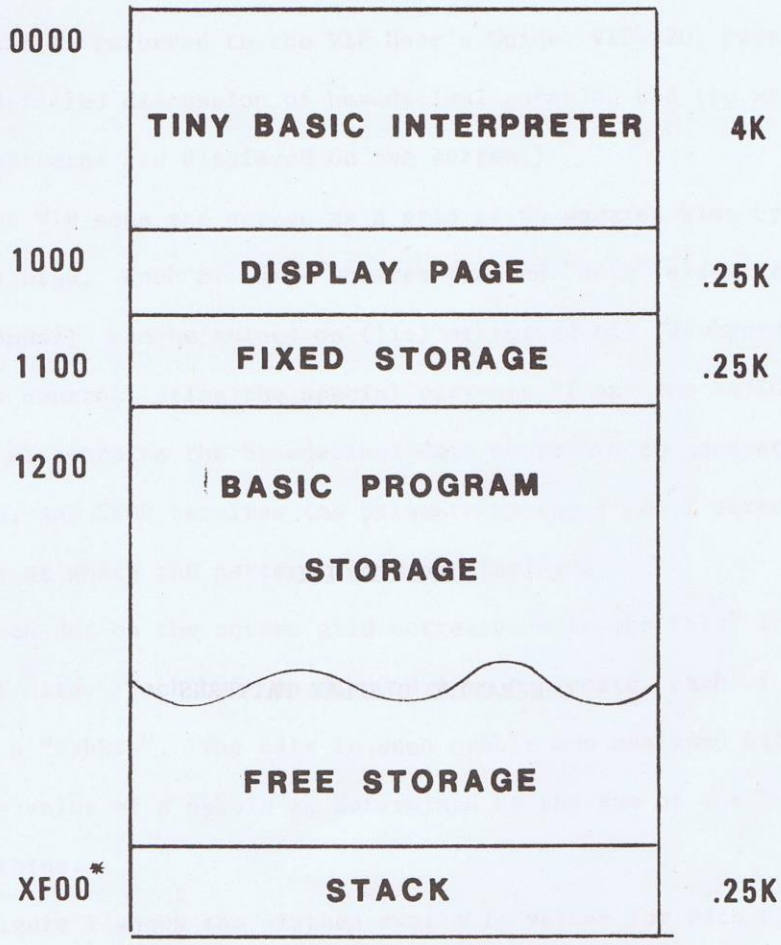
Term	Definition
Blanks	Blanks (spaces) are not significant to BASIC except when used in strings which are enclosed in quotation marks or in REM statements (comments).
Operators	Arithmetic operators are used to perform specific arithmetic operations: + addition — subtraction * multiplication — / division Relational operators are used to detect relationships between variables and/or other variables, expressions, or values: > greater than — < less than = equal to — >< not equal to >= greater than or equal to — <= less than or equal to Parentheses are used to group data for accurate evaluation.
Operands	The data upon which the operators are to be used, or the data upon which a keyword is to operate.
Numbers	Numbers are integers (whole numbers) in the range -32768 to +32767.
Cursor	The white box which appears on the TV screen in the screen position in which the next typed character will be displayed.

Term	Definition
Bug	An error in programming - or in wiring - is called a "bug" - and the process of tracking down and eliminating the error is called "debugging" the program or the board (the computer parts).

APPENDIX II

BASIC MEMORY MAP

VIP BASIC MEMORY MAP



* XF=HIGHEST PAGE OF RAM (32K MAXIMUM)

APPENDIX III

GENERATING DISPLAY PATTERNS

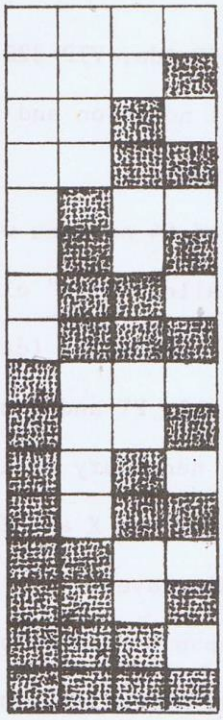
GENERATING A PATTERN

(The user is referred to the VIP User's Guide, VIP-320, pages 12-17, for a detailed discussion of hexadecimal notation and the manner in which patterns are displayed on the screen.)

The VIP sees the screen as a grid of 64 squares wide by 32 squares high. Each of these squares (called "dots" elsewhere in this manual) can be turned on (lit) or turned off (darkened) under program control, using the special variable PT and the BASIC keyword SHOW. PT contains the hexadecimal data necessary to generate a pattern, and SHOW requires (as parameters) the X and Y screen coordinates at which the pattern is to be displayed.

Each dot on the screen grid corresponds to one "bit" in an 8-bit byte of data. Each byte is divided into two parts, each of which is called a "nybble". The bits in each nybble are assigned bit numbers, and the value of a nybble is determined by the sum of the "on" bits it contains.

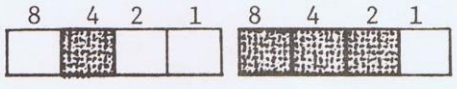
Figure 1 shows the sixteen available values for each nybble in a byte, along with its corresponding decimal values. The numbers at the top of the figure are the bit numbers; the values at the right are the hex values determined by finding the sum of the bit numbers for all the shaded squares in any given row. For example, to find the hex value of 7, add the numbers 4, 2, and 1. To find the hex value of D, add the bit numbers of the shaded squares: 8+4+1, or 13.



Hex	Decimal
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
A	10
B	11
C	12
D	13
E	14
F	15

FIGURE 1

These values are the values for one nybble - half of one byte. A full byte of data must be entered into PT for the pattern to be shown. The sums of two nybbles are <u>not</u> added together to form the value of a byte; they are entered one after the other. For example, a byte value of 4E represents two nybbles: the left one with a value of 4, the right one with a value of E, as shown here:

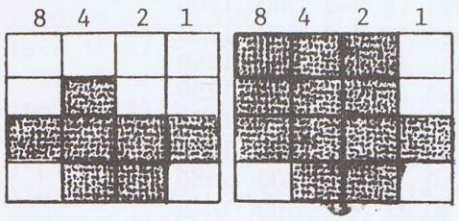


Hex 4E

FIGURE 2

Appendix III-3

The program below will place a pattern on the screen. The pattern is first drawn on paper, the hex values determined, and then the hex values are entered in the PT variable.



OE
4E
FF
66

```
10 CLS
20 PT=0E4EFF66
30 SHOW 0,0,1,1
40 PRINT
50 GOTO 50
```

FIGURE 3

This pattern requires 4 bytes of data to be stored in PT.

Note that a pattern may be as many as fifteen bytes high - but only one byte wide. To get the pattern to be displayed in more than one horizontal byte, you must specify the number of bytes in which it is to be displayed as one of the parameters in the SHOW keyword (the third parameter). Try SHOW 0,0,2,1 to see the effect (in line 30 of the program) and then try SHOW 0,0,1,2 - to see the effect of multiplying the number of vertical bits.

Appendix III-4

TIC-TAC-TOE IN VIP BASIC

A Two-Player Game

```
 10 CLS
 20 I=1
 30 A(I)=0
 40 I=I+1
 50 IF I < 19 THEN 30
 60 K=0
 70 A=20
 80 B=29
 90 C=39
100 D=4
110 E=13
120 F=23
130 I=0

200 PRINT AT 23,4,"1 2 3"
210 PRINT AT 23, 14,"4 5 6"
220 PRINT AT 23,23,"7 8 9"
230 PT=0101010101010101
240 SHOW 21,3
250 SHOW 21,12
260 SHOW 21,21
270 PT=8080808080808080
280 SHOW 37,3
290 SHOW 37,12
300 SHOW 37,21
310 PT=FF
330 SHOW 21,11,3,1
340 SHOW 21,20,3,1

500 PT=10107C1010
510 I=I+1
520 S=0
530 X=0
540 SHOW 1,25
550 GOKEY 570
560 GOTO 540
570 IF HIT=1 THEN 590
580 SHOW 1,25
590 GOSUB 1000
600 IF X>0 THEN 530
610 IF I=5 THEN 1810
620 IF S=9 THEN 500
630 PT=7C4444447C
640 S=9
650 GOTO 530

1000 IF KEY < 49 THEN 1470
1010 IF KEY > 57 THEN 1470
1020 M=KEY-48
1030 GOSUB 1600
1040 IF X > 0 THEN 1460
1050 IF M=9 THEN 1200
1060 IF M=8 THEN 1230
1070 IF M=7 THEN 1260
1080 IF M=6 THEN 1290
1090 IF M=5 THEN 1320
1100 IF M=4 THEN 1350
1110 IF M=3 THEN 1380
1120 IF M=2 THEN 1410
1130 IF M=1 THEN 1440

1200 PRINT AT C,F," "
1210 SHOW C,F
1220 RETURN
1230 PRINT AT B,F," "
1240 SHOW B,F
1250 RETURN
1260 PRINT AT A,F," "
1270 SHOW A,F
1280 RETURN
1290 PRINT AT C,E," "
1300 SHOW C,E
1310 RETURN
1320 PRINT AT B,E," "
1330 SHOW B,E
1340 RETURN
1350 PRINT AT A,E," "
1360 SHOW A,E
1370 RETURN
1380 PRINT AT C,D," "
1390 SHOW C,D
1400 RETURN
1410 PRINT AT B,D," "
1420 SHOW B,D
1430 RETURN
1440 PRINT AT A,D," "
1450 SHOW A,D
1460 RETURN

1470 FQ=100
1480 TO=8
```

Tic-Tac-Toe In VIP BASIC

```
1490 PRINT AT 0,20,"???"
1500 X=X+1
1510 PRINT AT 0,20,"   "
1520 IF X < 3 THEN 1480

1600 IF A(M)=1 THEN 1470
1610 IF A(M+9)=1 THEN 1470
1620 A(M+S)=1
1630 IF I < 3 THEN 1730
1640 IF A(1+S)+A(5+S)+A(9+S)=0 THEN 1730
1650 IF A(1+S)+A(2+S)+A(3+S)=3 THEN 1800
1660 IF A(1+S)+A(4+S)+A(7+S)=3 THEN 1800
1670 IF A(1+S)+A(5+S)+A(9+S)=3 THEN 1800
1680 IF A(2+S)+A(5+S)+A(8+S)=3 THEN 1800
1690 IF A(3+S)+A(5+S)+A(7+S)=3 THEN 1800
1700 IF A(3+S)+A(6+S)+A(9+S)=3 THEN 1800
1710 IF A(4+S)+A(5+S)+A(6+S)=3 THEN 1800
1720 IF A(7+S)+A(8+S)+A(9+S)=3 THEN 1800
1730 RETURN

1800 K=1
1810 CLS
1820 IF I=5 THEN 1880
1830 SHOW 15,5
1840 PRINT AT 10,15,"WINS!"
1850 GOSUB 1910
1860 CLS
1870 END
1880 IF K=1 THEN 1830
1890 PRINT AT 5,5,"NO ONE"
1900 GOTO 1840
1910 X=X+1
1920 IF X < 5 THEN 1910
1930 RETURN
```

This game uses the following keywords and variables:

LET	GOTO
IF/THEN	GOSUB
PRINT AT	KEY
PT	HIT
SHOW	RETURN
TO	FQ
CLS	END

To help you understand what's happening in this game, the names of the variables used, and their functions, are:

I	Number of Turns	S	Player ID
K	Win Flag	M	Key pressed
A	Leftmost X Coord	D	Topmost Y Coord
B	Center X Coord	E	Center Y Coord
C	Rightmost X Coord	F	Lowest Y Coord
A(I)	Occupied Squares	X	Time index

Lines 10-130	initialize the variables
Lines 200-340	display the Tic-Tac-Toe form on the screen
Lines 500-650	display each player's token (+ or 0) in turn, wait for the player to make a move, and checks to be sure the player hasn't made an error or that all the plays haven't been made.
Lines 1000-1130	determine whether the player pressed a valid key.
Lines 1200-1460	displays the player's token in the appropriate Tic-Tac-Toe square.
Lines 1470-1520	display an error message and sound a tone if the player made an error.
Lines 1600-1730	determine whether the player's move is a winning move.
Lines 1800-1930	display the winner's token and the winning message (if there was a winner) on the screen.

APPENDIX IV

QUICK REFERENCE

QUICK REFERENCE

NEW	N.	Clears program storage area
LIST	L.	Lists program, starting at lowest line number
LIST Z		Lists program, starting with line Z
RUN	R.	Starts program execution at lowest line
SAVE	S.	Saves program on cassette tape
LOAD	LO.	Loads program from tape
CLS	CL.	Clears the screen
COLOR	C.	Sets color
END	E.	Halts program execution
GOTO Z	G.	Branch to line number Z
GOKEY	GOK.	Branch to line number Z if any key is pressed
GOSUB Z	GOS.	Branch to subroutine at line number Z (Requires RETURN)
IF A	I.	Evaluates expression A (Requires THEN)
THEN	T.	Branch to line Z if condition A is true (Requires IF)
INPUT	IN.	Allows entry of data from keyboard while program runs
LET	LE.	Assigns a value to a variable
TI		Sets internal timer
FQ	F.	Sets frequency of tone generator
TO		Turns on tone generator and sets tone timer
PT		Stores pattern for later use by SHOW statement
PRINT PRINT	? P.	Prints value of variable or expression on the screen
PRINT AT X,Y ? AT P AT		Same as PRINT but display begins at screen X, Y
REM	RE.	Comments
RETURN	RET.	Returns from subroutine (Requires GOSUB)

NOTE: X & Y can be integer or variable, Z must be integer only.

SHOW X,Y	SH.	Displays pattern in PT at screen X,Y
TVOFF		Turns off TV display and disables tone & internal timers
TVON		Turns on TV display and enables tone & internal timers
ABS(Z)		Absolute value of expression Z
HIT	H.	Ø if last PRINT AT or SHOW had no hit, 1 if hit occurred
KEY	K.	ASCII value of key pressed in GOKEY statement
MEM	M.	Number of free bytes of program storage area
RND		Random number from Ø to 255

FORMATS:

PRINT AT xh,yv,list	xh = screen horizontal coordinate
SHOW xh,yv	yv = screen vertical coordinate
SHOW xh,yv,nh,nv	nh = number of blocks of pattern, hor
COLOR color, xh,yv,nh,nv	nv = number of blocks of pattern, ver

COLOR CODES:

Ø	black	4	green
1	red	5	yellow
2	blue	6	aqua
3	violet	7	white

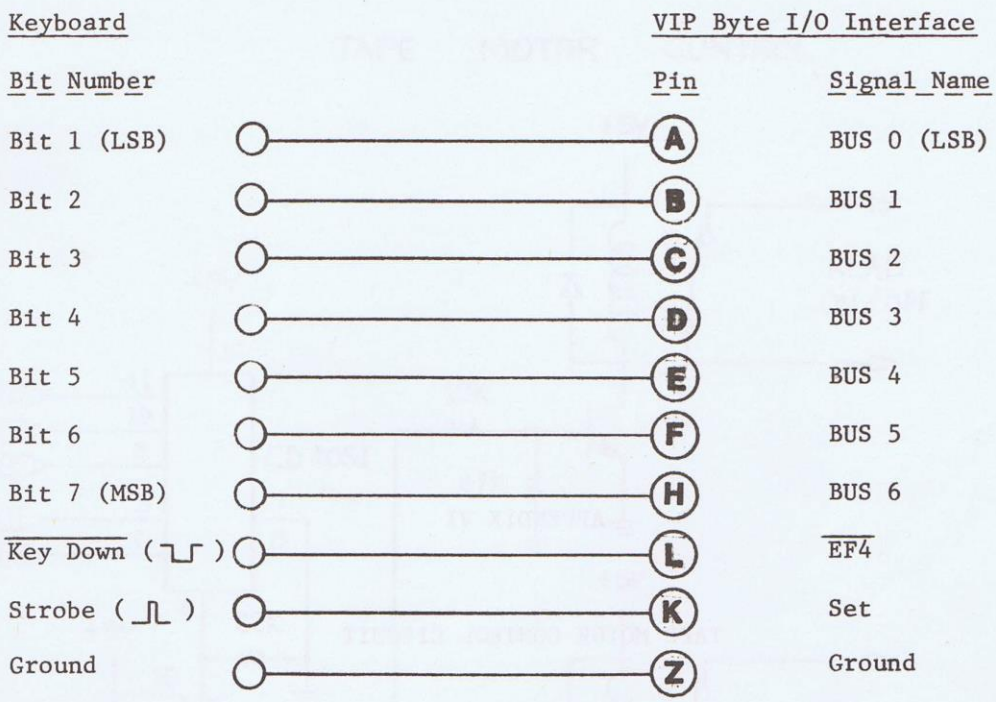
ERROR MESSAGES:

WHAT?	Syntax error; typing error
HOW?	Non-existent line number
SORRY?	Out of memory space
LD ERR	Cassette load error
ERR	Data for INPUT statement is invalid

APPENDIX V

ASCII KEYBOARD HOOK-UP TO VIP INPUT PORT

ASCII KEYBOARD HOOK-UP TO VIP INPUT PORT



Notes:

1. If $\overline{\text{Key Down}}$ is not available, $\overline{\text{Strobe}}$ may be substituted.
2. If data from keyboard is valid for as long as $\overline{\text{Key Down}}$ is true, no connection to Pin K (Set) is necessary.

APPENDIX VI

TAPE MOTOR CONTROL CIRCUIT

TAPE MOTOR CONTROL