SVI™
SPECTRAVIDEO

SVI-728-UM-1

PRINTED IN HONG KONG

SVI™
SPECTRAVIDEO

SVI·728

PERSONAL COMPUTER USER'S MANUAL

MSX

SPECTRAVIDEO'S USER'S MANUAL STATEMENT

This equipment generates and uses radio frequency energy and if not installed and used properly, that is, in strict accordance with the manufacturer's instructions, may cause interference to radio and television reception. It has been type tested and found to comply with the limits for a Class B computing device in accordance with the specifications in Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference in a residential installation. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause interference to radio or television reception, which can be determined by turning equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

- Reorient the receiving antenna
- Relocate the computer with respect to the receiver
- Move the computer away from the receiver
- Plug the computer into a different outlet so that computer and receiver are on different branch circuits

If necessary, the user should consult the dealer or an experienced radio/television technician for additional suggestions. The user may find the following booklet prepared by the Federal Communications Commission helpful:
"How to Identify and Resolve Radio-TV Interference Problems"
This booklet is available from the U.S. Government Printing Office, Washington, DC 20402. Stock No. 004-000-00345-4.

WARNING:
This equipment has been certified to comply with the limits for a class B computing device, pursuant to Subpart J of Part 15 of FCC Rules. Only peripherals (computer input/output devices, terminals, printers, etc.) certified to comply with the Class B limits may be attached to this computer. Operation with non-certified peripherals is likely to result in interference to radio and TV reception.

Published by
SPECTRAVIDEO INTERNATIONAL LTD.

First Printing 1984
Printed in Hong Kong

Every effort has been made to supply complete and accurate information in this manual. Spectravideo International Ltd. reserves the right to change Technical Specifications and Characteristics at any time without notice.

US $13.95
SVI 728 UM

TABLE OF CONTENTS

APPENDICES

INTRODUCTION

Welcome to the world of tomorrow. You have just made a purchase that will reward you for a long time to come. The SVI-MSX 728 home Computer that you have purchased will open the doors to the future both today and in the future. The SPECTRAVIDEO family of personal computers contain features that will allow you to take advantage of all current computing technology and will also allow you to expand your horizons as new technology evolves.

The SVI-MSX 728 is the tool that will allow you to deal with the computer revolution of the 1980's and this is the manual for the operation of this tool.

Again, welcome to tomorrow. Make it what you want it to be with you SVI-MSX 728. Just keep one thought in mind as you progress on your journey: A computer is only as smart as you make it. Without you, it's only a rather complex bunch of wires, chips, metal and plastic. You are the most important component of your new computer system !

SYSTEM OVERVIEW

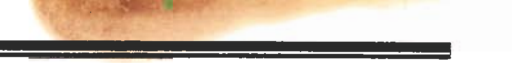
1 SYSTEM OVERVIEW

The terms in the following system overview might not be familiar to those of you who are just beginning to learn about computers. But don't worry. This manual was written especially for first time computer users to teach them the powerful MSX BASIC language that is built into the SVI-728. The overview is intended to give the beginner as well as the experienced user an appreciation of the power and beauty of the SVI-728: the first affordable and expandable personal computer.

Your SVI-728 computer contains 3 powerful processors that allow you to display exciting pictures, play music or sound effects and control a program all at the same time!

A Z80A 8-bit Microprocessor controls all the system components. It is responsible for storing, retrieving and executing programs. A video display processor (VDP) serves to generate all necessary video, control and synchronization signals. The VDP is capable of displaying 16 colors. It also supports manipulations of 32 sprites (shapes) and a maximum or 40 column text mode.
In addition, there is a programmable sound generator (PSG) which can produce music or a wide variety of complex sounds.

The MSX BASIC language is stored in the 32K bytes of Read Only Memory (ROM). Your instructions (or "programs" in computerese) to command the computer are stored in one



64K bytes of Random Access Memory (RAM). The remaining 16K is strictly for use by the VDP to store information with which it creates colorful graphic displays.

SUPER MSX BASIC

The powerful MSX BASIC language written by Microsoft® is an extension of the same BASIC language that is available on computers costing three times as much as the SVI-728.

If you are new to programming, you will find MSX BASIC to be a versatile, easy to learn and easy to use language. Experienced programmers will appreciate its powerful features and the flexibility it permits when writing, editing, debugging and running programs.

The MSX BASIC has the following powerful built-in features:

(a) Screen editor.
(b) Full graphic and sound manipulation.
(c) New SPRITE commands.
(d) Default double precision math package for business application programs.
(e) Special machine interrupt handling commands for real time BASIC programming.
(f) Special function keys for often used commands (user programmable).
(g) True upper and lower case display and printing.

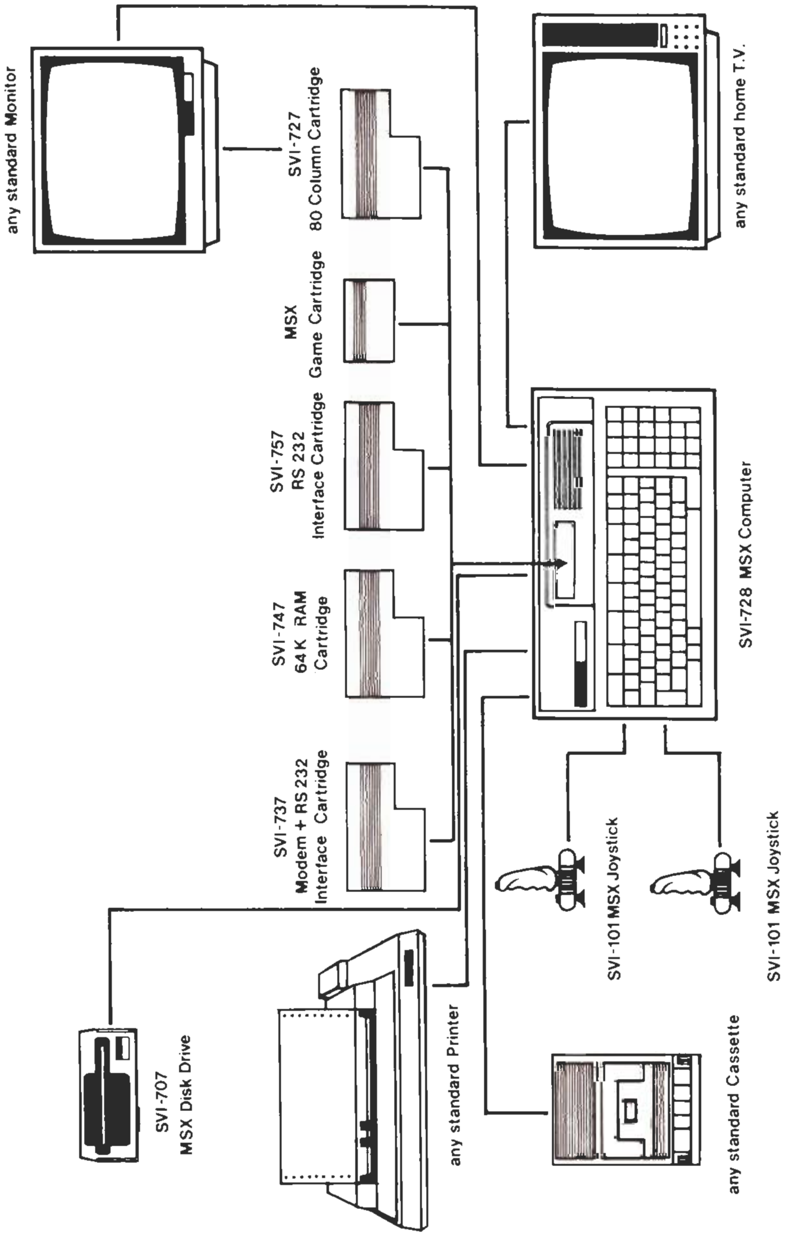
REMEMBER:

The secret to mastering the SVI-728's power is to take your time reading this book and trying out all the exercises on your computer. Like driving a car, you must learn by doing. By correcting your mistakes and learning not to feel bad about them, you will succeed and grow with the Spectravideo's line of expandable and affordable computer products.

EXPANDABILITY

To upgrade the computer, you can easily add more peripherals through the expansion module interface or game cartridge slot. You may install the SVI-707 MSX Disk Drive and the SVI-727 80 Column Video Cartridge to run CP/M programs; or else the speech synthesizer, the SVI-737 MSX 300 Baud Modem w/RS232. Besides, a wide variety of peripherals from other companies can also be used. Just name a few: music synthesizer, light pen etc.

MSX is a trademark of Microsoft Corporation
Microsoft® is a trademark of Microsoft Corporation.
CP/M is the trademark of Digital Research.

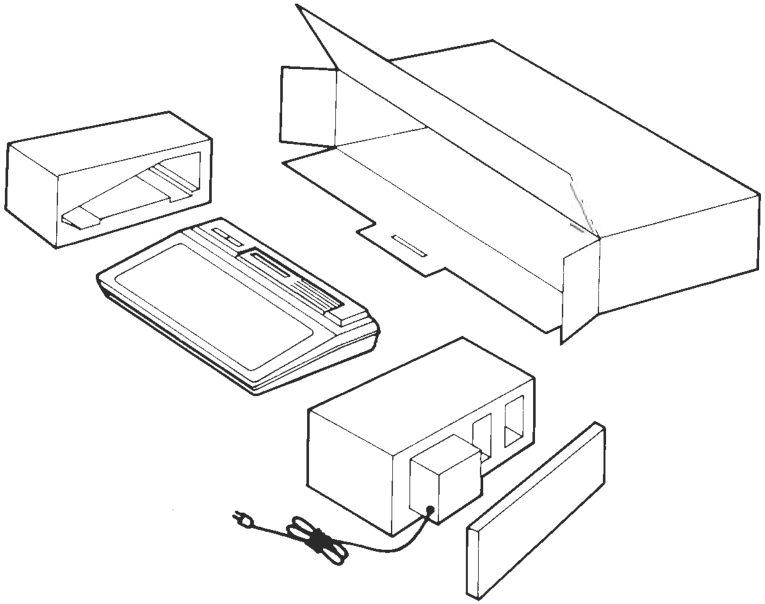


THE SVI-728 SYSTEM PERIPHERAL MAP

■ UNPACKING
■ SYSTEM INSTALLATION
■ POWER SUPPLY AND CABLES
■ POWER-ON SELF-TEST

UNPACKING THE SVI-728

The SVI-728 computer, video and audio cables, power supply and switch box (for USA version only) are all securely packed in a foam cushioned carton. Please note: Save all packaging materials in case you must ship the unit for maintenance or repairs.



Check the contents of your SVI-728 package. You should find the following items:

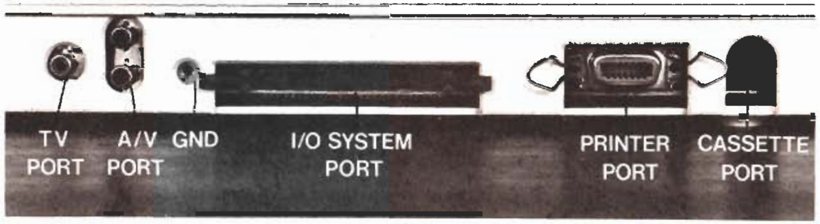
1. SVI-728 Home Computer
2. Video cable and audio cable
3. Switch box (for USA version)
4. Power supply
5. This instruction manual
6. Warranty registration card

If any items are missing, please check with your dealer immediately.

SYSTEM INSTALLATION

To connect the system, you will need two proper electrical outlets for your SVI-728 Computer and monitor or television set. Choose a comfortable position for operation, away from any source of extreme heat (sunlight, heaters, etc).

Study the photos below carefully:



TV PORT Connects your TV to the SVI-728 computer.

AUDIO/VIDEO PORT This port connects your monitor to the SVI-728 computer.

GND Ground.

I/O SYSTEM PORT Connects the SVI-707 disk drive to the SVI-728 computer

PRINTER PORT Connects to a Centronics-type printer.

CASSETTE PORT Connects to a standard cassette recorder.

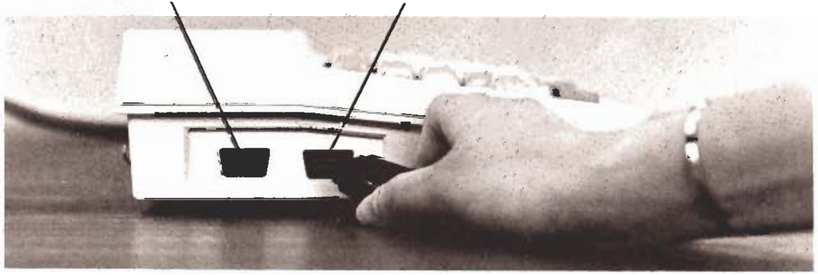
GAME SLOT For MSX game cartridge and other peripherals.

POWER SUPPLY SOCKET Connects the power supply unit to the SVI-728 The other end of the supply unit is connected to an electrical wall outlet.

POWER SWITCH Turns on the power to the SVI-728

JOYSTICK PORT 1 For connection to a joystick.

JOYSTICK PORT 2 For connection to second joystick.



CONNECTION TO A MONITOR

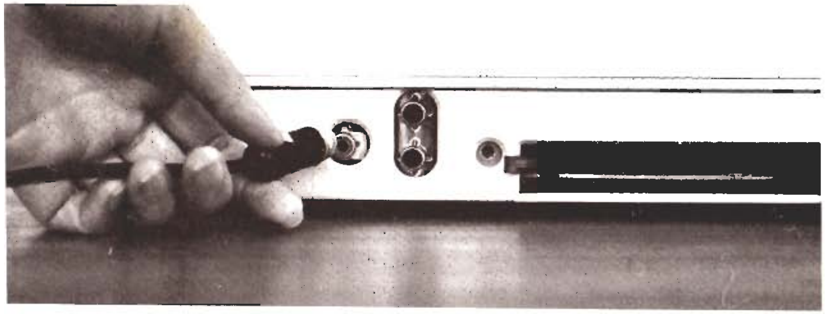
The video cable that is required in the SVI-728 computer system consists of 2 RCA phono plugs for video and audio on both ends.

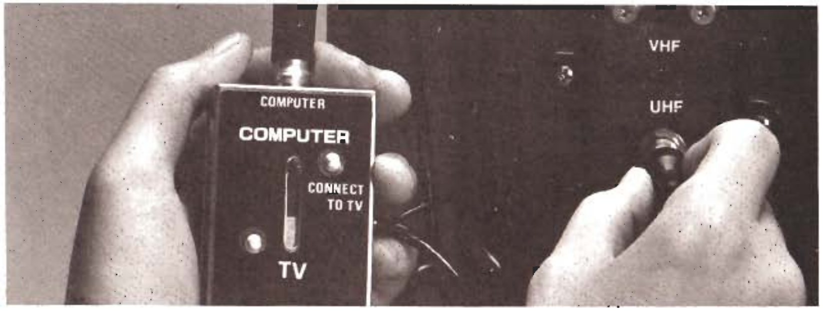
1. Connect one end of the RCA plugs into the video and audio ports on the rear of the SVI-728.
2. Connect the RCA plug marked "video" to the video port and the RCA plug marked "audio" to the audio port on the rear of the monitor.

NOTE: Some monitors require an inexpensive adapter to add to the RCA plugs before connecting them to the monitor. That is available from your local electronics dealer. We recommend the Radio Shack RCA Mini-Adapter (Model #274-330).

CONNECTION TO THE TELEVISION

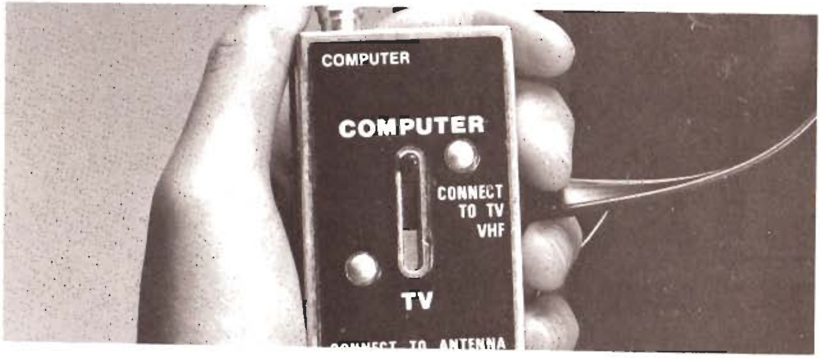
1. Connect one end of the RF Cable to the TV port at the rear of the computer. and the other end to the TV RF input.



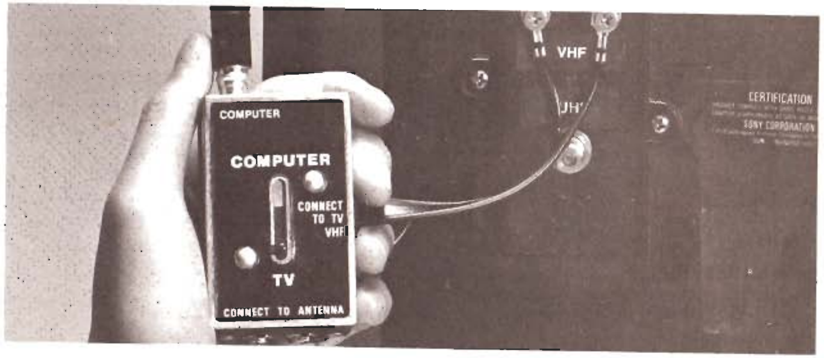


FOR USA VERSION

2. Connect the other end of the shielded video cable to the TV switch box.

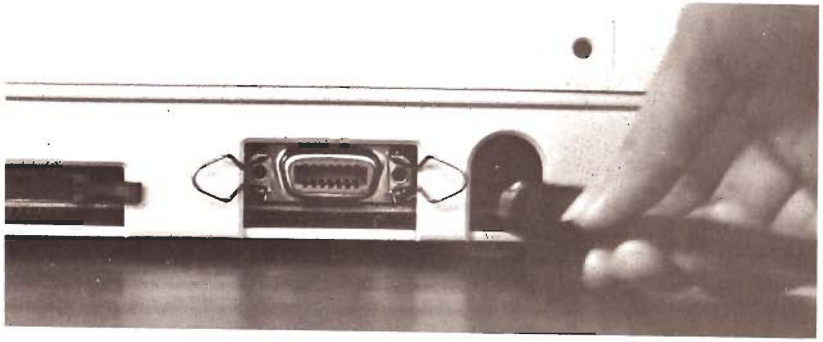


3. Disconnect the VHF TV antenna and reconnect it to the switch box's connector marked TV.

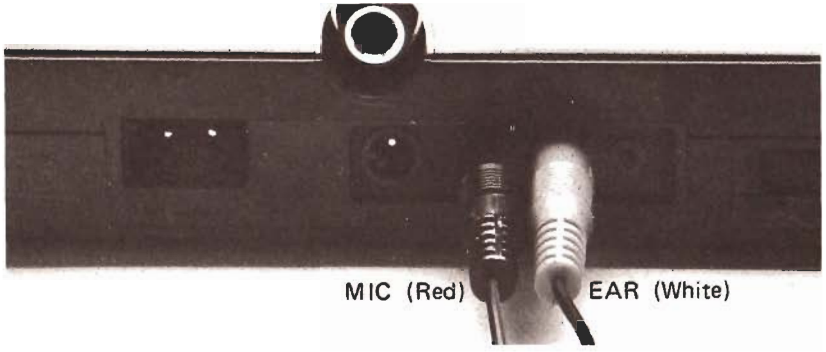


CONNECTION TO THE STANDARD CASSETTE RECORDER

1. Connect one end of the cassette cable to the cassette port at the rear of the computer.

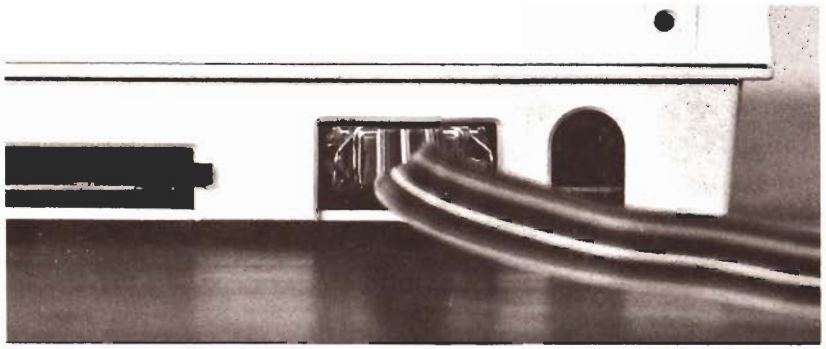


2. Insert the plugs on the other end of the cable to the corresponding jacks on the cassette recorder.

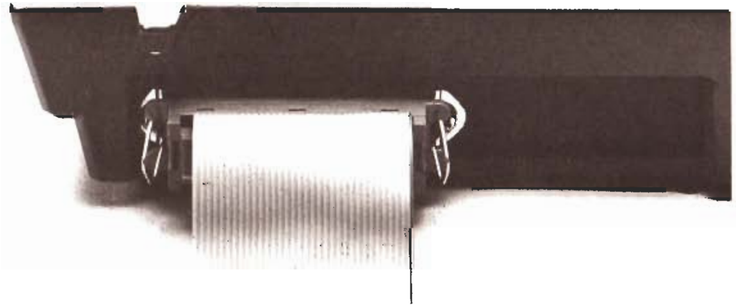


CONNECTION TO THE PRINTER

1. Connect one end of the SVI-207 Centronics Interface Cable to the printer port at the rear of the computer.



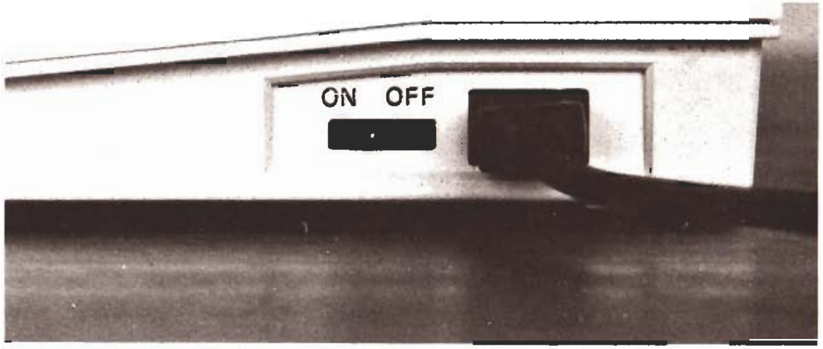
2. Connect the other end of the cable to the connector of the printer.



3. Plug the power cable of the printer to an electrical wall outlet.

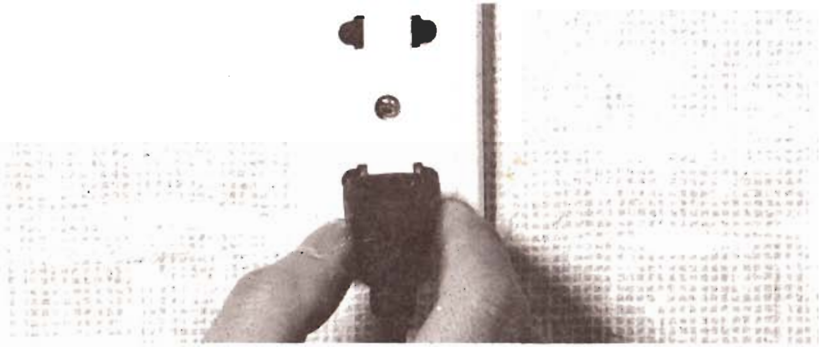
POWER SUPPLY AND CABLE CONNECTIONS

1. Before connecting the power supply, please check and be sure the power switch on the right side of the unit is OFF.



2. Connect the cord coming from the power supply as shown.

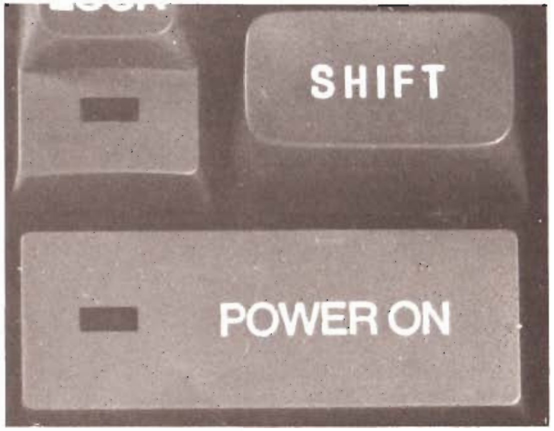
3. Connect the other end of the power supply to any wall outlet.



TURNING ON THE POWER

After you have connected the SVI-728 to your power supply, first turn on your TV then turn the SVI-728 power switch to the "ON" position. There is a channel switch 3-4 located at the bottom of keyboard. Select the proper channel (for USA users only).

The POWER ON indicator will light up.



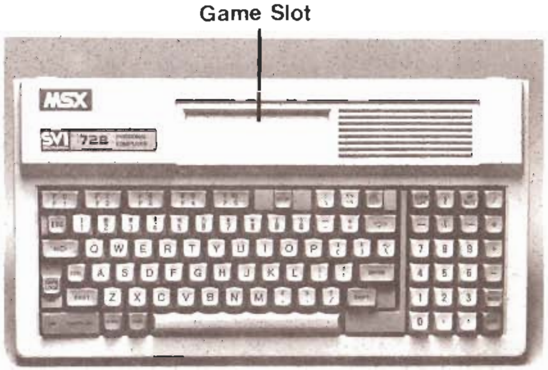
POWER ON SELF-TEST

The SVI-728 has a built-in diagnostic check that will automatically check the function of the system.

If the system still does not start up properly, refer to the trouble shooting chart. (Appendix G)

GAME SLOT

The MSX GAME SLOT is located conveniently on the upper middle position.
To insert or remove a MSX Cartridge, first be sure the power switch is in the OFF position.

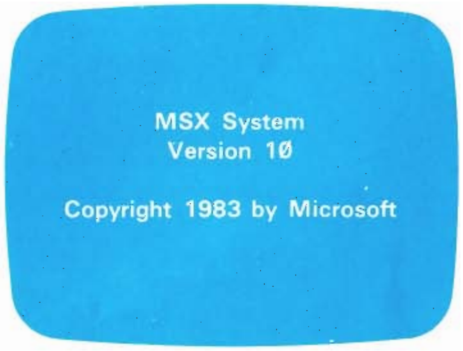


2 THE KEYBOARD

THE KEYBOARD

Programming is generally done by sending instructions to the computer through the keyboard. Your instructions and the computer's responses are visible on a TV screen, which is connected to the keyboard. The computer's keyboard should look somewhat familiar to you because it resembles that of a typewriter. However, the keyboard contains additional keys that are necessary to effectively communicate with the computer.

Turn on you computer (remember, the ON/OFF switch is located on the right side of the computer). You are off to a good start if the screen looks like this when you turn the computer on.

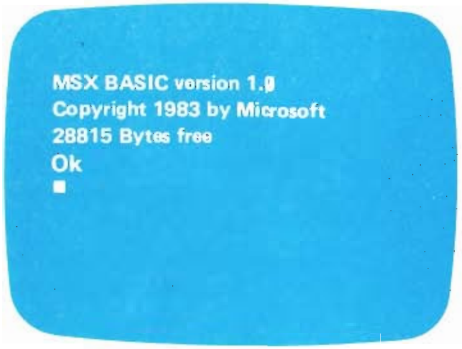


```
MSX System
Version 1Ø

Copyright 1983 by Microsoft
```

If you do not see anything on the screen immediately after turning on the power, turn to the trouble shooting chart in Appendix G for assistance.

After several seconds you will see the following information appear on the screen.



```
MSX BASIC version 1.0
Copyright 1983 by Microsoft
28815 Bytes free
Ok
■
```

Ok

CURSOR

The word "Ok" is the message to you from the computer that it is ready to accept your commands. The white square underneath the word "Ok" is called the "cursor." Its position on the screen informs you of the location of the next letter you type. Let's start typing and get acquainted with the SVI-728 and its special features that aid you in working with the computer.

Begin by pressing the following keys:

PRINT

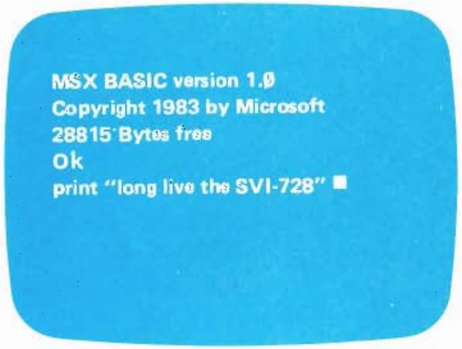
The cursor moves one position to the right every time you press a key.

Now find the SHIFT key (on the lower left side of the keyboard), and while holding it down, press the double quotation mark, " , key. This should cause the quotation marks to appear as on a typewriter.

Type the following message:

" LONG LIVE THE SVI-728"

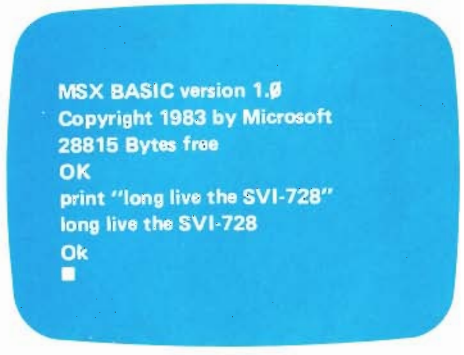
Now make another double quotation mark at the end of the message by pressing SHIFT and " . The screen should now look like this:



```
MSX BASIC version 1.0
Copyright 1983 by Microsoft
28815 Bytes free
Ok
print "long live the SVI-728" ■
```

If you ever make a mistake while typing - for example, you accidentally type the single quote mark instead of the double quotation mark - all you need to do is press the backspace key, and retype by pressing the SHIFT and while holding it down, press the double quotation mark, " Key. The backspace key permits easy retyping by erasing the character immediately to the left of the cursor.

Now press the ENTER key and look at the screen. It will look like this:



```
MSX BASIC version 1.0
Copyright 1983 by Microsoft
28815 Bytes free
OK
print "long live the SVI-728"
long live the SVI-728
Ok
■
```

When you press the ENTER key while in immediate mode, you are telling the computer that you have finished working and that you want the computer to begin working. The

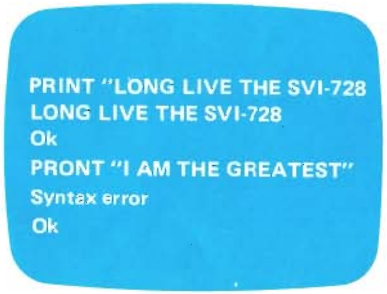
your message on the screen. When it has finished obeying your instructions, it will display the Ok message to inform you that it is ready for more instructions.

If you wish to type using upper case letters, you should press the SHIFT key while pressing a letter key. Should you desire to type using only upper case letters for an extended period of time, you should press the CAPS/LOCK key, located on the lower left side of the keyboard.

The CAPS/LOCK key toggles between the upper and lower case letters. When you wish to return to lower case letters, press the CAPS/LOCK key a second time to unlock it.

Now it's your turn to enter other messages in immediate mode. Don't forget to press the ENTER key when you have placed the closing double quotation mark at the end of each message. If a message is longer than a single screen line, the computer will automatically advance to the next screen line, thus alleviating the need for a return key as on a typewriter (more on this last point later).

If you misspell a word that is a BASIC command, such as PRINT, the following message will appear on the screen:



```
PRINT "LONG LIVE THE SVI-728
LONG LIVE THE SVI-728
Ok
PRONT "I AM THE GREATEST"
Syntax error
Ok
```

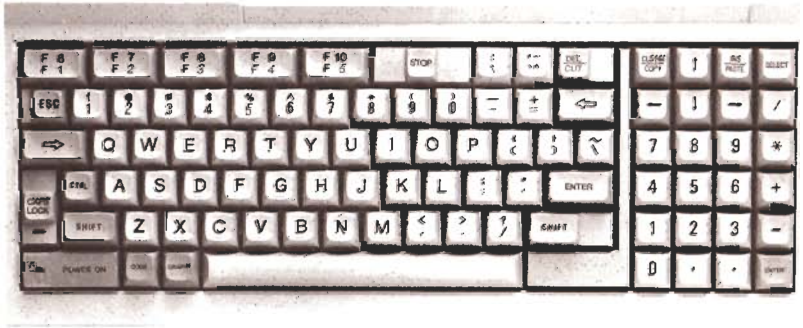
The Syntax error message tells you that you spelled an instruction incorrectly. Don't worry, the computer is very forgiving and patient. We will explain how to correct this mistake in the next chapter.

You can clear the entire screen at once by pressing the CLS/HM key located at the top right hand corner and the SHIFT key simultaneously. The abbreviation "CLS" stands for ClearScreen.

When you have finished practising entering messages for the computer to print, please continue reading.

KEYBOARD LAYOUT

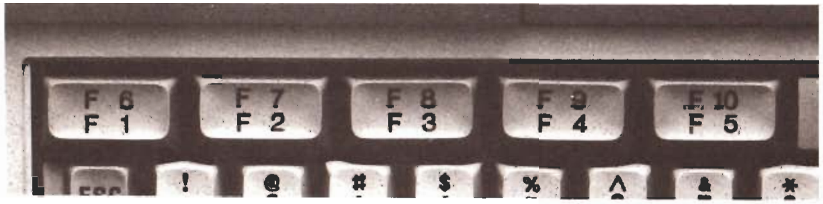
So far we have introduced you to the following keys: letter and number keys, the backspace key, the ENTER key and the CLS/HM key. All keys are shown below.



The SVI-728 has many convenient features that are operated by certain keys. You will find these special keys to be both helpful and time saving. The remainder of this chapter will describe these keys, most of which will be new and some of which will be a review.

FUNCTION KEY

Look at the top row of keys on the keyboard, the ones highlighted below.



These keys are called "function" keys and each one is marked with the letter "F" . They are a labor-saving device because they allow you to instruct the computer to perform a frequently used function by pressing only one key instead of having to type many keys.

Here is a list of each key, the function it performs and a brief description of the function.

KEY	PRE-DEFINED FUNCTION
F1	color
F2	auto [1Ø , 1Ø] ENTER
F3	goto
F4	list
F5	run ENTER
F6	color 15,4,7
F7	cload"
F8	cont ENTER
F9	list. ENTER
F10	CLS run ENTER

Function keys **F1** through **F5** are operated by pressing the appropriate key. Function keys **F6** through **F10** are operated by pressing the SHIFT key and holding it down while simultaneously pressing the appropriate key.

F1 COLOR

The COLOR command is used to change the text, background and border colors on your TV or monitor.

F2 AUTO

The AUTO command is used to make the computer generate program line numbers automatically. This command is used very often, since all programming statements must be preceded by line numbers.

F3 GOTO

GOTO is a command which provides you with the ability to execute your program from any place (line number) you desire.

F4 LIST

This command instructs your SVI-728 to print all of your immediately preceding program statements on the screen. LIST is probably the most often used computer command.

F5 RUN

RUN tells the computer to take the program you have written and perform the commands you have indicated.

F6 COLOR

This tells the computer to print white letters on a blue background with a cyan border. These colors are the colors of the screen when you turn the computer on.

F7 CLOAD"

CLOAD instructs the computer to input (load) data from a cassette recorder (which can be easily connected to your SVI-728.

F8 CONT

This command is used to tell the computer to "continue" program execution after the last executed line.

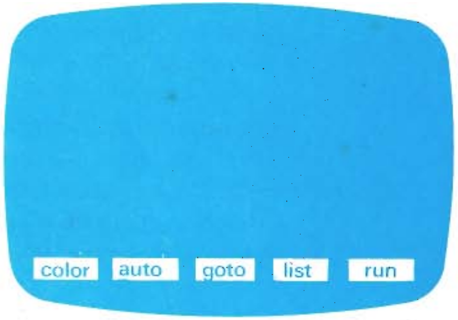
F9 LIST.

With this LIST. (with a period next to it) only the last line you were working on (whether programming, editing, etc.) will be displayed on the screen.

F10 RUN

This command is similar to the standard RUN command. However this command also clears the screen before it "runs" your program.

On the bottom of your TV screen, the SVI-728 lists the function that each key performs.



color
cload"
cont
list.
run

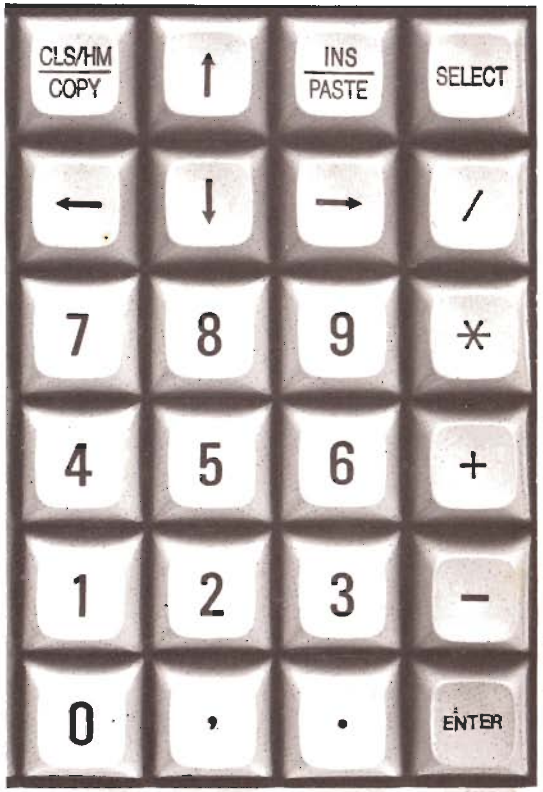
SHIFT

The SVI-728 will normally display the function of keys F1 through F5, and whenever you press the SHIFT key it displays the function of keys F6 through F10.

Any of these pre-defined functions can be quickly changed for your own convenience to a function that you frequently use.

NUMERIC KEYPAD and CURSOR CONTROL KEY

The numeric and cursor control keypad contains keys that are primarily used for simple numeric entry, word processing and cursor control. The followings are the commands that are accessed by this keypad.



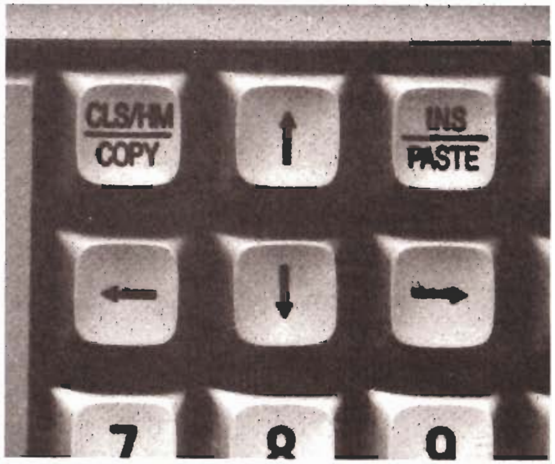
Numeric Keypad

The numeric keys **(0-9)** are the same as the keys on the top of the regular keyboard. These are used when performing rapid entry of numeric data. This keypad also contains the mathematical functions keys (+, −, *, /) which can be used to enter formulae and to perform quick calculations.

SELECT and PRINT KEYS

The SELECT and PRINT keys are also included on this keypad to allow the advantage of using these functions that are often available in word processing and data entry software packages. These keys have no function in BASIC programming and are only accessed from programs such as those mentioned above.

Arrow Keys (Cursor Control)



The arrow keys **(up, down, left & right)** control the movement of the cursor on the display screen. By pressing a combination of the up and left arrow keys, you will cause the cursor to move towards the upper left corner of the display screen. Other combinations will work in the same fashion giving you 8 directions of cursor movement using these keys.

PROGRAM CONTROL KEY

The followings are the program control keys used to control the operation of computer programs.

STOP

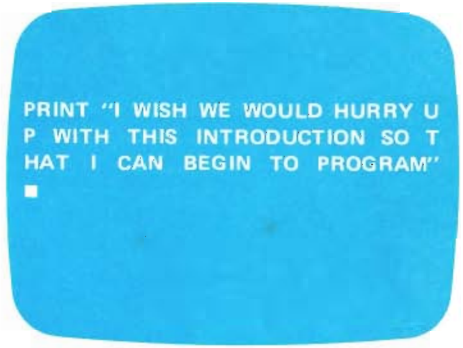
The STOP key. Press this key to pause the computer after you have instructed it to run or to perform a function (which makes it begin working on your program). Press the STOP key a second time to instruct the computer to resume working on your program or a function.

CTRL
STOP

The **CONTROL** key. This key is used in conjunction with the STOP key. In effect, this tells the computer to stop what it's doing and turn control back over to you (so that you can issue further instructions). Press the CTRL key while simultaneously pressing the STOP key.

ENTER

The ENTER key. Press this key at the end of each instruction you type. By pressing this key you are telling the computer to enter the instruction you just typed into its work space. As we previously mentioned, the ENTER key is not used to advance the cursor to the next screen line and therefore should not be confused with the return key on a typewriter. In the event that an instruction contains more characters than can fit on a single screen line, the computer will automatically advance the cursor to the next screen line. For example:



```
PRINT "I WISH WE WOULD HURRY U
P WITH THIS INTRODUCTION SO T
HAT I CAN BEGIN TO PROGRAM"
■
```

This long instruction cannot possibly fit onto a single screen line which has room for only 29 characters. Whenever this happens, the computer will automatically advance you to the next line. In the above example, the ENTER key should only be pressed after you have typed the closing double quotation mark that follows the word PROGRAM.

MISCELLANEOUS KEY

CAPS
LOCK

The CAPS/LOCK key – pressing this key will toggle the display characters from lower case to upper case or upper case to lower case.

CLS/HM
COPY

The CLS/HM key – pressing this key will move the cursor to the upper lefthand corner of the screen. When pressed together with the

SHIFT key, it will move the cursor to the upper lefthand position (home) and clear the screen.

INS PASTE

The INS/PASTE key – this key is used when you wish to insert characters within a line. Just move the cursor to the location where you wish to insert, then press this key and the text you type will be inserted.

DEL CUT

The DEL/CUT key – press this key to delete the character under the cursor.

ESC

This key is often used in software application programs. Its usual function is to interrupt the operation of a program or to continue operation following an interrupt (Escape).

⇨

This key is not used in BASIC. It is often used in a word processor or similar application program to space forward 5 spaces to begin a paragraph.

⇦

This key backs up the cursor one space. It deletes the character immediately to the left of the cursor prior to the key press.

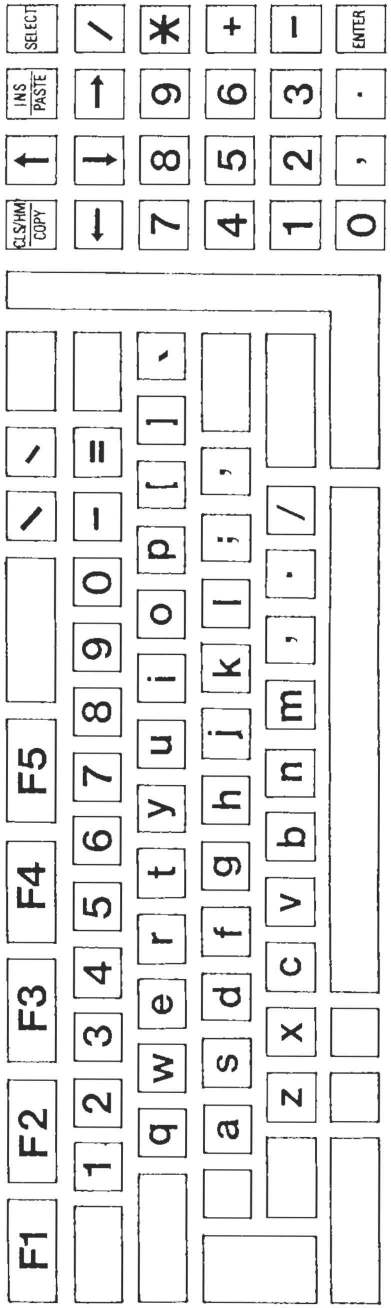
GRPH

The GRPH key is used to select the graphic symbols that correspond to the keys. If you press the GRPH key and hold it down while simultaneously pressing one of the letter keys, the graphic symbol of the corresponding key displayed.

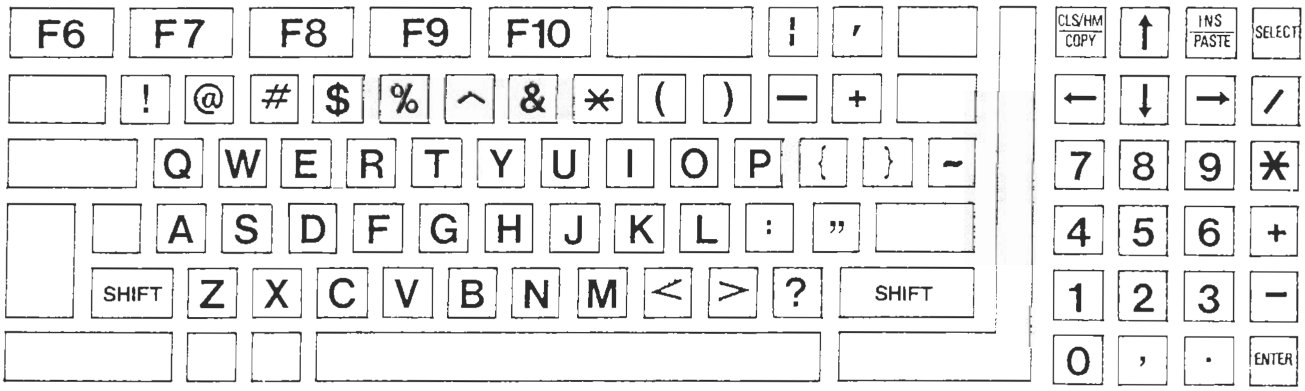
CODE

The CODE key is used to select the characters other than English that correspond to the keys. It you press the CODE key and hold it down while simultaneously pressing one of the letter keys, the different language character of the corresponding key will be displayed.

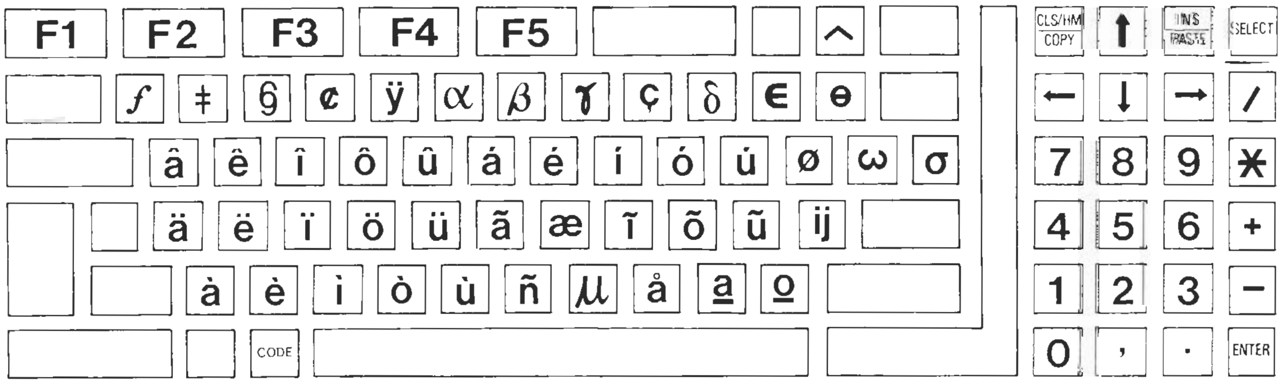
The details of the keyboard layout and templates are shown below:



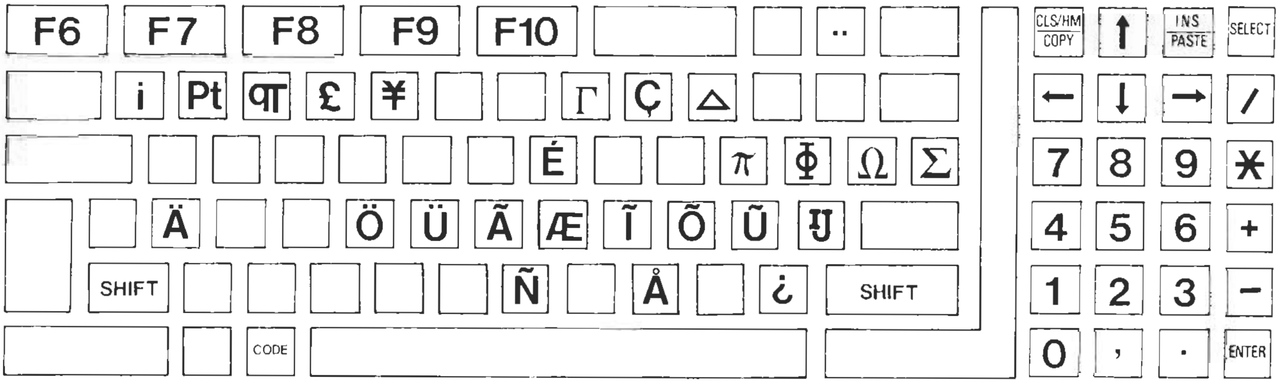
STANDARD KEYBOARD LAYOUT



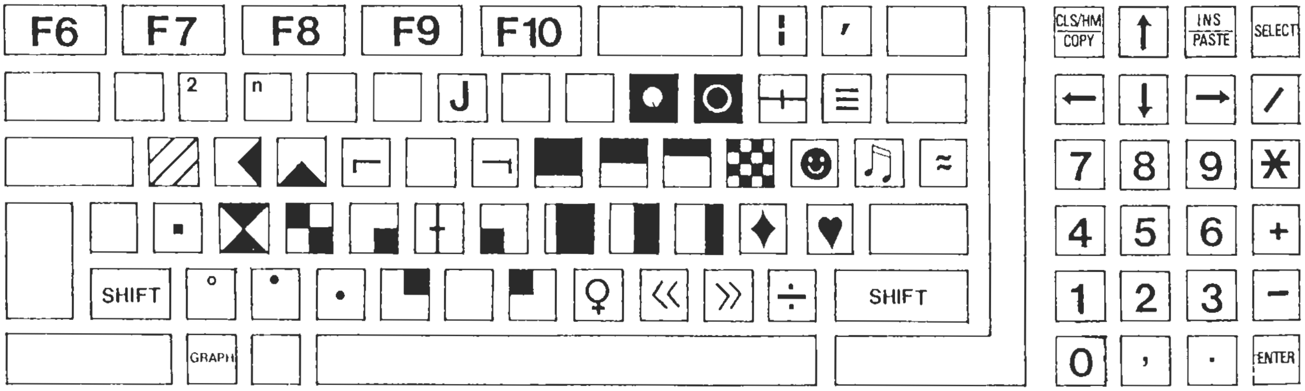
STANDARD KEYBOARD LAYOUT
WITH SHIFT KEY PRESSED



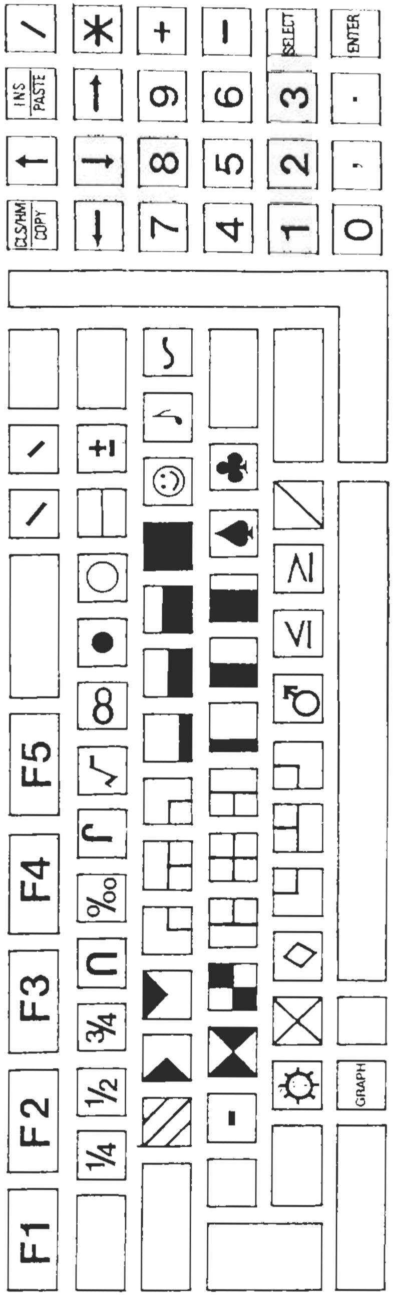
STANDARD KEYBOARD LAYOUT
WITH CODE KEY PRESSED



STANDARD KEYBOARD LAYOUT
WITH SHIFT & CODE KEYS PRESSED



STANDARD KEYBOARD LAYOUT
WITH SHIFT & GRAPH KEYS PRESSED



STANDARD KEYBOARD LAYOUT
WITH GRAPH KEYS PRESSED

Now that you have learned how to hook up your computer and are familiar with the keyboard, you can proceed further in the manual. However, if you are anxious to begin using your computer, please turn to **Appendix I** for some simple demonstrations.

After completing these exercises, return to chapter 3 to continue.

EASY EDITING

3 EASY EDITING

The BASIC Screen Editor lets you change a line anywhere on the screen. You can change only one line at a time. The Screen Editor can be used after an Ok prompt appears and before a RUN command is issued. By using the Cursor Control keys and the editing keys, you can move quickly around the screen, making corrections where necessary.

SCREEN EDITOR

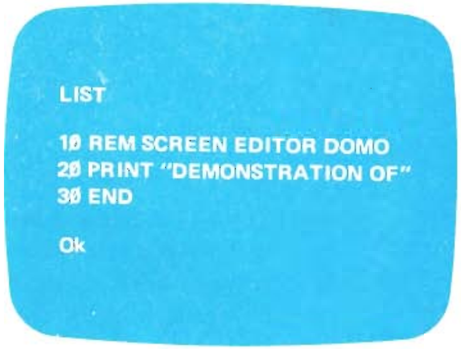
Here is an example to show you how to use the Screen Editor.

Let's enter the following program.

```
1ØREM SCREEN EDITOR DOMO ENTER
2ØPRINT "DEMONSTRATION OF" ENTER
3ØEND ENTER
```

Note: Remember that a program line must always begin with a line number. If you make a mistake, just press ENTER and retype the line.

After you have finished, press the CLS/HM and SHIFT keys simultaneously, then type LIST or F4 and press ENTER You should see:

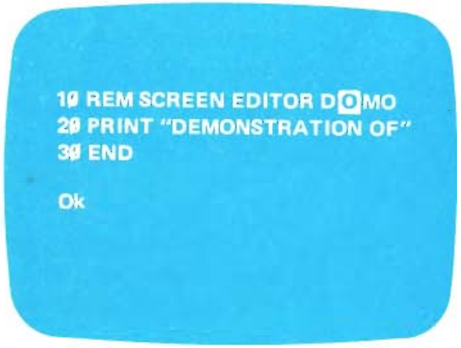


```
LIST

1Ø REM SCREEN EDITOR DOMO
2Ø PRINT "DEMONSTRATION OF"
3Ø END

Ok
```

Now correct the word DOMO in line 10. First use the cursor key's UP direction key to move to line 10 and then the RIGHT direction key to move the cursor to the top of the letter "O" of "DOMO".



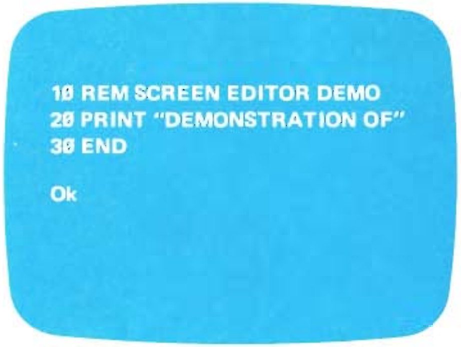
```
1Ø REM SCREEN EDITOR DOMO
2Ø PRINT "DEMONSTRATION OF"
3Ø END

Ok
```

Press the letter "E" to change the word to "DEMO", then press ENTER . The line will be stored now as:

1Ø REM SCREEN EDITOR DEMO

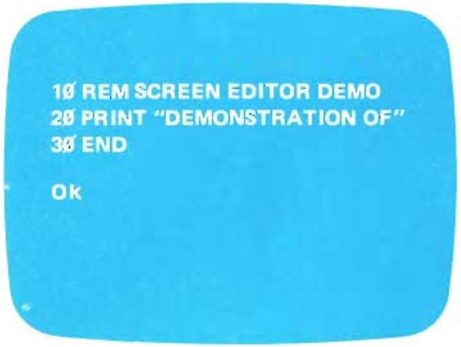
You have just replaced the character "O" with the character "E". To verify this, press SHIFT and CLS/HM simultaneously, then F4 (list), ENTER . You should see:



```
1Ø REM SCREEN EDITOR DEMO
2Ø PRINT "DEMONSTRATION OF"
3Ø END

Ok
```

The next step is to insert the two words SCREEN EDITOR into line 20. We do this by moving the cursor to the second quote of Line 20.



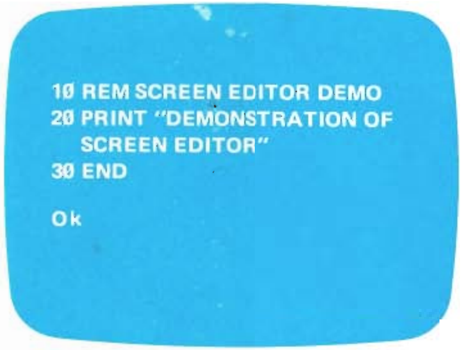
```
1Ø REM SCREEN EDITOR DEMO
2Ø PRINT "DEMONSTRATION OF"
3Ø END

Ok
```

INSERT MODE

Now press the INS/PASTE , key and the cursor will become half as tall as before. This means you are in the "INSERT" mode. Type "SCREEN EDITOR", bring the cursor back to the beginning of line 2Ø, and press ENTER .

You have just inserted the words "SCREEN EDITOR" .Follow the steps you used to verify line 1Ø and you will see:



```
1Ø REM SCREEN EDITOR DEMO
2Ø PRINT "DEMONSTRATION OF
   SCREEN EDITOR"
3Ø END

Ok
```

Besides using the Screen Editor, you can also change a line by entering a new one with the same line number. BASIC will automatically replace that line.

4 A BASIC INTRODUCTION

ABOUT THIS BASIC TUTORIAL GUIDE

To be able to control a computer, you must be able to communicate your instructions in a language that the computer understands. The SVI-728, like most personal computers, understands a language called BASIC (Beginner's All Purpose Symbolic Instruction Code). This language, which is built right into the SVI-728, is a set of English words with which you can instruct the computer to perform certain functions.

This manual differs in many ways from other manuals written to teach BASIC. One of the major differences is that most of the information presented in this tutorial has already been used successfully to teach BASIC in the classroom.

This guide begins by describing those BASIC commands which allow you to design simple pictures and see them displayed on the TV screen. Our reason for introducing you to BASIC through graphics is simple. Learning BASIC is like learning a foreign language. If we can relate the new words of BASIC to the knowledge that you already possess about drawing, you can move smoothly into the computer age and have fun at the same time.

In the next few chapters we will explain several commands which allow you to create pictures. But there is much more to learn about the SVI-728's graphic capabilities. It will be explained in greater depth in the latter chapters of this manual.

WHAT IS PROGRAMMING?

Programming is the act of writing the instructions and information that must be given to the computer in order for it to perform a task. Programs differ from one another in that the instructions and information necessary for managing a household's checkbook, for example, are different from the instructions and information needed to control a video game.

Programs written in one computer language will not contain the instructions and structure that a program in another computer language possesses. That is why professional programmers generally must specialize in one or two languages. It is the rare individual who is even aware of all the different computer languages that have been developed in the past twenty years.

There are two different ways to type a BASIC instruction into your computer: In "program" mode or in "immediate" mode.

As its name implies, you are in "program" mode when you write a program. A BASIC program is a set of instructions typed one instruction after the other with a line number beginning each instruction line. For example:

```
1Ø PRINT "I AM"
2Ø PRINT "STARTING TO"
3Ø PRINT "LEARN BASIC"
```

Line numbers are generally in intervals of ten to allow for easy reference when corrections are required or when additional lines need to be inserted. The computer executes your program after you type the word RUN.

The second way of instructing the computer is called "immediate" mode, because after each instruction is typed and the ENTER key is pressed, the computer will immediately respond. Do not precede the single line of instruction with a line number when you are in "immediate" mode.

ONE WORD BEFORE WE BEGIN

For most of the time spent with this tutorial, you will be in program mode. If you are serious about programming, then you will continue to write programs and very infrequently be in the immediate mode. The immediate mode is generally reserved for housekeeping details like saving programs on a cassette tape or disk, requesting a catalog of your programs stored on a disk and loading information from a cassette or disk into the computer.

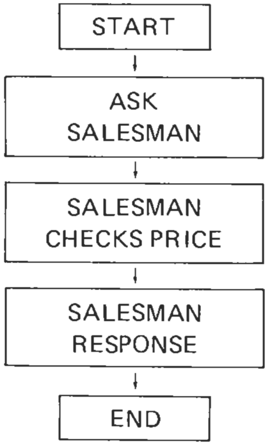
We realize that these words and concepts are new for many of you, but don't worry. These ideas will become clearer as you continue reading.

The model that we are about to present is critical to your fully understanding how a computer works and what it does with the program you type in.

Your interaction with the computer is similar to the following scenario.

If you were interested in buying a car you would probably call a car dealer for a price on a particular model. You might ask the salesman for a price on a Cadillac with power windows, power steering, leather seats, deep pile carpeting, and tinted windows. The salesman will then proceed to check his price lists, write the numbers down on a worksheet, add the numbers together and quote you the total price.

Procedures to get information about a car.



By the same token, when you work with a computer, the computer is the salesman. You tell the computer exactly what you want just as you tell the salesman and in both cases you get an answer. Similarly, in both instances you do not see the many calculations that both the salesman and the computer perform before informing you fo their answer. The instructions you give to the salesman or to the computer is the 'INPUT' they need to act on.

The calculations done on the salesman's worksheet are analogous to the wok performed in the computer's erasable memory (called RAM or Random Access Memory), and the answer you get is called, the 'OUTPUT'.

By reading and practising the new language you have started to learn, you will be entering the ever expanding computer world. BASIC is only the beginning.

ENJOY.

5 WRITE YOUR FIRST PROGRAM

GETTING STARTED

The easiest way to draw pictures on the computer is similar to the way one plays the game Battleship^{TM}.

The goal of Battleship is to try to guess the positions of your opponent's ships. The game board for Battleship looks like this:

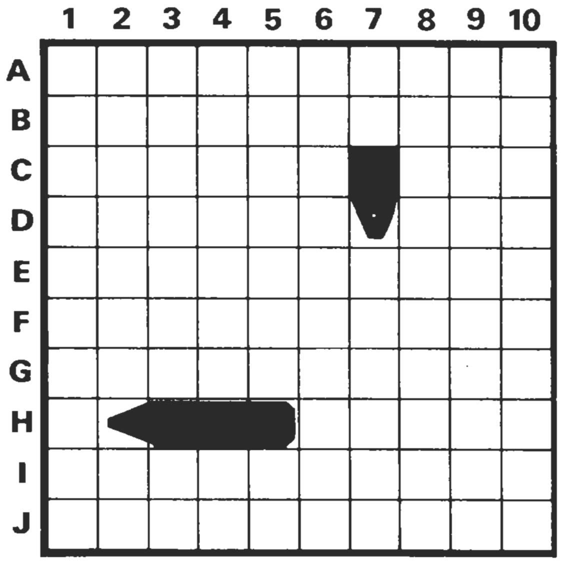


Figure 1 Battleship Game Board

Figure 1 shows how your opponent has his board set up. One of his ships is at positions H-2, H-3, H-4, and H-5. His other ship is at positions C-7 and D-7.

You must seek out his ships by calling the names of positions on the board. In this example, if you were to call H-2 he would reply that you hit his ship, but if you said H-1, he would have told you that you missed.

The important lesson to learn here is the way you give names to each of the positions on the board. In essence, each position on the Battleship game board is a point on a graph.

GRID and COORDINATES

To draw on the SVI-728 you must also give a name to each box of the grid that is shown in Figure 2.

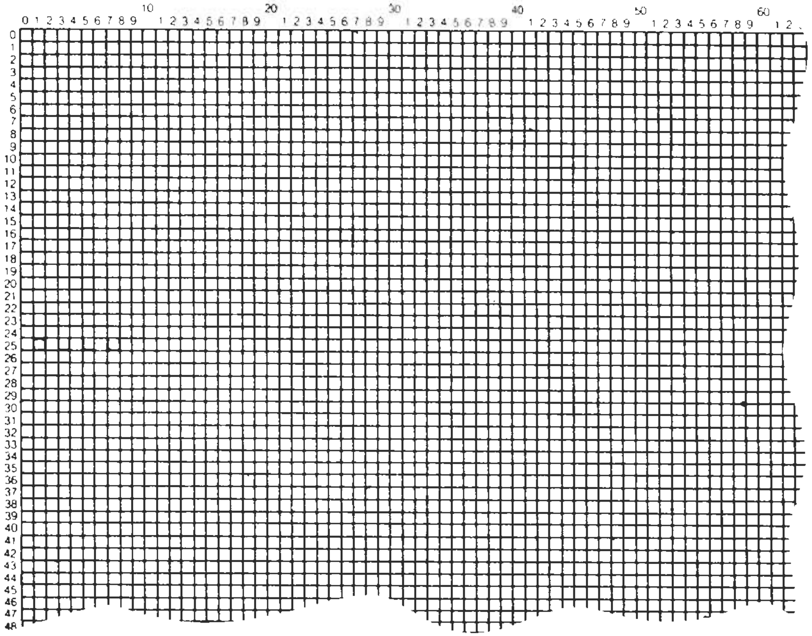


Figure 2

Each box has a name. The name is composed of two numbers: the number at the top of a column, and the number at the side designating each row. The first number is the horizontal location and the second number is the vertical location. These two numbers are called: the coordinate numbers of a point.

Understanding how to name boxes is very important for creating simple pictures. It is especially important because we are going to introduce through which pictures can be created. Spend a moment on the following exercise before proceeding to the next section.

In Figure 3, several boxes are darkened. This signifies that they have been lit. Do you know the names of these boxes? Write them down on a piece of paper.

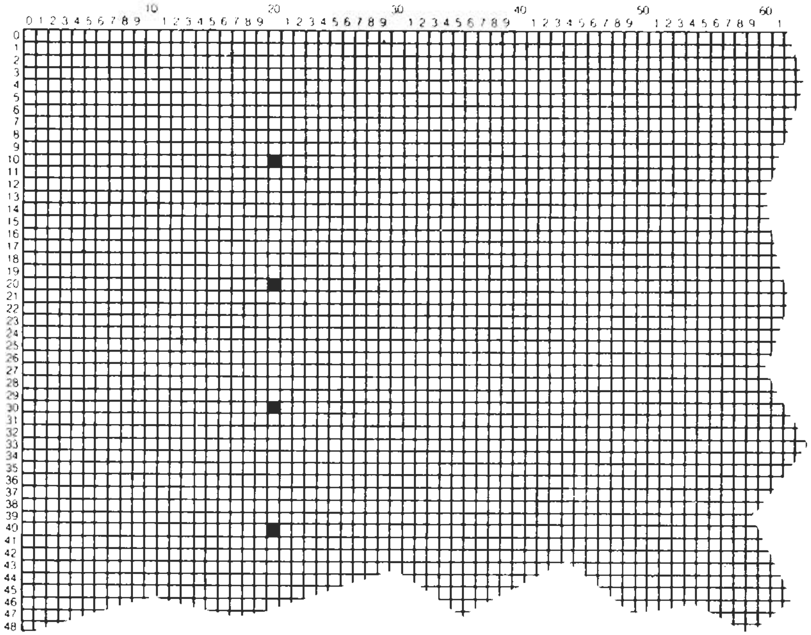


Figure 3

Your answers should have been
A: (2Ø, 1Ø)
B: (2Ø, 2Ø)
C: (2Ø, 3Ø)
D: (2Ø, 4Ø)

Now we will use these names in a program which will light the boxes in Figure 3. Type on the keyboard exactly what is listed below.

Don't forget to press ENTER at the end of each line. We will not always remind you from this point on. Pressing ENTER after each instruction should become second nature to you by the time you finish this book.

```
10 SCREEN 3
20 PSET (70,50)
30 PSET (66,54)
40 PSET (74,54)
50 PSET (62,58)
60 PSET (78,58)
70 PSET (66,62)
80 PSET (74,62)
90 PSET (70,66)
100 END
```

After you have checked to make certain that your program matches our program, type RUN.

What's the matter? Did the picture flash by too quickly? Don't worry we expected that little problem to occur. Do the following:

LIST

1. Type :

   LIST ENTER

   Ok

Your program is displayed again on the screen. As its name implies, LIST will list your whole program from beginning line number to the last number you used. If you typed LIST when you were in *immediate mode* and not using the line numbers, you would not see a program listed.

2. Type in the following new line.

95 GOTO 95 ENTER

3. Type LIST ENTER

Notice that line 95 was inserted between lines 90 and 100 automatically by the computer. Remember what we had said earlier in this guide about line numbers and why we usually use numbers with an interval of ten between them. The insertion of additional lines is a natural occurrence when you write programs.

4. Type RUN ENTER

The computer not only draws the shape we want, but it also keeps the picture on the screen forever and does not allow you to type in anything at the keyboard. Try typing. The characters you type don't appear on the TV. Why? Because the machine is still RUNning your program and will continue to do so until you press the CTRL + STOP keys simultaneously. Let's look at the program again.

```
 10 SCREEN 3
 20 PSET (70,50)
 30 PSET (66,54)
 40 PSET (74,54)
 50 PSET (62,58)
 60 PSET (78,58)
 70 PSET (66,62)
 80 PSET (74,62)
 90 PSET (70,66)
 95 GOTO 95
100 END
```

Before we added line 95, the picture appeared momentarily and then disappeared because the program did exactly what it was supposed to do (very quickly) and then disappeared. By adding Line 95, we told the computer to stay right where it is. To stop the program, press the CTRL + STOP keys simultaneously.

The CTRL + STOP keys combination breaks the computer out of its continuous loop, and will stop the computer from whatever program happens to be RUNning.

You will see whatever you type appear on the TV. Remember to press ENTER after you finish each line.

```
1Ø SCREEN 3
2Ø PSET (1Ø,2Ø)
3Ø PSET (2Ø,2Ø)
4Ø PSET (3Ø,2Ø)
5Ø PSET (4Ø,2Ø)
6Ø GOTO 6Ø
7Ø END
```

Line 1Ø tells the computer that you are going to draw a picture on SCREEN 3.

SCREEN

There are 4 screens:

SCREEN Ø - This screen allows you to enter 23 lines of text with a maximum of 39 characters on each line before the display starts to "scroll" or move up.

SCREEN 1 - The screen the computer displays when you turn it on. This screen allows you to enter 23 lines of text with a maximum of 29 characters on each line before the display starts to "scroll" or move up.

Both of the two screens allows you to communicate with your computer and tell him what to do for you: type in your commands in the immediate mode, or type in your programs.

SCREEN 2 - The High Resolution graphics screen which you use to draw High Resolution graphics.

SCREEN 3 - The Low Resolution graphics screen which you use for drawing Low Resolution graphics.

(More about High and Low Resolution you will find in Appendix F)

Whenever you CTRL-STOP a RUNning program which uses SCREEN 2 or SCREEN 3, the computer will automatically return to SCREEN Ø or 1 (the previous text screen) program you by displaying the "Ok" message and the cursor. This will allow you to review (LIST) your program and make your modifications.

PSET

Line 2Ø introduces the PSET instruction. PSET tells the computer to turn on the box that is named (1Ø,2Ø). Remember, the first number in parentheses refers to the column and the second number refers to the row of the box.

Lines 3Ø-5Ø continue to turn on the boxes we see darkened in Figure 3. Line 6Ø will soon be explained.

Now type RUN ENTER and your picture will be displayed. When you finish viewing your work, press the CTRL and STOP keys simultaneously to stop the program.

What you typed into the computer was your first program. Giving several instructions to the computer at once is generally more efficient than giving the computer one instruction at a time in immediate mode. (Remember that in *immediate mode* the computer executes the command you type immediately after you press the ENTER key.) If we didn't use line numbers to signify that we were writing a program but rather told the computer to turn on one box at a time, it would be like going to a supermarket ten times in one day and each time buying only one item. That would be a terrible waste of gas and time.

After all this talk, it is time for you to practise what we have been preaching.

Type NEW ENTER

NEW tells the computer to forget about the program you had previously typed.

Try to write a program that draws the shape in Figure 4.

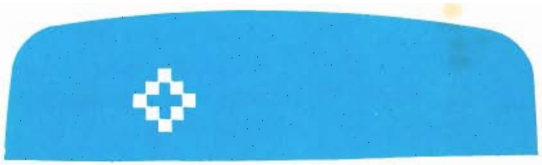


Figure 4

LET'S ADD SOME COLOR

Each of the boxes you light up can be in one of 16 colors. Here is a list of the 16 colors the computer uses and the corresponding number for each color.

COLOR #	COLOR
Ø	TRANSPARENT
1	BLACK
2	MEDIUM GREEN
3	LIGHT GREEN
4	DARK BLUE
5	LIGHT BLUE
6	DARK RED
7	CYAN
8	MEDIUM RED
9	LIGHT RED
1Ø	DARK YELLOW
11	LIGHT YELLOW
12	DARK GREEN
13	MAGENTA
14	GRAY
15	WHITE

Before describing how to add colors to your drawings we will first demonstrate your computer's vivid colors.

Press the CLS/HM and SHIFT keys to clear the screen and type the following:

COLOR 4, 15 ENTER

(Note that function key #1 F1 can generate the word "**color**" for you]

The colors of the background and text are now reversed. The numbers 4 and 15 represent two of the 16 colors listed above.

Let's experiment with the COLOR command. Begin by typing:

COLOR 4, 4 ENTER

The text disappeared and the only thing you see is a blue screen. Now type your name. You should hear the clicking sound but no characters are displayed on the screen. Here is why:

The first number following the COLOR command is the color you are instructing the computer to use for the display of text while the second number is the color the computer will use for the background. Since you instructed the computer to use the same blue color (4) for both the text and the background the text seemed to disappear. In other words, what you type is printed on the screen, you just can't see it! In order to return to "visible" text you need to type a new COLOR command, but it would not be easy without seeing the cusor or what you typed.

F6

Now you could appreciate the usefulness of function key #6.

First make sure that the cursor (which you cannot see) is positioned at the beginning of a "clean" new line. You do this simply by pressing ENTER

You will still not see anything changed, but you will hear a beep. Since the last thing you typed was your name, by pressing the ENTER key you caused the computer to generate a "Syntax error" message which is always accompanied by a short "warning" beep. This of course, wouldn't happen if your name is a correct command in the BASIC language.

Now, press the SHIFT key and while holding it down press function key # (**F6**). The display returns to normal and you should see now all the words you previously typed plus the "Syntax error" and "Ok" messages.

Take a look at the last color command on the screen:

COLOR, 15,4, 4

this is the command you gave the computer by using F6 (Note that you didn't have to ENTER this command, since F6 ENTERs automatically. You see that there is a third number added to the color command. We already know that the first number (15) is the text color, and that the second number (4) is the background color. The third number (7) is the color the computer uses for the border of the screen. The border is usually not displayed in the text mode (screen Ø and 1). That's why the need to specify a border color exists mostly when you are using SCREEN 2 or SCREEN 3.

BORDER BACKGROUND and FOREGROUND

We may think of the screen's display as of three layers, one on top of the other. At the bottom there is the **Border,** above it there is the **Background** which in the text mode, or SCREEN Ø or 1, covers the Border totally; and in the graphic screen (2 or 3) "grows down" in size and "exposes" the Border at the top and bottom of the screen .
Above the Background comes the **Foreground** which might be described as a clear acetate layer that "carries" all the images that appear on the screen: on SCREEN Ø or 1 it's the text, and on SCREEN 2 or 3 it's the graphic image.

If you understood this concept you should feel comfortable with the following fomat description: the format of the COLOR command is

COLOR, <foreground color #> ,
<background color #> ,
< border color #>

Now, experiment with the numbers to get familiar with all the colors.

O.k. Let's get back to the PSET command

PSET (1Ø,2Ø) 3 ENTER

Now change the color,

PSET (1Ø,2Ø), 8 ENTER

The PSET command only scratches the surface of the almost unlimited graphic capabilities of the SVI-728. The majority of commands to create exciting pictures on your computer will be described after we have introduced you to the BASIC language.

GOTO

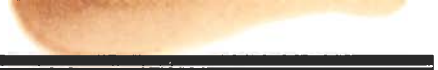
Before we end this chapter, type in this popular little program that uses the GOTO command in a way that is easily understood.

First type NEW ENTER

and then

```
10 PRINT "I LOVE MY SVI-728
    COMPUTER"
20 GOTO 10
```

now type RUN.



Press the CTRL + STOP keys combination to stop the runaway program. Do you understand why the machine printed "I LOVE MY SVI-728 COMPUTER" again and again? Here's why.

Line 1Ø tells the computer to PRINT the message between the parentheses on the screen. And Line 2Ø tells the computer to go back to line 1Ø and print the message again. After it is printed a second time, the computer reaches line 2Ø again and is sent back again to line 1Ø and then again and again . . . to infinity. Now try inserting other messages in between the parentheses in line 1Ø. The quickest way to change line 1Ø is as follows:

First type LIST

to see your program. Don't forget to press ENTER

Next retype

```
1Ø PRINT "
```

Now add any message you want, such as your name, and put a closing quotation mark at the end of your message. After you have finished retyping line 10 and pressed ENTER , the change has been entered in the computer's memory.

Now type RUN

and the revised message should appear on the screen. We will have more to say about the PRINT command in a later chapter.

We suggest that you read appendix F now for clarification on screen modes, before go to next chapter.

6 GOING THROUGH THE LOOPS

After using the PSET command in the previous chapter, you probably thought: there has to be a way to turn on the lights of particular boxes that is quicker than having to name each box individually. Well there is! But before we show it to you, first type in the following program:

In MSX-BASIC there is a nice shortcut that saves you from always having to type out the word command **PRINT**- just type in a **question mark (?)**-it serves the very same purpose!

```
1Ø CLS
2Ø PRINT "TELL ME A NUMBER"
3Ø INPUT Y
4Ø SCREEN 3
5Ø PSET (3Ø,Y)
6Ø PSET (45,Y)
7Ø PSET (6Ø,Y)
8Ø PSET (75,Y)
9Ø GOTO 9Ø
```

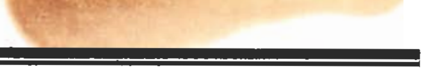
Now type RUN

Your program should have disappeared and the message "TELL ME A NUMBER" should suddenly appear on a cleared screen. The screen was cleared, thanks to the CLS command which stands for "ClearScreen", which is on line 1Ø.

The message is output to the screen by the PRINT command in line 2Ø, and the cursor should appear just after a question mark.

INPUT

The question mark was put there by the command INPUT. INPUT tells the computer



to wait and not go on to the next instruction of the program until you type something in at the keyboard. Whatever you type in is stored in a "container" in the computer's erasable memory.

CONTAINERS and VARIABLES

The type of a variable container discussed in this chapter is a numeric container. This is used for the storage of numeric values only. Non-numeric variables or Alpha-numeric variables will be discussed in chapter 8.

The more technically oriented of our readers will recognize that the container we are referring to is a **variable.** The following explanation about the way a computer uses containers to store information is meant for those readers who aren't comfortable with the term "variable".

Before we continue with the description of containers, let's first observe a container in action. First type in a number between 0-191 and press ENTER . (If you press a number greater than 191, you will get a blank screen.)

If you typed in a number between 0-191 and pressed ENTER , a line should have been drawn on the screen. The location of that line depends on the number you typed in. Here is why.

The number you typed is stored in a container called "Y", as designated in line 3Ø. Think of "Y" as a cup, and the number you typed as a mark on a piece of paper that is placed in the cup. When the computer reaches line 5Ø and it is time to turn on the box that is specified by the name (3Ø,Y), the computer looks in the cup marked "Y" and substitutes the number you typed in for the letter Y wherever "Y" appears. So if you had typed in 27, the number 27 would be stored in the container "Y". When the computer reaches line 5Ø it will light up box (3Ø, 27); at line 6Ø box (45, 27); at line 7Ø box (6Ø, 27); and finally at line 8Ø, box (75, 27).

CONTAINERS IN THE COMPUTER'S MEMORY

It would be very difficult to store and retrieve information if we did not have a clear way of referring to the information we use. Thus we have chosen the term "container" to designate a particular location in the computer's erasable memory where information is stored. The letter "Y" is only one of many possible symbols we can use to "name" the containers.

A container's name can be thought of as the number part of the address of your home. If a friend mails a letter to you and the envelope bears only the name of the street you live on but not the specific number of your house, it is quite possible that your mail will be placed in the wrong box. Similary, if we feed into the computer information that is not "correctly addressed", we will have a hard time finding the information we need later on. Our ability to communicate with the computer will then be severely limited.

ONE LAST WORD ABOUT CONTAINERS

The SVI-728, like most personal computers, uses two different sets of containers. One set of containers holds only numbers, the second set holds both numbers and words.

A container whose name begins with the letters A-Z holds only numbers but when a dollar sign, "$", is placed after the letters it can hold both numbers and words. A container name can be up to 2 characters in length-it can't be a word of BASIC Language-for example, SV, DL, HF, L1, etc, are legal containers names. Type:

```
1Ø CLS
2Ø PRINT "WHATEVER YOU TYPE IN
WHEN THE QUESTION MARK
APPEARS, I WILL REPEAT"
3Ø INPUT A$
4Ø PRINT A$
5Ø END
```

Type RUN

Then type a number in response to the dollar sign, and watch the computer echo whatever you typed (remember to press [ENTER] when you finish your input).

Then type RUN

again and this time type in a word or two in response to the question mark (and press [ENTER]).

What happens when you mix both numbers and words in the same container? Is the mixture accepted? Try it by RUNning the program again.

You may store a maximum of 255 characters (letters) in any one container. Likewise, you can type a maximum of 255 characters on any one line of your BASIC program. Experiment to see what happens when you try typing more than 255 characters. RUN the program again, and this time type more than 255 characters after the question mark and then press [ENTER] . You should see that the extra characters were ignored.

CONTAINER STORAGE AND RECALL

There are various commands that tell the computer what information to put into a specific container. We have already seen that INPUT puts the information that you type into a container. Here is another way. Type,

```
1Ø CLS
2Ø LET A = 1Ø
3Ø LET B = 2Ø
4Ø LET C = A + B
5Ø PRINT C
6Ø END
```

Now type RUN

LET

The number 30 should have been printed out on the screen. This program illustrates how one assigns information to containers. The command LET prefixed the name of the container you wish to use and the container is placed on the left side of the equal (=) sign. The information you wish to place into the container is written on the right side of the equal (=) sign. After the number 10 has been placed in "A" and the number 20 has been placed in "B", the computer adds the numbers together and places the sum into container "C". Thus the computer's answer is found in container "C" and the contents of container "C" are printed on the screen.
You do not have to use the command LET to assign information to a container. The following program does exactly the same thing:

```
1Ø CLS
2Ø A = 1Ø
3Ø B = 2Ø
4Ø C = A + B
5Ø PRINT C
6Ø END
```

The use of LET generally helps to improve the readability of your program. This is helpful when someone else looks at it, or when you look at it after not reading it for a long period of time.

Now we'll show you another way to use containers . . . one that allows you to draw quickly.

The following techniques will not only help you to draw quickly – they will also prove to be invaluable in almost any program you write. Type and RUN the following program:

```
1Ø CLS
2Ø FOR X = Ø TO 64
3Ø PRINT X
4Ø NEXT X
5Ø END
```

Surprised at the result? You really should not be. This program uses the same kind of loop and container that we have been discussing.

FOR/NEXT

After line 1Ø clears the screen, line 2Ø instructs the computer to place all the numbers between Ø and 64 into the "X" container one at a time. The obedient computer begins by putting a Ø in the "X" container. The instruction to PRINT X on line 3Ø forces the computer to look up the number stored in the "X" container and display it on the TV. Then on line 4Ø the computer is instructed to take the next number after Ø and place that in the "X" container. Since a container can hold only one item at a time, the Ø is erased from the "X" location in the computer's erasable memory and the number 1 is now placed in the "X" container. Then the computer is sent back to line 2Ø to begin tne process again.

Since the number 1 is one of the numbers designated to be placed in "X", the computer proceeds to line 3Ø where it finds the number 1 stored in "X", and proceeds to PRINT the number 1 on the screen. Line 4Ø causes the number 2 in the container and returns the computer to line 2Ø where the same events occur. This loop continues until the number 64 has been placed in "X" and has been displayed on the screen. Since 65 is not one of the designated numbers on line 2Ø, the computer sees the END command on line 5Ø and stops.

The FOR-NEXT commands are always a pair. Never use one without the other. The FOR-NEXT loop is fundamentally different from the loop we can create with the GOTO command. As your programming skills develop, you will learn when the use of each is appropriate.

Do you think you can change the program which printed out the numbers between Ø-255 to draw a straight line across the whole screen? Before you read how it's done, try it yourself. Hint: Two lines have to be changed and lines added.

```
1Ø SCREEN 3
2Ø FOR X = Ø TO 255
3Ø PSET (X,25)
4Ø NEXT X
5Ø END
```

Type RUN and watch how effortlessly the computer draws that line!

The computer is told on line 2Ø to put all the numbers between Ø-255 into "X" container one at a time. A Ø is placed in "X" in line 2Ø. And when the instruction to turn on the box named (X,25) is given in line 3Ø, the computer checks to see what number is stored in "X" and substitutes the Ø for the container name in the name of the box, so the first box lit is box (Ø,25).

NEXT X on line 4Ø places the number 1 in "X". Since 1 is a designated number, the second box the computer lights up is box (1,25). The number one stored in "X" is substituted for the "X" in line 4Ø. This loop continues until all 255 boxes in row 25 have been lit.

Now try changing the loop program which draws a horiztonal line on row 25 to one that will draw a vertical line at column 3Ø from the top of the screen. Your program should look like this:

```
1Ø CLS
2Ø SCREEN 3
3Ø FOR D = Ø TO 191
4Ø PSET (3Ø,D)
5Ø NEXT D
6Ø END
```

Notice that we use a different container name. "D" is the designated container that will hold all the numbers between Ø-191. This looping program works exactly like the program above, that draws the horizontal line.

We have spent much time explaining to you how containers work in a non-technical way. This was important because the more you know about how the computer works and the precise cause-and-effect of every command in BASIC, the better use you can make of the SVI-728 features.

SOME TIME SAVING HINTS

We have seen how loops can save you time. Another little time saving tip is this. You need not always place the END command at the end of every BASIC program. Although it is good practice to do so, you will quickly learn where END is necessary.

```
1Ø CLS
2Ø PRINT "I LOVE MY SVI-728"
3Ø GOTO 2Ø
```

END

The END command is not needed at the end of this program because the way the program is structured the computer will never pass line 3Ø. It will be caught in a continuous loop until you press the CTRL + STOP keys combination.

Similarly, you need not always precede your program with

```
1Ø CLS
```

You should decide when you have to work with a clear screen and when you don't.

When using the SCREEN command you do not need to use the CLS command because the SCREEN command automatically clears the screen.

SETTING THINGS IN MOTION

7

SETTING THINGS IN MOTION

Now that you know how to use containers to draw a line, let's add one line to the program you did in the previous chapter. This addition will cause a dot to move across the screen.

```
1Ø SCREEN 3
2Ø FOR X = Ø TO 255
3Ø PSET (X,25),11
4Ø PRESET (X,25)
5Ø NEXT X
```

RUN

Was that a bit too quick for you? Before we show you how to slow the bouncing ball down, let's review this program to learn how we create the illusion of motion.

PRESET

The new command PRESET on line 4Ø, acts just the opposite of PSET. PSET turns boxes on and PRESET turns boxes off. In our program above, PRESET turns off the very same box that PSET turned on. In line 3Ø we told PSET with what color to light the boxes. In line 4Ø we don't have to tell PRESET a color number. PRESET merely turns the box off to the background color of the screen.

How can we slow our bouncing ball down? Well, it makes sense to somehow slow down the computer so it will be delayed between line 3Ø and line 4Ø, because the slower the boxes get turned off, the longer they will stay on and the easier it will be for an observer to follow the path of the ball. We will add what is known in computer jargon as a time delay between line 3Ø and line 4Ø.

Add the following lines:

```
35 FOR T = Ø TO 5Ø
37 NEXT T
```

Now type LIST

and your program should look like this:

```
1Ø SCREEN 3
2Ø FOR X = Ø TO 255
3Ø PSET (X,25), 11
35 FOR T = Ø TO 5Ø
37 NEXT T
4Ø PRESET (X,25)
5Ø NEXT X
```

Now type RUN.

Your program should now be considerably easier to your eyes because the dot will move more slowly across the screen. Lines 35 and 37 should look familiar to you. They are an example of your typical FOR-NEXT loop, with an important difference.

In the previous chapter we stuck a command (such as PSET or PRINT) between the FOR X and the NEXT X commands. In the bouncing ball program we did no such thing. Rather, by placing the FOR T and the NEXT T commands on consecutive lines, we caused the computer to wait a certain amount of time before proceeding to line 4Ø. Lines 35 and 37 caused the computer to stop and count from Ø to 5Ø. The computer merely placed the numbers between Ø and 5Ø into the container called "T" one after the other, and that took up time. Hence, this use of the FOR-NEXT loop is called a "time delay".

To speed up the moving dot, simply change the last number you want to be placed in the "T" container to a number less than 50. For example,

```
35 FOR T = Ø TO 25
37 NEXT T
```

To slow down the bouncing ball, simply increase the last number to be placed in the T container to a number greater than 5Ø. For example,

```
35 FOR T = Ø TO 1ØØ
37 NEXT T
```

WALKING BACKWARD

Now that you have mastered how to make the ball move forward, from left to right, it is time to make the ball move backward, from right to left. Add the following lines to the bouncing ball program:

```
6Ø FOR Q = 255 TO Ø STEP-1
7Ø PSET (Q,25), 11
8Ø FOR F = Ø TO 1Ø
9Ø NEXT F
1ØØ PRESET (Q,25)
11Ø NEXT Q
```

STEP

The new addition to the revised bouncing ball program is the line 6Ø. You are probably wondering where the STEP – 1 came from. Here's why.

The instruction

```
FOR X = Ø TO 255
```

means that the computer is instructed to place all the numbers between Ø and 256 into the "X" container one at a time. The "one at a time" instruction need not be written explicitly.

```
FOR Q = Ø  TO 255
```

is equivalent to

```
FOR Q = Ø  TO 255 STEP 1
```

Which means move from Ø to the next number, one at a time. When you do not specify how many numbers to STEP to, the computer automatically assumes you mean to advance by one number at a time. We could just as easily have instructed the computer to jump from Ø to 2 with the instruction,

```
FOR Q = Ø TO 255 STEP 2
```

Therefore, line 6Ø,
FOR Q = 255 TO Ø STEP–1

is interpreted by the computer to mean placing all the numbers between 255 and Ø into container "Q" one number at a time in descending order. That is how we get the ball to move backward.

By deleting just one line you can see a drastically different result. The present program bounces the ball back and forth across the screen. Try it. Delete line 1ØØ by typing.

```
1ØØ ENTER
```

Now type LIST.

To prove to yourself that line 1ØØ was erased, type:

```
RUN
```

When you deleted line 1ØØ, the boxes that are lit up by line 7Ø are no longer turned off. Now change the program by adding one line which will make it get stuck in a continuous loop. Try it before you read on.

You should have added,

```
12Ø GOTO 2Ø
```

This addition tells the computer to return to line 2Ø and start drawing the ball and the line again. It looks like a little yo-yo.

FIRST CHALLENGING PROGRAM

Using all the materials you have learned up to this point, try writing a program that will create a picture in which a ball tumbles down a staircase. This program is tricky, but remember we have placed it here to really challenge you. It should be easier to tackle if you first instruct the computer to draw a staircase and then concentrate on making a ball move down the stairs. The finished product should resemble the picture in Figure 5.

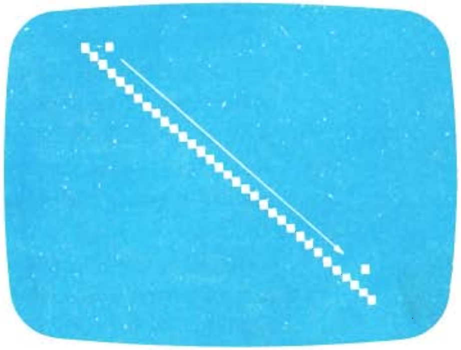


Figure 5

Our program to accomplish this task is listed below.

```
  5  REM THIS SECTION OF THE
     PROGRAM DRAWS A
     STAIRCASE
 1Ø  SCREEN 3
 2Ø  FOR X = Ø TO 191
 3Ø  PSET (X,Y), 6
 4Ø  Y = Y + 1
 45  NEXT
 5Ø  REM THIS SECTION OF THE
     PROGRAM DRAWS THE BALL
     TUMBLING DOWN THE STAIRS
 6Ø  Y = Ø
 7Ø  FOR X = 4 TO 191
 8Ø  PSET (X,Y), 15
1ØØ  FOR T = Ø TO 1ØØ
1Ø5  NEXT T
11Ø  PRESET (X,Y)
12Ø  Y = Y + 1
125  NEXT X
135  CLS
14Ø  Y = Ø
15Ø  GOTO 2Ø
```

REM

The above program does exactly what we had set out to do. The only new command introduced here is REM. REM is the abbreviated form of remark. When REM appears at the beginning of a program line, (in this program, lines 5 and 55), the computer understands that it should ignore whatever follows the REM command on the same line. REM statements are comments which are used to increase the readability of a program.

Lines 4Ø and 12Ø use containers in a way that is familiar to people who remember basic algebra. The instruction "Y = Y + 1" tells the computer to add 1 to the number that is stored in the "Y" container and then place the new number back in the "Y" container.

If you were not able to complete the staircase program, do not be discouraged. As mentioned at the beginning of this guide, we differ from other tutorial texts in many ways. In addition to offering in-depth, non-technical explanations of the BASIC commands we also supply you with challenging exercises.

8 DECISION MAKING

Up until now we have been introducing you to the BASIC language through graphics. Now we will begin to explore many other BASIC commands in different settings.

IF/THEN

In order to program the computer to make decisions, we use the instruction known as the IF-THEN statement. (Note: this is the first time we have used the word "program" as a verb. Previously we used this word as a noun, but the verb for creating a program is programming and we will start using this official term.) Let's see how it works. Enter (this is the verb we will now be using alternatively for the word "type") the following program:

```
 1Ø CLS
 2Ø PRINT "TRY TO GUESS THE
    NUMBER THAT I AM THINKING
    OF. I'LL GIVE YOU A HINT. IT
    IS BETWEEN Ø AND 1ØØ."
 3Ø A = 5Ø
 4Ø INPUT X
 5Ø IF X = A THEN GOTO 1ØØ
 6Ø PRINT "SORRY, YOU DID NOT
    GUESS IT. TRY AGAIN."
 7Ø GOTO 4Ø
1ØØ PRINT "GREAT GUESS. YOU ARE
    CORRECT, THE NUMBER I WAS
    THINKING OF IS 5Ø."
```

Now RUN the program. The INPUT command in line 4Ø will cause a question mark to appear on the screen. Enter your guess after the question mark. Line 5Ø of the program compares your guess that is stored in container "X" with the number that is stored in container "A". If you guessed the number 5Ø then the computer will jump to line 1ØØ. If you did not guess the number 5Ø, and container "X" holds any other number, the computer automatically continues to line 6Ø. There, it informs you of the result and sends you back to line 4Ø which places another question mark on the screen and waits patiently for your next guess. If you entered 5Ø as your guess, then the test on line 5Ø sends you to line 1ØØ which prints out the appropriate message. Let's take a closer look at the IF-THEN statement. Like the FOR-NEXT loop, the IF-THEN command is used very frequently.

The IF-THEN statement performs a test. In our example the command meant: if X = A was true then the computer must jump to line 1ØØ. If the test proved to be false (the number stored in "X" did not equal the number stored in "A"), then no action was to be taken (e.g. do not jump to line 1ØØ). Instead the computer just continues with the next line in order.

STRING VARIABLE

In chapter 6 we mentioned that your computer uses 2 different types of containers: One kind stores only numbers and is called a "Numeric Variable," while the other store both letters (words) and/or numbers as long as they are placed between double quotation marks. This variable is called a "String Variable" (Because a word is a group of characters that are strung together).

To differentiate between the two types of variables a dollar sign ($) is placed after the name you choose for the variable.

```
Example: A$ = "VARIABLE"
         B$ = "2 Variables"
         C$ = "1234"
```

We will now challenge you to write a program that requires the use of a String Variable.

ANOTHER CHALLENGING PROGRAM

Listed below is the English description of a program that you should try writing and running. The three steps below describe the three major components of this program. The first two steps can each be translated into one line of BASIC command that the computer understands. Step 3 will require several lines of BASIC commands to ensure that the computer does what is described. The program you write should behave as described in the three steps.

STEP 1: The computer asks you (by displaying the message on the TV) whether it should list all the numbers between Ø and 1ØØ for you on the screen.

STEP 2: The computer waits for your answer and stores it in the type of container that holds words.

STEP 3: The computer checks your response. If you type in the word "YES" it will proceed to print the numbers between Ø and 1ØØ and if you type "NO" the computer will say "GOODBYE" to you rather than print the numbers.

Try writing and running this program. Remember that very few people ever get a program written 1ØØ% correct the first time. Part of the beauty of programming is that it continually allows you to learn from your mistakes. We provided several hints in the description of the three steps. Read the description carefully before you begin. Good

luck. Just in case you get stuck, we wrote our program to meet the requirements of steps 1-3.

```
  5  CLS
 1Ø  PRINT "DO YOU WANT ME TO
     LIST ALL THE NUMBERS
     BETWEEN Ø AND 1ØØ? TELL
     ME YES OR NO."
 2Ø  INPUT A$
 3Ø  IF A$ = "NO" THEN GOTO 1ØØ
 4Ø  FOR I = Ø TO 1ØØ
 5Ø  PRINT I
 6Ø  NEXT I
 7Ø  END
1ØØ  PRINT "GOODBYE"
```

We wrote END on line 7Ø so that if you type "YES" and the computer lists all the numbers between Ø and 1ØØ, it will not continue onto line 1ØØ. The END serves as a barrier to stop the computer from continuing to line 1ØØ.

What happens if you enter a word other than "yes" or "no" in response to the computer's question?
Type RUN and find out.

If you just tried this experiment you should have found out that the computer lists all the numbers from Ø-1ØØ anyway, because as it stands now the program only checks for the presence of the word "no" in the A$ container. If any other word is in there, the computer still continues to line 4Ø. Try to repair this program. Here is what we suggest.

Add,

```
35 IF A$ = "YES" THEN GOTO 4Ø
37 PRINT "YOU DID NOT TELL ME
     YES OR NO, TRY AGAIN."
38 GOTO 2Ø
```

If you type LIST, your revised program should resemble ours.

```
  5 CLS
 1Ø PRINT "DO YOU WANT ME TO
    LIST ALL THE NUMBERS
    BETWEEN Ø AND 1ØØ? TELL ME
    YES OR NO."
 2Ø INPUT A$
 3Ø IF A$ = "NO" THEN GOTO 1ØØ
 35 IF A$ = "YES" THEN GOTO 4Ø
 37 PRINT "YOU DID NOT TELL ME
    YES OR NO, TRY AGAIN."
 38 GOTO 2Ø
 4Ø FOR I = Ø TO 1ØØ
 5Ø PRINT I
 6Ø NEXT I
 7Ø END
1ØØ PRINT "GOODBYE"
```

Now, if you type in any words other than "YES" or "NO", the computer will tell you that what you typed is not what it expected and will let you keep on trying until you type either "YES" or "NO".

MORE DECISIONS

Your computer is capable of making other kinds of decisions, beyond simple IF-THEN tests. Here is another one. Enter the following program:

```
10 CLS
20 REM THIS PROGRAM WILL
   GATHER INFORMATION ABOUT
   FAMILIES
30 PRINT "DO YOU HAVE A
   BROTHER? TYPE YES OR NO."
40 INPUT A$
50 PRINT "DO YOU HAVE A SISTER?
   TYPE YES OR NO."
60 INPUT B$
70 IF A$ = "YES" AND B$ = "YES"
   THEN PRINT "THEN THERE ARE
   AT LEAST THREE CHILDREN IN
   YOUR FAMILY."
80 IF A $ = "YES" OR B$ = "YES"
   THEN PRINT "THERE ARE AT
   LEAST TWO CHILDREN IN YOUR
   FAMILY."
90 IF A$ < > "YES" AND B$ = "YES"
   THEN PRINT "AREN'T YOU
   LUCKY THAT YOU DO NOT HAVE
   ANY BROTHERS."
100 IF A$ = "YES" AND B$ < > "YES"
   THEN PRINT "AREN'T YOU
   LUCKY THAT YOU DO NOT HAVE
   ANY SISTERS."
110 IF A$ < > "YES" AND B$ < >
   "YES" THEN PRINT "YOU ARE
   AN ONLY CHILD."
120 END
```

Now RUN

and answer the questions. If you typed the program in Capital Letters make sure that when you respond to the question mark sign you type the words "YES" or "NO" in capital letters. If you typed the program in Lower case your "yes" or "no" response should also be typed in Lower case letters. Then RUN the program and again type in different responses to see the results.

The program is very straightforward. The new commands AND, OR, < > (not equal) are used in BASIC the same way they are used in everyday speech.

PRINT IT THE WAY YOU WANT

We can have the PRINT command place information on different parts of the screen by adding a comma or semicolon to the word PRINT. Enter the following program.

```
10 FOR I = Ø TO 1ØØ
20 PRINT "HELLO"
30 NEXT I
```

Type RUN. You should recognize this form of the PRINT statement. This is what we introduced to you a few chapters ago. Now change line 2Ø to read

```
10 FOR I = Ø TO 1ØØ
20 PRINT "HELLO",
30 NEXT I
```

Now type RUN again and compare the results you get with those of the previous program.

Now change line 2Ø again as follows;

```
10 FOR I = Ø TO 1ØØ
20 PRINT "HELLO";
30 NEXT I
```

There are three very distinct ways that output can be displayed. Later on, we will apply these different PRINT styles in our programs.

TAB

By using the TAB command, you can tell the computer where you want the information to be put. Enter the following lines in immediate mode (without a line number).

```
PRINT TAB (2Ø); "HELLO"
```

and now enter,

```
PRINT TAB (3Ø); "GOODBYE"
```

TAB starts the printing of the message at the column of text specified in the parentheses. The left most column of the text screen is Ø, and the right most column is 39.

9 SOME "RANDOM" THOUGHTS

Many games that people play involve an element of chance, from board games like monopoly to the game of craps played in casinos. The excitement and the high risk involved in these games occurs by rolling disc and getting a random, or unexpected number.

It is very easy for us to roll the dice and receive a random number. A random number is one that occurs as if you placed your hand in a barrel full of numbers and picked one out. Unless you were a prophet, you wouldn't know what number to expect, and would be surprised at the result.

Random numbers are important in computer programs especially in game or quiz programs, as well as some mathematic and scientific applications.

While it is easy for us to roll random numbers with dice, it is not so easy for your computer to simulate dice and pick a truly random number. Many microcomputers don't even afford you the opportunity to easily spin a truly random number. When you tell these other machines to pick a random number they will produce a fake random number. Their numbers are not truly random because each time they start picking a number they always begin with the same number. It's as if the barrel that they picked from only had one number in it.

However, your SVI-728 can pick truly random numbers. To do so, always insert the following line at the beginning of a program that calls for random numbers.

```
5 N = RND ( – TIME)
```

RND, INT and–TIME

The new commands that are introduced are INT, RND and - TIME.

To pick a truly random number the SVI-728 takes a look at its internal clock, which measures the passage of time in microseconds. Each time that the computer receives the instruction – TIME, it looks at its internal clock and uses the time on the clock as the truly random number from which to pick more random numbers. Since time never stands still, each time the computer checks its clock it will report a truly random number.

Type the following short program to see how to use line 5 in a program and what the INT and RND instructions do.

```
1Ø REM THIS PROGRAM PICKS
     A RANDOM NUMBER
2Ø N = RND ( – TIME)
3Ø X = INT (RND (1) * 1Ø)
4Ø PRINT X
```

The new line here is line 3Ø. The first instruction that the computer responds to on this line is RND (1). The number one in parentheses is called a dummy variable (container). Any number could have been used in the parentheses. In each case the computer picks a truly random number that is between zero and one, for example, .2345. The next instruction that the computer performs on line 3Ø is multiplying the decimal number (.2345) by 1Ø. Since we want a whole number (one that doesn't have any numbers to the right of the decimal point) we then use the INT instruction to chop off the numbers to the right of the decimal point leaving us with X = 2.

We can now present the program that simulates the spinning of dice to achieve truly random numbers.

```
1Ø R EMTHIS PROGRAM SIMULATES
     THE ROLLING OF DICE
2Ø N = RND ( – TIME)
3Ø X = INT (RND (1) * 6 + 1)
35 IF X = 7 THEN GOTO 3Ø
4Ø Y = INT (RND (1) * 6 + 1)
45 IF Y = 7 THEN GOTO 4Ø
5Ø R = X + Y
6Ø PRINT R
```

Lines 3Ø and 4Ø pick the random numbers. Let's take an example. If the instruction RND (1) picks the number.9678, we then multiply this number by 6 and get 5.8Ø68. The computer then adds 1 to 5.8Ø68 and gets 6.8Ø68. Finally it takes the INT of 6.8Ø68 which is 6. So "X" dice spun a 6. The computer then repeats this process to get the number of the "Y" dice and then adds the "X" and "Y" die together and places the total in container "R" which is then printed. In all honesty, the explanation of random numbers is a little complicated and we suggest that you review the material a second time if you are not quite sure how the RND function works.

JUMPING AROUND

The command GOTO was shown to cause the computer to jump from one line to another. Right now we will show you another way of jumping around in your program and then explain the difference between these two methods.

Enter the following program:

```
 1Ø REM THIS PROGRAM
    CONVERTS YEARS TO MONTHS
 2Ø CLS
 3Ø PRINT "HOW MANY YEARS OLD
 4Ø INPUT N
 5Ø GOSUB 2ØØ
 6Ø PRINT "HOW MANY YEARS OLD
    IS YOUR FATHER?"
 7Ø INPUT N
 8Ø GOSUB 2ØØ
 9Ø PRINT "HOW MANY YEARS OLD
    IS YOUR MOTHER?"
1ØØ INPUT N
11Ø GOSUB 2ØØ
12Ø END
2ØØ PRINT N; "YEARS IS EQUAL TO";
    N * 12; "MONTHS"
21Ø RETURN
```

RUN the program and answer the questions. Do you understand how this program works? Let's review it.

GOSUB/ RETURN

After asking you the question on line 3Ø and accepting your answer on line 4Ø, the computer encounters GOSUB 2ØØ, which means GOSUB subroutine 2ØØ. A subroutine is a section of a program, either one line or several lines, that is referred to frequently by the main part of the program. Therefore, it is called a SUBroutine, since it is subordinate, or secondary, to the bulk of the program.

When the computer sees GOSUB 2ØØ on line 5Ø, the computer jumps to line 2ØØ. When the computer completes the conversion of years to months on line 2ØØ and prints it on the screen, the program continues to line 21Ø. There, the command RETURN sends the computer back to the line that follows the one which contained the GOSUB 2ØØ. In our program above, that would RETURN the computer to line 6Ø.

The same thing happens when GOSUB 2ØØ is encountered on line 8Ø and line 11Ø. That is, the computer RETURNs to line 9Ø and 12Ø respectively.

What is the difference between the GOTO command and the GOSUB-RETURN command? If we had used the command GOTO 2ØØ instead of GOSUB 2ØØ, then line 21Ø would have had to say GOTO 6Ø to have the computer continue from where it stopped when it jumped to 2ØØ. But line 21Ø would have also had to say GOTO 9Ø and GOTO 12Ø if we had used GOTO 2ØØ on lines 8Ø and 11Ø. But the computer could not have understood a line 21Ø that would have said,

```
21Ø GOTO 6Ø, GOTO 9Ø, GOTO 12Ø
```

This problem stems from the fact that when the computer listens to a GOTO command it does not remember the line that it stopped at before it jumped to the specified GOTO line number. On the other hand, the beauty of the GOSUB-RETURN command is that it remembers precisely the point from which it jumped. When it is finished with the subroutine and sees the command RETURN, it knows exactly where to return to without being told again.

The next program employs another command that is a variation of the GOSUB-RETURN statement. Enter,

```
 1Ø REM THIS PROGRAM
    DEMONSTRATES A SIMPLE
    TELEPHONE BOOK HELPER
 3Ø PRINT "IF YOU WANT TO FIND A
    PHONE NUMBER QUICKLY, JUST
    PICK ONE OF THE CHOICES
    BELOW AND I WILL DO THE
    REST."
 4Ø PRINT "1. DOCTOR"
 5Ø PRINT "2. POLICE"
 6Ø PRINT "3. SPECTRAVIDEO"
 7Ø INPUT N
 8Ø ON N GOSUB 1ØØ, 2ØØ, 3ØØ
 9Ø GOTO 1Ø
1ØØ PRINT "THE DOCTOR'S NUMBER
    IS 555-1234"
11Ø RETURN
2ØØ PRINT "THE POLICE'S NUMBER
    IS 555-1245"
21Ø RETURN
3ØØ PRINT "SPECTRAVIDEO'S
    NUMBER IS 555-1256"
31Ø RETURN
```

RUN the program and try it out. The variation of GOSUB-RETURN is on line 8Ø. The command ON N GOSUB 1ØØ, 2ØØ, 3ØØ means that if "N" (the container that is holding the number you have chosen) is the number 1, then the computer will jump to the subroutine on line 1ØØ. If "N" is holding the number 2, then the program jumps to the subroutine on line 2ØØ, and if "N" is holding the number 3, it jumps to the subroutine on line 3ØØ.

THE COMPUTER AS A CALCULATOR

You can make the computer function as a calculator.

For addition enter

PRINT 6 + 3

and the computer will output

9

For subtraction enter

PRINT 6 –3

and the computer will output

3

For multiplication enter

PRINT 6 * 3

and the computer will output

18

For division enter

PRINT 6/3

and the computer will output

2

If you forget to use the word PRINT when you use the computer as a calculator, the computer will not output the answer. The computer will have calculated the answer and placed it into a container in its memory automatically, but it will not inform you of its answer.

ARITHMETIC OPERATIONS

Most of you probably remember the basic rules of arithmetic from your elementary school days, so we will not dwell upon these types of calculations any further.

The following table lists the order of precedence for the mathematical and logical commands that your computer will calculate. That means if any of these signs or words are used in your programs, the order listed here is the order in which they will be executed

1. ( )
2. NOT
3. * /
4. + –
5. > < = >= <= <>
6. AND
7. OR

If there is more than one operation listed on any line of BASIC commands, and these operations are both listed on the same level in the above table, then the operation closer to the left side of the screen will be performed first. For example, if you entered

PRINT 3 + 2 – 1

the computer will output,
4
because it first added 3 + 2 and then did 5 – 1.
Similarly, if you entered,
PRINT 3 – 2 + 1

the computer will output,
2
because it first perform 3 – 2 and then did
1 + 1.

PROCESSING INFORMATION

At the beginning of this tutorial we described a computer program as a set of instructions that processed (acted on) information. The information of the program needs to begin processing is called the **input** to the program and the result of the processing is called the **output.**

The following program demonstrates how to tell the computer the information you want it to process. Most of the commands in BASIC, like PRINT, FOR-NEXT, IF-THEN, PSET, etc. tell the computer what to do with information. We have already shown you some words that tell the computer what information to work on commands like INPUT and LET X = 20. Now we will introduce you to a new set of words, READ and DATA, which as the names imply, tell the computer that what follows is the information to be processed.

Enter the following program.

```
 5 CLS
1Ø REM THE FOLLOWING PROGRAM
   WILL READ AND PRINT THE
   NAMES OF THE FIRST FIVE
   MONTHS OF THE YEAR
2Ø FOR X = 1 TO 5
3Ø READ F$
4Ø PRINT F$
5Ø NEXT X
6Ø DATA JANUARY, FEBRUARY,
   MARCH, APRIL, MAY
```

Type RUN.

Your program should have output the names of the first five months of the year. Now let's take a closer look and see how the program works.

READ/DATA

Line 2Ø prepares a loop that will do something 5 times. That something is told to the computer in line 3Ø, where we instruct the computer to READ some information and place it in container F$. When the computer comes across the READ command, it immediately searches the whole program for the first line that begins with the command DATA. Since DATA appears on line 6Ø, the computer READs the first word, "January " and places it in container F$. Then the program continues to line 4Ø and PRINTs the word in F$ (January). NEXT X on line 5Ø sends the computer back to line 2Ø to begin this process of READing and PRINTing again.

The second time around, the computer looks again for the DATA statement that tells the computer that on this line it will find the information it needs to READ. So the computer goes to the information listed on the DATA line after the place where it previously stopped and stores in F$. When the computer puts the word "February" into container F$, it first erases the word "January", which was previously stored in F$. So when the instruction PRINT F$ is encountered on line 40, the only word in F$ is "February" and so it is PRINTed on the TV.

The line that begins with the command DATA and contains the information we want the computer to READ, can be placed anywhere in the program. The READ command searches the whole program, starting at the beginning, for the first line in which the command DATA appears. Therefore, our READ/DATA program could have been written in the following way:

```
 5 CLS
1Ø REM THE FOLLOWING PROGRAM
   WILL READ AND PRINT THE
   NAMES OF THE FIRST FIVE
   MONTHS OF THE YEAR.
15 DATA JANUARY, FEBRUARY,
   MARCH, APRIL, MAY
2Ø FOR X =1 TO 5
3Ø READ F$
4Ø PRINT F$
5Ø NEXT X
```

OR

```
 2 DATA JANUARY, FEBRUARY,
   MARCH, APRIL, MAY
 5 CLS
1Ø REM THE FOLLOWING PROGRAM
   WILL READ AND PRINT THE
   NAMES OF THE FIRST FIVE
   MONTHS OF THE YEAR
2Ø FOR X = 1 TO 5
3Ø READ F$
4Ø PRINT F$
5Ø NEXT X
```

The information that is placed on the DATA line is separated by commas. The words or numbers between commas are treated as one piece of information. Enter and RUN the following program to see what we mean.

```
1Ø CLS
2Ø REM THIS PROGRAM READS
   AND PRINTS OUT THE NAMES
   OF FIVE STATES
3Ø FOR X = 1 TO 5
4Ø READ F$
5Ø PRINT F$
6Ø NEXT X
7Ø DATA NEW YORK, NEW JERSEY,
   NORTH DAKOTA, NEW
   HAMPSHIRE, NEW MEXICO
```

How does the following program, which reads and prints the name of 5 states, differ from the program that did the same thing above?

```
1Ø CLS
2Ø FOR X = 1 TO 5
3Ø READ F$: READ G$
4Ø PRINT F$, G$
5Ø NEXT X
6Ø DATA NEW, YORK, NEW, JERSEY,
   NORTH, DAKOTA, NEW,
   HAMPSHIRE, NEW, MEXICO
```

You should be able to tell the difference from just comparing the two programs without even having to enter the second version and running it.

The first program, which read and printed the names of 5 states, did so by reading each set of two words into one container, F$. In the second program, the name of each state was broken into two parts. The first half of the name was stored in container F$ and the second half of the name was stored in container G$.

OUT OF DATA message

Enter the above revised program into the two containers F$ and G$, if you have not yet done so. Run the program. Now add line 7Ø:
7Ø GOTO 2Ø
And RUN the program again. Did you get OUT OF DATA message? That error message resulted form the fact that you tried to READ information again but since the computer finished READing the list after you ran the program the first time, the computer did not find any more information on the DATA line after the name MEXICO.
The way to fix that problem is to tell the computer to return to the beginning of the DATA before it is time to read the names again. The command you use to do this is RESTORE. Here is how the repaired program looks.

```
1Ø CLS
2Ø CLEAR 5ØØ
3Ø FOR X = 1 TO 5
4Ø READ F$: READ G$
5Ø PRINT F$, G$
6Ø NEXT X
7Ø RESTORE
8Ø GOTO 3Ø
9Ø DATA NEW, YORK, NEW, JERSEY,
   NORTH, DAKOTA, NEW,
   HAMPSHIRE, NEW, MEXICO
```

CLEAR

Here is how this program works. We have added another command called CLEAR on line 2Ø. Whenever you use a READ-DATA command you should also use CLEAR, which CLEARs away a lot of room (e.g., 5ØØ spaces) in the computer's memory. Then you can

work with the containers that store words without running out of space, because the containers that store words and text generally take up much more room than the containers that store only numbers.

RESTORE

Then on line 7Ø, the RESTORE command instructs the computer to return to the beginning of the list of information on the DATA line before the GOTO 3Ø command on line 8Ø sends the computer back to the beginning of the loop on line 3Ø. The program should print out the names of the states without receiving an OUT OF DATA message.

ANOTHER TIME SAVER

Our programs are beginning to get larger. That is we are writing more lines of instructions in each program. Some people like to tell the computer to automatically write the line number for each line of the program. This is accomplished by typing:

AUTO ENTER

AUTO

In immediate mode, the computer will respond with the number 1Ø and it will wait for you to begin typing on line 1Ø. When you have finished line 1Ø and pressed ENTER the computer will automatically print line 2Ø and wait for you to continue.

You can make the computer start the automatic line numbers from a number other than 1Ø, and it can increase the number for the next line by more than or less than 1Ø. In other words, the AUTO command can be told what line number to start with and the increase for each subsequent line.
For example:
AUTO 2Ø, 4Ø
This will start the first line number at 2Ø and cause the second line number to be 6Ø and the third line number to be 1ØØ.

As with all the commands we have introduced to you in this tutorial guide, it is necessary for you to read the material in other parts of this manual to get a full understanding of each BASIC command.

10 ARRAYS-A WAY OF ORGANIZING MANY CONTAINERS

We hope that you are beginning to feel comfortable with the concept of containers and how your computer uses them to store and manipulate information. Previously, we have always used a container as an individual unit. That is, we have thought of each container as a distinct unit. "A", "B", "C$" and "D$" are examples of individual containers into which we put numbers and words. However, we did not have the need to group several individual containers into one set.

There are times when it is necessary to arrange a series of containers into a larger unit. If you wanted to keep track of 5 test scores for each of the 25 students in your class, it would make sense to organize the containers that hold the scores into a larger unit. Then you work with 25 sets of containers, rather than with 125 individual containers. (Each set would be a table with 5 lines of information.) Here is an example of what the table for a student named John might look like.

JOHN

(1) 75
(2) 82
(3) 94
(4) 68
(5) 1ØØ

Thus if we had before us 25 different tables representing the scores of all 25 students, we could then refer to John's first test score as John (1), his second test score as John (2) etc.

The following program demonstrates how we can instruct the computer to group John's 5 test scores into a table.

```
 1Ø REM THIS PROGRAM
     ORGANIZES JOHN'S TEST
     SCORES INTO A TABLE
 2Ø CLS
 3Ø DIM J (5)
 4Ø FOR X = 1 TO 5
 5Ø READ J (X)
 6Ø NEXT X
 7Ø PRINT "WHICH TEST GRADE DO
     YOU WANT?"
 8Ø INPUT Z
 9Ø PRINT J (Z)
1ØØ DATA 75, 98, 94, 68, 1ØØ
```

Enter this program and RUN it. This is how it works.

DIM

Line 3Ø DIMensions a block of five containers for the set of containers called "J". That means the computer is instructed to set aside enough room to store 5 numbers in 5 different containers. "J" is the name for this group of containers. On line 5Ø the computer READs in one test score at a time from the data line and places it into one container.

When the program reaches line 7Ø, it has already placed all 5 test grades into 5 different containers. This information is organized in the computer in a way that is similar to the table of John's test scores. So when we INPUT the test score that we want in line 8Ø, the computer is told on line 9Ø to print that particular line of the table.

Oh, we almost forgot to tell you: The offical computer term for the set of containers that we just worked with is an **"array".**

WORKING WITH A LARGER SET OF CONTAINERS

It is a little trickier to work with a large set of containers that can store two items each than it is to work with the simple container we demonstrated above. If you wished to store the scores of 5 students for one test, you would need to have one set of containers that stores their names, and one that stores their respective grades. The table that represents this information looks like this:

TEST #1

(1) JOHN	75
(2) BOB	81
(3) JOAN	65
(4) HARRY	96
(5) LINDA	98

Figure 7

The program that will READ and PRINT out a table just like the one in Figure 7 is listed below. Enter the program, RUN it and try to understand it before reading our explanation.

```
 1Ø REM THIS PROGRAM
    ORGANIZES NAMES AND
    SCORES FOR ONE TEST
 2Ø CLS
 3Ø DIM V$ (5,2)
 4Ø FOR N = 1 TO 5
 5Ø FOR S = 1 TO 2
 6Ø READ V$ (N,S)
 7Ø NEXT S: NEXT N
 8Ø REM THIS SECTION PRINTS OUT
    THE TABLE
 9Ø PRINT "NAME", "GRADE" :
    PRINT
1ØØ FOR N = 1 TO 5
11Ø PRINT V$ (N, 1), V$ (N, 2)
12Ø NEXT N
13Ø DATA JOHN, 75, BOB, 81,
    JOAN, 65, HARRY, 96, LINDA, 98
```

Don't get discouraged. This is the most difficult program you will see in this tutorial! Let's take a closer look at it.

Line 3Ø DIMensions or sets aside 5 containers that will store the number corresponding to the student's names listed in Figure 7, and the 2 containers that correspond to each name or number.

Lines 1ØØ-12Ø PRINT the table on the TV screen in the same form as appears in Figure 7.

The computerese term for this extended container is a **"double array"**,which is a fancy way of describing a two-column table.

Is there another way that we could have written this program? Yes, there is. Generally, almost all programs can be written in more than one way. Different people plan and program according to their own styles. Of course there are important differences among the various styles. You will learn to discern which option to choose as your knowledge increases with practice.

Here is another way you could have written the program that reads and prints the table in Figure 7. This time we use two separate sets of containers: One to store the names, and one to store the grades. Each one uses a non-column array, not the double column array that we just used.

```
 1Ø CLS
 2Ø DIM A$ (5), B (5)
 3Ø FOR I = 1 TO 5
 4Ø READ A$ (I), B (I)
 5Ø NEXT I
 6Ø PRINT "NAME", "GRADE": PRINT
 7Ø FOR K = 1 TO 5
 8Ø PRINT A$ (K), B (K)
 9Ø NEXT K
1ØØ DATA JOHN, 75, BOB, 81, JOAN,
    65, HARRY, 96, LINDA, 98
```

RUN the revised program to make sure that it does what it is supposed to.

ON THE END OF A LONG STRING

11 ON THE END OF A LONG STRING

STRING

Congratulations! You have made it this far and we trust you are doing well. We will now cover what is known in computerese as "String Manipulations". A string is a group of characters that usually is just a word. Accordingly, a word is considered to be a string of characters that can easily be changed and rearranged.

Enter the following program.

```
10 CLEAR 200
20 INPUT "TYPE IN A SENTENCE
    THAT IS NO MORE THAN 10
    WORDS LONG.";S$
30 PRINT "THIS IS THE NUMBER OF
    CHARACTERS IN THE SENTENCE
    THAT YOU JUST TYPED."
40 PRINT LEN (S$)
```

Now RUN the program. As its name implies, the LEN command calculates the number of characters in a string of characters. In this example, the whole sentence was stored in container S$.

INPUT

We hope you have noticed that line 20 looks a bit unusual. This is because we used the INPUT command to print the message on the screen without the help of the PRINT command, and placed the container name at the end of the line. This line is a very good example of a short cut in programming that combines several commands on one line.

LEFT$
RIGHT$
MID$

The following commands manipulate character strings in various ways; RIGHT$, LEFT$ and MID$. The programs below will introduce these commands and their use to you.

```
10 INPUT “TYPE A WORD THAT IS AT
   LEAST 6 LETTERS LONG”; W$
2Ø PRINT “THE FIRST LETTER IS:”;
   LEFT$ (W$,1)
3Ø PRINT “THE LAST TWO LETTERS
   ARE:”; RIGHT$ (W$,2)
4Ø GOTO 1Ø
```

The LEFT$ (W$,1) command on line 2Ø told the computer to print the first character to the left of the string of characters stored in container W$. If we had written LEFT$ (W$,3) then the computer would have printed the 3 characters closest to the left.

Similarly, the instruction RIGHT$ (W$,2) on line 3Ø causes the computer to print the 2 characters to the right, or the end of the string of characters to the right, or the end of the string of characters that is stored in container W$.

The MID$ is more powerful than the LEFT$ and RIGHT$ commands. Enter and run the following program.

```
1Ø W$ = “SPECTRAVIDEO”
2Ø PRINT MID$ (W$,4,3)
```

This MID$ command on line 2Ø started at the fourth character from the left of the word SPECTRAVIDEO (which was stored in container W$), and counted the next three characters and printed out these characters, i.e., CTR.

But the MID$ command can also serve a very useful function.

If you wanted to search through a sentence for a specific word, you could use MID$. It is demonstrated in the following program. Enter and run the program.

```
 1Ø CLEAR 5ØØ
 2Ø INPUT “TYPE A SENTENCE THAT
    IS NO MORE THAN 10 WORDS
    LONG”; S$
 3Ø INPUT “TYPE A WORD THAT
    APPEARS IN THE SENTENCE
    YOU ENTERED”; W$
 4Ø X = LEN (W$)
 5Ø FOR T = 1 TO LEN (S$)
 6Ø IF MID$ (S$,T,X) = W$ THEN
    GOTO 1ØØ
 7Ø NEXT T
 8Ø PRINT “THE WORD YOU
    SEARCHED FOR IS NOT IN THE
    SENTENCE”
 9Ø END
1ØØ PRINT W$ “APPEARS IN THE
    SENTENCE AT CHARACTER
    NUMBER ”;T
```

LEN

In this program, MID$ is used to test various sized groups of characters to see if they are identical to the word we are searching for. The program calculated the LENgth of W$, and placed the number of characters in W$, into the container called “X” (line 4Ø). Then the computer starts at the left position of the characters in W$ and counts the number of characters stored in “X”. If a match is found, you are told that the word has been found in your sentence.

ADVANCED GRAPHICS AND SOUND PROGRAMMING

12 ADVANCED GRAPHICS AND SOUND PROGRAMMING

The material in this chapter will demonstrate the advanced graphics and sound capability that is built into the SVI-728 which separate it from its competitors. This power is not available and accessable from BASIC in any other personal computer.

This chapter is separated into two parts. The first part expands on the introduction to graphics that you have received earlier. The second part explains the simple approach to sound programming using the PLAY command. A more complicated approach to sound programming is possible using the SOUND command which is explained in Appendix H.

PART ONE– ADVANCED GRAPHICS

CIRCLE

To begin exploring the graphics capability of the SVI-728, type in the following lines, pressing **ENTER** after each is completed:

```
1Ø SCREEN 2
2Ø CIRCLE (128,8Ø),6Ø,11
3Ø PAINT (128,8Ø),11
4Ø GOTO 3Ø
```

Now, RUN the program and you will see the yellow circle appear on the screen and then it will be filled in by the SVI-728's yellow paintbrush. To understand how this happens, let's look at each line individually.

```
1Ø SCREEN 2
```

This line causes the computer to display its graphic screen.

```
2Ø CIRCLE (128,8Ø),6Ø,11
```

Here, you are telling the computer to draw a circle around a center point that is 128 columns from the left side of the screen, 8Ø rows down from the top of the screen, with a radius (distance from the center of the circle) of 6Ø points and using the yellow (the number 11) outline.

PAINT

```
3Ø PAINT (128,8Ø),11
```

This line introduces you the **PAINT** command. This command tells the computer to use its "paint brush" to fill certain areas. In line 3Ø the computer is instructed to fill the circle you have just outlined in line 2Ø. In order to paint (fill) an object you must give the computer the coordinates numbers (in parentheses) which designate any point inside the object (as we just did giving the coordinates of the center point of our circle). If you had used coordinates which designate a point outside the object, the computer would have painted the whole background but not fill the object itself.
However, the fill color MUST be the same as the outline colour. In our case the number 11 is the same yellow color used for the circle outline from line 2Ø. The PAINT "recognizes" outlines of objects as borders only if their color matches the PAINT color. A different PAINT color will "ignore" the outlines and will paint (fill) the whole screen-covering the object.

```
4Ø GOTO 3Ø
```

The last line of this program causes the computer to repeat line 3Ø so the circle will not disappear. To stop the program press the **CTRL-STOP** keys combination.

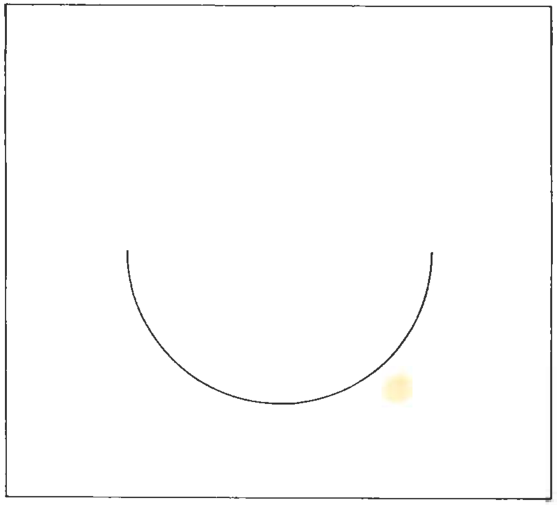
You can experiment with the numbers in this program to vary the location, size or color of the circle being painted.

You can create a vast array of different sized circles and geometric shapes by adding a few more instructions to the CIRCLE command. We will give you another example of using the circle command.

Change the program to read:

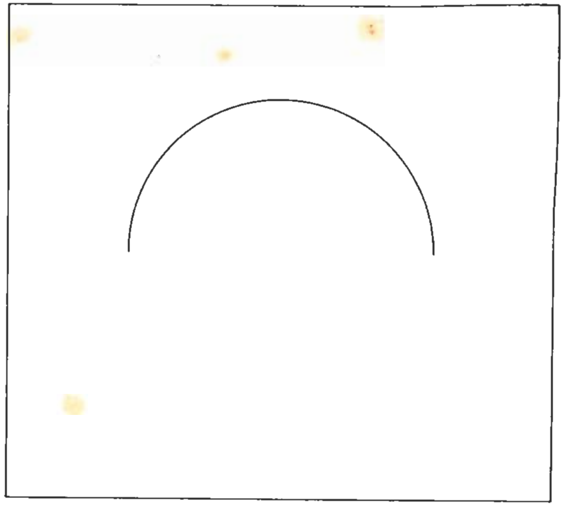
```
1Ø SCREEN 2
2Ø CIRCLE (128,96),8Ø,13,3.14,6.28
3Ø GOTO 3Ø
```

RUNning the program will give you the following result:



You should see the bottom half of the circle. Should you change line 2Ø to be:

```
2Ø CIRCLE (128,96),8Ø,13,6.28,3.14
```



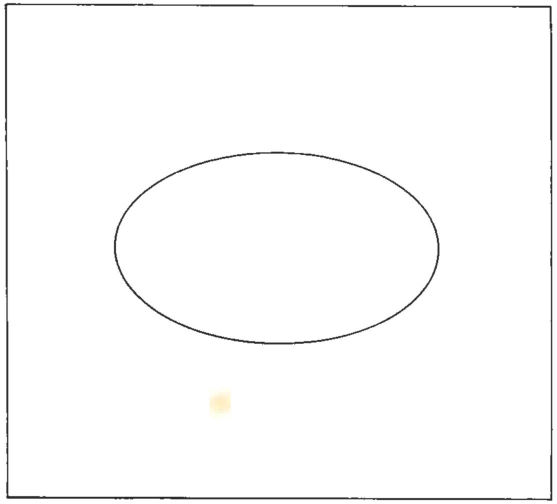
you will see that the top half of the circle is drawn. Another way of constructing a whole circle is with the following changes in line 2Ø:

```
2Ø CIRCLE (128,96), 80,13,Ø,6.28
```

Those of you who remember your geometry will recall that 3.14 is pi and 6.28 is 2pi (approximate). The two pi numbers which follow the color number (#13) on line 2Ø tell the computer where you would like the computer to begin and end the circle (how much of the circle you want drawn).

You can also specify the kind of shape drawn. For example, you can draw an ellipse (a distorted circle for those of you unfamiliar with geometry) with the following added feature on line 2Ø.

```
2Ø CIRCLE (128;96), 8Ø, 13,,,1/4
```

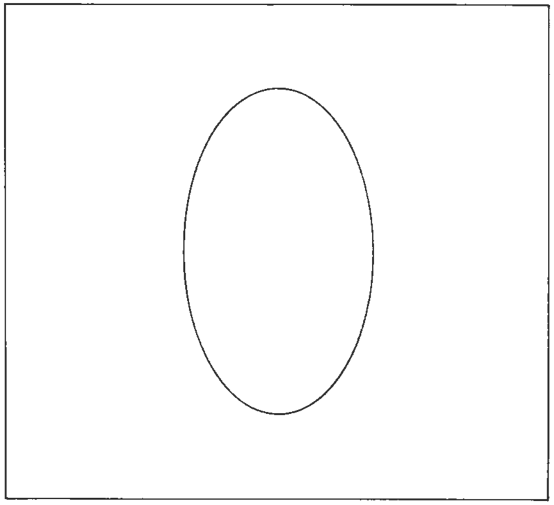


How did we get an ellipse? Well, the three commas after the number 13 are necessary to inform the computer that we will not be specifying the starting and ending points of the shape and are therefore leaving them blank. The computer knows what to do when we leave it blank. It assumes that we want the whole shape drawn. The 1/4 at the end of line 2Ø tells the computer the height/width ratio that we want.

Generally, the width of the circle is the same as the diameter you specify. However, if the ratio number at the end of the CIRCLE command line is less than one (1), the circle will be wider than it is high, as in the example above where the ratio is "1/4". If the ratio is greater than one (1), the circle will be higher than it is wide, as in the following example:

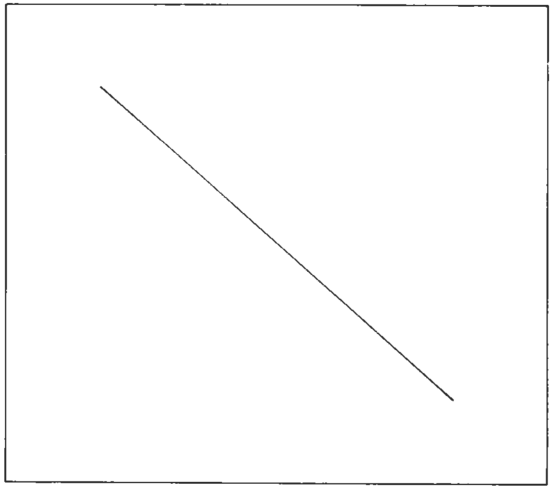
```
2Ø CIRCLE (128,96), 8Ø,13,,,2
```

LINE



Now that you have seen what your SVI-728 can do with circles and its paintbrush, we'll take a look at lines and boxes. The computer has the same simple method for drawing them as it does for circles. First, type NEW to clear the memory of the program we were using before. Now, enter the following

```
1Ø SCREEN 2
2Ø LINE (5Ø,4Ø) -- (2ØØ,15Ø),14
3Ø GOTO 3Ø
```



When you run this program, you will see that a line has been drawn from high on the left side of the screen to a low point on the right side of the screen. The line that causes this to happen is line 2Ø:

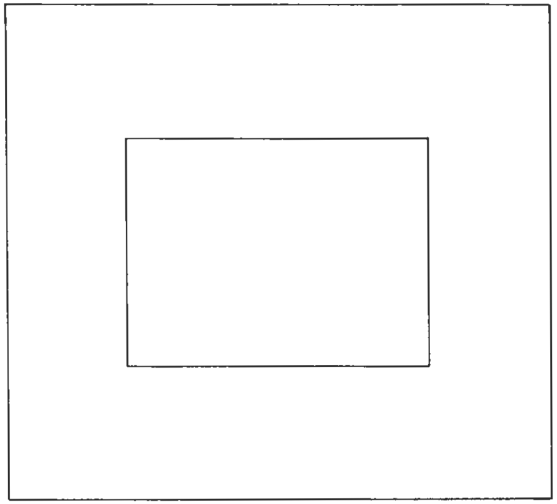
```
3Ø LINE (5Ø,4Ø) — (2ØØ,15Ø),14
```

This line tells the computer to draw a line from a position 5Ø points from the left margin on the screen and 4Ø points down from the top over to a position that is 2ØØ points from the left margin and 15Ø points down from the top. The number 14 designates the color of the line to be white.

BOX (B)

By simply adding the letter "B" following a comma to the end of line 2Ø you will convert this line into a box.

```
2Ø LINE (5Ø,4Ø) -- (2ØØ,15Ø),14,B
```



RUNning the program now, will show a box (outlined rectangle) on the screen. The "B" tells the computer to draw the box at the same coordinates as the line.

BOX FILL (BF)

To tell the computer to use the paintbrush (fill the box) simply add the letter "F" immediately to the right of the "B" in line 2Ø:

```
2Ø LINE (5Ø,4Ø) — (2ØØ,15Ø),14,BF
```

Now, you will see that the program draws the same box and paints the inside with the same color as the outline.

As an interesting summary of the graphics commands you have learned so far RUN the following program:

```
 1Ø COLOR, 1,9
 2Ø SCREEN 2
 3Ø CIRCLE (126,11Ø),6Ø,9,,,1.3
 4Ø CIRCLE (11Ø,96),1Ø,2
 5Ø PAINT (11Ø,96),2
 6Ø CIRCLE (142,96),1Ø,2
 7Ø PAINT (142,96),2
 8Ø LINE (1ØØ,125) — (1Ø5,135),14
 9Ø LINE (152,135) — (147,135),14
1ØØ LINE (1Ø5,135) — (147,135),14
11Ø CIRCLE (74,11Ø),25,11,,,5
12Ø CIRCLE (178,11Ø),25,11,,,5
13Ø Y = 85:R = 5Ø:C = 1
14Ø FOR Q = 1 TO 1Ø
15Ø Y = Y — 5:R = R — 4:C = C + 1
16Ø CIRCLE (126,Y),R,C,,,.2
17Ø NEXT Q
18Ø FOR T = 1 TO 5ØØ: NEXT T
19Ø N = RND (— TIME)
2ØØ FOR T = 1 TO  3Ø
21Ø X = X + 1Ø:Y = 1ØØ
22Ø C = INT (RND(1)*15) + 1
23Ø LINE (X,Y) — (X + 35, Y + 35),C,BF
24Ø LINE (X,Y) — (X + 35, Y + 35),1,BF
25Ø NEXT T
26Ø GOTO 26Ø
```

This program demonstrates all the concepts introduced thus far. We will continue with some more graphics commands.

DRAW

The **DRAW** command is actually the door to an actual mini-language within BASIC called "Graphic Macro Language (**GML**)".
Start by clearing the computer's memory.
(Type: NEW then press **ENTER**)
Then type the following lines:

```
1Ø SCREEN 2
2Ø PSET (5Ø,6Ø), 1
3Ø DRAW "D5Ø R5Ø U5Ø L5Ø"
4Ø GOTO 4Ø
```

Line 2Ø positions your graphic cursor at the X, Y coordinates (5Ø,6Ø), and designates the color to be black (1).

Line 3Ø starts the line drawing at the point specified in the **PSET** command. It can moves relative to the point, according to the distance and directional commands specified in the **DRAW** statement.
Example:

DRAW "U5ØR5Ø"

This tells the computer to draw fifty units to to RIGHT.

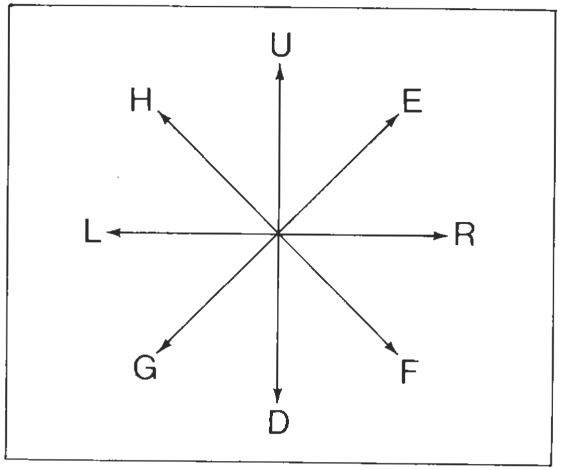
Instructions are in quotes because the DRAW command acts on a character string. Remember, a character string is a variable (container) that holds characters. Therefore, we could have written the DRAW example used above in the following manner.

```
3Ø T$ = "U5ØR5ØD5ØL5Ø"
4Ø DRAW T$
5Ø GOTO 5Ø
```

This second way of DRAWing first defines the object we wish to DRAW, and then places it in T$ and then DRAWs T$.

Please note you can draw diagonally using

E = diagonally up & right
F = diagonally down & right
G = diagonally down & left
H = diagonally up & left



Now we will look at several other additional graphic commands you can use to create exciting screen displays. They are **SCALE, LOCATE** and **BLANK MOVE.**

SCALE

Type in and RUN the following program:

```
1Ø COLOR, 1, 1
2Ø SCREEN 2 : X = 14 : Y = 12Ø :
   CO = 2 : Z = 1
3Ø PSET (X, Y)
4Ø A$="S=Z;C=CO;F15R5ØE15
   U45H15L3ØH15U2ØE1ØR2ØF1Ø
   R1ØU12H15L3ØG2ØD35F25R25
   F1ØD18G1ØL35H1ØL1ØD14SØ"
5Ø DRAW A$
6Ø X = X + 16 : Y = Y + 8
   CO = CO + 1 : Z = Z + 1
7Ø IF Z > 1Ø THEN 20
9Ø GOTO 3Ø
```

In this example, the variable **A$** contains the statement S = Z which sets the command SCALE equal to Z. As the program continues, Z is raised by 1 each time line 6Ø is reached. This causes the scale to be raised by 1.

LOCATE

Now try this program:

```
1Ø COLOR, 1, 1
11 CLS
12 REM LOCATE IS USED TO
   POSITION CURSOR
13 LOCATE 1Ø, 17Ø : PRINT
   “THIS IS AN”
14 LOCATE 1Ø, 18Ø : PRINT
   “EXAMPLE OF LOCATE”
15 FOR I = 1 TO 1ØØØ : NEXT
2Ø X = 14 : Y = 12Ø : CO = 2 : Z = 1
25 SCREEN 2
3Ø PSET (X, Y)
4Ø A$ = “S = Z;C = CO;F15R5ØE15U45
   H15L3ØH15U2ØE1ØR2ØE1ØR2ØF1Ø
   R 1ØU12H15L3ØG2ØD35F25R25
   F1ØD18G1ØL35H1ØL1ØD14SØ”
5Ø DRAW A$
6Ø X = X + 16 : Y = Y + 8: CO = CO + 1
   : Z = Z + 1
7Ø IF Z > 1Ø THEN 2Ø
9Ø GOTO 3Ø
```

The command **LOCATE** is used to position the cursor displayed on the screen.

BLANK MOVE

Now, type NEW and enter the following program:

```
 1Ø  REM DRAWING
 2Ø  SCREEN 2
 3Ø  A$ = “BM3Ø,156C9F15R5Ø E15U
     45H15L3ØH15U2ØE1ØR2ØF1ØR
     1ØU12H15L3ØG2ØD35F25R25F
     1ØD18G1ØL35H1ØL1ØD14”
 4Ø  DRAW A$
 5Ø  PAINT (42,154),9
 6Ø  LINE (15Ø,171) – (21Ø,171),2
 7Ø  LINE (21Ø,171) – (23Ø,34),2
 8Ø  LINE (23Ø,34) – (21Ø,34),2
 9Ø  LINE (21Ø,34) – (19Ø,151),2
1ØØ  LINE (19Ø,151) – (17Ø,151),2
11Ø  LINE (17Ø,151) – (15Ø,34),2
12Ø  LINE (15Ø,34) – (13Ø,34),2
13Ø  LINE (13Ø,34) – (15Ø,171),2
14Ø  PAINT (165, 165),2
15Ø  GOTO 15Ø
```

In this example, line 3Ø contains the statement **BM3Ø,156.** This is the command that causes the cursor to begin drawing at those specified X and Y coordinates. Without using this command, all draw statements will begin executing at the upper left corner of the screen.

SPRITE$

Now that we have learned how to deal with the simpler shapes, we will examine the more complex type of graphics generation called **SPRITE** generation. The best way to understand a sprite is to imagine it as a magic genie you can create and easily control.

Unlike the graphic commands we have encountered up till now which can only create one type of an object, like a line or circle – the manipulation of sprites is a lot more flexible.

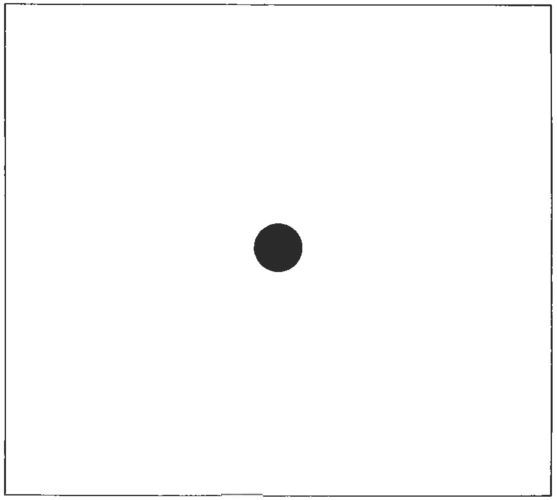
In order to see a sprite on the screen you must do the following:

STEP 1: Pick one genie "to talk to." (There are 32 different genies available to do your bidding).
STEP 2: You must tell the genie "what you want it to wear." In other words, what shape you want it to assume).
STEP 3: You must tell it what color to take the shape that it will wear.
STEP 4: You must tell it where to appear on the screen.

The importance of this genie metaphor cannot be overstated. Whenever you do not get the results you expected when commanding a genie it is probably because you did not provide all four pieces of information necessary to make the genie appear.

We will now demonstrate how to instruct a genie. Enter the following program and RUN it.

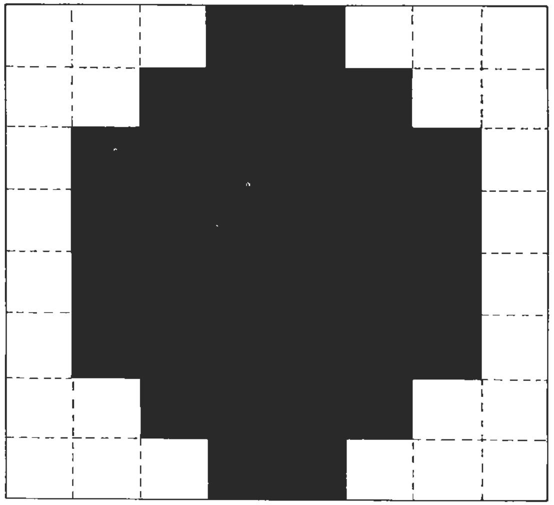
```
 1Ø SCREEN 2
 2Ø FOR T = 1 TO 8
 3Ø READ A$
 4Ø S$ = S$ + CHR$(VAL("&B" + A$))
 5Ø NEXT T
 6Ø SPRITE$ (1) = S$
 7Ø PUT SPRITE Ø, (128,96),8, 1:
    GOTO 7Ø
1ØØ DATA ØØØ11ØØØ
11Ø DATA ØØ1111ØØ
12Ø DATA Ø111111Ø
13Ø DATA Ø111111Ø
14Ø DATA Ø111111Ø
15Ø DATA Ø111111Ø
16Ø DATA ØØ1111ØØ
17Ø DATA ØØØ11ØØØ
```



You should see a red ball appear at the center of the screen. This ball is the sprite that we created in the above program. Here is how it works.

Lines 1ØØ–17Ø: These lines design the clothes that the genie will wear (or in straighter language, they contain the design of the sprite's shape). Each line of the data statement has eight characters on it. They represent the size of the sprite. The zeros are to make the display transparent at that point of the shape while the ones are the points of the display that are lit.

If we took an 8 by 8 grid , the shape would look like this:



Lines 2Ø–5Ø: These lines complete the design of the shape of the clothes. They set up a loop that will read the set of data lines, convert them into binary strings, append each one to the former and then store this one shape unit in S$.

Line 6Ø: This line picks the sprite that we will command (#1) and tells it to carry the shape contained in S$.

Line 7Ø: This line tells the sprite what color to make the shape, and where to appear on the screen. This line:

```
7Ø PUT SPRITE Ø, (128,96), 8,1
```

is read like this (It's a long sentence, but you'll be able to follow it):

PUT the SPRITE that is specified at the end of line, which is #1, on plane (surface) Ø, at position (128,96) which is the center of the screen, using color #8. The sprite to use is #1.

(Note: In the command, PUT SPRITE (sprite plane), (X,Y), (color #), (sprite pattern #), the use of different plane numbers allows the user to place more than one sprite on the screen at once.)

The way we create sprites is very logical. If you are familiar with any other computer's BASIC, you will immediately notice how much easier the sprite manipulation is on the SVI-728. That's because other systems force you to go PEEKing and POKing around their computer's memory.

Sprites are not limited to only 8 by 8 pixels. Sprites can also be placed within a 16 by 16 box. The sprite size after the screen mode determines the size of sprite. The sprite size should be set to Ø to select 8 X 8 pixels, 1 to select 8 X 8 pixels (magnifed by 2), 2 to select 16 X 16 pixels, 3 to select 16 X 16 pixels (magnified by 2).

Hence, if sprite size 1 is selected the sprite becomes 8 X 8 pixels (magnified by 2); however, if sprite size 2 is selected, then the use of a 16 X 16 box is allowed. But, the computer fills the 16 X 16 box differently than it fills the 8 X 8 box. The following program should illustrate how this works:

```
 1Ø  screen 2, 2
 2Ø  for X = 1 to 32
 3Ø  read a$
 4Ø  restore
 5Ø  s$ = s$ + chr$ (val("&b" + a$))
 6Ø  sprite$ (Ø) = s$
 7Ø  put sprite Ø, (128, 96), 1, Ø
 8Ø  next x
 9Ø  goto 9Ø
1ØØ  data 11111111
```

Notice tha the computer first fills a box 8 × 16 and then fills another 8 × 16 box alongside this one to make 16 × 16 box. Therefore, when making a 16 × 16 sprite, careful manipulation of data statements (32 of them) is required.
This program demonstrates the use of the joystick to move sprites. When run, a small spaceship-like sprite will appear. This sprite can be moved anywhere within the confines of the screen and will also fire a bullet when the trigger is pressed.

```
 1Ø  Color 15, 1, 1
 2Ø  Screen 2,2
 3Ø  Rem This section reads in the sprites
 4Ø  For T = 1 to 8
 5Ø  Read A$
 6Ø  S$ = S$ + CHR$(VAL ("&b" + A$))
 7Ø  Next T
 8Ø  SPRITE $ (1) = S$
 9Ø  For T = 1 to 8
1ØØ  Read B$
11Ø  U$ = U$ + CHR$ (VAL("&b" + B$))
12Ø  Next T
13Ø  SPRITE $ (2) = U$
14Ø  Rem This section sets the initial
     location of the Sprite
15Ø  X = 128 : Y = 96
16Ø  Put SPRITE 1, (X,Y), 9,1
17Ø  D = STICK (Ø)
18Ø  F = STICK (Ø)
19Ø  Rem This section takes the given
     joystick information and uses it
     to make the sprite move
2ØØ  If F <> Ø then GOSUB 46Ø
21Ø  If D = 1 then X = X : Y = Y – 1
22Ø  If D = 2 then X = X + 1: Y = Y – 1
23Ø  If D = 3 then X = X + 1: Y = Y
24Ø  If D = 4 then X = X + 1: Y = Y + 1
25Ø  If D = 5 then X = X : Y = Y + 1
26Ø  If D = 6 then X = X – 1: Y = Y + 1
27Ø If D = 7 then X  = X – 1: Y = Y
28Ø If D = 8 then X = X – 1: Y = Y – 1
29Ø GOTO 16Ø
3ØØ DATA ØØ1111ØØ
31Ø DATA Ø1ØØØØ1Ø
32Ø DATA 1ØØØØØØ1
33Ø DATA 11111111
34Ø DATA Ø1ØØØØ1Ø
35Ø DATA 1ØØØØØØ1
36Ø DATA 1ØØØØØØ1
37Ø DATA 1ØØØØØØ1
38Ø DATA ØØØ1ØØØØ
39Ø DATA ØØ1Ø1ØØØ
4ØØ DATA ØØ1Ø1ØØØ
41Ø DATA ØØ111ØØØ
42Ø DATA ØØØØØØØØ
43Ø DATA ØØØØØØØØ
44Ø DATA ØØØØØØØØ
45Ø DATA ØØØØØØØØ
46Ø For I = Y – 3 to – 2Ø step – 2
47Ø Put SpriteØ, (X,I),9,2
48Ø Next I
49Ø Return
```

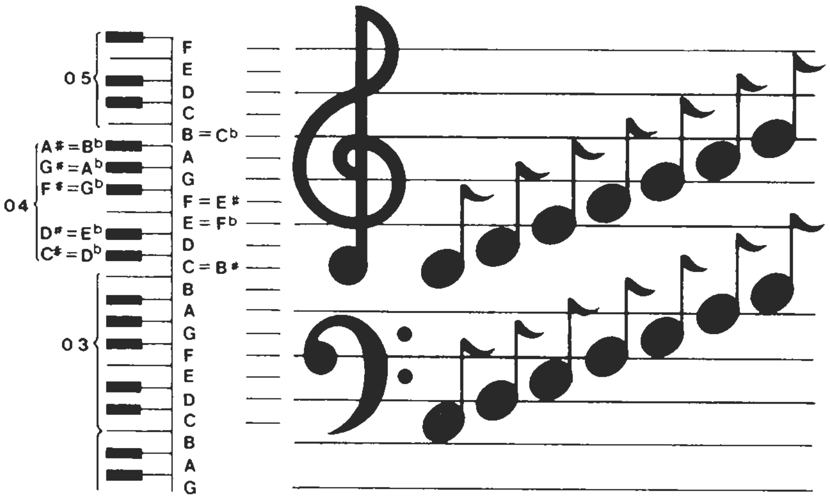
If you have understood all of the previous graphics examples and concepts, you should be well on your way to creating exciting graphics to enhance the programs you create using your SVI-728.

That ends our introduction to BASIC graphics. Now it is time to move on to the extraordinary sound capabilities of the SVI-728.

PART TWO– SOUND PROGRAMMING

There is also a very powerful music synthesizer built-in to the SVI-728 that is easily used by simple BASIC commands to produce music. In addition to the power of this synthesizer, it is most important to realize that it can do its work independently of the main microprocessor. What this means is that you can program the synthesizer to do one thing while the screen, printer, modem or other peripheral is doing something else.

The following figure represents the available musical scale that can be accessed by the SVI-728's synthesizer.



PLAY

The command that opens the door to this synthesizer is the BASIC keyword **PLAY.** For example, Typing the BASIC statement:

```
PLAY "CDE"
```

followed by pressing the **ENTER** key, will produce musical tones from your SVI-728 through the speaker on your television or monitor. You could achieve the similar results by writing a BASIC program with the following lines:

```
1Ø PLAY "CDE"
2Ø GOTO 1Ø
```

There are numerous other things that can be done with sound using the synthesizer in your SVI-728. We will look at the simple ones in this section and progress to the more complex ones in later pages. We will continue to work with the BASIC program listed above and make changes in it as we go along.

O (OCTAVE)

First, change line 1Ø to read:

```
1Ø PLAY "O1CDE"
```

Now, when you run the program, you will hear that the sounds produced are at a very low pitch when compared with the first ones you made. This is because you have set the **OCTAVE** by adding the "O1" before the "CDE". This is the command that allows you to access 8 octaves with the synthesizer. Now add this line:

```
11 PLAY "O4CDE"
```

When the program is run, you will hear three low notes followed by three higher notes. The octaves you can access using the "O" command can range from 1 (lowest) to 8 (highest).

T (TEMPO)

Now, change line 1Ø to read:

```
1Ø PLAY "T32O1CDE"
```

The program will now play the same note you heard before but at a much slower rate. What you did by typing the "T32" before the "O1CDE" was to set the **TEMPO** or speed of the music. The values for "T" can range from 32 (slowest) to 255 (fastest).

You will also notice that the notes in line 11 also play at the slower pace. This is because the synthesizer will play at whatever tempo you set until you tell it to play at a different tempo. To see this in action, change line 11 to read:

```
11 PLAY "T255O 4CDE"
```

Now, as you can hear, the notes from line 11 play at a much faster pace than those in line 10.

L (LENGTH)

You can also control the length of each note individually. To see this, change line 1Ø to read:

```
1Ø PLAY "T255O1CDL1E"
```

This change the "E" note to a much longer duration than "C" or "D" and also causes the notes in line 11 to play for a longer time. This length command can be placed in front of any note to control its length. The length of the notes can be varied from 1 (longest) to 64 (shortest).

Two other BASIC commands that can be applied to sound are the **"S"** command and the **"M"** command. These two commands determine the tonal qualities of the note being played. These are more commonly referred to as the **"ENVELOPE"** characteristics of a note. Everything that creates a sound has unique characteristics.

For example: the same note played on a piano and a trumpet may be at the same pitch but will have two distinctly different sounds. These two commands allow you to shape the notes you are creating in the same way.

S (SHAPE)

The **"S"** command controls the shape of the note. As an illustration of this, change line 1Ø to read:

```
1Ø PLAY "S1O4CDE"
```

and eliminate line 11.

Now, run the program to see the differences in the sounds you hear. These shape commands can be considered the voices of the synthesizer. There are 14 of them built-in to the SVI-728. This means that the numbers used to set the **"S"** command can range from **1** to 14.

M (TONE)

The **"M"** command controls the tone period or to be more specific, the amount of time that you will hear each note based on its tonal qualities. To see how this works, change line 1Ø to read:

```
1Ø PLAY "S1ØM5ØØØO4CDE"
```

As you will hear, this changes the sound dramatically. The values used to set **"M"** can range from 1 to 65535.

R (REST)

You can also insert pause between notes by using the **"R"** command. Change line 1Ø to read:

```
1Ø PLAY "O4CR1DR1ØE"
```

This causes the "C" note to play and then silence is heard for a while then the "D" note plays, then a shorter period of silence, then the "E" note followed immediately by the "C" note again.

V (VOLUME)

The final command we will examine in this section is the "V" command. This command is used to set the volume of the sound being produced. Change line 1Ø to read:

```
1Ø PLAY "O4V5CV1ØDV15E"
```

You will now hear that each note gets louder than the one before it. You can set the volume from Ø to 15.

USING 3 CHANNELS OF SOUND

So far, we have only used one note at a time to demonstrate the use of the synthesizer. However, the SVI-728 has three separate channels of sound that can be programmed individually to play together to create chords. Change line 1Ø to read:

```
1Ø PLAY "O1CDE", "O3EFC", "O5GAB"
```

What you hear is three notes being played in combination to create a chord. You can also have each channel play something entirely different from the others to create a melody and harmony part in the music you create.

There are also other ways of addressing the sound and music generation capabilities of the SVI-728 and they will be covered in the appendix, which is called "USING THE PROGRAMMABLE SOUND GENERATOR".

APPENDIX A
ASCII CHARACTER CODES

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	BLANK (NULL)	┼	BLANK (Space)	Ø	@	P	`	p	Ç	É	á	Ã	▂	◀	α	≡
1	☺	┴	!	1	A	Q	a	q	ü	æ	í	ã	▚	⧗	β	±
2	☻	┬	"	2	B	R	b	r	é	Æ	ó	Ĩ	▆	⧓	Γ	≥
3	♥	┤	#	3	C	S	c	s	â	ô	ú	ĩ	▀	▘	π	≤
4	♦	├	$	4	D	T	d	t	ä	ö	ñ	Õ	▪	▗	Σ	⌠
5	♣	┼	%	5	E	U	e	u	à	ò	Ñ	õ	▀	▝	σ	⌡
6	♠	│	&	6	F	V	f	v	å	û	a̲	Ũ	▎	▖	μ	÷
7	•	─	´	7	G	W	g	w	ç	ù	o̲	ũ	▞	▓	τ	≈
8	◘	┌	(	8	H	X	h	x	ê	ÿ	¿	Ĳ	▌	Δ	Φ	°
9	○	┐	)	9	I	Y	i	y	ë	Ö	⌐	ĳ	▕	ǂ	θ	•
A	◙	└	*	:	J	Z	j	z	è	Ü	¬	¾	▐	ω	Ω	•
B	♂	┘	+	;	K	[	k	{	ï	¢	½	~	▨	█	δ	√
C	♀	╳	,	<	L	\	l	¦	î	£	¼	◇	▧	▄	∞	^{n}
D	♪	╱	–	=	M	]	m	}	ì	¥	ī	‰	▼	▌	ø	^{2}
E	♫	╲	.	>	N	^	n	~	Ä	Pt	«	¶	▲	▐	∈	■
F	☼	┼	/	?	O	_	o	BLANK (DEL)	Å	ƒ	»	§	▶	▀	∩	BLANK (FF)

APPENDIX B
MATHEMATICAL FUNCTIONS

Derived Functions

Functions that are not available in MSX BASIC can be derived by using the following formulae:

Function	Microsoft BASIC Equivalent
SECANT	= 1/COS(X)
COSECANT	= 1/SIN(X)
COTANGENT	= 1/TAN(X)
INVERSE SINE	= ATN (X/SQP(–X*X + 1))
INVERSE COSINE	= –ATN (X/SQR(–X*X+1)) + 1.5708
INVERSE SECANT	= ATN (1/SQR (X*X – 1) + (SGN(X) – 1)* 1.5708
INVERSE COSECANT	= ATN (X/SQR (X*X – 1) + (SGN(X) – 1)*1.5708
INVERSE COTANGENT	= –ATN(X) + 1.5708
HYPERBOLIC SINE	= (EXP(X) – EXP (--X))/2
HYPERBOLIC COSINE	= (EXP(X) + EXP(–X))/2
HYPERBOLIC TANGENT	= (EXP(X) – EXP(–X))/(EXP(X) + EXP(–X))
HYPERBOLIC SECANT	= 2/(EXP(X) + EXP (–X))
HYPERBOLIC COSECANT	= 2/(EXP(X) – EXP(–X))
HYPERBOLIC COTANGENT	= (EXP(X) + EXP(–X))/(EXP(X) – EXP(–X))
INVERSE HYPERBOLIC SINE	= LOG(X + SQR(X*X + 1))
INVERSE HYPERBOLIC COSINE	= LOG(X + SQR(X*X – 1))
INVERSE HYPERBOLIC TANGENT	= LOG ((1 + X)/(1 – X))/2
INVERSE HYPERBOLIC SECANT	= LOG ((SQR(–X*X + 1) + 1)/X)
INVERSE HYPERBOLIC COSECANT	= LOG ((SGN (X)*SQR (X*X + 1) + 1)/X)
INVERSE HYPERBOLIC COTANGENT	= LOG ((X + 1)/(X – 1))/2

APPENDIX C
ERROR CODES AND ERROR MESSAGES

Code Message

1 **NEXT without FOR**
A variable in a NEXT statement does not correspond to any previously executed, unmatched FOR statement variable.

2 **Syntax error**
A line is encountered that contains some incorrect sequence of characters (such as unmatched parentheses, misspelled command or statement, incorrect punctuation, etc.)

3 **RETURN without GOSUB**
A RETURN statement is encountered for which there is no previous, unmatched GOSUB statement.

4 **Out of DATA**
A READ statement is executed when there is no DATA statement with unread data remaining in the program.

5 **Illegal function call**
A parameter that is out of the range is passed to a math or string function. An FC error may also occur as the result of:

1. a negative or unreasonably large subscript
2. a negative or zero argument with LOG
3. a negative argument to SQR
4. an improper argument to MID$, LEFT$, RIGHT$, INP, OUT, PEEK, POKE, TAB, SPC, STRING$, SPACE$, INSTR$ or ON . . . . GOTO.

6 **Overflow**
The result of a calculation is too large to be represented in BASIC's number format.

7 **Out of memory**
A program is too large, has too many files, has too many FOR loops or GOSUBs, too many variables or expressions that are too complicated.

8 **Undefined line number**
A line reference in a GOTO, GOSUB, IF . . . THEN . . . ELSE is to a nonexistent line.

9 **Subscript out of range**
An array element is referenced either with a subscript that is outside the dimensions of the array, or with the wrong number of subscripts.

10 **Redimensioned array**
Two DIM statements are given for the same array, or DIM statement is given for an array after the default dimension of 10 has been established for that array.

11 **Division by zero**
A division by zero is encountered in an expression, or the operation of involution results in zero being raised to a negative power.

12 **Illegal direct**
A statement that is illegal in direct mode is entered as a direct mode command.

13 **Type mismatch**
A string variable name is assigned a numeric value or vice versa; a function that expects a numeric argument is given a string argument or vice versa.

14 **Out of string space**
String variables have caused BASIC to exceed the amount of free memory remaining. BASIC will allocate string space dynamically, until it runs out of memory.

15 **String too long**
An attempt is made to create a string more than 255 characters long.

16 **String formula too complex**
A string expression is too long or too complex. The expression should be broken into smaller expressions.

17 **Can't continue**
An attempt is made to continue a program that:
1. has halted due to an error,
2. has been modified during a break in execution, or
3. does not exist.

18 **Underfined user function**
FN function is called before defining it with the DEF FN statement.

19 **Device I/O error**
An I/O error occurred on a cassette, disk drive, printer, or CRT operation. It is a fatal error; i.e., BASIC cannot recover from the error.

20 **Verify error**
The current program is different from the program saved on the cassette.

21 **No RESUME**
An error trapping routine is entered but contains no RESUME statement.

22 **Unprintable error**
A RESUME statement is encountered before an error trapping routine is entered.

23 **Unprintable error**
An error message is not available for the error condition which exists. This is usually caused by an ERROR with an undefined error code.

24 **Missing operand**
An expression contained an operator with no operand following it.

25 **Line buffer overflow**
An entered line has too many characters.

26 : 49 **Unprintable errors**
These codes have no definitions. Should be reserved for future expansion in BASIC.

DISK ERRORS

50 **FIELD overflow**
A FIELD statement attempts to allocate more bytes than were specified for the record length of a random file in the OPEN statement. Or, the end of the FIELD buffer is encountered while doing sequential I/O (PRINT#, INPUT#) to a random file.

51 **Internal error**
An internal malfunction has occurred. Report to Microsoft the conditions under which the message appeared.

52 **Bad file number**
A statement or command references a file with a file number that is not OPEN or is out of the range of file numbers specified by MAXFILES statement.

53 **File not found**
A LOAD, KILL or OPEN statement references a file that does not exist in the memory.

54 **File already open**
A sequential output mode OPEN is issued for a file that is already open; or a KILL is given for a file that is open.

55 **Input past end**
An INPUT statement is executed after all the data in the file has been INPUT, or for null (empty) file. To avoid this error, use the EOF function to detect the end of file.

56 **Bad file name**
An illegal form is used for the file name with LOAD, SAVE, KILL, NAME, etc.

57 **Direct statement in file**
A direct statement is encountered while LOADing an ASCII format file. The LOADing is terminated.

58 **Sequential I/O only**
A statement to random access is issued for a sequential file.

59 **File not OPEN**
The file specified in a PRINT#, INPUT#, etc. hasn't been OPENed.

60
⋮
255 **Unprintable error**
These codes have no definitions. Users may place their own error code definition at the high end of this range.

APPENDIX D
MSX BASIC RESERVED WORDS

BASIC statements and function names are reserved. That is, the key words cannot be used in variable names. This appendix lists all of the MSX BASIC language words that are reserved. If you attempt to use any of the words listed below as the name of the variable, an error is indicated by the computer.

ABS
ASC
ATN
AUTO
BASE
BEEP
BIN$
BLOAD
BSAVE
CALL
CDBL
CHR$
CINT
CIRCLE
CLEAR
CLOAD
CLOAD?
CLOSE
CLS
COLOR
CONT
COS
CSAVE
CSNG
CSRLIN
DATA
DEF FN
DEFINT

DEFDBL
DEFSNG
DEFSTR
DEFUSR
DELETE
DIM
DRAW
END
EOF
ERASE
ERL
ERR
ERROR
EXP
FIX
FOR
FRE
GOSUB
GOTO
HEX$
IF GOTO
IF THEN
INKEY$
INP
INPUT
INPUT$
INPUT#
INSTR

INT
INTERVAL ON
INTERVAL OFF
INTERVAL STOP
KEY
KEY LIST
KEY (n) ON
KEY (n) OFF
KEY (n) STOP
KEY ON
KEY OFF
LEFT$
LEN
LET
LINE
LINE INPUT
LINE INPUT#
LIST
LLIST
LOAD
LOCATE
LOG
LPOS
LPRINT
LPRINT USING
MAXFILES
MERGE
MID$

MOTOR ON
MOTOR OFF
NEW
NEXT
OCT$
ON ERROR GOTO
ON GOSUB
ON GOTO
ON INTERVAL GOSUB
ON KEY GOSUB
ON SPRITE GOSUB
ON STOP GOSUB
ON STRIG GOSUB
OPEN
OUT
PAD
PAINT
PDL
PEEK
PLAY
POINT
POKE
POS
PRESET
PRINT
PRINT USING
PRINT#
PRINT# USING

PSET
PUT SPRITE
READ
REM
RENUM
RESTORE
RESUME NEXT
RESUME 0
RETURN
RIGHT$
RND
RUN

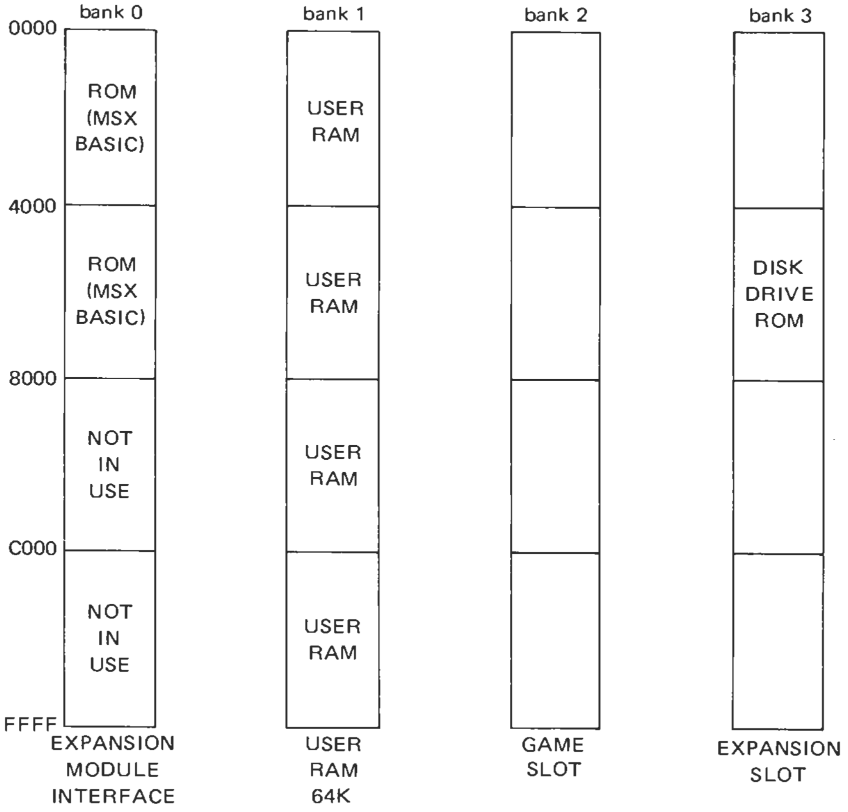
SAVE
SCREEN
SGN
SIN
SOUND
SPACE$
SPC
SPRITE ON
SPRITE OFF
SPRITE STOP
SPRITE$
SQR

STR$
STICK
STOP
STOP ON
STOP OFF
STOP STOP
STRIG
STRIG ON
STRIG OFF
STRIG STOP
STRING$
SWAP

TAB
TAN
TIME
TROFF
TRON
USR
VAL
VARPTR
VDP
VPEEK
VPOKE
WIDTH
WAIT

APPENDIX E
MEMORY MAPS & I/O PINOUTS

MEMORY MAP



	bank 0	bank 1	bank 2	bank 3
0000	ROM (MSX BASIC)	USER RAM		
4000	ROM (MSX BASIC)	USER RAM		DISK DRIVE ROM
8000	NOT IN USE	USER RAM		
C000	NOT IN USE	USER RAM		
FFFF	EXPANSION MODULE INTERFACE	USER RAM 64K	GAME SLOT	EXPANSION SLOT

I/O EXPANSION MODULE INTERFACE AND GAME SLOT PIN LAYOUT

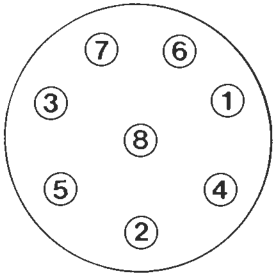
PIN NO.	NAME	I/O*	PIN NO.	NAME	I/O*
1	$\overline{\text{CS 1}}$	O	2	$\overline{\text{CS 2}}$	O
3	$\overline{\text{CS 12}}$	O	4	$\overline{\text{SLTSL}}$	O
5	reserved	—	6	$\overline{\text{RFSH}}$	O
7	$\overline{\text{WAIT}}$ (note 2)	I	8	$\overline{\text{INT}}$	I
9	$\overline{\text{M 1}}$	O	10	$\overline{\text{BUSDIR}}$	I
11	$\overline{\text{IORQ}}$	O	12	$\overline{\text{MERQ}}$	O
13	$\overline{\text{WR}}$	O	14	$\overline{\text{RD}}$	O
15	$\overline{\text{RESET}}$	O	16	reserved	—
17	A 9	O	18	A 15	O
19	A 11	O	20	A 10	O
21	A 7	O	22	A 6	O
23	A 12	O	24	A 8	O
25	A 14	O	26	A 13	O
27	A 1	O	28	A 0	O
29	A 3	O	30	A 2	O
31	A 5	O	32	A 4	O
33	D 1	I / O	34	D 0	I / O
35	D 3	I / O	36	D 2	I / O
37	D 5	I / O	38	D 4	I / O
39	D 7	I / O	40	D 6	I / O
41	GND	—	42	CLOCK	O
43	GND	—	44	SW 1	—
45	+ 5V	—	46	SW 2	—
47	+ 5V	—	58	+ 12V	—
49	SUNDIN	I	50	− 12V	—

Note: (1) Input & output is measured with respect to SVI-728.
(2) Open collector output.

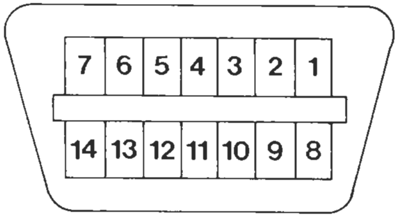
SVI-728 EXPANSION MODULE INTERFACE AND GAME SLOT SIGNAL DESCRIPTION

PIN NO.	NAME	DESCRIPTION
1	$\overline{\text{CS 1}}$	ROM 4000 ~ 7FFF select signal
2	$\overline{\text{CS 2}}$	ROM 8000 ~ BFFF select signal
3	$\overline{\text{CS 12}}$	ROM 4000 ~ BFFF select signal
4	$\overline{\text{SLTSL}}$	Slot selected signal signal. Fixed select signal for each slot.
5		Reserved for future use only
6	$\overline{\text{RFSH}}$	Refresh signal
7	$\overline{\text{WAIT}}$	Wait signal to CPU
8	$\overline{\text{INT}}$	Interrupt request signal
9	$\overline{\text{M 1}}$	Fetch cycle signal of CPU
10	$\overline{\text{BUSDIR}}$	This signal controls the direction of external data bus buffer when the cartridge is selected. It is low level when the data is sent by the cartridge.
11	$\overline{\text{IORQ}}$	I/O request signal
12	$\overline{\text{MERQ}}$	Memory request signal
13	$\overline{\text{WR}}$	Write signal
14	$\overline{\text{RD}}$	Read signal
15	$\overline{\text{RESET}}$	System reset signal
16		Reserved for future use only
17~32	A0~A15	Address bus
33~40	D0~D7	Data bus
41	GND	Ground
42	CLOCK	CPU clock 3579MHz
43	GND	Ground
44, 46	SW1, SW2	Insert/remove protect
45, 47	+ 5V	+ 5V power supply
48	+ 12V	+ 12 power supply
49	SUNDIN	Sound input (−5dbm)
50	−12V	−12V power supply

CASSETTE

PIN	NAME	I/O	LAYOUT
1	GND	–	
2	GND	–	
3	GND	–	
4	CMTOUT	O	
5	CMTIN	I	
6	REM +	O	
7	REM –	O	
8	GND	–	

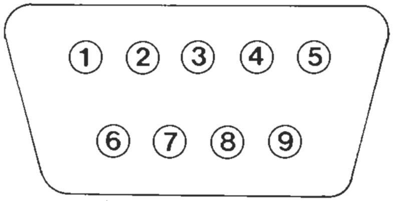
PRINTER

PIN	NAME	I/O	LAYOUT
1	PSTB	O	
2	PDBϕ	O	
3	PDB1	O	
4	PDB2	O	
5	PDB3	O	
6	PDB4	O	
7	PDB5	O	
8	PDB6	O	
9	PDB7	O	
10	NC	–	
11	BUSY	I	
12	NC	–	
13	NC	–	
14	GND	–	

KEYBOARD

PIN	NAME	I/O	PIN	NAME	I/O
1	INPUT 7	I	2	INPUT 6	I
3	INPUT 5	I	4	INPUT 4	I
5	INPUT 3	I	6	INPUT 2	I
7	INPUT 1	I	8	INPUT 0	I
9	OUTPUT 9	O	10	OUTPUT 8	O
11	OUTPUT 7	O	12	OUTPUT 6	O
13	OUTPUT 5	O	14	OUTPUT 4	O
15	OUTPUT 3	O	16	OUTPUT 2	O
17	OUTPUT 1	O	18	OUTPUT 0	O
19	CAPS	O	20	+5V	–
21	GND	–			

JOYSTICK

PIN	NAME	I/O	LAYOUT
1	FORWARD/SCK	I	
2	BACKWARD/SI	I	
3	LEFT/CS	I	
4	RIGHT/SENCTR	I	
5	+5V	–	
6	TRIGGER 1	I/O	
7	TRIGGER 2	I/O	
8	OUTPUT	O	
9	GND	–	

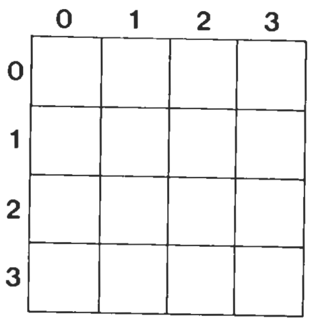
APPENDIX F
NOTES ON SCREENS

TEXT

The screen Ø command allows the display of 39 characters per line; while the screen 1 command caters for 29 characters. Both screens hold 24 rows per screen. The default is 29 characters per line. If screen Ø is on, by means of the "WIDTH 4Ø" command, 4Ø characters per line can be displayed.

HIGH AND LOW RESOLUTION

The 64 x 48 grid you see in figure 2-3 in Chapter 5 can be treated as a separate screen. The 64 by 48 grid is called a "low resolution graphics screen" and the 256 by 192 is called a "high resolution graphic screen." Every 4 boxes of the 256 by 192 grid is treated as one unit on the 64 x 48 grid. The box numbers in Figure 2 in Chapter 5 should be multiplied by 4 to appear in the proper places on the screens. This will maintain compatibility between the low resolution and high resolution screen.



Each of the four boxes is treated as an individual point on a high resolution screen.

The screen 3 command draws on the low resolution screen, and the screen 2 command draws on the high resolution screen. Thus if you PSET one of these four boxes on a high resolution screen, only that one will be lit up. But on a low resolution screen, if you turn on any one of these boxes all of them will be lit.

APPENDIX G
TROUBLE SHOOTING CHART

SYMPTOM	POSSIBLE CAUSE	REMEDY
NO POWER	Power Switch not turned ON.	Turn on power switch on the righthand side of the machine.
	Power cable not connected	Be sure the power cable is connected to the computer and the wall sockets.
	Blown fuse in the computer	Return the system to an authorized dealer for replacement.
NO SOUND OR PICTURE	Wrong TV hook up	Hook up the computer to the "VHF" antenna terminals.
	Loose video cable	Be sure all video cables are securely fastened.
NO SOUND	TV volume too low	Adjust the volume control of TV.
NO COLOR		Adjust TV Color level and fine tune the TV.

APPENDIX H
PROGRAMMABLE SOUND GENERATOR (PSG)

Other than the **PLAY** statement which allows you to create musical notes, you can use the **SOUND** statement to directly control the various capabilities of the Programmable Sound Generator which we will refer to as the PSG.

A PSG SOUND statement take the form of:

SOUND < Register of PSG > < value >

where < register of PSG > is one of the 13 available registers the PSG uses, and < value > is a number between 1 to 255. The function of creating a specific sound or sound effect logically follows the control sequence listed below:

OPERATION	REGISTER	FUNCTION
Tone generator control	RØ-R5	Program tone periods
Noise generator control	R6	Program noise period
Mixer control	R7	Enable tone and or noise on selected channels
Amplitude control	R8-R1Ø	Select "fixed" or "envelope variable" amplitudes
Envelope generator control	R11-R13	Program envelope period and select envelope pattern

TONE GENERATOR CONTROL

The PSG has 3 tone channels A, B and C. The frequency for each channel is obtained by counting down the input clock by 16 times the value of the frequency wanted.

For example:

Desired value = 3579545/ (16 * frequency)

Low register = <Desired value> AND 255

High register = <Desired value> / 256

The high and low registers correspond to the register pair used by each control.

Channel	Register
A	1,Ø
B	3,2
C	5,4

Program example follow:

```
1Ø INPUT "ENTER FREQUENCY";A
2Ø F = 3579545 / (16 * A)
3Ø H = 1 / 256
4Ø L = F AND 255
5Ø SOUND Ø,L
6Ø SOUND 1,H
7Ø SOUND 8,15: PRINT" VOLUME
   CONTROL OF CHANNEL A"
8Ø SOUND 7,254 : PRINT" BINARY
   1111111Ø TO ENABLE CHANNEL A"
9Ø END
```

AMPLITUDE CONTROL

In the previous example program, you should notice we used register 8 to enable volume of channel A. The PSG has three separate registers to control the amplitude of the different channels.

Channel	Register
A	8
B	9
C	1Ø

Each channel can have a volume from Ø to 15 with 15 being the loudest.

For example:

```
10 SOUND Ø, 1ØØ
20 SOUND 1,Ø
3Ø SOUND 7,254 : REM TURN
   ON CHANNEL A (MIXER)
40 FOR I = 15 TO Ø STEP – 1
50 SOUND 8,1
6Ø FOR J = 1 TO 2ØØ : NEXT J :
   REM DELAY
70 NEXT I
```

You will hear a high pitched sound fading away, because we changed the volume of channel A from 15 through Ø.

The amplitude control register can also be used to direct the envelope period of each channel, by setting the amplitude channel to a value of 16, the amplitude of the corresponding channel would be controlled by register 11, 12 and 13. For more information on this, refer to Envelope Period Control Registers.

MIXER CONTROL

The **MIXER** register, register number 7, controls the three Noise / Tone channels. The Mixers, as previously described, combine the noise and tone frequencies for each of the three channels. The determination of combining neither, either or both noise and tone frequencies on each channel is made by the state of Bit Ø – Bit 5 of register # 7. Bit 6 and 7 are for I/O ports connected through the PSG, and these are ignored by BASIC.

REGISTER 7

B7	B6	B5	B4	B3	B2	B1	BØ
Not used		Noise channel			Sound channel		
//////		C	B	A	C	B	A

Bits logical value

1 if channel is disabled **Ø if channel is enabled**

For example:

```
SOUND 7,&B1111111Ø
```

will turn on tone channel A.

```
SOUND 7, &B1111Ø11Ø
```

will enable both noise and tone channel A.

ENVELOPE PERIOD CONTROL REGISTER

The generation of fairly complex envelope patterns can be accomplished by two different ways in BASIC. First, it is possible to vary the frequency of the envelope using register 11 and 12 as a 16 bit register; and second, the relative shape and cycle of the envelope can be varied by using register 13.

For example:

< Desired envelope freq > = 3479545/ (256*freq)

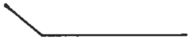
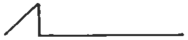
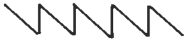
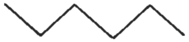
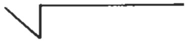
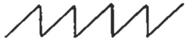
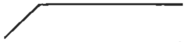
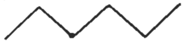
Program example:

```
 1Ø SOUND Ø,1ØØ
 2Ø SOUND 1,Ø : REM Tone channel A
 3Ø SOUND 7, &B1111111Ø:REM Enable A
 4Ø SOUND 8,16 : REM Value of 16 to
    enable E/P reg.
 5Ø SOUND 13,14 : REM Shape select
 6Ø S = .5 : REM .5 Hz
 7Ø K = 3579545
 8Ø L = K/ (256*S) AND 255
 9Ø H = K/ (256*S) /256
1ØØ SOUND 11,L
11Ø SOUND 12,H
12Ø END
```

NOTE: Register 13 is the shape register described next .

SHAPE REGISTER

You can select 9 different shapes for the envelope period output, by programming the shape register #13.

Selected value	Shape
Ø,1,2,3,9	
4,5,6,7 15	
8	
1Ø	
11	
12	
13	
14	

→ ← Envelope Period

For example:

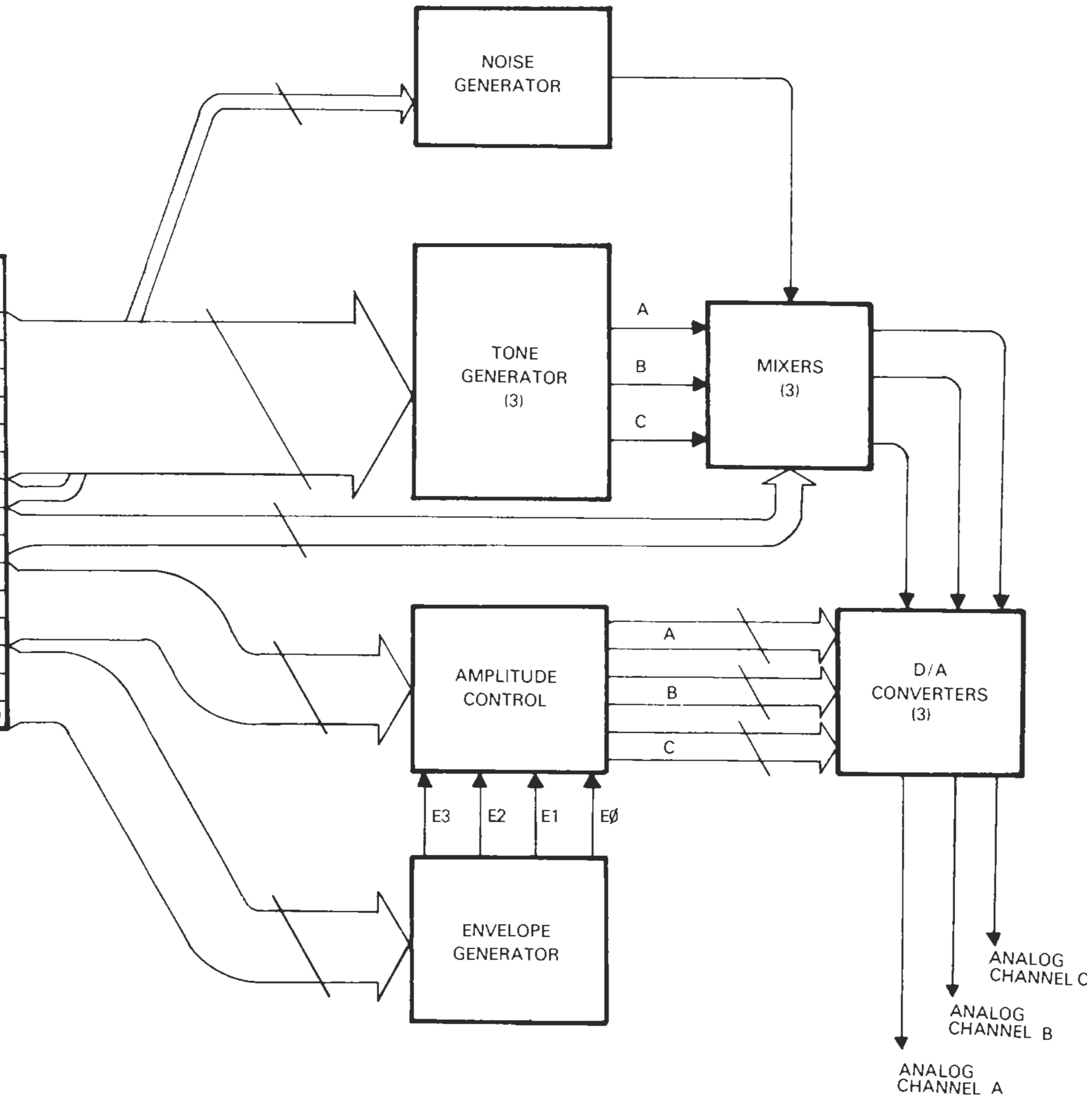
```
SOUND 13, 14
```

will create a tone modulating up and down according to the envelope period set in registers 11 and 12, when the enable bit 4 of register 8 (SOUND 8, 16) is set.

PSG BLOCK DIAGRAM

REGISTERED ARRAY (14 READ/WRITE CONTROL REGISTERS)

REGISTER \ BIT		B7	B6	B5	B4	B3	B2	B1	BØ
RØ	Channel A Tone Period	8-Bit Fine Tune A							
R1		▨	▨	▨	▨	4 Bit Coarse Tune A			
R2	Channel B Tone Period	8-Bit Fine Tune B							
R3		▨	▨	▨	▨	4 Bit Coarse Tune B			
R4	Channel C Tone Period	8 Bit Fine Tune C							
R5		▨	▨	▨	▨	4 Bit Coarse Tune C			
R6	Noise Period	▨	▨	▨	5 Bit Period Control				
R7	Enable	IN/OUT		Noise			Tone		
		I/OB	I/OA	C	B	A	C	B	A
R8	Channel A Amplitude	▨	▨	▨	M	L3	L2	L1	LØ
R9	Channel B Amplitude	▨	▨	▨	M	L3	L2	L1	LØ
R1Ø	Channel C Amplitude	▨	▨	▨	M	L3	L2	L1	LØ
R11	Envelope Period	8 Bit Fine Tune E							
R12		8 Bit Coarse Tune E							
R13	Envelope Shape/Cycle	▨	▨	▨	▨	CONT	ATT	ALT	HOLD



Remarks: CONT = Continue
ATT = Attack
ALT = Alternate

APPENDIX I

INTRODUCTION TO BASIC PROGRAMMING (BASIC GRAPHICS and BASIC SOUND)

In learning to use your SVI-728 computer, you use many different commands and programming techniques. However, if you are like most people, you would like to be able to do something NOW to see your computer in action. To help you do this, we will guide you through some very simple examples to prepare you for the things you will be exposed to later in the user's manual.

We are going to assume that you have read the systems overview and set-up procedures in your SVI-728 owner's manual, and have set up your computer properly. First, turn the television or monitor on and then turn on you SVI-728. Watch and you will see that the SPECTRAVIDEO logo appears on the screen, changing colors 3 times. The screen then clears and displays the amount of memory available. When the square cursor appears below this information, it is your indication that the SVI-728 is waiting for your instructions.

The first command that we will explore is one that controls the color of the screen and the characters that appear on it.

COLOR

This command allows you to vary the colors of the display in any manner you choose. You can choose from 16 colors, denoted by the numbers Ø to 15. As an example, type in the following line:

```
COLOR 6, 1
```

Now press the **ENTER** key. When you do this, you will see that the screen will change from its normal colors of blue background with white letters to a black screen with orange letters. You can explore the various colors that are available by varying the numbers you type after the "COLOR" command. When you have completed this exploration, turn the computer off and then back on. This will clear all the memory and return the system to its original or "default" condition. There are other ways of accomplishing this, by the way, and they are explained in greater depth in the user's manual.

CLS

Now, let's program a line in to the computer that will cause it to print you name on the top of the screen. Type in the following line:

```
1Ø CLS
```

Now, type RUN and press the **ENTER** key. You will see that the screen has cleared and that the word "Ok" is printed on line 1 with the cursor (white square) below. The command "CLS" (Clear Screen) is the command that clears the screen anytime the computer finds it in a program.

Now, type this line:

```
2Ø PRINT "SPECTRAVIDEO"
```

Type RUN and press **ENTER** again, and you will see that the word "SPECTRAVIDEO" now appears on the top line of the screen with the word "Ok" and the cursor below. What you have just done is to write a two-line program! Now, we will edit that program. First, type LIST and press the **ENTER** key. This will cause the program that you just wrote to be displayed on the screen. Now, using the joystick cursor control, move the cursor so that it is placed on the "S" of "SPECTRAVIDEO." Presss the **INS/PASTE** key on the keyboard and you will see that the cursor is now only half as tall as it was. This indicates that it is in the "INSERT" mode. To begin this edit, simply type in your name. The line should now look like this:

```
2Ø PRINT "YOUR NAMESPECTRAVIDEO"
```

The half-height cursor should still be located over the "S" in "SPECTRAVIDEO." Now, press the DEL/CUT key once and you will see that the "S" has disappeared and that the cursor is now located over the "P". You have deleted the "S". Do the same thing for all of the letters remaining until you get to the quotation mark. When the cursor is located over this mark, press the **ENTER** key.

Your edit is now completed. If you have followed all of the above steps, listing the program should cause the screen display to look like this:

```
10 CLS
2Ø PRINT "YOUR NAME"
```

"RUN" the program and you will see that your name is now at the top of the screen. Congratulations! You have created and edited a program using the built-in editor of your SVI-728.

BASIC GRAPHICS

Another area that will allow you to quickly explore the power in your SVI-728 is that of graphics. To see this in action, type in the following line, pressing **ENTER** after each is completed:

```
1Ø CLS
2Ø SCREEN 2
3Ø CIRCLE (128,8Ø), 6Ø, 11
4Ø PAINT (128,8Ø), 11
5Ø GOTO 4Ø
```

Now, type RUN and you will see a yellow circle appear on the screen and then it will be filled in by the SVI-728's yellow paint brush. To understand how this happens, let's look at each line individually.

```
1Ø CLS
```

As explained above, this line clears the screen. It is a good idea to place this line near the beginning of any program, so that you begin with a clean screen.

```
2Ø SCREEN 2
```

This line causes the computer to display its graphics screen (#2). You always have a choice of four screens using BASIC and the screen you were using in the first example was screen 1. This is the screen that is always displayed when the computer is turned on (which is why you didn't have to add a line to the first program to use this screen).

```
3Ø CIRCLE (128,8Ø), 6Ø, 11
```

Here, you are telling the computer to draw a circle at a point that is 128 points from the left side of the screen, 8Ø points down from the top of the screen, with a radius (distance from the center of the circle) of 6Ø points and with a yellow (the number 11) border.

```
4Ø PAINT (128, 8Ø), 11
```

This is the line that tells the SVI-728 to use its paintbrush to fill in the circle you have just outlined in line 3Ø. You must use the same coordinates (128,8Ø) and the same color (11) as used in the previous line.

```
5Ø GOTO 4Ø
```

The last line in the program causes the program to repeat line 4Ø until you stop the program by pressing the **CTRL** and **STOP** keys at the same time. You will also notice that when you do this, the screen clears and the cursor moves to the top left hand corner. This is because the display has returned to screen 1 referred to above. You can experiment with the numbers in this program to vary the location, size or color of the circle being painted.

The first number following the "CIRCLE" command (the X axis) may range from Ø to 255 and the second number (the Y axis) may range from Ø to 191.

LINE & BOX DRAWING

Now that you have seen what your SVI-728 can do with circles and its paintbrush, we'll take a look at lines and boxes. The computer has the same simple method for drawing them as it does for circles. First, type NEW to clear the memory of the program we were using before. Now, enter the following lines:

```
1Ø CLS
2Ø SCREEN 2
3Ø LINE (5Ø,4Ø) – (2ØØ, 15Ø)
4Ø GOTO 4Ø
```

When you run this program, you will see that a line has been drawn from high on the left side of the screen to a low point on the right side of

the screen. The line that causes this to happen is line 3Ø:

```
3Ø LINE (5Ø,4Ø) – (2ØØ, 15Ø)
```

This line tells the computer to draw a line from a position 5Ø points from the left margin of the screen and 4Ø points down from the top over to a position that is 2ØØ points from the left margin and 15Ø points down from the top.

Now, you can simply convert this same line into a box command by changing the line to read as follows:

```
3Ø LINE (5Ø,4Ø) – (2ØØ, 15Ø), 1Ø,B
```

By running the program now, you will see a box on the screen. By adding the ",1Ø" following the parentheses, you defined the color of the box border and the "B" tells the computer to draw a box at the same coordinates as the line. To tell the computer to use the paintbrush, change the line to read:

```
3Ø LINE (5Ø,4Ø) – (2ØØ, 15Ø), 1Ø,BF
```

Now you will see that the program draws the same box and paints the inside with the same color as the border.

BASIC MUSIC

PLAY

There is also a very powerful music synthesizer built into the SVI-728 that can be easily used, through simple BASIC commands, to produce music. The key to this synthesizer is the BASIC statement:

```
PLAY “ABC”
```

followed by pressing the enter key. This will produce musical tones from your SVI-728 through the speaker on your television or monitor. You could achieve similar results by writing a BASIC program with the following lines:

```
2Ø PLAY “ABC”
2Ø GOTO 1Ø
```

There are numerous other things that can be done with sound using the synthesizer in your SVI-728. We will look at the simple ones in this section and progress to the more complex ones in later pages. We will continue to work with the BASIC program listed above, and make changes in it as we go along.

O (OCTAVE)

First, change line 1Ø to read:

```
1Ø PLAY “O1ABC”
```

Now, when you run the program, you will hear that the sounds produced are at a very low pitch when compared with the first ones you made. This is because you have set the OCTAVE by adding the “O1” before the “ABC.” This is the command that allows you to access 8 octaves with the synthesizer. Now add this line:

```
11 PLAY “O4ABC”
```

When the program is run, you will hear three low notes followed by three higher notes. The octaves you can access using the “O” command can range from Ø (lowest) to 7 (highest).

T (TEMPO)

Now, change line 1Ø to read:

```
1Ø PLAY “T32O1ABC”
```

The program will now play the same note you heard before but at a much slower rate. What you did by typing the “T32” before the “O1ABC” was to set the TEMPO or speed of the music. The values for “T” can range from 32 (slowest) to 255 (fastest).

You will also notice that the notes in line 11 also play at the slower pace. This is because the synthesizer will play at whatever tempo you set until you tell it to play at a different tempo. To see this in action, change line 11 to read:

```
11 PLAY “T255O4ABC”
```

Now, as you can hear, the notes from line 11 play at a much faster pace than those in line 1Ø.

L (LENGTH)

You can also control the length of each note individually. To see this, change line 1Ø to read:

```
1Ø PLAY “T255O1ABL1C”
```

This changes the “C” note to a much longer duration than “A” or “B” and also causes the notes in line 11 to play for a longer time. This length command can be placed in front of any note to control the length of the note. The lengths of the notes can be varied from 1 (longest) to 64 (shortest).

S(SHAPE) & M(TONE)

Two other BASIC commands that can be applied to sounds are the "S" command and the "M" command. These two commands determine the tonal qualities of the note being played. As an example of this, the same note played on a piano and a trumpet may be at the same pitch but will have two distinctly different sounds. These two commands allow you to shape the notes you are creating in the same way.

The "S" command controls the shape of the note. As an illustration of this, change line 1Ø to read:

```
1Ø PLAY "S1O4ABC"
```

and eliminate line 11 by typing 11 and pressing ENTER

Now, run the program to see the differences in the sounds you hear. These shape commands can be considered the voices of the synthesizer. There are 14 of them built into the SVI-728. This means that the number used to set the "S" command can range from 1 to 14.

The "M" command controls the tone period or to be more specific, the amount of time that you will hear each note based on its tonal qualities. To see how this works, change line 1Ø to read:

```
1Ø PLAY "S1ØM5ØØØO4ABC"
```

As you will hear, this changes the sound dramatically. The values used to set "M" can range from 1 to 65535.

R (REST)

You can also insert pauses between notes by using the "R" command. Change line 1Ø to read:

```
1Ø PLAY "O4AR1BR1ØC"
```

This causes the "A" note to play, followed by a brief period of silence. Then the "B" note plays, then a shorter period of silence, then the "C" note followed immediately by the "A" note again.

V (VOLUME)

The final command we will examine in this section is the "V" command. This command is used to set the volume of the sound being produced. Change line 1Ø to read:

```
1Ø PLAY "O4V5AV1ØBV15C"
```

You will now hear that each note gets louder than the one before it. You can set the volume from Ø to 15.

USING 3 CHANNELS OF SOUND

So far, we have only used one note at a time to demonstrate the use of the synthesizer. However, the SVI-728 has three separate channels of sound that can be programmed individually to play together to create chords. Change line 1Ø to read:

```
1Ø PLAY "O1ABC", "O3CDE", "O5FAG"
```

What you hear now is three notes being played in combination to create a chord. You can also have each channel play something entirely different from the others to create melody and harmony parts in the music you create.

These are just a few of the exciting things that your SVI-728 Computer System can do for you. To begin exploring your machine in greater depth, read each chapter in this user's manual and follow the examples. Have Fun and Good Luck!!!

APPENDIX J
GLOSSARY

Although most of the words that appear in the glossary were not used in this manual, we have included them for your future reference:

access time The time between the instant that an address is sent to a memory location and the instant data returns.

accumulator One of several registers which temporarily store, or "accumulate" the results of various operations.

address The digital number used by the CPU to specify a location in memory.

ALU *Arithmetic Logic Unit.* The part of a CPU that adds, subtracts, shifts, ANDs, ORs and performs other computational and logical operations.

alphanumeric A device or a system that includes both alphabetical and numerical characters.

architecture The organizational structure of a computer system.

array A list of values stored in a series of memory locations.

ASCII *American Standard Code for Information Interchange.* Consists of 128 letters, numbers, punctuation marks, and special symbols. Each of which consists of a binary pattern that uses eight digits.

assembler A software program which converts symbolic or mnemonic language into machine. language.

BASIC *Beginners All Purpose Symbolic Instruction Code.* A high level programming language designed for the beginning programmer.

baud A unit by which signal speeds are measured. In microprocessing, the baud rate refers to the number of bits per second.

binary A numbering system that has 2 as its base, and that uses only the digits Ø and 1. Used by digital computers to perform the tasks in data processing.

bit *Binary digit.* Single element of a binary number with a value of either Ø or 1.

boolean logic Mathematical logic processes based on a system of algebra developed in the early nineteenth century by an English mathematician George Boole.

bootstrap A technique or device for loading the first instructions or words of a routine into memory. These instructions are used then to bring in the rest of the routine.

branch A way of re-routing a program so that it branches to another set of instructions to perform another task.

bug An error or defect in the hardware or software of the computer, causing a malfunction.

bus A set of wires of conductors arranged in parallel, used to transmit data, signals or power between parts of a computer system.

byte A group of eight bits, operated upon as a unit.

clock A device or circuit that sends out timing pulses to synchronize the action of the processor.

COBOL — *Common Business Oriented Language.* A high level language used in many business applications.

command — An instruction to the computer that causes something to happen.

compiler — A program to convert a high level language into assembler or machine language (understood by the computer).

controller — An interface which allows the control of an I/O device by the central processing unit.

CPU — *Central Processing Unit.* The part of the computer that controls all execution of instructions and arithmetic operations.

CRT — Cathode Ray Tube. The display on which information is shown after program execution.

cursor — A symbol displayed on the screen indicating where the next character is to be displayed.

data — Essentially, information that is input to the computer.

data bus — An electrical path along which information passes.

debug — To delete any errors in program.

DMA — *Direct Memory Access.* The accessing of memory without intervention of the central processing unit.

disk — A plate resembling a record album with a magnetic surface used to store data or programs. Also known as "floppy disk".

DOS — *Disk Operating System.* The program used for implementation of a disk drive.

dump — The transfer of information from one piece of equipment to another.

editor — A program used for the creating and/or altering of text in another program.

execute — The final step in running a program. An execution will perform the operation specified in the program.

expression — A particular grouping of numbers, letters or variables that comprise a single quantity.

fetch — Refers to the reading out of an instruction/data from an addressed memory location.

file — A collection of organized records that are usually on one particular subject.

firmware — The programs that are built into the ROM of a microcomputer.

floppy disk drive — A peripheral device used to store data from and input data to the computer. It is an input/output device.

flowchart — A diagram used in the development of a computer program. A flowchart shows the sequence of steps to take.

format — The particular arrangement or layout of data on a data medium, such as a screen or a diskette.

FORTRAN — *FORmula TRANslation.* A high level language using algebraic notation.

gate — An electrical signal circuit, with two (or more) inputs and one output, that behaves as a switch to create a particular state (either a binary one or zero).

hardware — The physical components that make up a particular computer system include all the peripheral devices.

hexadecimal A numbering system used in computers. Use the digits Ø-9 and the letters A-F.

high level language A programming language that is easier to understand and more convenient for the programmer. BASIC, FORTRAN, PASCAL and PL-1 are some examples of high level languages.

I/O devices *Input/output devices.* These would include the disk drive, data cassette, keyboard, printer, TV/ monitor, etc.

instruction A command telling the computer to perform a specific task.

instruction set This is the set of instructions built into the firmware of the microcomputer. This instruction set is used by the programmer.

interface This is the way in which peripheral devices are linked to the mainframe console of the microcomputer.

interpreter A program that converts one instruction at a time into machine language understood by the computer.

keyboard This is the console of the computer through which data is input to the CPU.

kilobyte or K Equivalent to 1Ø24 bytes.

library A collection of files or records that a person can access easily.

load To enter a program into a computer's memory.

location The portion in memory in which a data word or an instruction is stored.

logic A particular way of reasoning using thought processes.

loop A series of instructions that allow the programmer to repeat a particular sequence of events in a program.

LSI *Large Scale Integration.* An integrated circuit that has thousands of components packed in one chip.

machine language This is the language of the computer that is the lowest level the computer itself can accept as a program. This language is used with either binary, octal or hexadecimal numbering systems.

memory The part of the computer that stores data and instructions. Each instruction uses a particular address which tells the CPU where to fetch from.

menu This is a list much like the one in your local restaurant, except this type of menu lists what the computer is ready to do for you.

microprocessor This is also known as the Central Processing Unit. It is comprised of one or more LSI circuits that control all the processes of the computer.

mnemonics These are abbreviated terms for instructions used so that the programmer can easily remember them.

modem *MOdulater DEModulator.* This is a device used to convert data to signals that can be transmitted over telephone lines and then back to data again at the receiving end.

octal A numbering system used in computers employing the digits Ø-7.

on-line Whenever a peripheral device is interacting with its host computer, it is said to be "on line". For example, a printer is said to be "on-line" when it is doing a computer printout.

operating system The "OS" of a computer system is the program that directs such things as input/output, memory allocation, interrupt processing. It controls the overall operation of the computer.

output When data is said to be "output" it usually refers to the printout from a printer. Output may also be programs or data saved on a floppy diskette.

page A grouping of memory locations by higher order address bits. In an 8-bit microprocessor, a page usually has 256 bytes.

peripheral Any device external from the host computer but used in conjunction with the computer to perform operations such as printouts, data storage and retrieval, CRT displays, telecommunications, graphics, etc.

pixel The measurement of one dot on the display screen. The number and arrangement of pixels is what determines screen resolution.

pointer This is the register in the CPU that contains the memory address.

program The sequence of instructions that tell the computer what task to perform.

program counter This is the register in the CPU that specifies the address of the next instruction to be executed.

prompt This is the symbol on the screen which shows the user that the computer is ready to accept input from the keyboard.

RAM *Random Access Memory.* This is the portion of memory that can be written into and read from. When the computer's power is turned off, anything written will be lost.

register A circuit used to store or manipulate bits or bytes of data in the CPU.

ROM *Read Only Memory.* This is the part of memory that may only be read from. It is said to be "non-volatile", meaning that when power is turned off the ROM retains its information.

routine A specific program designed to do a particular function.

software Software pertains to the programs that are input to the computer by the user.

source program A program written in a language that is easily understood.

sprite This is a shape designed by the programmer when using a computer's graphic capabilities.

subroutine A routine in a program may be used again to perform a specific function.

syntax The rules governing a command line. If the command line is not in proper syntax, a "syntax error" message will occur.

terminal An input/output device usually consisting of a keyboard, CRT and printer used as a work station.

time sharing The process of (more than one user) sharing the use of a CPU via time robin.

truth table A truth table shows the different values that an AND, OR, NAND, NOR or other logic gate will have, according to two select inputs.

utility program This is a program that helps the user perform various specific utility functions, such as a debug program to find mistakes in programs.

variable A variable is any number or set of numbers, assigned a particular value, that is to be operated upon in a program.

volatile storage The type of storage which, when power is removed, the program or data in memory is lost. RAM is said to be volatile.

windows Several smaller screens displayed on one CRT screen at the same time.

INDEX

SVI-728 MSX COMPUTER QUICK REFERENCE CARD

FORMAT NOTATION

The following rules apply to the format of a statement or command:

1 Items in capital must be input as shown
2. Items in lower case letters enclosed in bracket < > are to be supplied by the user.
3 Items in square brackets [ ] are optional
4. All punctuation except angle brackets and square brackets (i.e., commas, parentheses, semicolons, hypens, equal signs) must be included where shown.
5. Items followed by an ellipse ( ) may be repeated any number of times (up to the length of line).
6. 'string' means a string expression
7. "exp" means a numeric expression, either constant or variable
8 'line' and "line number" both means line number
9. "n" means an integer
10. 'x, y' denotes x, y co-ordinates of the screen

VARIABLE TYPE DECLARATION CHARACTER

Variable	Range	No. of Byte
$ String	0 to 255 characters	3+No of characters
% Integer	–32768 to 32767	2
! Single precision	7 to 1 digit floating integer	4
# Double precision	16 to 8 digit floating point	8

Others:

&B	Binary figure
&H	Hexadecimal figure
&O	Octal figure

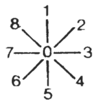
CHARACTER SET

Apart from the alphabet (A to Z, a to z) and numeric (0 to 9) characters, other special characters are tabulated:

Character	Action
	Blank
=	Equal sign or assignment symbol
+	Plus sign
–	Minus sign
*	Asterisk or multiplication symbol
/	Slash or division symbol
^	Up arrow or exponentiation symbol
(	Left parenthesis
)	Right parenthesis
%	Percent
#	Number sign or double precision declaration character
$	Dollar or string variable sign
!	Exclamation point or single precision declaration
[	Left bracket
]	Right bracket
	Comma
	Period or decimal point
	Single quotation mark (apostrophe) or same as REM
;	Semicolon
	Colon or used to seperate statements typed on the same line
&	Ampersand
?	Question mark or command PRINT
<	Less than
>	Greater than
@	At sign
_	Underscore
⇦	Delete last character typed
ESC	Escape
⇨	Move print position to next tab stop (TAB stops are set every eight columns)
ENTER	Terminate input of a line
HM	Move cursor to home position
CLS	Clear screen and move cursor to home position
DEL	Delete character at cursor
INS	Toggle insert/overtype mode
STOP	Toggle pause/resume program execution

JOYSTICK/CURSOR CONTROL PAD

There are eight possible movements for both joystick and cursor keys



For directions 2, 4, 6 and 8, a combination of two cursor keys should be pressed.

PROGRAMMABLE FUNCTION KEY

Key Number	Initial Definition
F1	color
F2	auto [10, 10]
F3	goto
F4	list
F5	run ENTER
F6	color 15, 4, 7 ENTER
F7	cload"
F8	cont ENTER
F9	list. ENTER
	CLS run ENTER

CONTROL CHARACTER

The ASCII control key is entered by pressing the key while holding down the CTRL Key.

Hex. Code	Control Key	Special Key	Function
01 *	A		Ignored
02 *	B		Move cursor to start of previous word
03 *	C		Break when BASIC is waiting for input
04 *	D		Ignored
05 *	E		Truncate line (clear text to end of logical line)
06 *	F		Move cursor to start of next word
07 *	G		Beep
08	H	⇦	Delete characters passed over
09 *	I	⇨	Move to next TAB stop
0A *	J		Line feed
0B *	K	HM	Move cursor to home position
0C *	L	CLS	Clear screen
0D *	M	ENTER	Enter current logical line
0E *	N		Append to end of line
0F *	O		Ignored
10 *	P		Ignored
11 *	Q		Ignored
12 *	R	INS	Toggle insert/typeover mode
13 *	S		Ignored
14 *	T		Ignored
15 *	U		Clear logical line
16 *	V		Ignored
17 *	W		Ignored
18 *	X	SELECT	Only function for word processor
19 *	Y		Ignored
1A *	Z		Ignored
1B	[	ESC	Ignored
1C *	\		Cursor right
1D *	]		Cursor left
1E *	^		Cursor up
1F *	_		Cursor down
7F	DEL	DEL	Delete character at cursor

Note: Those keys marked with asterisk cancel insert mode when editor is in insert mode.

BASIC COMMAND

AUTO [< line number >] [, < increment >]
Generate line numbers automatically.

BEEP
Make a beep sound.

BLOAD " [< device descriptor >] [< filename >] " [, R] [, < load address >]
Load a machine language program into memory from the specified device.

BSAVE " [< device descriptor >] <filename >", < start address > , < end address > [, < execution address >]
Save a memory image at the specified memory location to the device.

CLEAR [< string space > [, < highest location >]]
Set all numeric variables to zero, all string variables to null and close all open files and optionally, set the end of memory

CLOAD [" < filename > "]
Load a BASIC program file from the cassette

CLOAD? [" < filename > "]
Verify a BASIC program on cassette with one in memory

COLOR [< foreground >] [, < background >] [, < border >]
Set colors for the screen display.

CONT
Continue program execution after break or stop.

CSAVE " < filename > "
Save a BASIC program currently resided in memory to the cassette tape.

DELETE [< line number >] [– < line number >]
Delete program lines.

KEY < function key # > , " < string > "
Set a string to a function key.

KEY LIST
List the contents of all the programmable function keys.

KEY ON
Turn on function key display on the 24th line of text screen.

LIST [< line number >] [– < line number >]
List all or part of the program on the screen.

LLIST [< line number >] [– < line number >]
List all or part of the program on the printer.

LOAD " [< device descriptor >] < filename > "
Load a BASIC program (an ASCII file) from the device.

MAXFILES = n
Specify the maximum number of files opened at a time.

MERGE " [< device descriptor >] < filename > "
Merge the lines from an ASCII program into the program currently in memory.

MOTOR ON/OFF
Turn cassette motor on or off.

NEW
Delete entire program from working memory and reset all variables.

RENUM [< new number >] [, < old number >] [, < increment >]
Renumber program lines

RUN [< line number >]
Execute a program.

SAVE " < device descriptor > < filename > "
Save the program in memory with name filename as an ASCII file.

SOUND ON/OFF
Turn sound of the cassette audio on or off.

TRON
Turn on trace for program execution

TROFF
Turn trace off.

WIDTH n
Set the display width on screen during text mode. n may be 1 ~ 40

BASIC STATEMENT

CLOSE [#] < filenumber > [, < filenumber >]
Close the channel and release the buffer associated with it.

DATA < list of constants >
Store the numeric and/or string constants that are accessed by the READ statement(s).

DEF FN < name > [< parameter list >] = < function definition >
Define and name an arithmetic or string function

DEF USR n = < address >
Define the entry address for the nth machine language routine.

DEFINT < ranges of letters >
DEFSNG < ranges of letters >
DEFSTR < ranges of letters >
Declare variable type as integer, single precision, double precision or string.

DIM < list of subscripted variables >
Specify the maximum values for array variable subscripts and allocate storage accordingly.

DRAW < string >
Draw a figure according to the Graphic Macro Language.

END
Terminate program execution, close all files and return to command level

ERASE < list of array variables >
Eliminate arrays from a program.

ERROR n
Generate error message of code n.

FOR < variable > = x TO y [STEP z]
NEXT [< variable >]
Allow a series of instructions to be performed in a loop a given number of times

GOSUB < line number >
RETURN [< line number >]
Branch to a subroutine and then return from it. If < line number > after RETURN is not specified, return routine to the statement following the last GOSUB executed

GOTO < line number >
Branch out of the normal program sequence to the specified line number.

IF < exp > THEN < statement/line > [ELSE < statement/line >]
Make a decision regarding program flow based on the result returned by an expression.

LET < variable > = < exp >
Assign value of an expression to a variable. LET may be omitted.

INPUT [" < prompt string > " ;] < list of variables >
Allow input from keyboard during program execution.

INPUT # < filenumber > , < variable list >
Read data items from the specified channel and assign them to program variables.

INPUT $ (n , [#] < filenumber >)
Return a string of n characters, read from the file.

LINE INPUT [" < prompt string > " ;] < string variable >
Input an entire line (up to 254 characters) to a string variable.

LINE INPUT # < filenumber > , < string variable >
Read an entire line (up to 254 characters) from a sequential file to a string variable.

LPRINT [< list of exp >]
Print data at the printer.

LPRINT USING " < string > " ; < list of exp >
Print data at the printer using a specified format.

MID$ (< string > , n [, m]) = < string >
Replace a portion of the first string exp with the second string exp starting in the first string's nth character with m number of characters.

OPEN " < device descriptor > < filename > " [FOR < mode >] AS [#] < filenumber >
Allocate a buffer for I/O and set the mode that will be used with the buffer.

OUT < port > , < byte >
Send a byte to a machine output port.

PLAY < string for voice 1 > [, < string for voice 2 >] [, < string for voice 3 >]
Play music according to Music Macro Language.

POKE < memory address > , < byte >
Write a byte into a memory location

PRINT [< exp >]
Output data to the terminal

PRINT # < filenumber > , < exp >
Write data to the specified channel.

PRINT USING " < string > " ; < list of exp >
Print numeric or string expression using a specified format

PRINT # < filenumber > , USING " < string > " ; < list of exp >
Write data to the specified channel using a specified format.

READ < list of variables >
Read values from a DATA statement and assign them to variables.

REM < remark >
Allow explanatory remark to be inserted in a program.

RESTORE [< line number >]
Allow DATA statements to be reread from a specified line.

RESUME [0]
Resume execution at the statement causing an error.

RESUME < line number >
Resume execution at the specified line number.

RESUME NEXT
Resume execution at the statement immediately following the one causing an error.

SOUND < register of PSG > , < value >
Put a value into a register to control the Programmable Sound Generator

STOP
Terminate program execution and return to command level. The "Break in < line number >" message is printed.

SWAP < variable > , < variable >
Exchange the value of two variables.

WAIT < port > , < mask > [, < select >]
Suspend program execution read input at port until [< input bit > XOR < select > AND < mask >] returns a non-zero value.

INTERRUPT CONTROL COMMAND AND STATEMENT

INTERVAL ON/OFF/STOP
To enable, disable, or terminate the BASIC timer interrupt trapping.

KEY n ON/OFF/STOP
To enable, disable, or terminate interrupts caused by a function key.

ON ERROR GOSUB < linenumber >
Define the line number of the subroutine to execute when an error has been detected.

ON ERROR GOTO < linenumber >
Enable error trapping and specify the first line of the error handling subroutine.

ON < exp > GOTO < list of linenumbers >
ON < exp > GOSUB < list of linenumbers >
Branch to one of several specified line numbers, depending on the value returned when an expression is evaluated.

ON INTERVAL = < exp > GOSUB < list of linenumbers >
Set the line number to execute at every other machine interrupt cycle (60 per second) specified by < exp >

ON KEY (n) GOSUB < list of linenumbers >
Specify the line number corresponding to the line offset n in the statement to execute whenever a function key number n has been depressed.

ON STRIG GOSUB < linenumber >
Define the starting line of the subroutine employed when any of the joystick button is depressed. (0 for space bar, 1 for Joystick 1 or 2 for Joystick 2)

ON SPRITE GOSUB < linenumber >
Define jump address when sprites collision occurs.

ON STOP GOSUB < linenumber >
Define jump address when the CTRL + STOP keys are pressed.

STOP ON/OFF/STOP
To enable, disable or terminate CTRL + STOP trapping

STRIG ON/OFF/STOP
To enable, disable, or terminate joystick button or space bar trapping

SPRITE ON/OFF/STOP
To enable, disable, or terminate the interrupt generated when two or more sprites collide

GRAPHIC COMMAND AND STATEMENT

CLS
Clear graphic screen

CIRCLE (x, y), < radius > [, < color > [, < start > , < end >] [, < aspect ratio >]
Draw an ellipse with a center, radius, foreground color, starting and ending angles, height to width ratio as specified.

COLOR [< foreground >] , [< background >] , [< border >]
Set colors for the screen

DRAW < string >
Draw figure on the screen according to the Graphic Macro Language

GET [(x1, y1) – (x2, y2)] [, < color >] [, B [F]]
Read points from the graphic screen or from defiend area of the graphic screen. Array must be dimensioned large enough to hold the data

LINE [(x1, y1)] – (x2, y2) [, [< color >] [B [F]]
Draw straight line connecting the two coordinate pairs specified or if B [F] is present draw or fill rectangle

LOCATE x, y
Position the graphic cursor to the starting coordiante pointed to by x, y, can used with LINE, POINT PRINT

PAINT (x, y) [, < color > [, < low res mode border attribute >]]
Fill in an arbitrary graphic figure of the specified fill color. Paint must not have border for high res while border must be specified in low res

POINT (x, y)
Read the color of a pixel in the graphic mode

PSET (x, y) [, < color >]
Draw a dot at the assigned position on the screen using the foreground color or < color > if specified

PRESET (x, y) [, < color >]
Same as PSET except draw in background color if < color > not specified

PUT (x, y), < arrayname > [, < operation >]
Output graphic patterns in the array to assigned position on the screen, operation can be:

PSET:	output pattern as is
PRESET:	reverse pattern foreground/background color
AND:	combine graphic pattern color with screen pattern
OR:	graphic pattern overlapping the screen data
XOR:	perform XOR with screen data; if the matching pixel from the array and the screen are the same then that pixel will be displayed in background color else it will be displayed in background color else it will be displayed in the foreground color

PUT SPRITE < sprite plane > , (x, y) [, < color >] [, < pattern no. >]
Set up sprite attribute, when a field is omitted, the current value is used. see SPRITE$.

SCREEN [< exp 1 >] [, < exp 2 >]
Use < exp 1 > to select graphic mode: 1 for high res., 2 for low res. < exp 2 > selects the size of sprite (if used)

SPRITE$ (n) = < string >
Define a pattern as sprite, the number of string character depends on the sprite size, n must be less than 256 when size of sprite is 0 or 1 (8 bytes), less than 64 when size of sprite is 2 or 3 (32 bytes)

SPRITE$ (n)
Return either an 8 byte character string or 32 byte character string of the sprite number n depending on the size of sprite selected

VPEEK (< vram address >)
Return byte value from the specified location in video ram

VPOKE < vram address > , < byte >
Put a byte into video ram address specified.

GRAPHIC MACRO LANGUAGE (GML)

The followings are used in conjuction with DRAW command "n" is a number

U	n	Move up
D	n	Move down
L	n	Move left
R	n	Move right
E	n	Move diagonally up and right
F	n	Move diagonally down and right
G	n	Move diagonally down and left
H	n	Move diagonally up and left
S	n	Set scaling factor
B	n	Move without plotting
M	x v	Move absolute or relative distance If x has a plus sign (+) or a minus sign (–) in front of it, it is relative Otherwise it is absolute
N	n	Move and return to original position
A	n	Set the angle n

n	0	1	2	3
deg	0	90	180	270

C	n	Set color number
X	<string>	Execute string, must be terminated by a semicolon

SOUND COMMAND AND STATEMENT

SOUND statement is used to directly control the various capabilities of the Programmable Sound Generator (PSG). It takes the form of

SOUND <register of PSG> < value >

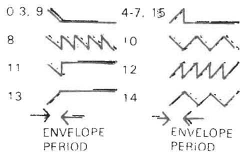
The functions of the 13 PSG registers are given below

PSG	Register	Function
0 1		Tone of channel A
2 3		Tone of channel B
4, 5		Tone of channel C
6		Noise generator
7		Mixer

B7	B6	B5	B4	B3	B2	B1	B0
Not used		Noise Channel			Tone Channel		
		C	B	A	C	B	A

Bit logical value 1 to disable and 0 to enable

8	Bit 0 to 3:amplitude control of channel A Bit 4 fixed amplitude control
9	Bit 0 to 3.amplitude control of channel B Bit 4 fixed amplitude control
10	Bit 0 to 3:amplitude control of channel C Bit 4 fixed amplitude control
11 12	Envelope period control range
13	Envelope cycle/shape control



MUSIC MACRO LANGUAGE

The PLAY command is used in conjunction with the following character string to play a melody

String	Function
A to G [# or +][–]	Play the indicated note in the current octave A number sign (#) or plus sign (+) afterwards indicates a sharp a minus sign (–) indicates a flat
I n	Set the length of the note n may range from 1 to 64.
M n	Modulation sets period of envelope n may range from 1 to 65535
N n	Play note n n may range from 0 to 96 in 7 octaves 7 n = 0 means rest
O n	Octave Set the octave of the notes to be played. Default is 4
R n	Pause or rest n may range 1 to 64
S n	Set shape of noise output n can be 0 to 15
T n	Tempo Set the number of quarter notes in a minute n ranges from 32 to 255 Default is 120
V n	Volume Set the volume of output n may range from 0 to 15 The default is 8
	Dot or period after each note causes it to be played 3/2 times the period determined by L times T
X	Execute specified string

PRINT USING FORMAT FIELD SPECIFIER

String or numeric expression can be printed using a specified format.

Specifier	Possible Digit	Field Character	Definition
NUMERIC			
#	1	1	Numeric field
	0	1	Decimal point
+	0	1	Print leading or trailing sign. Positive numbers will have "+", negative numbers will have #
–	0	1	Trailing sign Print "–" if negative, otherwise blank.
**	2	2	Leading asterisk
$$	1	2	Floating dollar sign. $ is placed in front of the leading digit.
**$	2	3	Asterisk and floating dollar sign
	1	1	Use comma every three digits (left of decimal point only).
^^^^	0	4	Exponential format Number is aligned so leading digit is non-zero
—	0	1	Next character literal
STRING			
!			Single character
/ /			< 2 + number of spaces > character field
&			Variable length field

BASIC FUNCTION

Function	Action
ASC (< string >)	Return the ASCII value of the first character of string.
BIN$ (n)	Return a string which represents the binary value of n
CHR$ (I)	Return a character having the ASCII code I.
FRE (< exp >)	Returns remaining memory free space.
HEX$ (< exp >)	Convert a number to a hexadecimal string.
INKEY$	Return a one character string input from keyboard or null string if no input.
INPUT$ (length [[#] m])	Return a string having specified length read from the console or a file.
LEFT$ (< string > < length >)	Return a length of leftmost characters of the string expression
LEN (< string >)	Return the number of characters in string.
MID$ (< string > < start > [< length >])	Return characters from the middle of the string starting at the position specified to the end of the string or for the specified length.
OCT$ (< exp >)	Convert a number to an octal string.
RIGHT$ (< string > < length >)	Return rightmost characters of the string expression
SPACE$ (I)	Return a string of I spaces.
STR$ (< exp >)	Convert a numeric expression to a string.
STRING$ (< length > < string >)	Return a string with specified length containing first character of string.
STRING$ (< length> I)	Return string with specified length containing first character with ASCII code I.
VAL (< string >)	Convert the string representation of a number to its numeric value.
CVI (< string >)	Convert a 2-character string to an integer.
CVS (< string >)	Convert a 4-character string to a single precision number.
CVD (< string >)	Convert an 8-character string to a double precision number.
MKI$ (< value >)	Convert an integer to a 2-character string.
MKS$ (<value >)	Convert a single precision value to a 4-character string.
MKD$ (< value >)	Convert a double precision value to an 8-character string.
BASE (n)	Current base address for each table.
CSRLIN	Return the vertical coordinate of the cursor
EOF (< filenumber >)	Return (–1) if the end of a sequential file has been reached Otherwise, return (0)
ERL	Error line number.
ERR	Error code number.
PAD (n)	Return various status of touch pad.
PEEK (I)	Read a byte from memory location I.
POS (n)	Return the current horizontal position of cursor. n is a dummy argument
SPC (I)	Print I blanks on the screen
STICK (n)	Return the direction of a joystick.
STRIG (n)	Return the status of a trigger button of a joystick
TAB (I)	Space to position I on the console
TIME	An unsigned integer generated from the system internal timer
USR [< digit >] (argument)	Call the user s machine language subroutine with the specified argument.
VARPTR (< variable >) VARPTR (# < filenumber >)	Return the address of the first byte of data identified with < variable >
VDP (n)	For n = 0 to 7, this specifies the current value of VDP's write only register. For n = 8 it specifies the status register of VDP

OPERATOR

Symbol	Function
=	Assignment or equality sign
–	Negation or subtraction
+	Addition or string concatenation
*	Multiplciation or asterisk
/	Division (floating point result)
^	Exponentiation symbol
\	Integer division
MOD	Integer modulus
NOT	One s complement
AND	Bitwise AND
OR	Bitwise OR
XOR	Bitwise exclusively OR
EQV	Bitwise equivalence
IMP	Bitwise implication
= < , > > = = < > = = < <> > <	Relational tests (result is either true (–1) or false (0))

The precedence of operators is
(1) Expressions in parentheses
(2) Exponentiation
(3) Negation
(4) Multiplication or Division
(5) Integer Division
(6) MOD
(7) Addition or Subtraction
(8) Relational operators
(9) NOT
(10) AND
(11) OR
(12) XOR
(13) IMP
(14) EQV

ARITHMETIC FUNCTION

Function	Action
ABS (< exp >)	Absolute value of expression.
ATN (< exp >)	Arctangent of the expression.
CDBL (< exp >)	Convert the expression to a double precision number.
CINT (< exp >)	Convert the expression to an integer
COS (< exp >)	Cosine of the expression (in radians).
CSNG (< exp >)	Convert the expression to a single precision number
EXP (< exp >)	Raise the constant e to the power of expression.
FIX (< exp >)	Return truncated integer of expression.
FRE (< exp >)	Give memory free space not used by BASIC.
INT (< exp >)	Evaluate the expression for the largest integer calculated.
LOG (< exp >)	Give the natural logarithm of the expression.
RND ([< exp >)	Generate a random number [< exp >] < 0 Seed new sequence [< exp >] = 0 Return previous random number [≤ exp >] >0 or omitted Return new random number
SGN (< exp >)	1 if expression > 0 0 if expression = 0 –1 if expression < 0
SIN (< exp >)	Sine of the expression (in radians)
SQR (< exp >)	Square root of expression
TAN (< exp >)	Tangent of the expression (in radians)

Angle is expressed or returned in radians.

SVI-728 MSX COMPUTER QUICK REFERENCE CARD

FORMAT NOTATION

The following rules apply to the format of a statement or command:

1. Items in capital must be input as shown
2. Items in lower case letters enclosed in bracket < > are to be supplied by the user
3. Items in square brackets [ ] are optional.
4. All punctuation except angle brackets and square brackets (i.e., commas, parentheses, semicolons, hypens, equal signs) must be included where shown.
5. Items followed by an ellipse (. . .) may be repeated any number of times (up to the length of line).
6. "string" means a string expression.
7. "exp" means a numeric expression, either constant or variable
8. "line" and "line number" both means line number.
9. "n" means an integer.
10. "x, y" denotes x, y co-ordinates of the screen

VARIABLE TYPE DECLARATION CHARACTER

Variable	Range	No. of Byte
$ String	0 to 255 characters	3+No.of characters
% Integer	–32768 to 32767	2
! Single precision	7 to 1 digit floating integer	4
# Double precision	16 to 8 digit floating point	8

Others:

&B	Binary figure
&H	Hexadecimal figure
&O	Octal figure

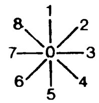
CHARACTER SET

Apart from the alphabet (A to Z , a to z) and numeric (0 to 9) characters, other special characters are tabulated:

Character	Action
	Blank
=	Equal sign or assignment symbol
+	Plus sign
–	Minus sign
*	Asterisk or multiplication symbol
/	Slash or division symbol
^	Up arrow or exponentiation symbol
(	Left parenthesis
)	Right parenthesis
%	Percent
#	Number sign or double precision declaration character
$	Dollar or string variable sign
!	Exclamation point or single precision declaration
[	Left bracket
]	Right bracket
,	Comma
	Period or decimal point
	Single quotation mark (apostrophe) or same as REM
;	Semicolon
:	Colon or used to seperate statements typed on the same line
&	Ampersand
?	Question mark or command PRINT
<	Less than
>	Greater than
@	At sign
—	Underscore
⇐	Delete last character typed
ESC	Escape
⇒	Move print position to next tab stop (TAB stops are set every eight columns)
ENTER	Terminate input of a line
HM	Move cursor to home position
CLS	Clear screen and move cursor to home position
DEL	Delete character at cursor
INS	Toggle insert/overtype mode
STOP	Toggle pause/resume program execution

JOYSTICK/CURSOR CONTROL PAD

There are eight possible movements for both joystick and cursor keys.



For directions 2, 4, 6 and 8, a combination of two cursor keys should be pressed.

PROGRAMMABLE FUNCTION KEY

Key Number	Initial Defioition
F1	color
F2	auto [10, 10]
F3	goto
F4	list
F5	run ENTER
F6	color 15, 4, 7 ENTER
F7	cload"
F8	cont ENTER
F9	list. ENTER
	CLS run ENTER

CONTROL CHARACTER

The ASCII control key is entered by pressing the key while holding down the CTRL Key.

Hex. Code		Control Key	Special Key	Function
01	*	A		Ignored
02	*	B		Move cursor to start of previous word
03	*	C		Break when BASIC is waiting for input
04	*	D		Ignored
05	*	E		Truncate line (clear text to end of logical line)
06	*	F		Move cursor to start of next word
07	*	G		Beep
08		H	⇐	Delete characters passed over
09	*	I	⇒	Move to next TAB stop
0A	*	J		Line feed
0B	*	K	HM	Move cursor to home position
0C	*	L	CLS	Clear screen
0D	*	M	ENTER	Enter current logical line
0E	*	N		Append to end of line
0F	*	O		Ignored
10	*	P		Ignored
11	*	Q		Ignored
12	*	R	INS	Toggle insert/typeover mode
13	*	S		Ignored
14	*	T		Ignored
15	*	U		Clear logical line
16	*	V		Ignored
17	*	W		Ignored
18	*	X	SELECT	Only function for word processor
19	*	Y		Ignored
1A	*	Z		Ignored
1B		[	ESC	Ignored
1C	*	\		Cursor right
1D	*	]		Cursor left
1E	*	^		Cursor up
1F	*	_		Cursor down
7F		DEL	DEL	Delete character at cursor

Note: Those keys marked with asterisk cancel insert mode when editor is in insert mode.

BASIC COMMAND

AUTO [< line number >] [, < increment >]
Generate line numbers automatically.

BEEP
Make a beep sound.

BLOAD " [< device descriptor >] [< filename >] " [, R] [,< load address >]
Load a machine language program into memory from the specified device.

BSAVE " [< device descriptor >] <filename >", < start address > , < end address > [, < execution address >]
Save a memory image at the specified memory location to the device.

CLEAR [< string space > [, < highest location >]]
Set all numeric variables to zero, all string variables to null and close all open files and optionally, set the end of memory.

CLOAD [" < filename > "]
Load a BASIC program file from the cassette.

CLOAD? [" < filename > "]
Verify a BASIC program on cassette with one in memory

COLOR [< foreground >] [, < background >] [, < border >]
Set colors for the screen display.

CONT
Continue program execution after break or stop.

CSAVE " < filename > "
Save a BASIC program currently resided in memory to the cassette tape.

DELETE [< line number >] [– < line number >]
Delete program lines.

KEY < function key # > , " < string > "
Set a string to a function key.

KEY LIST
List the contents of all the programmable function keys.

KEY ON
Turn on function key display on the 24th line of text screen.

LIST [< line number >] [– < line number >]
List all or part of the program on the screen.

LLIST [< line number >] [– < line number >]
List all or part of the program on the printer.

LOAD " [< device descriptor >] < filename > "
Load a BASIC program (an ASCII file) from the device.

MAXFILES = n
Specify the maximum number of files opened at a time.

MERGE " [< device descriptor >] < filename > "
Merge the lines from an ASCII program into the program currently in memory.

MOTOR ON/OFF
Turn cassette motor on or off.

NEW
Delete entire program from working memory and reset all variables.

RENUM [< new number >] [, < old number >] [, < increment >]
Renumber program lines.

RUN [< line number >]
Execute a program.

SAVE " < device descriptor > < filename > "
Save the program in memory with name filename as an ASCII file.

SOUND ON/OFF
Turn sound of the cassette audio on or off.

TRON
Turn on trace for program execution.

TROFF
Turn trace off.

WIDTH n
Set the display width on screen during text mode. n may be 1 ∽ 40.

BASIC STATEMENT

CLOSE [#] < filenumber > [, < filenumber > .]
Close the channel and release the buffer associated with it.

DATA < list of constants >
Store the numeric and/or string constants that are accessed by the READ statement(s).

DEF FN < name > [< parameter list >] = < function definition >
Define and name an arithmetic or string function.

DEF USR n = < address >
Define the entry address for the nth machine language routine.

DEFINT < ranges of letters >
DEFSNG < ranges of letters >
DEFSTR < ranges of letters >
Declare variable type as integer, single precision, double precision or string.

DIM < list of subscripted variables >
Specify the maximum values for array variable subscripts and allocate storage accordingly.

DRAW < string >
Draw a figure according to the Graphic Macro Language.

END
Terminate program execution, close all files and return to command level.

ERASE < list of array variables >
Eliminate arrays from a program.

ERROR n
Generate error message of code n.

FOR < variable > = x TO y [STEP z]
NEXT [< variable >]
Allow a series of instructions to be performed in a loop a given number of times.

GOSUB < line number >
RETURN [< line number >]
Branch to a subroutine and then return from it. If < line number > after RETURN is not specified, return routine to the statement following the last GOSUB executed.
GOTO < line number >
Branch out of the normal program sequence to the specified line number.

IF < exp > THEN < statement/line > [ELSE < statement/line >]
Make a decision regarding program flow based on the result returned by an expression.

LET < variable > = < exp >
Assign value of an expression to a variable. LET may be omitted.

INPUT [" < prompt string > " ;] < list of variables >
Allow input from keyboard during program execution.

INPUT # < filenumber > , < variable list >
Read data items from the specified channel and assign them to program variables.

INPUT $ (n , [#] < filenumber >)
Return a string of n characters, read from the file.

LINE INPUT [" < prompt string > " ;] < string variable >
Input an entire line (up to 254 characters) to a string variable.

LINE INPUT # < filenumber > , < string variable >
Read an entire line (up to 254 characters) from a sequential file to a string variable.

L[illegible] < list of exp >]
[illegible] data at the printer.

LPR[illegible]ING " < string > " ; < list of exp >
[illegible]ata at the printer using a specified format

MIDS (< [illegible]ing > , n [, m]) = < string >
Replace a portion of the first string exp with the second string exp starting in the first string s nth character with m number of characters.

OPEN " < device descriptor > < filename > " [FOR < mode >] AS [#]< filenumber >
Allocate a buffer for I/O and set the mode that will be used with the buffer.

OUT < port > < byte >
Send a byte to a machine output port.

PLAY < string for voice 1 > [, < string for voice 2 >] [, < string for voice 3 >]
Play music according to Music Macro Language.

POKE < memory address > , < byte >
Write a byte into a memory location.

PRINT [< exp >]
Output data to the terminal.

PRINT # < filenumber > , < exp >
Write data to the specified channel.

PRINT USING " < string > " ; < list of exp >
Print numeric or string expression using a specified format.

PRINT # < filenumber > , USING " < string > " ; < list of exp >
Write data to the specified channel using a specified format.

READ < list of variables >
Read values from a DATA statement and assign them to variables

REM < remark >
Allow explanatory remark to be inserted in a program.

RESTORE [< line number >]
Allow DATA statements to be reread from a specified line.

RESUME [0]
Resume execution at the statement causing an error.

RESUME < line number >
Resume execution at the specified line number.

RESUME NEXT
Resume execution at the statement immediately following the one causing an error.

SOUND < register of PSG > , < value >
Put a value into a register to control the Programmable Sound Generator.

STOP
Terminate program execution and return to command level. The "Break in < line number >" message is printed.

SWAP < variable > , < variable >
Exchange the value of two variables.

WAIT < port > , < mask > [, < select >]
Suspend program execution read input at port until [< input bit >XOR < select > AND < mask >] returns a non-zero value.

INTERRUPT CONTROL COMMAND AND STATEMENT

INTERVAL ON/OFF/STOP
To enable, disable, or terminate the BASIC timer interrupt trapping.

KEY n ON/OFF/STOP
To enable, disable, or terminate interrupts caused by a function key.

ON ERROR GOSUB < linenumber >
Define the line number of the subroutine to execute when an error has been detected.

ON ERROR GOTO < linenumber >
Enable error trapping and specify the first line of the error handling subroutine.

ON < exp > GOTO < list of linenumbers >
ON < exp > GOSUB < list of linenumbers >
Branch to one of several specified line numbers, depending on the value returned when an expression is evaluated.

ON INTERVAL = < exp > GOSUB < list of linenumbers >
Set the line number to execute at every other machine interrupt cycle (60 per second) specified by < exp >.

ON KEY (n) GOSUB < list of linenumbers >
Specify the line number corresponding to the line offset n in the statement to execute whenever a function key number n has been depressed.

ON STRIG GOSUB < linenumber >
Define the starting line of the subroutine employed when any of the joystick button is depressed. (0 for space bar, 1 for Joystick 1 or 2 for Joystick 2)

ON SPRITE GOSUB < linenumber >
Define jump address when sprites collision occurs.

ON STOP GOSUB < linenumber >
Define jump address when the CTRL + STOP keys are pressed

STOP ON/OFF/STOP
To enable, disable, or terminate CTRL + STOP trapping

STRIG ON/OFF/STOP
To enable, disable, or terminate joystick button or space bar trapping.

SPRITE ON/OFF/STOP
To enable, disable, or terminate the interrupt generated when two or more sprites collide

GRAPHIC COMMAND AND STATEMENT

CLS
Clear graphic screen.

CIRCLE (x, y), < radius > [, < color > [, < start > , < end >] [, < aspect ratio >]
Draw an ellipse with a center, radius, foreground color, starting and ending angles, height to width ratio as specified.

COLOR [< foreground >] , [< background >] , [< border >]
Set colors for the screen.

DRAW < string >
Draw figure on the screen according to the Graphic Macro Language.

GET [(x1, y1) – (x2, y2)] [, < color >] [, B [F]]
Read points from the graphic screen or from defiend area of the graphic screen. Array must be dimensioned large enough to hold the data.

LINE [(x1, y1)] – (x2, y2) [, [< color >] [, B [F]]
Draw straight line connecting the two coordinate pairs specified or if B [F] is present draw or fill rectangle

LOCATE x, y
Position the graphic cursor to the starting coordiante pointed to by x, y; can used with LINE, POINT, PRINT

PAINT (x, y) [, < color > [, < low res mode border attribute >]]
Fill in an arbitrary graphic figure of the specified fill color. Paint must not have border for high res. while border must be specified in low res.

POINT (x, y)
Read the color of a pixel in the graphic mode.

PSET (x, y) [, < color >]
Draw a dot at the assigned position on the screen using the foreground color or < color > if specified.

PRESET (x, y) [, < color >]
Same as PSET except draw in background color if < color > not specified.

PUT (x, y), < arrayname > [, < operation >]
Output graphic patterns in the array to assigned position on the screen, operation can be:
PSET: output pattern as is
PRESET: reverse pattern foreground/background color
AND: combine graphic pattern color with screen pattern
OR: graphic pattern overlapping the screen data
XOR: perform XOR with screen data; if the matching pixel from the array and the screen are the same then that pixel will be displayed in background color else it will be displayed in background color else it will be displayed in the foreground color

PUT SPRITE < sprite plane > , (x, y) [, < color >] [, < pattern no >]
Set up sprite attribute, when a field is omitted, the current value is used, see SPRITE$.

SCREEN [< exp 1 >] [, < exp 2 >]
Use < exp 1 > to select graphic mode: 1 for high res.; 2 for low res. < exp 2 > selects the size of sprite (if used).

SPRITE$ (n) = < string >
Define a pattern as sprite, the number of string character depends on the sprite size, n must be less than 256 when size of sprite is 0 or 1 (8 bytes), less than 64 when size of sprite is 2 or 3 (32 bytes).

SPRITE$ (n)
Return either an 8 byte character string or 32 byte character string of the sprite number n depending on the size of sprite selected.

VPEEK (< vram address >]
Return byte value from the specified location in video ram.

VPOKE < vram address > , < byte >
Put a byte into video ram address specified.

GRAPHIC MACRO LANGUAGE (GML)

The followings are used in conjuction with DRAW command. "n" is a number.

U n	Move up
D n	Move down
L n	Move left
R n	Move right
E n	Move diagonally up and right
F n	Move diagonally down and right
G n	Move diagonally down and left
H n	Move diagonally up and left
S n	Set scaling factor
B n	Move without plotting
M x, y	Move absolute or relative distance If x has a plus sign (+) or a minus sign (–) in front of it, it is relative. Otherwise, it is absolute.
N n	Move and return to original position
A n	Set the angle. n

n	0	1	2	3
deg	0	90	180	270

C n	Set color number
X <string>	Execute string, must be terminated by a semicolon.

SOUND COMMAND AND STATEMENT

SOUND statement is used to directly control the various capabilities of the Programmable Sound Generator (PSG) It takes the form of:
SOUND <register of PSG > < value >

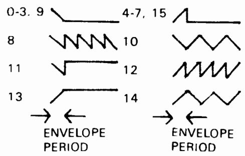
The functions of the 13 PSG registers are given below

PSG Register	Function
0, 1	Tone of channel A
2, 3	Tone of channel B
4, 5	Tone of channel C
6	Noise generator
7	Mixer

B7	B6	B5	B4	B3	B2	B1	B0
Not used		Noise Channel			Tone Channel		
		C	B	A	C	B	A

Bit logical value: 1 to disable and 0 to enable

8	Bit 0 to 3:amplitude control of channel A Bit 4: fixed amplitude control
9	Bit 0 to 3:amplitude control of channel B Bit 4· fixed amplitude control
10	Bit 0 to 3:amplitude control of channel C Bit 4· fixed amplitude control
11, 12	Envelope period control range
13	Envelope cycle/shape control



MUSIC MACRO LANGUAGE

The PLAY command is used in conjunction with the following character string to play a melody.

String	Function
A to G [# or +][–]	Play the indicated note in the current octave. A number sign (#) or plus sign (+) afterwards indicates a sharp, a minus sign (–) indicates a flat
L n	Set the length of the note. n may range from 1 to 64.
M n	Modulation sets period of envelope n may range from 1 to 65535.
N n	Play note n. n may range from 0 to 96 in 7 octaves 7. n = 0 means rest.
O n	Octave Set the octave of the notes to be played. Default is 4.
R n	Pause or rest n may range 1 to 64.
S n	Set shape of noise output n can be 0 to 15.
T n	Tempo Set the number of quarter notes in a minute. n ranges from 32 to 255 Default is 120
V n	Volume. Set the volume of output n may range from 0 to 15 The default is 8.
	Dot or period after each note causes it to be played 3/2 times the period determined by L times T
X	Execute specified string.

PRINT USING FORMAT FIELD SPECIFIER

String or numeric expression can be printed using a specified format.

Specifier	Possible Digit	Field Character	Definition
NUMERIC			
#	1	1	Numeric field
	0	1	Decimal point
+	0	1	Print leading or trailing sign. Positive numbers will have "+", negative numbers will have "#"
–	0	1	Trailing sign. Print "–" if negative, otherwise blank.
**	2	2	Leading asterisk
$$	1	2	Floating dollar sign $ is placed in front of the leading digit.
**$	2	3	Asterisk and floating dollar sign.
	1	1	Use comma every three digits (left of decimal point only).
∧∧∧∧	0	4	Exponential format Number is aligned so leading digit is non-zero
—	0	1	Next character literal.
STRING			
|			Single character
/ /			< 2 + number of spaces > character field
&			Variable length field.

BASIC FUNCTION

Function	Action
ASC (< string >)	Return the ASCII value of the first character of string.
BIN$ (n)	Return a string which represents the binary value of n.
CHR$ (I)	Return a character having the ASCII code I.
FRE (< exp >)	Returns remaining memory free space.
HEX$ (< exp >)	Convert a number to a hexadecimal string.
INKEY$	Return a one character string input from keyboard or null string if no input.
INPUT$ (length [. [#] m])	Return a string having specified length read from the console or a file.
LEFT$ (< string > , < length >)	Return a length of leftmost characters of the string expression.
LEN (< string >)	Return the number of characters in string.
MID$ (< string > . < start > [. < length >])	Return characters from the middle of the string starting at the position specified to the end of the string or for the specified length.
OCT$ (< exp >)	Convert a number to an octal string.
RIGHT$ (< string > < length >)	Return rightmost characters of the string expression.
SPACE$ (I)	Return a string of I spaces.
STR$ (< exp >)	Convert a numeric expression to a string
STRING$ (< length > . < string >)	Return a string with specified length containing first character of string.
STRING$ (< length>, I)	Return string with specified length containing first character with ASCII code I.
VAL (< string >)	Convert the string representation of a number to its numeric value.
CVI (< string >)	Convert a 2-character string to an integer.
CVS (< string >)	Convert a 4-character string to a single precision number.
CVD (< string >)	Convert an 8-character string to a double precision number.
MKI$ (< value >)	Convert an integer to a 2-character string.
MKS$ (<value >)	Convert a single precision value to a 4-character string.
MKD$ (< value >)	Convert a double precision value to an 8-character string.
BASE (n)	Current base address for each table.
CSRLIN	Return the vertical coordinate of the cursor.
EOF (< filenumber >)	Return (–1) if the end of a sequential file has been reached. Otherwise, return (0).
ERL	Error line number.
ERR	Error code number.
PAD (n)	Return various status of touch pad.
PEEK (I)	Read a byte from memory location I.
POS (n)	Return the current horizontal position of cursor. n is a dummy argument.
SPC (I)	Print I blanks on the screen.
STICK (n)	Return the direction of a joystick.
STRIG (n)	Return the status of a trigger button of a joystick
TAB (I)	Space to position I on the console
TIME	An unsigned integer generated from the system internal timer.
USR [< digit >] (argument)	Call the user's machine language subroutine with the specified argument.
VARPTR (< variable >) VARPTR (# < filenumber >)	Return the address of the first byte of data identified with < variable >
VDP (n)	For n = 0 to 7, this specifies the current value of VDP's write only register. For n = 8, it specifies the status register of VDP

OPERATOR

Symbol	Function
=	Assignment or equality sign
–	Negation or subtraction
+	Addition or string concatenation
*	Multiplciation or asterisk
/	Division (floating point result)
∧	Exponentiation symbol
\	Integer division
MOD	Integer modulus
NOT	One's complement
AND	Bitwise AND
OR	Bitwise OR
XOR	Bitwise exclusively OR
EQV	Bitwise equivalence
IMP	Bitwise implication
= < , > , > = , = < , > = , = < , <> , > <	Relational tests (result is either true (–1) or false (0))

The precedence of operators is
(1) Expressions in parentheses
(2) Exponentiation
(3) Negation
(4) Multiplication or Division
(5) Integer Division
(6) MOD
(7) Addition or Subtraction
(8) Relational operators
(9) NOT
(10) AND
(11) OR
(12) XOR
(13) IMP
(14) EQV

ARITHMETIC FUNCTION

Function	Action
ABS (< exp >)	Absolute value of expression.
ATN (< exp >)	Arctangent of the expression.
CDBL (< exp >)	Convert the expression to a double precision number.
CINT (< exp >)	Convert the expression to an integer.
COS (< exp >)	Cosine of the expression (in radians).
CSNG (< exp >)	Convert the expression to a single precision number
EXP (< exp >)	Raise the constant e to the power of expression.
FIX (< exp >)	Return truncated integer of expression.
FRE (< exp >)	Give memory free space not used by BASIC.
INT (< exp >)	Evaluate the expression for the largest integer calculated.
LOG (< exp >)	Give the natural logarithm of the expression.
RND ([< exp >)	Generate a random number [< exp >] < 0 Seed new sequence [< exp >] = 0 Return previous random number [≤ exp >] >0 or omitted Return new random number
SGN (< exp >)	1 if expression > 0 0 if expression = 0 –1 if expression < 0
SIN (< exp >)	Sine of the expression (in radians)
SQR (< exp >)	Square root of expression
TAN (< exp >)	Tangent of the expression (in radians)

Angle is expressed or returned in radians.