ATARI® 400/800™

BASIC REFERENCE MANUAL

ATARI®
A Warner Communications Company

ERROR CODES

ERROR CODE	ERROR CODE MESSAGE
2	Memory Insufficient
3	Value Error
4	Too Many Variables
5	String Length Error
6	Out of Data Error
7	Number greater than 32767
8	Input Statement Error
9	Array or String DIM Error
10	Argument Stack Overflow
11	Floating Point Overflow/ Underflow Error
12	Line Not Found
13	No Matching FOR Statement
14	Line Too Long Error
15	GOSUB or FOR Line Deleted
16	RETURN Error
17	Garbage Error
18	Invalid String Character

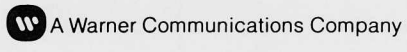
Note: The following are INPUT/OUTPUT errors that result during the use of disk drives, printers, or other accessory devices. Further information is provided with the auxiliary hardware.

19	LOAD program Too Long
20	Device Number Larger
21	LOAD File Error
128	BREAK Abort
129	IOCB
130	Nonexistent Device
131	IOCB Write Only
132	Invalid Command
133	Device or File not Open
134	BAD IOCB Number
135	IOCB Read Only Error
136	EOF
137	Truncated Record
138	Device Timeout
139	Device NAK
140	Serial Bus
141	Cursor Out of Range

ERROR CODE	ERROR CODE MESSAGE
142	Serial Bus Data Frame Overrun
143	Serial bus data frame checksum error
144	Device done error
145	Read after write compare error
146	Function not implemented
147	Insufficient RAM
160	Drive number error
161	Too many OPEN files
162	Disk full
163	Unrecoverable system data I/O error
164	File number mismatch
165	File name error
166	POINT data length error
167	File locked
168	Command invalid
169	Directory full
170	File not found
171	POINT invalid

For explanation of Error Messages see Appendix B.

BASIC REFERENCE MANUAL



Atari Software Support Group
P.O. Box 427
Sunnyvale, CA 94086

Printed in USA

CONTENTS

PREFACE

This manual assumes the user has read the *Atari BASIC — A Self-Teaching Guide* or some other book on BASIC. This manual is not intended to "teach" BASIC. It is a reference guide to the commands, statements, functions, and special applications of Atari® BASIC.

The programs and partial programming examples used in this manual are photostats of listings printed on the Atari 820™ Printer. Some of the special symbols in the Atari character set do not appear the same on the printer; e.g., the clear screen symbol "↰" appears as a "}". The examples in the text were chosen to illustrate a particular function — not necessarily "good" programming techniques.

Each of the sections contains groups of commands, functions, or statements dealing with a particular aspect of Atari BASIC. For instance, Section 9 contains all the statements pertaining to Atari's unique graphics capabilities. The appendices include quick references to terms, error messages, BASIC keywords, memory locations, and the ATASCII character set.

As there is no one specified application for the Atari Personal Computer System, this manual is directed at general applications and the general user. Appendix H contains programs that illustrate a few of the Atari system's capabilities.

1

GENERAL INFORMATION

This section explains BASIC terminology, special notations, and abbreviations used in this manual, and the special keys on the ATARI 400™ and ATARI 800™ Personal Computer Systems keyboard. It also points to other sections where BASIC commands deal with specific applications.

TERMINOLOGY

BASIC: Beginner's **A**ll-purpose **S**ymbolic **I**nstruction **C**ode.

BASIC Keyword: Any reserved word "legal" in the BASIC language. May be used in a statement, as a command, or for any other purpose. (See Appendix A for a list of all "reserved words" or keywords in ATARI BASIC.)

BASIC Statement: Usually begins with a keyword, like **LET, PRINT,** or **RUN.**

Constant: A constant is a value expressed as a number rather than represented by a variable name. For example, in the statement X = 100, X is a variable and 100 is a constant. (See **Variable**.)

Command String: Multiple commands (or program statements) placed on the same numbered line separated by colons.

Expression: An expression is any legal combination of variables, constants, operators, and functions used together to compute a value. Expressions can be either arithmetic, logical, or string.

Function: A function is a computation built into the computer so that it can be called for by the user's program. A function is *NOT* a statement; it is part of an expression. It is really a subroutine used to compute a value which is then "returned" to the main program when the subroutine returns. **COS** (Cosine), **RND** (random), **FRE** (unused memory space), and **INT** (integer) are examples of functions. In many cases the value is simply assigned to a variable (stored in a variable) for later use. In other cases it may be printed out on the screen immediately. See Section 6 for more on functions. Examples of functions as they might appear in programs are:

```
10 PRINT RND(0)
```

(print out the random number returned)

```
10 X=100+COS(45)
```

(add the value returned to 100 and store the total in variable X)

Logical Line: A logical line consists of one to three physical lines, and is terminated either by a RETURN or automatically when the maximum logical line limit is reached. Each numbered line in a BASIC program consists of one logical line when displayed on the screen. When entering a line which is longer than one physical line, the cursor will automatically go to the beginning of the next physical line when the end of the current physical line is reached. If RETURN is not entered, then both physical lines will be part of the same logical line.

Operator: Operators are used in expressions. Operators include *addition (+), subtraction (-), multiplication (*), division (/), exponentiation (^), greater than (>), less than (<), equal to (=), greater than or equal to (>=), less than or equal to (<=),* and *not equal to (<>).* The logical keywords **AND, NOT** and **OR** are also operators. The + and - operators can also be used as unary operators; e.g., -3. Do *not* put several unary operators in a row; e.g.,——3, as the computer will interpret it incorrectly.

Physical Line: One line of characters as displayed on a television screen.

String: A string is a group of characters enclosed in quotation marks. "ABRACADABRA" is a string. So are "ATARI MAKES GREAT COMPUTERS" and "123456789". A string is much like a constant, as it too, may be stored in a variable. A string variable is different, in that its name must end in the character **$**. For example, the string "ATARI 800" may be assigned to a variable called A**$** using (optional) **LET** like this:

```
10 LET A$="ATARI 800"          (note quotation marks)
                  OR
10 A$="ATARI 800"              (LET is optional; the
                                quotes are required.)
```

Quotation marks may *not* be used within a string. However, the closing quotation can be omitted if it is the last character on a logical line. (See Section 7 - **STRINGS**).

Variable: A variable is the name for a numerical or other quantity which may (or may not) change. Variable names may be up to 120 characters long. However, a variable name must start with an alphabetic letter, and may contain only capital letters and numerical digits. It is advisable *not* to use a keyword as a variable name or as the first part of a variable name as it may not be interpreted correctly. Examples of storing a value in a variable:

```
LETC123DUB=1.234
LETVARIABLE112=267.543
LETA=1
LETF5TH=6.5
LETTHISNO = 59.809
```

Note: **LET** is optional and may be omitted)

Variable Name Limit: ATARI BASIC limits the user to 128 variable names. To bypass this problem, use individual elements of an array instead of having separate variable names. BASIC keeps all references to a variable which has been deleted from a program, and the name still remains in the variable name table.

If the screen displays an ERROR-4 (Too Many Variables) message, use the following procedure to make room for new variable names:

```
LIST filespec
NEW
ENTER filespec
```

The LIST filespec writes the untokenized version of the program onto a disk or cassette. NEW clears the program and the table areas. The program is then re-entered, re-tokenized, and a new variable table is built. (The tokenized version is Atari BASIC's internal format. The untokenized versions in ATASCII which is the version displayed on the screen).

Arrays and Array Variables: An array is a list of places where data can be filed for future use. Each of these places is called an *element*, and the whole array or any element is an array variable. For example, define "Array A" as having 6 elements. These elements are referred to by the use of subscripted variables such as **A(2), A(3), A(4),** etc. A number can be stored in each element. This may be accomplished element by element (using the **LET** statement), or as a part of a **FOR/NEXT** loop (see Chapter 8).

Note: Never leave blanks between the element number in parentheses and the name of the array.

Correct	**Incorrect**
A(23)	**A (23)**
ARRAY(3)	**ARRAY (3)**
X123(38)	**X123 (38)**

SPECIAL NOTATIONS USED IN THIS MANUAL

Line Format: The format of a line in a BASIC program includes a line number (abbreviated to lineno) at the beginning of the line, followed by a statement keyword, followed by the body of the statement and ending with a line terminator command (RETURN key). In an actual program, the four elements might look like this:

	STATEMENT		
Line Number	**Keyword**	**Body**	**Terminator**
100	**PRINT**	**A/X * (Z+4.567)**	**RETURN**

Several statements can be typed on the same line provided they are separated by colons (:). See **IF/THEN** in Section 5, and Section 11.

Capital Letters: In this book, denote keywords to be typed by the user in upper case form exactly as they are printed in this text. Reverse-video characters will not work except in the case of the **RUN** command. Here are a few examples:

PRINT INPUT LIST END GOTO GOSUB FOR NEXT IF

Lower Case Letters: In this manual, lower case letters are used to denote the various classes of items which may be used in a program, such as variables (**var**), expressions (**exp**), and the like. The abbreviations used for these classes of items are shown in Table 1.1.

Items in Brackets: Brackets, [ ], contain optional items which may be used, but are not required. If the item enclosed in brackets is followed by three dots [**exp,...**], it means that *any* number of expressions may be entered, but none are required.

Items stacked vertically in braces: Items stacked vertically in braces indicate that any one of the stacked items may be used, but that only one at a time is permissible. In the example below, type either the **GOTO** or the **GOSUB**.

$$100\begin{Bmatrix}\text{GOTO}\\\text{GOSUB}\end{Bmatrix}2000$$

Command abbreviations in headings: If a command or statement has an abbreviation associated with it, the abbreviation is placed following the full name of the command in the heading; e.g., **LET (L.)**.

ABBREVIATIONS USED IN THIS MANUAL

The following table explains the abbreviations used throughout this manual:

TABLE 1.1 ABBREVIATIONS

avar	**Arithmetic Variable:** A location where a numeric value is stored. Variable names may be from 1 to 120 alphanumeric characters, but must start with an alphabetic character, and all alpha characters must be unreversed and upper case.
svar	**String Variable:** A location where a string of characters may be stored. The same name rules as avar apply, except that the last character in the variable name must be a **$**. String variables may be subscripted. See Section 7, **STRINGS.**
mvar	**Matrix Variable:** Also called a *Subscripted Variable.* An element of an array or matrix. The variable name for the array or matrix as a whole may be any legal variable name such as **A, X, Y, ZIP,** or **K.** The subscripted variable (name for the particular element) starts with the matrix variable, and then uses a number, variable, or expression in parentheses *immediately* following the array or matrix variable. For example, **A(ROW), A(1), A(X+1).**
var	**Variable:** Any variable. May be mvar, avar, or svar.
aop	**Arithmetic operator.**
lop	**Logical operator.**
aexp	**Arithmetic Expression:** Generally composed of a variable, function, constant, or two arithmetic expressions separated by an arithmetic operator.
lexp	**Logical Expression:** Generally composed of two arithmetic or string expressions separated by a logical operator. Such an expression evaluates to either a 1 (logical true) or a 0 (logical false). For example, the expression 1<2 evaluates to the value 1 (true) while the expression "LEMON" = "ORANGE" evaluates to a zero (false) as the two strings are not equal.

sexp **String Expression:** Can consist of a string variable, string literal (constant), or a function that returns a string value.

exp Any expression, whether sexp or aexp.

lineno **Line Number:** A constant that identifies a particular program line in a deferred mode BASIC program. Must be any integer from 0 through 32767. Line numbering determines the order of program execution.

adata **ATASCII Data:** Any ATASCII character excluding commas and carriage returns. (See Appendix C.)

filespec **File Specification:** A string expression that refers to a device such as the keyboard or to a disk file. It contains information on the type of I/O device, its number, a colon, an optional file name, and an optional filename extender. (See **OPEN**, Section 5.)

Example filespec: "D1:NATALIE.ED"

OPERATING MODES

Direct Mode: Uses no line numbers and executes instruction immediately after RETURN key is pressed.

Deferred Mode: Uses line numbers and delays execution of instruction(s) until the RUN command is entered.

Execute Mode: Sometimes called Run mode. After RUN command is entered, each program line is processed and executed.

Memo Pad Mode: A non-programmable mode that allows the user to experiment with the keyboard or to leave messages on the screen. Nothing written while in Memo Pad mode affects the RAM-resident program.

SPECIAL FUNCTION KEYS

Reverse (Inverse) Video key, or "ATARI LOGO KEY". Pressing this key causes the text to be reversed on the screen (dark text on light background). Press key a second time to return to normal text.

CAPS/LOWR **Lower Case key:** Pressing this key shifts the screen characters from upper case (capitals) to lower case. To restore the characters to upper case, press the SHIFT key and the CAPS/LOWR key simultaneously.

ESC **Escape key:** Pressing this key causes a command to be entered into a program for later execution.

Example: To clear the screen, you would enter:

10 PRINT " ESC CTRL CLEAR "

and press RETURN .

Escape is also used in conjunction with other keys to print special graphic control characters. See Appendix F and back cover for the specific keys and their screen-character representations.

BREAK **Break key:** Pressing this key during program execution causes execution to stop. Execution may be resumed by typing CONT followed by pressing RETURN .

SYSTEM RESET **System Reset key:** Similar to BREAK in that pressing this key stops program execution. Also returns the screen display to Graphics mode 0, clears the screen, and returns margins and other variables to their default values.

SET-CLR-TAB **Tab key:** Press SHIFT and the SET-CLR-TAB keys simultaneously to set a tab. To clear a tab, press the CTRL and SET-CLR-TAB keys simultaneously. Used alone, the SET-CLR-TAB advances the cursor to the next tab position. In Deferred mode, set and clear tabs by preceding the above with a line number, the command PRINT, a quotation mark, and press the ESC key.

Examples:

100 PRINT " ESC SHIFT SET-CLR-TAB "
200 PRINT " ESC CTRL SET-CLR-TAB "

Default tab settings are placed at columns 7, 15, 23, 31, and 39.

INSERT **Insert key:** Press the SHIFT and INSERT keys simultaneously to insert a line. To insert a single character, press the CTRL and INSERT keys simultaneously.

DELETE BACK S **Delete key: Press the** SHIFT and DELETE keys simultaneously to delete a line. To delete a single character, press CTRL and DELETE simultaneously.

DELETE BACK S **Back Space key:** Pressing this key replaces the character to the left of the cursor with a space and moves cursor back one space.

CLEAR **Clear key:** Pressing this key while holding down the SHIFT or CTRL key blanks the screen and puts the cursor in the upper left corner.

RETURN **Return key:** Terminator to indicate and end of a line of BASIC. Pressing this key causes a numbered line to be interpreted and added to a BASIC program RAM. An unnumbered line (in Direct mode) is interpreted and executed immediately. Any variables are placed in a variable table.

ARITHMETIC OPERATORS

The Atari Personal Computer System uses five arithmetic operators:

+ addition (also unary plus; e.g., +5)
– subtraction (also unary minus; e.g., –5)
* multiplication
/ division
^ exponentiation

LOGICAL OPERATORS

The logical operators consists of two types: *unary* and *binary*. The unary operator is **NOT**. The binary operators are:

AND Logical AND
OR Logical OR

Examples:

```
10 IF A=12 AND T=0 THEN PRINT "GOOD"
```
Both expressions must be true before GOOD is printed.

```
10 A=(C>1) AND (N<1)
```
If both expressions true, A = +1; otherwise A = 0.

```
10 A = (C+1) OR (N-1)
```
If either expression true, A = +1; otherwise A = 0.

```
10 A = NOT(C+1)
```
If expression is false, A = +1; otherwise A = 0.

The rest of the binary operators are relational.

< The first expression is less than the second expression.
> The first expression is greater than the second.
= The expressions are equal to each other.
<= The first expression is less than or equal to the second.
>= The first expression is greater than or equal to the second.
<> The two expressions are not equal to each other.

These operators are most frequently used in **IF/THEN** statements and logical arithmetic.

OPERATOR PRECEDENCE

Operations within the innermost set of parentheses are performed first and proceed out to the next level. When sets of parentheses are enclosed in another set, they are said to be "nested". Operations on the same nesting level are performed in the following order:

Highest precedence	<, >, =, <=, >=, <>	Relational operators used in string expressions. Have same precedence and are performed from left to right.
	-	Unary minus
	^	Exponentiation.
	* , /	Multiplication and division have the same precedence level and are performed from left to right.
	+, -	Addition and subtraction have the same precedence level and are performed from left to right.
	<, >, =, <=, >=, <>	Relational operations in numeric expressions have the same precedence level from left to right.
	NOT	Unary operator
	AND	Logical AND
Lowest precedence	OR	Logical OR

BUILT-IN FUNCTIONS

The section titled **FUNCTION LIBRARY** explains the arithmetic and special functions incorporated into Atari BASIC.

GRAPHICS

Atari graphics include 9 graphics modes. The commands have been designed to allow maximum flexibility in color choice and pattern variety. Section 9 explains each command and gives examples of the many ways to use each.

SOUND AND GAMES CONTROLLERS

The Atari Personal Computer is capable of emitting a large variety of sounds, including simulated explosions, electronic music, and "raspberries." Section 10 defines the commands for using the SOUND function and for controlling paddle, joystick, and keyboard controllers.

WRAPAROUND AND KEYBOARD ROLLOVER

The ATARI Personal Computer System has screen wraparound thus allowing greater flexibility. It also allows the user to type one key ahead. If the user presses and holds any key, it will begin repeating after ½ second.

ERROR MESSAGES

If a data entry error is made, the screen display shows the line reprinted preceded by the message ERROR- and the offending character is highlighted. After correcting the character in the original line, delete the line containing the ERROR- *before* pressing RETURN. Appendix B contains a list of all the error messages and their definitions.

2

COMMANDS

Whenever the cursor (□) is displayed on the screen, the computer is ready to accept input. Type the command (in either Direct or Deferred mode), and press RETURN. This section describes the commands used to clear computer memory and other useful control commands.

The commands explained in this section are the following:

BYE	**NEW**
CONT	**REM**
END	**RUN**
LET	**STOP**
LIST	

BYE (B.)

Format: BYE
Example: BYE

The current function of the BYE command is to exit BASIC and put the computer in Memo Pad mode. This allows the user to experiment with the keyboard or to leave messages on the screen without disturbing any BASIC program in memory. To return to BASIC, press SYSTEM RESET.

CONT (CON.)

Format: CONT
Example: CONT

Typing this command followed by a RETURN causes program execution to resume. If a BREAK, STOP, or END is encountered, the program will stop until CONT RETURN is entered. Execution resumes at the next sequential *line number* following the statement at which the program stopped.

Note: If the statement at which the program is halted has other commands on the same numbered line which were not executed at the time of the BREAK, STOP, or END, they will *not* be executed. On CONT, execution resumes at the next numbered line. A loop may be incorrectly executed if the program is halted before the loop completes execution.

This command has no effect in a Deferred mode program.

END

Format: END
Example: 1000 END

This command terminates program execution and is used in Deferred mode. In Atari BASIC, an END is not required at the end of a program. When the end of the program is reached, Atari BASIC automatically closes all files and turns off sounds (if any). END may also be used in Direct mode to close files and turn off sounds.

LET (LE.)

Format: [LET] var = exp
Example: LET X = 3.142 * 16
LET X = 2

This statement is optional in defining variables. It can just as easily be left out of the statement. It may be used, however, to set a variable name equal to a value.

LIST (L.)

Format: LIST [lineno [, lineno]]
LIST [filespec [,lineno [,lineno]]]

Examples:

```
LIST
LIST 10
LIST,10,100
LIST "P.",20,100

LIST "P"
LIST "D:DEMO.LST"
```

This command causes the computer to display the source version of all lines currently in memory if the command is entered without line number(s), or to display a specified line or lines. For example, LIST 10,100 RETURN displays lines 10 through 100 on the screen. If the user has not typed the lines into the computer in numerical order, a LIST will automatically place them in order.

Typing L."P will print the RAM-resident program on the printer.

LIST can be used in Deferred mode as part of an error trapping routine (See TRAP in Section 4).

The LIST command is also used in recording programs on cassette tape. The second format is used and a filespec is entered. (See Section 5 for more details on peripheral devices.) If the entire program is to be listed on tape, no line numbers need be specified.

Example: LIST "C1"
1000 LIST "C1"

NEW

Format: NEW
Example: NEW

This command erases the program stored in RAM. Therefore, before typing NEW, either SAVE or CSAVE any programs to be recovered and used later. NEW clears BASIC's internal symbol table so that no arrays (See Section 8) or strings (See Section 7) are defined. Used in Direct mode.

REM (R. or . SPACE)

Format: REM text
Example: 10 REM ROUTINE TO CALCULATE X

This command and the text following it are for the user's information only. It is ignored by the computer. However, it is included in a LIST along with the other numbered lines. Any statement on the same numbered line which occurs after a REM statement will be ignored.

RUN (RU.)

Format: RUN [filespec]
Examples: RUN
RUN "D:MENU"

This command causes the computer to begin executing a program. If no filespec is specified, the current RAM-resident program begins execution. If a filespec is included, the computer retrieves the specified, tokenized program from the specified file and executes it.

All variables are set to zero and all open files and peripherals are closed. All arrays, strings, and matrices are eliminated and all sounds are turned off. Unless the TRAP command is used, an error message is displayed if any error is detected during execution and the program halts.

RUN can be used in Deferred mode.

Examples:
```
10 PRINT "OVER AND OVER AGAIN."
20 RUN
```

Type RUN and press RETURN. To end, press BREAK.

To begin program execution at a point other than the first line number, type **GOTO** followed by the specific line number, then press RETURN.

STOP (STO.)

Format: STOP
Example: 100 STOP

When the STOP command is executed in a program, BASIC displays the message **STOPPED AT LINE** _____, terminates program execution, and returns to Direct mode. The **STOP** command does not close files or turn off sounds, so the program can be resumed by typing CONT RETURN.

NOTES

3

EDIT FEATURES

In addition to the special function keys described in Section 1, there are cursor control keys that allow immediate editing capabilities. These keys are used in conjunction with the SHIFT or CTRL keys.

The following key functions are described in this section:

CTRL	CTRL INSERT	CTRL 1
SHIFT	CTRL DELETE	CTRL 2
CTRL ↑	SHIFT INSERT	CTRL 3
CTRL ↓	SHIFT DELETE	BREAK
CTRL →	SHIFT CAPS/LOWR	ESC
CTRL ←		

SCREEN EDITING

The keyboard and display are logically combined for a mode of operation known as screen editing. Each time a change is completed on the screen, the RETURN key must be pressed. Otherwise, the change is not made to the program in RAM.

Example:

```
10 REM PRESS RETURN AFTER LINE EDIT
20 PRINT :PRINT
30 PRINT "THIS IS LINE 1 ON THE SCREEN."
```

To delete line 20 from the program, type the line number and press the RETURN key. Merely deleting the line from the screen display does **not** delete it from the program.

The screen and keyboard as I/O devices are described in Section 5.

CTRL

Control key. Striking this key in conjunction with the arrow keys produces the cursor control functions that allow the user to move the cursor anywhere on the screen without changing any characters already on the screen. Other key combinations control the setting and clearing of tabs, halting and restarting program lists, and the graphics control symbols. Striking a key while holding the CTRL key will produce the upper-left symbol on those keys having three functions.

SHIFT

Shift key: This key is used in conjunction with the numeric keys to display the symbols shown on the upper half of those keys. It is also used in conjunction

with other keys to insert and delete lines, return to a normal, upper case letter display, and to display the function symbols above the subtraction, equals, addition, and multiplication operators as well as the brackets, [ ], and question mark,?.

DOUBLE-KEY FUNCTIONS

Cursor Control Keys

CTRL ↑	Moves cursor up one physical line without changing the program or display.
CTRL →	Moves cursor one space to the right without disturbing the program or display.
CTRL ↓	Moves cursor down one physical line without changing the program or display.
CTRL ←	Moves cursor one space to the left without disturbing the program or display.

Like the other keys on the Atari keyboard, holding the cursor control keys for more than ½ second causes the keys to repeat.

Keys Used With

CTRL INSERT	Inserts one character space.
CTRL DELETE	Deletes one character or space.
CTRL 1	Stops temporarily and restarts screen display without "breaking out" of the program.
CTRL 2	Rings buzzer.
CTRL 3	Indicates end-of-file.

Keys Used With SHIFT

SHIFT INSERT	Inserts one physical line.
SHIFT DELETE	Deletes one physical line.
SHIFT CAPS/LOWR	Returns screen display to upper-case alphabetic characters.

Special Function Keys

BREAK	Stops program execution or program list, prints a READY on the screen, and displays cursor.
ESC	Allows commands normally used in Direct mode to be placed in Deferred mode; e.g., In Direct mode, CTRL CLEAR clears the screen display. To clear the screen in Deferred mode, type the following after the program line number. Press ESC then press CTRL and CLEAR together.

PRINT " ESC CTRL CLEAR "

4

PROGRAM STATEMENTS

This section explains the commands associated with loops, conditional and unconditional branches, error traps, and subroutines and their retrieval. It also explains the means of accessing data and the optional command used for defining variables.

The following commands are described in this section:

FOR, TO, STEP/NEXT	**IF/THEN**	**POP**
GOSUB/RETURN	**ON, GOSUB**	**RESTORE**
GOTO	**ON, GOTO**	**TRAP**

FOR (F.), TO, STEP/NEXT (N.)

Format: FOR avar = aexp1 TO aexp2 [STEP aexp3]
NEXT avar

Examples:
```
FOR X = 1 TO 10
NEXT X
FOR Y = 10 TO 20 STEP 2
NEXT Y
FOR INDEX = Z TO 100 * Z
NEXT INDEX
```

This command sets up a loop and determines how many times the loop is executed. The loop variable (avar) is initialized to the value of aexp1. Each time the NEXT avar statement is encountered, the loop variable is incremented by the aexp3 in the STEP statement. The aexp3 can be positive or negative integers, decimals, or fractional numbers. If there is *no* STEP aexp3 command, the loop increments by one. When the loop completes the limit as defined by aexp2, it stops and the program proceeds to the statement immediately following the NEXT statement; it may be on the same line or on the next sequential line.

Loops can be nested, one within another. In this case, the innermost loop is completed before returning to the outer loop. The following example illustrates a nested loop program.

```
10 FOR X=1 TO 3
20 PRINT "OUTER LOOP"
30 Z=0
40 Z=Z+2
50 FOR Y=1 TO 5 STEP Z
60 PRINT "    INNER LOOP"
70 NEXT Y
80 NEXT X
90 END
```

Figure 4-1. Nested Loop Program

In Figure 4-1, the outer loop will complete three passes (X = 1 to 3). However, before this first loop reaches its NEXT X statement, the program gives control to the inner loop. Note that the NEXT statement for the inner loop must precede the NEXT statement for the outer loop. In the example, the inner loop's number of passes is determined by the STEP statement (**STEP Z**). In this case, **Z** has been defined as 0, then redefined as **Z+2.** Using this data, the computer must complete three passes through the inner loop before returning to the outer loop. The aexp3 in the step statement could also have been defined as the numerical value 2.

The program run is illustrated in Figure 4-2.

```
OUTER LOOP
    INNER LOOP
    INNER LOOP
    INNER LOOP
OUTER LOOP
    INNER LOOP
    INNER LOOP
    INNER LOOP
OUTER LOOP
    INNER LOOP
    INNER LOOP
    INNER LOOP
```

Figure 4-2. Nested Loop Execution

The return address for the loops are placed in a special group of memory addresses referred to as a *stack*. The information is "pushed" on the stack and when used, the information is "popped" off the stack (see **POP**.)

GOSUB (GOS.) RETURN (RET.)

Format: GOSUB lineno
lineno
RETURN

Example: 100 GOSUB 2000
2000 PRINT "SUBROUTINE"
2010 RETURN

A subroutine* is a program or routine used to compute a certain value, etc. It is generally used when an operation must be replaced several times within a program sequence using the same or different values. This command allows the user to "call" the subroutine[1], if necessary. The last line of the subroutine must contain a RETURN statement. The RETURN statement goes back to the physical line *following* the GOSUB statement.

Like the preceding **FOR/NEXT** command, the **GOSUB/RETURN** command uses a stack for its return address. If the subroutine is not allowed to complete normally; e.g., a GOTO lineno before a RETURN, the GOSUB address must be "popped" off the stack (see **POP**) or it could cause future errors.

* Generally, a subroutine can do anything that can be done in a program. It is used to save memory and program-entering time, and to make programs easier to read and debug.

To prevent accidental triggering of a subroutine (which normally follows the main program), place an **END** statement preceding the subroutine. The following program demonstrates the use of subroutines.

```
10 PRINT "}"                                    (Clear screen)
20 REM EXAMPLE USE OF GOSUB/RETURN
30 X=100
40 GOSUB 1000
50 X=120
60 GOSUB 1000
70 X=50
80 GOSUB 1000
90 END
1000 Y=3*X
1010 X=X+Y
1020 PRINT X,Y
1030 RETURN
```

Figure 4-3. GOSUB/RETURN Program Listing

In the above program, the subroutine, beginning at line 1000, is called three times to compute and print out different values of X and Y. Figure 4-4 illustrates the results of executing this program.

```
 400          300
 480          360
 200          150
```

Figure 4-4. GOSUB/RETURN Program Run

GOTO (G.)

Format: {GO TO / GOTO} aexp

Examples: 100 GOTO 50
500 GOTO (X+Y)

The GOTO command is an unconditional branch statement just like the GOSUB command. They both immediately transfer program control to a target line number or arbitrary expression. However, using anything other than a constant will make renumbering the program difficult. If the target line number is non-existent, an error results. Any GOTO statement that branches to a preceding line may result in an "endless" loop. Statements following a GOTO statement will not be executed. Note that a conditional branching statement (see **IF/THEN**)can be used to break out of a GOTO loop. The following program illustrates two uses of the GOTO command.

```
10 PRINT
20 PRINT :PRINT "ONE"
30 PRINT "TWO"
40 PRINT "THREE"
50 PRINT "FOUR"
60 PRINT "FIVE"
65 GOTO 100
70 PRINT "$$$$$$$$$$$$$$$$"
80 PRINT "%%%%%%%%%%%%%%%%"
90 PRINT "????????????????"
95 END
100 PRINT "SIX"
110 PRINT "SEVEN"
120 PRINT "EIGHT"
130 PRINT "NINE"
140 PRINT "TEN"
150 GOTO 70
```

Figure 4-5. GOTO Program Listing

Upon execution, the numbers in the above listing will be listed first followed by the three rows of symbols. The symbols listed on lines 70, 80, and 90 are ignored temporarily while the program executes the GOTO 100 command. It proceeds with the printing of the numbers "SIX" through "TEN", then executes the second GOTO statement which transfers program control back to line 70. (This is just an example. This program could be rewritten so that no GOTO statements were used.) The program, when executed, looks like the following:

```
ONE
TWO
THREE
FOUR
FIVE
SIX
SEVEN
EIGHT
NINE
TEN
$$$$$$$$$$$$$$$$
%%%%%%%%%%%%%%%%
????????????????
```

Figure 4-6. GOTO Program Run

IF/THEN

Format: IF aexp THEN { lineno | statement [:statement...] }

Examples:
```
IF X = 100 THEN 150
IF A$ = "ATARI" THEN 200
IF AA = 145 and BB = 1 THEN PRINT AA, BB
IF X = 100 THEN X = 0
```

The IF/THEN statement is a conditional branch statement. This type of branch occurs only if certain conditions are met. These conditions may be either arithmetical or logical. If the aexp following the IF statement is true (non-zero), the program executes the THEN part of the statement. If, however, the aexp is false (a logical 0), the rest of the statement is ignored and program control passes to the next numbered line.

In the format, IF aexp THEN lineno, lineno must be a constant, not an expression and specifies the line number to go to if the expression is true. If several statements occur after the THEN, separated by colons, then they will be executed if *and only* if the expression is true. Several IF statements may be nested on the same line. For example:

```
100 IF X=5 THEN IF Y=3 THEN R=9:GOTO200
```

The statements R=9: GOTO 100 will be executed only if X=5 and Y=3. The statement Y=3 will be executed if X=5.

The following program demonstrates the IF/THEN statement.

```
5 GRAPHICS 0:? :? "                IF DEMO"
10 ? :? "ENTER A";:INPUT A
20 IF A=1 THEN 40:REM MULTIPLE STATEMENT
S HERE WILL NEVER BE EXECUTED!!
30 ? :? "A IS NOT 1.  EXECUTION CONTINUE
S HERE WHEN THE EXPRESSION IS FALSE."
40 IF A=1 THEN ? :? "A=1":? "YES, IT IS
REALLY 1.":REM MULTIPLE STATEMENTS HERE
WILL BE EXECUTED ONLY IF A=1!!
50 ? :? "EXECUTION CONTINUES HERE IF A<>
1 OR AFTER 'YES, IT IS REALLY 1' IS DISP
LAYED."
60 GOTO 10
```

Figure 4-7. IF/THEN Program

```
ENTER A                                    (entered 2)
A IS NOT 1.  EXECUTION CONTINUES HERE WH
EN THE EXPRESSION IS FALSE.
EXECUTION CONTINUES HERE IF A<>1 OR AFTE
R 'YES, IT IS REALLY 1' IS DISPLAYED.
ENTER A
                                           (entered 1)
A=1
YES, IT IS REALLY 1.
EXECUTION CONTINUES HERE IF A<>1 OR AFTE
R 'YES, IT IS REALLY 1' IS DISPLAYED.
ENTER A
```

Figure 4-8. IF/THEN Program Execution.

ON/GOSUB/ RETURN ON/GOTO

Format: ON aexp {GOTO / GOSUB} lineno [,lineno...]

Examples:
```
100 ON X GOTO 200, 300, 400
100 ON A GOSUB 1000, 2000
100 ON SQR(X) GOTO 30, 10, 100
```

Note: **GOSUB** and **GOTO** may not be abbreviated.

These two statements are also conditional branch statements like the IF/THEN statement. However, these two are more powerful. The aexp must evaluate to a positive number which is then rounded to the nearest positive integer (whole number) value up to 255. If the resulting number is 1, then program control passes to the first lineno in the list following the GOSUB or GOTO. If the resulting number is 2, program control passes to the second lineno in the list, and so on. If the resulting number is 0 or is greater than the number of linenos in the list, the conditions are not met and program control passes to the next statement which may or may not be located on the same line. With ON/GOSUB, the selected subroutine is executed and then control passes to the next statement.

The following routine demonstrates the ON/GOTO statement:

```
10 X=X+1
20 ON X GOTO 100,200,300,400,500
30 IF X>5 THEN PRINT "COMPLETE.":END
40 GOTO 10
50 END
100 PRINT "NOW WORKING AT LINE 100":GOTO
 10
200 PRINT "NOW WORKING AT LINE 200":GOTO
 10
300 PRINT "NOW WORKING AT LINE 300":GOTO
 10
400 PRINT "NOW WORKING AT LINE 400":GOTO
 10
500 PRINT "NOW WORKING AT LINE 500":GOTO
 10
```

Figure 4-9 ON/GOTO Program Listing

When the program is executed, it looks like the following:

```
NOW WORKING AT LINE 100
NOW WORKING AT LINE 200
NOW WORKING AT LINE 300
NOW WORKING AT LINE 400
NOW WORKING AT LINE 500
COMPLETE.
```

Figure 4-10 ON/GOTO Program Execution

POP

Format: POP

Example:
```
1000 POP
```

In the description of the FOR/NEXT statement, the *stack* was defined as a group of memory addresses reserved for return addresses. The top entry in the stack controls the number of loops to be executed and the RETURN target line for a GOSUB. If a subroutine is not terminated by a RETURN statement, the top memory location of the stack is still loaded with some numbers. If another GOSUB is executed, that top location needs to be cleared. To prepare the stack for a new GOSUB, use a POP to clear the data from the top location in the stack.

The POP command must be used according to the following rules:

1. It must be in the execution path of the program.
2. It must follow the execution of any GOSUB statement that is not brought back to the main program by a RETURN statement.

The following example demonstrates the use of the POP command with a GOSUB when the RETURN is not executed:

```
10 GOSUB 1000
15 REM LINE 20 WILL NOT BE EXECUTED
20 PRINT "NORMAL RETURN PRINTS THIS MESS
AGE."
30 PRINT "ABNORMAL RETURN PRINTS THIS ME
SSAGE."
40 POP
999 END
1000 PRINT "NOW EXECUTING SUBROUTINE."
1010 GOTO 30
1020 RETURN
```

Figure 4-11. GOSUB Statement With POP

RESTORE (RES.)

Format: RESTORE [aexp]
Example: 100 RESTORE

The Atari Personal Computer System contains an internal "pointer" that keeps track of the DATA statement item to be read next. Used without the optional aexp, the RESTORE statement resets that pointer to the first DATA item in the program. Used with the optional aexp, the RESTORE statement sets the pointer to the first DATA item on the line specifed by the value of the aexp. This statement permits repetitive use of the same data.

```
10 FOR N=1 TO 2
20 READ A
30 RESTORE
40 READ B
50 M=A+B
60 PRINT "TOTAL EQUALS ";M
70 NEXT N
80 END
90 DATA 30,15
```

Figure 4-12. Restore Program Listing

On the first pass through the loop, **A** will be 30 and **B** will be 30 so the total line 50 will print **SUM TOTAL EQUALS 60,** but on the second pass, **A** will equal 15

and **B**, because of the RESTORE statement, will still equal 30. Therefore, the PRINT statement in line 50 will display **SUM TOTAL EQUALS 45.**

TRAP (T.)

Format: TRAP aexp
Example: 100 TRAP 120

The TRAP statement is used to direct the program to a specified line number if an error is detected. Without a TRAP statement, the program stops executing when an error is encountered and displays an error message on the screen.

The TRAP statement works on any error that may occur after it has been executed, but once an error has been detected and trapped, it is necessary to reset the trap with another TRAP command. This TRAP command may be placed at the beginning of the section of code that handles input from the keyboard so that the TRAP is reset after each error. PEEK(195) will give you an error message (see Appendix B). 256*PEEK(187)+PEEK(186) will give you the number of the line where the error occurred. The TRAP may be cleared by executing a TRAP statement with an aexp whose value is from 32767 to 65535 (e.g., 40000).

5

INPUT/OUTPUT COMMANDS AND DEVICES

This section describes the input/output devices and how data is moved between them. The commands explained in this section are those that allow access to the input/output devices. The input commands are those associated with getting data into the RAM and the devices geared for accepting input. The output commands are those associated with retrieving data from RAM and the devices geared for generating output.

The commands described in this section are:

CLOAD	**INPUT**	**OPEN/CLOSE**	**READ/DATA**
CSAVE	**LOAD**	**POINT**	**SAVE**
DOS	**LPRINT**	**PRINT**	**STATUS**
ENTER	**NOTE**	**PUT/GET**	**XIO**

INPUT/OUTPUT DEVICES

The hardware configuration of each of the following devices is illustrated in the individual manuals furnished with each. The Central Input/Output (CIO) subsystem provides the user with a single interface to access all of the system peripheral devices in a (largely) independent manner. This means there is a single entry point and a device-independent calling sequence. Each device has a symbolic device name used to identify it; e.g., **K:** for the keyboard. Each device must be opened before access and each must be assigned to an Input/Output Control Block (IOCB). From then on, the device is referred to by its IOCB number.

ATARI BASIC contains 8 blocks in RAM which identifies to the Operating System the information it needs to perform an I/O operation. This information includes the command, buffer length, buffer address, and two auxiliary control variables. ATARI BASIC sets up the IOCB's, but the user must specify which IOCB to use. BASIC reserves IOCB #0 for I/O to the Screen Editor, therefore the user may not request IOCB #0. The GRAPHICS statement (see Section 9) opens IOCB #6 for input and output to the screen. (This is the graphics window **S:**). IOCB #7 is used by BASIC for the LPRINT, CLOAD, and CSAVE commands. The IOCB number may also be referred to as the device (or file) number. IOCB's 1 through 5 are used in opening the other devices for input/output operations. If IOCB #7 is in use, it will prevent LPRINT or some of the other BASIC I/O statements from being performed.

Keyboard: (K:) Input only device. The keyboard allows the user to read the converted (ATASCII) keyboard data as each key is pressed.

Line Printer: (P:) Output only device. The line printer prints ATASCII characters, a line at a time. It recognizes no control characters.

Program Recorder: (C:) Input and Output device. The recorder is a read/write device which can be used as either, but never as both simultaneously. The cassette has two tracks for sound and program recording purposes. The audio track cannot be recorded from the ATARI system, but may be played back through the television speaker.

Disk Drives: (D1:, D2:, D3:, D4:) Input and Output devices. If 16K of RAM is installed, the ATARi can use from one to four disk drives. If only one disk drive is attached, there is no need to add a number after the symbolic device code D.

Screen Editor: (E:) Input and Output device. This device uses the keyboard and display (see *TV Monitor*) to simulate a screen editing terminal. Writing to this device causes data to appear on the display starting at the current cursor position. Reading from this device activates the screen editing process and allows the user to enter and edit data. Whenever the RETURN key is pressed, the entire logical line within which the cursor resides is selected as the current record to be transferred by CIO to the user program. (See Section 9).

TV Monitor: (S:) Input and Output device. This device allows the user to read characters from and write characters to the display, using the cursor as the screen addressing mechanism. Both text and graphics operations are supported. See Section 9 for a complete description of the graphics modes.

Interface, RS-232: (R:) The RS-232 device enables the ATARI system to interface with RS-232-compatible devices such as printers, terminals, and plotters. It contains a parallel port to which the 80-column printer (ATARI 825™) can be attached.

CLOAD (CLOA.)

Format: CLOAD
Examples: CLOAD
100 CLOAD

This command can be used in either Direct or Deferred mode to load a program from cassette tape into RAM for execution. On entering CLOAD, one bell rings to indicate that the PLAY button needs to be pressed followed by RETURN. However, do not press PLAY until after the tape has been positioned. Specific instructions for CLOADing a program are contained in the *ATARI 410 Program Recorder Manual*. Steps for loading oversized programs are included in the paragraphs under **CHAINING PROGRAMS** at the end of this section.

CSAVE (CS.)

Format: CSAVE
Examples: CSAVE
100 CSAVE
100 CS.

This command is usually used in Direct mode to save a RAM-resident program onto cassette tape. CSAVE saves the tokenized version of the program. On entering CSAVE two bells ring to indicate that the PLAY and RECORD buttons must be pressed followed by RETURN. Do not, however, press these buttons until the tape has been positioned. It is faster to save a program using this command rather than a SAVE "C" (see **SAVE**) because short inter-record gaps are used.

Notes: Tapes saved using the two commands, SAVE and CSAVE, are not compatible

It may be necessary to enter an LPRINT (see **LPRINT**) before using CSAVE. Otherwise, CSAVE may not work properly.

For specific instructions on how to connect and operate the hardware, cue the tape, etc., see the *ATARI 410 Program Recorder Manual*.

DOS (DO.)

Format: DOS
Example: DOS

The DOS command is used to go from BASIC to the Disk Operating System (DOS). If the Disk Operating System has not been booted into memory, the computer will go into Memo Pad mode and the user must press SYSTEM RESET to return to Direct mode. If the Disk Operating System has been booted, the DOS Menu is displayed. To clear the DOS Menu from the screen, press SYSTEM RESET. Control then passes to BASIC. Control can also be returned to BASIC by selecting **B** (Run Cartridge) on the DOS Menu.

The DOS command is usually used in Direct mode; however, it may be used in a program. For more details on this, see the *Atari DOS Manual.*

ENTER (E.)

Format: ENTER filespec
Examples: ENTER "C
ENTER "D:DEMOPR.INS"

This statement causes a cassette tape to play back a program originally recorded using LIST (see Section 2, **LIST**). The program is entered in unprocessed (untokenized) form, and is interpreted as the data is received. When the loading is complete, it may be run in the normal way. The ENTER command may also be used with the disk drive. Note that both LOAD and CLOAD (see Section 2) clear the old program from memory before loading the new one. ENTER merges the old and new programs. This ENTER statement is usually used in Direct mode.

INPUT (I.)

Format: INPUT [#aexp {,/;}] {avar/svar} [,{avar/svar} ...]
Examples: 100 INPUT X
100 INPUT N$
100 PRINT "ENTER THE VALUE OF X"
110 INPUT X

This statement requests keyboard data from the user. In execution, the computer displays a ? prompt when the program encounters an INPUT statement. It is usually preceded by a PRINT statement that prompts the user as to the type of information being requested.

String variables are allowed only if they are not subscripted. Matrix variables are not allowed.

The #aexp is optional and is used to specify the file or device number from which the data is to be input (see Input/Output Devices). If no #aexp is specified, then input is from the screen editor (**E:**).

If several strings are to be input from the screen editor, type one string, press RETURN, type the next string, RETURN, etc. Arithmetic numbers can be typed on the same line separated by commas.

```
10 PRINT "ENTER 5 NUMBERS TO BE SUMMED"
20 FOR N=1 TO 5
30 INPUT X
40 C=C+X
50 NEXT N
60 PRINT "THE SUM OF YOUR NUMBERS IS ";C
70 END
```

Figure 5-1 Input Program Listing

LOAD (LO.)

Format: LOAD filespec
Example: LOAD "D1:JANINE.BRY"

This command is similar to CLOAD except the full file name system can be used. LOAD uses long inter-record gaps on the tape (see **CLOAD**) and uses the tokenized version of the program. When using only one disk drive, it is not necessary to specify a number after the "D" because the default is disk drive #1.

LPRINT (LP.)

Format: LPRINT [exp][{;,} exp...]
Example: LPRINT "PROGRAM TO CALCULATE X"
100 LPRINT X;" ";Y;" ";Z

This statement causes the computer to print data on the line printer rather than on the screen. It can be used in either Direct or Deferred modes. It requires no device specifier and no OPEN or CLOSE statement. (BASIC uses IOCB #7.)

The above program listing illustrates a program that will add 5 numbers entered by the user. To print a program listing on the line printer, see **LIST**.

NOTE (NO.)

Format: NOTE #aexp, avar, avar
Example: 100 NOTE #1, X, Y

This command is used to store the current disk sector number in the first avar and the current byte number within the sector in the second avar. This is the current read or write position in the specified file where the next byte to be read or written is located. This NOTE command is used when writing data to a disk file (see **POINT**). The information in the NOTE command is written into a second file which is then used as an index into the first file.

OPEN (O.)
CLOSE (CL.)

Formats: OPEN #aexp,aexp1,aexp2, filespec
CLOSE #aexp
Examples: 100 OPEN #2,8,0,"D1:ATARI800.BAS"
100 A$ = "D1:ATARI800.BAS"
110 OPEN #2,8,0,A$
150 CLOSE #2

Before a device can be accessed, it must be opened. This "opening" process links a specific IOCB to the appropriate device handler, initializes any CIO-related control variables, and passes any device-specific options to the device handler. The parameters for the OPEN command are defined as follows:

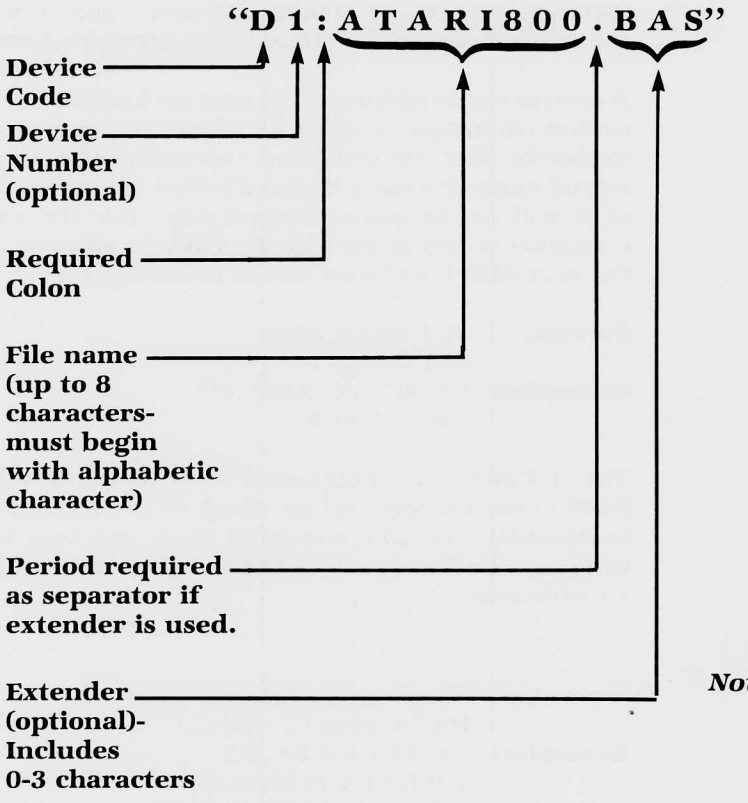
#	Mandatory character that must be entered by the user.
aexp	Reference IOCB or file number to same parameters for future use (as in **CLOSE** command). Number may be 1 through 7.

aexp1 Code number to determine input or output operation.

Code 4 = input operation
8 = output operation
12 = input and output operation
6 = disk directory input operation
(In this case, the filespec is the search specification.)
9 = end-of-file append (output) operation. Append is also used for a special screen editor input mode. This mode allows a program to input the next line from E: without waiting for the user to press RETURN.

aexp2 Device-dependent auxiliary code. An 83 in this parameter indicates sideways printing on a printer (see appropriate manuals for control codes).

filespec Specific file designation. Must be enclosed in quotation marks. The format for the filespec parameter is shown in Figure 5-2.



Note: Filenames are not used with the program recorder.

Figure 5-2 Filename Breakdown

The CLOSE command simply closes files that have been previously opened with an OPEN command. Note in the example that the aexp following the mandatory # character must be the same as the aexp reference number in the OPEN statement.

POINT (P.)

Format: POINT #aexp, avar, avar
Example: 100 POINT #2, A,B

This command is used when reading a file into RAM. The first avar specifies the sector number and the second avar specifies the byte within that sector where the next byte will be read or written. Essentially, it moves a software-controlled pointer to the specified location in the file. This gives the user "random" access to the data stored on a disk file. The POINT and NOTE commands are discussed in more detail in the *DOS Manual*.

PRINT (PR or ?)

Format: PRINT [#aexp]{ ; , } [exp] [,exp...]
Examples: PRINT X, Y, Z, A$
100 PRINT "THE VALUE OF X IS ";X
100 PRINT "COMMAS", "CAUSE", "COLUMN", "SPACING"
100 PRINT #3, A$

A PRINT command can be used in either Direct or Deferred mode. In Direct mode, this command prints whatever information is contained between the quotation marks exactly as it appears. In the first example, PRINT X,Y,Z,A$, the screen will display the current values of X,Y,Z, and A$ as they appear in the RAM-resident program. In the last example, PRINT #3,A$, the #3 is the file specifier (may be any number between 1 and 7) that controls to which device the value of A$ will be printed. (See **Input/Output Devices**.)

A comma causes tabbing to the next tab location. Several commas in a row cause several tab jumps. A semicolon causes the next aexp or sexp to be placed immediately after the preceding expression with no spacing. Therefore, in the second example a space is placed before the ending quotation mark so the value of X will not be placed immediately after the word "IS". If no comma or semicolon is used at the end of a PRINT statement, then a RETURN is output and the next PRINT will start on the following line.

PUT(PU.)/ GET(GE.)

Format: PUT #aexp, aexp
GET #aexp,
Examples: 100 PUT #6, ASC("A")
200 GET #1,X

The PUT and GET are opposites. The PUT command outputs a single byte from 0-255 to the file specified by #aexp. (# is a mandatory character in both these commands). The GET command reads one byte from 0-255 (using #aexp to designate the file, etc. on diskette or elsewhere) and then stores the byte in the variable avar.

READ (REA.) DATA (D.)

Formats: READ var [, var...]
DATA adata [, adata...]
Examples: 100 READ A,B,C,D,E
110 DATA 12,13,14,15,16
100 READ A$,B$,C$,D$,E$
110 DATA EMBEE, EVELYN, CARLA, CORINNE, BARBARA

These two commands are always used together and the DATA statement is always used in Deferred mode[1]. The DATA statement can be located anywhere

[1]A Direct mode READ will only read data if a DATA statement was executed in the program.

in the program, but must contain as many pieces of data as there are defined in the READ statement. Otherwise, an "out of data" error is displayed on the screen.

String variables used in READ statements must be dimensioned and cannot be subscripted. (See **STRINGS** Section). Neither may array variables may be used in a READ statement.

The DATA statement holds a number of string data for access by the READ statement. It cannot include arithmetical operations, functions, etc. Furthermore, the data type in the DATA statement must match the variable type defined in the corresponding READ statement.

The following program totals a list of numbers in a DATA statement:

```
10 FOR N=1 TO 5
20 READ D
30 M=M+D
40 NEXT N
50 PRINT "SUM TOTAL EQUALS ";M
60 END
70 DATA 30,15,106,17,87
```

Figure 5-3 Read/Data Program Listing

The program, when executed, will print the statement:

SUM TOTAL EQUALS 255.

SAVE (S.)

Format: SAVE filespec
Example: SAVE "D1:YVONNE.PAT"

The SAVE command is similar to the CSAVE command except that the full file name system can be used. The device code number is optional when using only one disk drive. The default is to disk drive #1. SAVE, like LOAD, uses long interrecord gaps on the cassette (see **CSAVE**) and the tokenized form of the program.

STATUS (ST.)

Format: STATUS #aexp,avar
Example: 350 STATUS #1,Z

The STATUS command calls the STATUS routine for the specified device (aexp). The status of the STATUS command (see **ERROR MESSAGES**, Appendix B) is stored in the specified variable (avar). This may be useful for future devices such as the RS-232 interface.

XIO (X.)

Format: XIO cmdno, #aexp, aexp1, aexp2, filespec
Example: XIO 18,#6,0,0,"S:"

The XIO command is a general input/output statement used for special operations. One example is its use to fill an area on the screen between plotted points

and lines with a color (see Section 9). The parameters for this command are defined as follows:

cmdno — Number that stands for the particular command to be performed.

cmdno	OPERATION	EXAMPLE
3	OPEN	Same as BASIC OPEN
5	GET RECORD	These 4 commands are similar to
7	GET CHARACTERS	BASIC INPUT GET, PRINT, and PUT
9	PUT RECORD	
11	PUT CHARACTERS	respectively.
12	CLOSE	Same as BASIC CLOSE
13	STATUS REQUEST	Same as BASIC STATUS
17	DRAW LINE	Same as BASIC DRAWTO
18	FILL	See Section 9
32	RENAME	XIO 32,#1,0,0,"D:TEMP.CAROL"
33	DELETE	XIO 33,#1,0,0,"D:TEMP.BAS"
35	LOCK FILE	XIO 35,#1,0,0,"D:TEMP.BAS"
36	UNLOCK FILE	XIO 36,#1,0,0,"D:TEMP.BAS"
37	POINT	Same as BASIC POINT
38	NOTE	Same as BASIC NOTE
254	FORMAT	XIO 254,#1,0,0,"D2:"

aexp — Device number (same as in OPEN). Most of the time it is ignored, but must be preceded by #.

aexp1
aexp2 — Two auxiliary control bytes. Their usage depends on the particular device and command. In most cases, they are unused and are set to 0.

filespec — String expression that specifies the device. Must be enclosed in quotation marks. Although some commands, like Fill (Section 9), do not look at the filespec, it must still be included in the statement.

CHAINING PROGRAMS

If a program requires more memory than is available, use the following steps to string programs of less than the maximum memory available into one program.

1. Type in the first part of the program in the normal way.
2. The last line of the first part of the program should contain only the line number and the command RUN"C:"
3. Cue the tape to the blank section. Write down the program counter number for later RUN purposes. Press PLAY and RECORD buttons on the deck so that both remain down.
4. Type SAVE"C:" and press RETURN .
5. When the beeping sound occurs, press RETURN again.
6. When the screen displays "READY", do not move tape. Type NEW RETURN .
7. Repeat the above instructions for the second part of the program.
8. As the second part of the program is essentially a totally new program, it is possible to re-use the line numbers used in the first part of the program.
9. If there is a third part of the program, make sure the last line of the second part is a RUN"C:" command.

To execute a "chained" program, use the following steps:

1. Cue the tape to the beginning of part 1 of the program.
2. Press PLAY button on the recorder.
3. Type RUN"C:" RETURN .
4. When the "beep" sounds, press RETURN again.

The computer automatically loads the first part of the program, runs it, and sounds a "beep" to indicate when to hit the space bar or RETURN to trigger the tape motor for the second LOAD/RUN. The loading takes a few seconds.

Note: A one-part program can be recorded and reloaded in the same way or CSAVE and CLOAD can be used.

Note: Remember to boot DOS *before* typing in your program.

MODIFYING A BASIC PROGRAM ON DISK

The procedure for modifying an existing BASIC program stored on a diskette is demonstrated in the following steps:

1. Turn off ATARI console and insert BASIC cartridge.
2. Connect disk drive and turn it on - without inserting diskette.
3. Wait for Busy Light to go out and for the drive to stop. Open disk drive door.
4. Insert diskette (with DOS) and close door.
5. Turn on console. DOS should boot in and the screen show READY.
6. To load program from disk, type
   LOAD "D:filename.ext
7. Modify program (or type in new program).
8. To save program on disk, type
   SAVE "D:filename.ext
9. Always wait for the Busy light to go out before removing diskette.
10. To get a Directory listing, do not remove diskette and type
    DOS
    Upon RETURN , the DOS Menu will be displayed. Select command letter **A**, type it, and press RETURN twice to list the directory on the screen; or type **A** followed by pressing RETURN then **P:** RETURN to list directory on the printer.
11. To return to BASIC, type **B** RETURN or press SYSTEM RESET .

NOTES

6

FUNCTION LIBRARY

This section describes the arithmetic, trigonometric, and special purpose functions incorporated into the ATARI BASIC. A function performs a computation and returns the result (usually a number) for either a print-out or additional computational use. Included in the trigonometric functions are two statements, radians (RAD) and degrees (DEG), that are frequently used with trigonometric functions. Each function described in this section may be used in either Direct or Deferred mode. Multiple functions are perfectly legal.

The following functions and statements are described in this section:

ABS
CLOG
EXP
INT
LOG
RND
SGN
SQR
ATN
COS
SIN
DEG/RAD
ADR
FRE
PEEK
POKE
USR

ARITHMETIC FUNCTIONS

ABS

Format: ABS(aexp)
Example: 100 AB = ABS (-190)

Returns the absolute value of a number without regard to whether it is positive or negative. The returned value is always positive.

CLOG

Format: CLOG (aexp)
Example: 100 C = CLOG(83)

Returns the logarithm to the base 10 of the variable or expression in parentheses. CLOG(0) should give an error and CLOG(1) should be 0.

EXP

Format: EXP (aexp)
Example: 100 PRINT EXP(3)

Returns the value of e (approximately 2.71828283), raised to the power specified by the expression in parentheses. In the example given above, the number returned is 20.0855365. In some cases, EXP is accurate only to six significant digits.

INT

Format: INT (aexp)
Examples: 100 I = INT(3.445) *(3 would be stored in I)*
100 X = INT(-14.66778) *(-15 would be stored in X)*

Returns the greatest integer less than or equal to the value of the expression. This is true whether the expression evaluates to a positive or negative number. Thus, in our first example above, I is used to store the number 3. In the second example, X is used to store the number - 15 (the first whole number that is less than or equal to – 14.66778). This INT function should not be confused with the function used on calculators that simply truncates (cuts off) all decimal places.

LOG

Format: LOG(aexp)
Example: 100 L = LOG(67.89/2.57)

Returns the natural logarithm of the number or expression in parentheses. LOG(0) should give an error and LOG(1) should be 0.

RND

Format: RND(aexp)
Example: 10 A = RND (0)

Returns a hardware-generated random number between 0 and 1, but never returns 1. The variable or expression in parentheses following RND is a dummy and has no effect on the numbers returned. However, the dummy variable must be used. Generally, the RND function is used in combination with other BASIC statements or functions to return a number for games, decision making, and the like. Here's a simple routine that returns a random number between 0 and 999.

```
10 X=RND(0)                    (0 is dummy variable)
20 RX=INT(1000*X)
30 PRINT RX
```

SGN

Format: SGN(aexp)
Example: 100 X = SGN(-199) *(-1 would be returned)*

Returns a -1 if aexp evaluates to a negative number; a 0 if aexp evaluates to 0, or a 1 if aexp evaluates to a positive number.

SQR

Format: SQR(aexp)
Example: 100 PRINT SQR(100) *(10 would be printed)*

Returns the square root of the aexp which must be positive.

TRIGONOMETRIC FUNCTIONS

ATN

Format: ATN(aexp)
Example: 100 X = ATN(65)

Returns the arctangent of the variable or expression in parentheses.

COS

Format: COS(aexp)
Example: 100 C = COS(X + Y + Z)

Note: Presumes X, Y, Z previously defined!

Returns the trigonometric cosine of the expression in parentheses.

SIN

Format: SIN(aexp)
Example: 100 X = SIN(Y)

Note: Presumes Y previously defined.

Returns the trigonometric sine of the expression in parentheses.

DEG/RAD

Format: DEG
RAD
Example: 100 DEG
100 RAD

These two statements allow the programmer to specify degrees or radians for trigonometric function computations. The computer defaults to radians unless DEG is specified. Once the DEG statement has been executed, RAD must be used to return to radians.

See Appendix E for the additional trigonometric functions that can be derived.

SPECIAL PURPOSE FUNCTIONS

ADR

Format: ADR(svar)
Example: ADR(A$)

Returns the decimal memory address of the string specified by the expression in parentheses. Knowing the address enables the programmer to pass the information to USR routines, etc. (See **USR** and Appendix D)

FRE

Format: FRE(aexp)
Examples: PRINT FRE (0)
100 IF FRE (0)<1000 THEN PRINT "MEMORY CRITICAL"

This function returns the number of bytes of user RAM left. Its primary use is in Direct mode with a dummy variable (0) to inform the programmer how much memory space remains for completion of a program. Of course FRE can also be used within a BASIC program in Deferred mode.

PEEK

Format: PEEK(aexp)
Examples: 1000 IF PEEK (4000) = 255 THEN PRINT "255"
100 PRINT "LEFT MARGIN IS"; PEEK (82)

Returns the contents of a specified memory address location (aexp). The address specified must be an integer or an arithmetic expression that evaluates to an integer between 0 and 65535 and represents the memory address in decimal notation (not hexadecimal). The number returned will also be a decimal integer with a range from 0 to 255. This function allows the user to examine either RAM or ROM locations. In the first example above, the PEEK is used to determine whether location 4000 (decimal) contains the number 255. In the second example, the PEEK function is used to examine the left margin.

POKE

Format: POKE aexp1, aexp2
Examples: POKE 82, 10
100 POKE 82, 20

Although this is not a function, it is included in this section because it is closely associated with the PEEK function. This POKE command inserts data into the memory location or modifies data already stored there. In the above format, aexp1 is the decimal address of the location to be poked and aexp2 is the data to be poked. Note that this number is a decimal number between 0 and 255. POKE cannot be used to alter ROM locations. In gaining familiarity with this command it is advisable to look at the memory location with a PEEK and write down the contents of the location. Then, if the POKE doesn't work as anticipated, the original contents can be poked into the location.

The above Direct mode example changes the left screen margin from its default position of 2 to a new position of 10. In other words, the new margin will be 8 spaces to the right. To restore the margin to its normal default position, press SYSTEM RESET.

USR

Format: USR (aexp1 [, aexp2][, aexp3...])
Example: 100 RESULT = USR (ADD1,A*2)

This function returns the results of a machine-language subroutine. The first expression, aexp1, must be an integer or arithmetic expression that evaluates to an integer that represents the decimal memory address of the machine language routine to be performed. The input arguments aexp2, aexp3, etc., are optional. These should be arithmetic expressions within a decimal range of 0 through 65535. A non-integer value may be used; however, it will be rounded to the nearest integer.

These values will be converted from BASIC's Binary Coded Decimal (BCD) floating point number format to a two-byte binary number, then pushed onto the hardware stack, composed of a group of RAM memory locations under direct control of the 6502 microprocessor chip. Figure 6-1 illustrates the structure of the hardware stack.

N	(Number of arguments on the stack-may be 0)
$\mathbf{X_1}$	(High byte of argument X)
$\mathbf{X_2}$	(Low byte of argument X)
$\mathbf{Y_1}$	(High byte of argument Y)
$\mathbf{Y_2}$	(Low byte of argument Y)
$\mathbf{Z_1}$	(High byte of argument Z)
$\mathbf{Z_2}$	(Low byte of argument Z)
.	
.	
.	
$\mathbf{R_1}$	(Low byte of return address)
$\mathbf{R_2}$	(High byte of return address)

Figure 6-1. Hardware Stack Definition

Note: X is the argument following the address of the routine, Y is the second, Z is the third, etc. There are N pairs of bytes.

See Section 11 for a description of the USR function in machine language programming. Appendix D defines the bytes in RAM available for machine language programming.

7

STRINGS

This section describes strings and the functions associated with string handling. Each string must be dimensioned (see **DIM** statement, Section 8) and each string variable must end with a **$**. A string itself is a group of characters "strung" together. The individual characters may be letters, numbers, or symbols (including the Atari special keyboard symbols.) A substring is a part of a longer string and any substring is accessible in Atari BASIC if the string has been properly dimensioned (see end of section). The characters in a string are indexed from 1 to the current string length, which is less than or equal to the dimensioned length of the string.

The string functions described in this section are:

ASC
CHR$
LEN
STR$
VAL

ASC

Format: ASC(sexp)
Examples: 100A = ASC(A$)

This function returns the ATASCII code number for the first character of the string expression (sexp). This function can be used in either Direct or Deferred mode. Figure 7-1 is a short program illustrating the ASC function.

```
10 DIM A$(3)
20 A$="E"
30 A=ASC(A$)
40 PRINT A
```

Figure 7-1. ASC Function Program

When executed, this program prints a 69 which is the ATASCII code for the letter "E". Note that when the string itself is used, it must be enclosed in quotation marks.

CHR$

Format: CHR$ (aexp)
Examples: 100 PRINT CHR$ (65)
100 A$ = CHR$ (65)

This character string function returns the character, in string format, represented by the ATASCII code number(s) in parentheses. Only one character is returned. In the above examples, the letter A is returned. Using the ASC and CHR$ functions, the following program prints the upper case and lower case letters of the alphabet.

```
10 FOR I=0 TO 25
20 PRINT CHR$(ASC("A")+I),CHR$(ASC("a")+
I)
30 NEXT I
```

Figure 7-2. ASC and CHR$ Program Example

Note: There can be only one STR$ and only one CHR$ in a logical comparison.

LEN

Format: LEN (sexp)
Example: 100 PRINT LEN(A$)

This function returns the length in bytes of the designated string. This information may then be printed or used later in a program. The length of a string variable is simply the index for the character which is currently at the end of the string. Strings have a length of 0 until characters have been stored in them. It is possible to store into the middle of the string by using subscripting. However, the beginning of the string will contain garbage unless something was stored (using **STO**) there previously.

The following routine illustrates one use of the LEN function:

```
10 DIM A$(10)
20 A$="ATARI"
30 PRINT LEN(A$)
```

Figure 7-3. LEN Function Example

The result of running the above program would be 5.

STR$

Format: STR$ (aexp)
Example: A$ = STR$(65)

This string from number function returns the string form of the number in parentheses. The above example would return the actual number 65, but it would be recognized by the computer as a string.

Note: There can only be one STR$ and only one CHR$ in a logical comparison. For example, A = STR$(1) > STR$(2) is not valid and will not work correctly.

VAL

Format: VAL(sexp)
Example: 100 A = VAL(A$)

This function returns a number of the same value as the number stored as a string. This is the opposite of a STR$ function. Using this function, the computer can perform arithmetic operations on strings as shown in the following example program:

```
10 DIM B$(5)
20 B$="10000"
30 B=SQR(VAL(B$))
40 PRINT "THE SQUARE ROOT OF ";B$;" IS "
;B
```

Figure 7-4. VAL Function Program

Upon execution, the screen displays THE SQUARE ROOT OF B$ IS 100.

It is not possible to use the VAL function with a string that does not start with a number, or that cannot be interpreted by the computer as a number. It can, however, intrepret floating point numbers; e.g.,VAL("1E9")would return the number 1,000,000,000.

STRING MANIPULATIONS

Strings can be manipulated in a variety of ways. They can be split, concatenated, rearranged, and sorted. The following paragraphs describe the different manipulations.

String Concatenation

Concatenation means putting two or more strings together to form one large string. Each string to be included in a larger string is called a *substring*. Each substring must be dimensioned (see **DIM**). In Atari BASIC, a substring can contain up to 99 characters (including spaces). After concatenation, the substrings can be stored in another string variable, printed, or used in later sections of the program. Figure 7-5 is a sample program demonstrating string concatenation. In this program, A$, B$, and C$ are concatenated and placed in A$.

```
10 DIM A$(100),B$(100),C$(100)
20 A$="STRINGS & SUBSTRINGS ARE DISCUSSE
D "
30 B$="IN 'ATARI BASIC--A SELF-TEACHING
GUIDE'"
40 C$="---CHAPTER 9."
50 A$(LEN(A$)+1)=B$
60 A$(LEN(A$)+1)=C$
70 PRINT A$
```

Figure 7-5. String Concatenation Example

String Splitting

The format of a subscript string variable is as follows:

svarname(aexp1[,aexp2])

The svarname is used to indicate the unsubscripted string variable name (with $). aexp1 indicates the starting location of the substring and aexp2 (if used) indicates the ending location of the substring. If no aexp2 is specified, then the end of the substring is the current end of the string. The starting location cannot be greater than the current length of the string. The two example programs in Figure 7-6 illustrate a split string with no end location indicated and a split string with an ending location indicated.

```
10 DIM S$(5)
20 S$="ABCD#"
30 PRINT S$(2)
40 END
```

Result is BCD.
(without ending location)

```
10 DIM S$(20)
20 S$="ATARI 800 BASIC"
30 PRINT S$(7,9)
40 END
```

Result is 800.
(with ending location)

Figure 7-6. Split String Examples

String Comparisons and Sorts

In string comparisons, the logical operators are used exactly the way they are with numbers. The second program in Appendix H is a simple example of bubble sort.

In using logical operators, remember that each letter, number, and symbol is assigned an ATASCII code number. A few general rules apply to these codes:

1. ATASCII codes for numbers are sized in order of the numbers' real values and are always lower than the codes for letters (see Appendix C).

2. Upper case letters have lower numerical values than the lower case letters. To obtain the ATASCII code for a lower case letter if you know the upper case value, add 32 to the upper case code.

Note: Atari BASIC's memory management system moves strings around in memory to make room for new statements. This causes the string address to vary if a program is modified or Direct mode is used.

8

ARRAYS AND MATRICES

An array is a one-dimensional list of numbers assigned to subscripted variables; e.g., A(0), A(1), A(2). Subscripts range from 0 to the dimensioned value. Figure 8-1 illustrates a 7-element array.

A(0)
A(1)
A(2)
A(3)
A(4)
A(5)
A(6)

Figure 8-1. Example of an Array

A matrix, in this context, is a two-dimensional table containing rows and columns. Rows run horizontally and columns run vertically. Matrix elements are stored by BASIC in row-major order. This means that all the elements of the first row are stored first, followed by all the elements of the second row, etc. Figure 8-2 illustrates a 7×4 matrix.

Columns

Rows				
	M(0,0)	M(0,1)	M(0,2)	M(0,3)
	M(1,0)	M(1,1)	M(1,2)	M(1,3)
	M(2,0)	M(2,1)	M(2,2)	M(2,3)
	M(3,0)	M(3,1)	M(3,2)	M(3,3)
	M(4,0)	M(4,1)	M(4,2)	M(4,3)
	M(5,0)	M(5,1)	M(5,2)	M(5,3)
	M(6,0)	M(6,1)	M(6,2)	M(6,3)

Figure 8-2. Example of a Matrix

This section describes the two commands associated with arrays, matrices, and strings, and how to load both arrays and matrices. The commands in this section are:

DIM
CLR

DIM (DI.)

Format: DIM { svar(aexp) / mvar(aexp[,aexp]) } [{ ,svar(aexp) / ,mvar(aexp[,aexp ...] }]

Examples:
```
DIM A(100)
DIM M(6,3)
DIM B$(20)        used with STRINGS
```

A DIM statement is used to reserve a certain number of locations in memory for a string, array, or matrix. A character in a string takes one byte in memory and a number in an array takes six bytes. The first example reserves 101 locations for an array designated A. The second example reserves 7 rows by 4 columns for a two-dimensional array (matrix) designated M. The third example reserves 20 bytes designated B$. **All strings, arrays, and matrices must be dimensioned.** It is a good habit to put all DIM statements at the beginning of the program. Notice in Figure 8-1 that although the array is dimensioned as DIM A(6), there are actually 7 elements in the array because of the 0 element. Although Figure 8-2 is dimensioned as DIM M(6,3), 28 locations are reserved.

Note: The ATARI Personal Computer does not automatically initialize array or matrix variables to 0 at the start of program execution. To initialize array or matrix elements to 0, use the following program steps:

```
250 DIM A(100)
300 FOR E=0 TO 100
310 A(3)=0
320 NEXT E
```

Arrays and matrices are "filled" with data by using FOR/NEXT statements, READ/DATA statements and INPUT commands. Figure 8-3 illustrates the "building" of part of an array using the FOR/NEXT loop and Figure 8-4 builds an array using the READ/DATA statements.

```
10 DIM A(100)
20 X=10
30 FOR E=1 TO 90
40 X=X+1
50 A(E)=X
60 NEXT E
70 FOR E=1 TO 90
80 PRINT E,A(E)
90 NEXT E
```

Figure 8-3. Use of FOR/NEXT to Build An Array

```
10 DIM A(3)
20 FOR E=1 TO 3
30 READ X
40 A(E)=X
50 PRINT A(E),
60 NEXT E
70 END
100 DATA 33,45,12
```

Figure 8-4. Use of READ/DATA to Build An Array

Figure 8-5 shows an example of building a 6×3 matrix.

```
10 DIM M(6,3)
20 FOR ROW=0 TO 6
30 FOR COL=1 TO 3
40 M(ROW,COL)=INT(RND(0)*1000)
50 NEXT COL:NEXT ROW
60 FOR ROW=0 TO 6
70 FOR COL=1 TO 3
80 PRINT M(ROW,COL)
90 NEXT COL:PRINT :NEXT ROW
```

Figure 8-5. Building A Matrix

Note that the words ROW and COLUMN are not BASIC commands, statements, functions, or keywords. They are simply variable names used here to designate which loop function is first. The program could just as easily have been written with X and Y as the variable names.

CLR

Format: CLR
Example: 200 CLR

This command clears the memory of all previously dimensioned strings, arrays, and matrices so the memory and variable names can be used for other purposes. It also clears the values stored in undimensioned variables. If a matrix, string, or array is needed after a CLR command, it must be redimensioned with a DIM command.

NOTES

9

GRAPHICS MODES AND COMMANDS

This section describes the Atari BASIC commands and the different graphics modes of the ATARI Personal Computer. Using these commands, it is possible to create graphics for game, graphics, and patterns.

The commands to be described in this section are:

GRAPHICS	**LOCATE**	**PUT/GET**
COLOR	**PLOT**	**SETCOLOR**
DRAWTO	**POSITION**	**XIO**

The PUT/GET and XIO commands explained in this section are special applications of the same commands described in Section 5.

GRAPHICS (GR.)

Format: GRAPHICS aexp
Example: GRAPHICS 2

This command is used to select one of the nine graphics modes. Table 9-1 summarizes the nine modes and the characteristics of each. The GRAPHICS command automatically opens the screen, S:(the graphics window), as device #6. So when printing text in the text window, it is not necessary to specify the device code. The aexp must be positive, rounded to the nearest integer. Graphics mode 0 is a full-screen display while modes 1 through 8 are split screen displays. To override the split-screen, add the characters +16 to the mode number (aexp) in the GRAPHICS command. Adding 32 prevents the graphics command from clearing the screen.

To return to graphics mode 0 in Direct mode, press SYSTEM RESET or type GR.0 and press RETURN

TABLE 9.1—TABLE OF MODES AND SCREEN FORMATS

		SCREEN FORMAT				
Gr. Mode	**Mode Type**	**Horiz. (Rows)**	**Vert. (Col) Split Screen**	**Vert. (Col) Full Screen**	**Number Of Colors**	**RAM Required (Bytes)**
0	TEXT	40	-	24	2	993
1	TEXT	20	20	24	5	513
2	TEXT	20	10	12	5	261
3	GRAPHICS	40	20	24	4	273
4	GRAPHICS	80	40	48	2	537
5	GRAPHICS	80	40	48	4	1017
6	GRAPHICS	160	80	96	2	2025
7	GRAPHICS	160	80	96	4	3945
8	GRAPHICS	320	160	192	1/2	7900

The following paragraphs describe the nine graphics modes.

GRAPHICS MODE 0

This mode is the 1-color, 2-luminance (brightness) default mode for the ATARI Personal Computer. It contains a 24 by 40 character screen matrix. The default margin settings at 2 and 39 allow 38 characters per line. Margins may be changed by poking LMARGN and RMARGN (82 and 83). See Appendix I. Some systems have different margin default settings. The color of the characters is determined by the background color. Only the luminance of the characters can be different. This full-screen display has a blue display area bordered in black (unless the border is specified to be another color). To display characters at a specified location, use one of the following two methods.

Method 1.

```
lineno POSITION aexp1, aexp2      Puts cursor at location
lineno PRINT sexp                 specified by aexp1 and aexp2.
```

Method 2

```
lineno GR. 0                      Specifies graphics mode.
lineno POKE 752,1                 Suppresses cursor.
lineno COLOR ASC(sexp)            Specifies character to be
                                  printed.
lineno PLOT aexp1,aexp2           Specifies where to print
                                  character.
lineno GOTO lineno                Start loop to prevent READY
                                  from being printed. (GOTO
                                  same lineno.)

                                  Press BREAK to terminate
                                  loop.
```

GRAPHICS 0 is also used as a clear screen command either in Direct mode or Deferred mode. It terminates any previously selected graphics mode and returns the screen to the default mode (GRAPHICS 0).

GRAPHICS MODES 1 AND 2

As defined in Table 9-1, these two 5-color modes are Text modes. However, they are both split-screen (see Figure 9-1) modes. Characters printed in Graphics mode 1 are twice the width of those printed in Graphics 0, but are the same height. Characters printed in Graphics mode 2 are twice the width and height of those in Graphics mode 0. In the split-screen mode, a PRINT command is used to display characters in either the text window or the graphics window. To print characters in the graphics window, specify device #6 after the PRINT command.

Example:

```
100 GR. 1
110 PRINT#6;"ATARI"
```

The default colors depend on the type of character input. Table 9-2 defines the default color and color register used for each type.

Table 9-2. Default Colors for Specific Input Types

Character Type	Color Register	Default Color
Upper case alphabetical	0	Orange
Lower case alphabetical	1	Light Green
Inverse upper case alphabetical	2	Dark Blue
Inverse lower case alphabetical	3	Red
Numbers	0	Orange
Inverse numbers	2	Dark Blue

Note: See **SETCOLOR** to change character colors.

Unless otherwise specified, all characters are displayed in upper case non-inverse form. To print lower case letters and graphics characters, use a **POKE 756,226**. To return to upper case, use **POKE 756,224**.

In graphics modes 1 and 2, there is no inverse video, but it is possible to get all the rest of the characters in four different colors (see end of section).

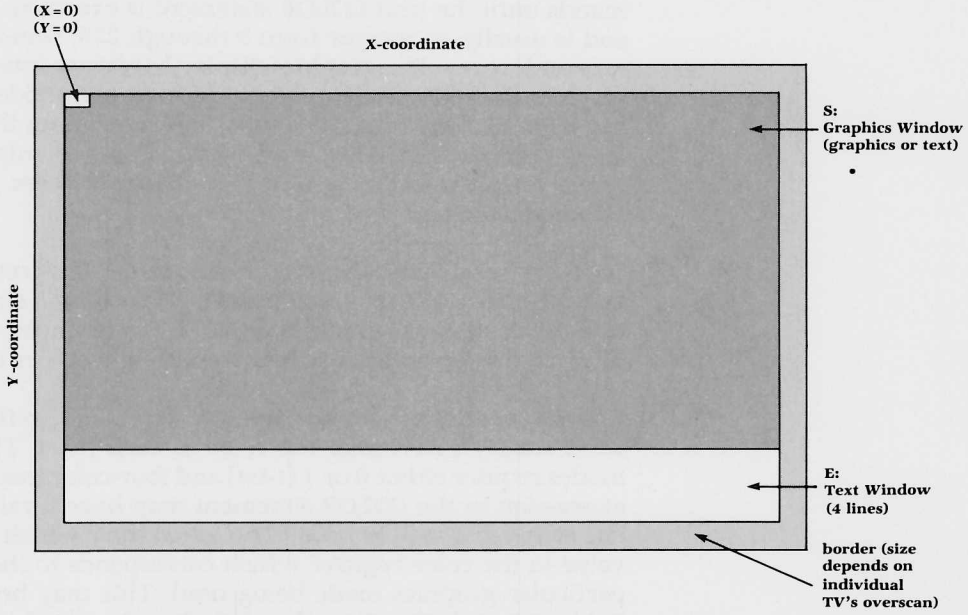


Figure 9-1. Split-Screen Display For Graphics Modes 1 and 2

The X and Y coordinates start at 0 (upper left of screen). The maximum values are the numbers of rows and columns minus 1 (see Table 9-1).

This split-screen configuration can be changed to a full screen display by adding the characters +16 to the mode number.

Example: GRAPHICS 1+16

GRAPHICS MODES 3, 5, AND 7

These three 4-color graphics modes are also split-screen displays in their default state, but may be changed to full screen by adding +16 to the mode number. Modes 3, 5, and 7 are alike except that modes 5 and 7 use more points (pixels) in plotting, drawing, and positioning the cursor; the points are smaller, thereby giving a much higher resolution.

GRAPHICS MODES 4 AND 6

These two 2-color graphics modes are split-screen displays and can display in only two colors while the other modes can display 4 and 5 colors. The advantage of a two-color mode is that it requires less RAM space (see Table 9-1). Therefore, it is used when only two colors are needed and RAM is getting crowded. These two modes also have a higher resolution which means smaller points than Graphics mode 3.

GRAPHICS MODE 8

This graphics mode gives the highest resolution of all the other modes. As it takes a lot of RAM to obtain this kind of resolution, it can only accomodate a maximum of one color and two different luminances.

COLOR (C.)

Format: COLOR aexp
Examples: 110 COLOR ASC("A")
110 COLOR 3

The value of the expression in the COLOR statement determines the data to be stored in the display memory for all subsequent PLOT and DRAWTO commands until the next COLOR statement is executed. The value must be positive and is usually an integer from 0 through 255. Non-integers are rounded to the nearest integer. The graphics display hardware interprets this data in different ways in the different graphics modes. In text modes 0 through 2, the number can be from 0 through 255 (8 bits) and determines the character to be displayed and its color. (The two most significant bits determine the color. This is why only 64 different characters are available in these modes instead of the full 256-character set.)

Tables 9-6 and 9-7 at the end of this section illustrate the internal character set and the character/color assignment. Table 9-2 is a simplified table which allows easy generation of some of the colors. For example, **COLOR ASC("A"): PLOT 5,5** will display an orange A character in graphics modes 1 or 2 at location 5,5.

Graphics modes 3 through 8 are not text modes, so the data stored in the display RAM simply determines the color of each pixel. Two-color or two-luminance modes require either 0 or 1 (1-bit) and four-color modes require 0, 1, 2, or 3. (The expression in the COLOR statement may have a value greater than 3, but only one or two bits will be used.) The actual color which is displayed depends on the value in the color register which corresponds to the data of 0, 1, 2, or 3 in the particular graphics mode being used. This may be determined by looking in Table 9-5, which gives the default colors and the corresponding register numbers. Colors may be changed by using SETCOLOR.

Note that when BASIC is first powered up, the color data is 0, and when a GRAPHICS command (without +32) is executed, all of the pixels are set to 0. Therefore, nothing seems to happen to PLOT and DRAWTO in GRAPHICS 3 through 7 when no COLOR statement has been executed. Correct by doing a COLOR 1 first.

DRAWTO (DR.)

Format: DRAWTO aexp1, aexp2
Example: 100 DRAWTO 10, 8

This statement causes a line to be drawn from the last point displayed by a PLOT (see **PLOT**) to the location specified by aexp1 and aexp2. The first expression represents the X coordinate and the second represents the Y-coordinate (see Figure 9-1). The color of the line is the same color as the point displayed by the PLOT.

LOCATE (LOC.)

Format: LOCATE aexp1, aexp2, var
Example: 150 LOCATE 12, 15, X

This command positions the invisible graphics cursor at the specified location in the graphics window, retrieves the data at that pixel, and stores it in the specified arithmetic variable. This gives a number from 0 to 255 for Graphics modes 0 through 2; 0 or 1 for the 2-color graphics modes; and 0, 1, 2, or 3 for the 4-color modes. The two arithmetic expressions specify the X and Y coordinates of the point. LOCATE is equivalent to:

POSITION aexp1, aexp2:GET #6,avar

Doing a PRINT after a LOCATE or GET from the screen may cause the data in the pixel which was examined to be modified. This problem is avoided by repositioning the cursor and putting the data that was read, back into the pixel before doing the PRINT. The following program illustrates the use of the LOCATE command.

```
10 GRAPHICS 3+16
20 COLOR 1
30 SETCOLOR 2,10,8
40 PLOT 10,15
50 DRAWTO 15,15
60 LOCATE 12,15,X
70 PRINT X
```

Figure 9-2. Example Program Using LOCATE

On execution, the program prints the data (**1**) determined by the COLOR statement which was stored in pixel 12, 15.

PLOT (PL.)

Format: PLOT aexp1, aexp2
Example: 100 PLOT 5,5

The PLOT command is used in graphics modes 3 through 8 to display a point in the graphics window. The aexp1 specifies the X-coordinate and the aexp2 the Y-coordinate. The color of the plotted point is determined by the hue and luminance in the color register from the last COLOR statement executed. To change this color register, and the color of the plotted point, use SETCOLOR. Points that can be plotted on the screen are dependent on the graphics mode being used. The range of points begins at 1 and extends to one less than the total number of rows (X-coordinate) or columns (Y-coordinate) shown in Table 9-1.

POSITION (POS.)

Format: POSITION aexp1, aexp2
Example: 100 POSITION 8, 12

The POSITION statement is used to place the invisible graphics window cursor at a specified location on the screen (usually precedes a PRINT statement). This statement can be used in all modes. Note that the cursor does not actually move until an I/O command which involves the screen is issued.

PUT/GET (PU./GE.)

Formats: PUT #aexp, aexp
GET #aexp, avar
Examples: 100 PUT #6, ASC("A")
200 GET #1, X

In graphics work, PUT is used to output data to the screen display. This statement works hand-in-hand with the POSITION statement. After a PUT (or GET), the cursor is moved to the next location on the screen. Doing a PUT to device #6 causes the one-byte input (second aexp) to be displayed at the cursor position. The byte is either an ATASCII code byte for a particular character (modes 0-2) or the color data (modes 3-8).

GET is used to input the code byte of the character displayed at the cursor position, into the specified arithmetic variable. The values used in PUT and GET correspond to the values in the COLOR statement. (PRINT and INPUT may also be used.)

Note: Doing a PRINT after a LOCATE or GET from the screen may cause the data in the pixel which was examined to be modified. To avoid this problem, reposition the cursor and put the data that was read, back into the pixel before doing the PRINT.

SETCOLOR (SE.)

Format: SETCOLOR aexp1, aexp2, aexp3
Example: 100 SETCOLOR 0, 1, 4

This statement is used to choose the particular hue and luminance to be stored in the specified color register. The parameters of the SETCOLOR statement are defined below:

aexp1 = Color register (0-4 depending on graphics mode)
aexp2 = Color hue number (0-15. See Table 9-3)
aexp3 = Color luminance (must be an even number between 0 and 14; the higher the number, the brighter the display. 14 is almost pure white.)

TABLE 9.3—THE ATARI HUE (SETCOLOR COMMAND) NUMBERS AND COLORS

COLORS	SETCOLOR (aexp2) NUMBERS
GRAY	0
LIGHT ORANGE (GOLD)	1
ORANGE	2
RED-ORANGE	3
PINK	4
PURPLE-BLUE	6
BLUE	7
BLUE	8
LIGHT BLUE	9
TURQUOISE	10
GREEN-BLUE	11
GREEN	12
YELLOW-GREEN	13
ORANGE-GREEN	14
LIGHT ORANGE	15

Note: Colors will vary with type and adjustment of TV or monitor used.

The ATARI display hardware contains five color registers, numbered from 0 through 4. The Operating System (OS) has five RAM locations (COLOR0 through COLOR4, see Appendix I - Memory Locations) where it keeps track of the current colors. The SETCOLOR statement is used to change the values in these RAM locations. (The OS transfers these values to the hardware registers every television frame.) The SETCOLOR statement requires a value from 0 to 4 to specify a color register. The COLOR statement uses different numbers because it specifies data which only *indirectly* corresponds to a color register. This can be confusing, so careful experimentation and study of the various tables in this section is advised.

No SETCOLOR commands are needed if the default set of five colors is used. Although 128 different color-luminance combinations are possible, not more than five can be displayed at any one time. The purpose of the color registers and SETCOLOR statement is to specify these five colors.

TABLE 9.4—TABLE OF SETCOLOR "DEFAULT" COLORS*

Setcolor (Color Register)	Defaults To Color	Luminance	Actual Color
0	2	8	ORANGE
1	12	10	GREEN
2	9	4	DARK BLUE
3	4	6	PINK OR RED
4	0	0	BLACK

*"DEFAULT" occurs if not **SETCOLOR** statement is used.

Note: Colors may vary depending upon the television monitor type, condition, and adjustment.

A program illustrating Graphics mode 3 and the commands explained so far in this section is shown below:

```
10 GRAPHICS 3
20 SETCOLOR 0,2,8:COLOR 1
30 PLOT 17,1:DRAWTO 17,10:DRAWTO 9,18
40 PLOT 19,1:DRAWTO 19,18
50 PLOT 20,1:DRAWTO 20,18
60 PLOT 22,1:DRAWTO 22,10:DRAWTO 30,18
70 POKE 752,1
80 PRINT :PRINT "    ATARI PERSONAL COMP
UTERS"
90 GOTO 90
```

The SETCOLOR and COLOR statements set the color of the points to be plotted (see Table 9.5). The SETCOLOR command loads color register 0 with hue 2 (orange) and a luminance of 8 ("normal"). The next 4 lines plot the points to be displayed. Line 90 suppresses the cursor and line 100 prints the string expression ATARI PERSONAL COMPUTERS in the text window (6 spaces in).

Note that the background color was never set because the default is the desired color (black).

If the program is executed, it will print the Atari logo in the graphics window and the string expression in the text window as in Figure 9-3.

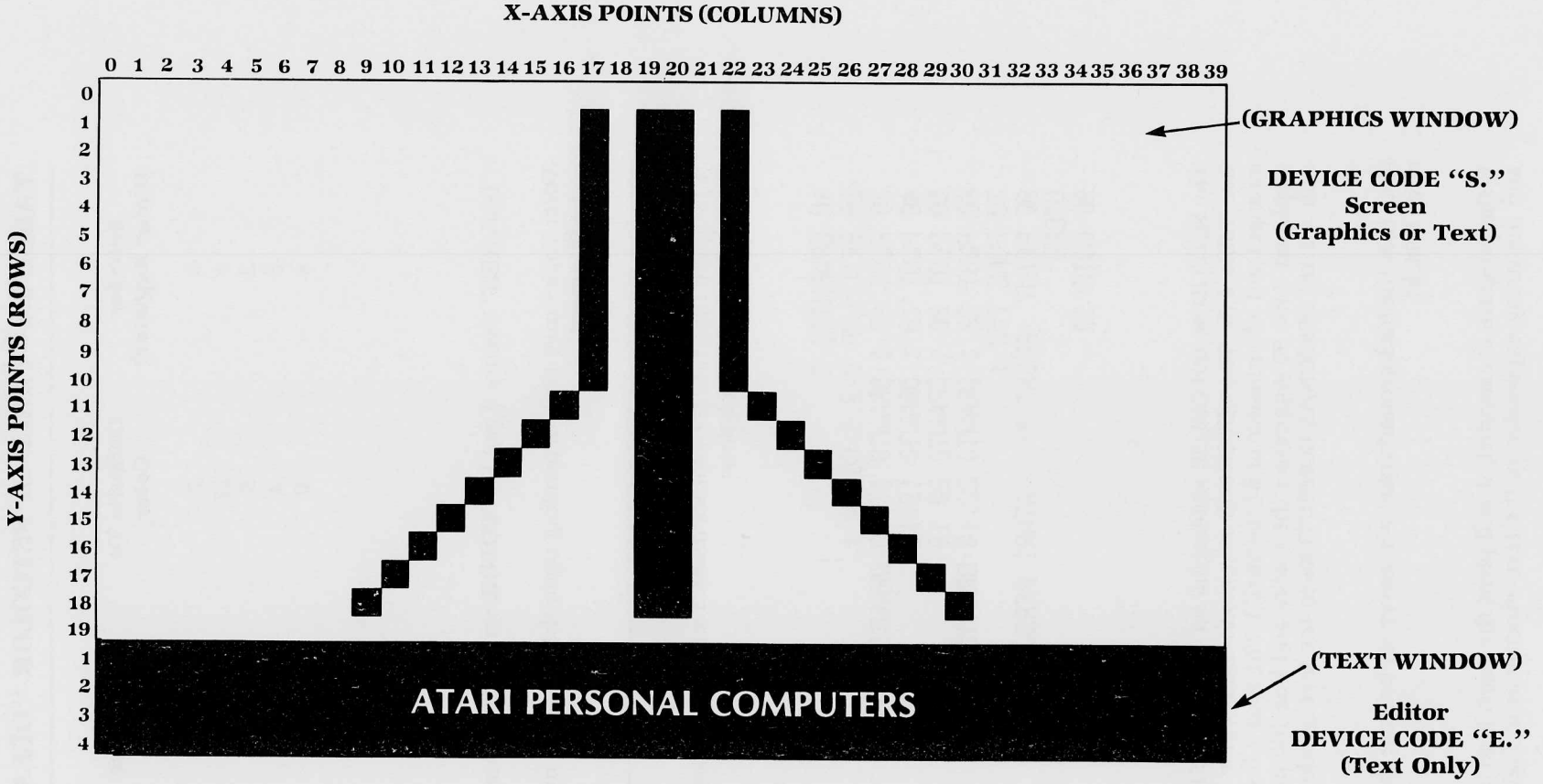


Figure 9-3. Atari Logo Program Execution

TABLE 9.5—MODE, SETCOLOR, COLOR TABLE

Default Colors	Mode or Condition	SETCOLOR (aexp1) Color Register No.	Color (aexp)	DESCRIPTION AND COMMENTS
LIGHT BLUE DARK BLUE BLACK	MODE 0 and ALL TEXT WINDOWS	0 1 2 3 4	COLOR data actually determines character to be plotted	— Character luminance (same color as background) Background — Border
ORANGE LIGHT GREEN DARK BLUE RED BLACK	MODES 1 and 2 (Text Modes)	0 1 2 3 4	COLOR data actually determines character to be plotted	Character Character Character Character Background, Border
ORANGE LIGHT GREEN DARK BLUE BLACK	MODES 3, 5, and 7 (Four-color Modes)	0 1 2 3 4	1 2 3 - 0	Graphics point Graphics point Graphics point — Graphics point (background default), Border
ORANGE BLACK	MODES 4 and 6 (Two-color Modes) 4	0 1 2 3 0	1 - - -	Graphics point — — — Graphics point (background default), Border
LIGHT GREEN DARK BLUE BLACK	MODE 8 (1 Color 2 Luminances)	0 1 2 3 4	- 1 0 - -	— Graphics point luminance (same color as background) Graphics point (background default) — Border

XIO (X.) SPECIAL FILL APPLICATION

Format: XIO 18, #aexp, aexp1, aexp2, filespec
Example: 100 XIO 18, #6, 0, 0, "S:"

This special application of the XIO statement fills an area on the screen between plotted points and lines with a non-zero color value. Dummy variables (0) are used for aexp1 and aexp2.

The following steps illustrate the fill process:

1. PLOT bottom right corner (point 1).
2. DRAWTO upper right corner (point 2). This outlines the right edge of the area to be filled.
3. DRAWTO upper left corner (point 3).
4. POSITION cursor at lower left corner (point 4).
5. POKE address 765 with the fill color data (1, 2, or 3).
6. This method is used to fill each horizontal line from top to bottom of the specified area. The fill starts at the left and proceeds across the line to the right until it reaches a pixel which contains non-zero data (will wraparound if necessary). This means that fill cannot be used to change an area which has been filled in with a non-zero value, as the fill will stop. The fill command will go into an infinite loop if a fill with zero (0) data is attempted on a line which has no non-zero pixels. BREAK or SYSTEM RESET can be used to stop the fill if this happens.

The following program creates a shape and fills it with a data (color) of 3. Note that the XIO command draws in the lines of the left and bottom of the figure.

```
10 GRAPHICS 5+16
20 COLOR 3
30 PLOT 70,45
40 DRAWTO 50,10
50 DRAWTO 30,10
60 POSITION 10,45
70 POKE 765,3
80 XIO 18,#6,0,0,"S:"
90 GOTO 90
```

Figure 9-4. Example "FILL" Program

Assigning Colors To Characters In Text Modes 1 and 2

This procedure describes the method of assigning colors to the Atari character set. First, look up the character number in Table 9-6. Then, see Table 9-7 to get the conversion of that number required to assign a color register to it.

Example: Assign SETCOLOR 0 to lower case "r" in mode 2 whose color is determined by register 0.

1. In Table 9-6, find the column and number for "r" (114-column 4).
2. Using Table 9-7, locate column 4. Conversion is the character number minus 32 (114 - 32 = 82).

Table 9.6—INTERNAL CHARACTER SET

Column 1				Column 2				Column 3				Column 4			
#	CHR	#	CHR	#	CHR	#	CHR	#	CHR	#	CHR	#	CHR	#	CHR
0	Space	16	0	32	@	48	P	64		80		96		112	p
1	!	17	1	33	A	49	Q	65		81		97	a	113	q
2	”	18	2	34	B	50	R	66		82		98	b	114	r
3	#	19	3	35	C	51	S	67		83		99	c	115	s
4	$	20	4	36	D	52	T	68		84		100	d	116	t
5	%	21	5	37	E	53	U	69		85		101	e	117	u
6	&	22	6	38	F	54	V	70		86		102	f	118	v
7	’	23	7	39	G	55	W	71		87		103	g	119	w
8	(	24	8	40	H	56	X	72		88		104	h	120	x
9	)	25	9	41	I	57	Y	73		89		105	i	121	y
10	*	26	:	42	J	58	Z	74		90		106	j	122	z
11	+	27	;	43	K	59	[	75		91 ①		107	k	123	
12	,	28	<	44	L	60	\	76		92		108	l	124	|
13	—	29	=	45	M	61	]	77		93		109	m	125 ①	
14	—	30	>	46	N	62	^	78		94		110	n	126 ①	
15	/	31	?	47	O	63	—	79		95		111	o	127 ①	

1. In mode 0 these characters must be preceded with an escape, CHR$(27), to be printed.

Table 9.7—CHARACTER/COLOR ASSIGNMENT					
		Conversion 1	**Conversion 2**	**Conversion 3**	**Conversion 4**
MODE 0	[2]SETCOLOR 2	#+32	#+32	#–32	NONE
		POKE 756,224		POKE 756,226	
MODE 1	SETCOLOR 0	–$32	#+32	#–32	#–32
OR	SETCOLOR 1	NONE	#+64	#–64	NONE
MODE 2	SETCOLOR 2	#+160	#+160	#+96	#+96
	SETCOLOR 3	#+128	#+192	#+64	#+128

2. Luminance controlled by SETCOLOR 1, 0, LUM.

3. POKE the Character Base Address (CHBAS) with 226 to specify lower case letters or special graphics characters; e.g.,

```
POKE 756,226
     or
CHBAS = 756
POKE CHBAS, 226
```

To return to upper case letters, numbers, and punctuation marks, POKE CHBAS with 224.

4. A PRINT statement using the converted number (82) assigns the lower case "r" to SETCOLOR 0 in mode 2 (see Table 9-5).

Graphic Control Characters

These characters are produced when the CTRL key is pressed with the alphabetic keys shown on back cover. These characters can be used to draw design, pictures, etc., in mode 0 and in modes 1 and 2 if CHBAS is changed.

10

SOUNDS AND GAME CONTROLLERS

This section describes the statement used to generate musical notes and sounds through the audio system of the television monitor. Up to four different sounds can be "played" simultaneously creating harmony. This SOUND statement can also be used to simulate explosions, whistles, and other interesting sound effects. The other commands described in this section deal with the functions used to manipulate the keyboard, joystick, and paddle controllers. These functions allow these controllers to be plugged in and used in BASIC programs for games, etc.

The command and functions covered in this section are:

SOUND	**PADDLE**	**STICK**
	PTRIG	**STRIG**

SOUND (SO.)

Format: SOUND aexp1, aexp2, aexp3, aexp4
Example: 100 SOUND 2, 204, 10, 12

The SOUND statement causes the specified note to begin playing as soon as the statement is executed. The note will continue playing until the program encounters another SOUND statement with the same aexp1 or an END statement. This command can be used in either Direct or Deferred modes.

The SOUND parameters are described as follows:

aexp1 = *Voice.* Can be 0-3, but each voice requires a separate SOUND statement.

aexp2 = *Pitch.* Can be any number between 0-255. The larger the number, the lower the pitch. Table 10-1 defines the pitch numbers for the various musical notes ranging from two octaves above middle C to one octave below middle C.

aexp3 = *Distortion.* Can be even numbers between 0-14. Used in creating sound effects. A 10 is used to created a "pure" tone whereas a 12 gives an interesting buzzer sound. A buzzing sound (like engines at a race track) can be produced using two separate SOUND commands with the distortion value (aexp3) alternating between 0 and 1. A value of 1 is used to force output to the speaker using the specified volume (see aexp4). The rest of the numbers are used for other special effects, noise generation, and experimental use.

aexp4 = *Volume control.* Can be between 1 and 15. Using a 1 creates a sound barely audible whereas a 15 is loud. A value of 8 is considered normal. If more than 1 sound statement is being used, the total volume should not exceed 32. This will create an unpleasant "clipped" tone.

Using the note values in Table 10-1, the following example demonstrates how to write a program that will "play" the C scale.

TABLE 10.1. TABLE OF PITCH VALUES FOR THE MUSICAL NOTES

HIGH NOTES	C	29
	B	31
	A# or B♭	33
	A	35
	G# or A♭	37
	G	40
	F# or G♭	42
	F	45
	E	47
	D# or E	50
	D	53
	C# or D♭	57
	C	60
	B	64
	A# or B	68
	A	72
	G# or A♭	76
	G	81
	F# or G♭	85
	F	91
	E	96
	D# or E♭	102
	D	108
	C# or D♭	114
MIDDLE C	C	121
	B	128
	A# or B♭	136
	A	144
	G# or A♭	153
	G	162
	F# G♭	173
	F	182
LOW NOTES	D	193
	D# or E♭	204
	D	217
	C# or Db	230
	C	243

```
10 READ A
20 IF A=256 THEN END
30 SOUND 0,A,10,10
40 FOR W=1 TO 400:NEXT W
50 PRINT A
60 GOTO 10
70 END
80 DATA 29,31,35,40,45,47,53,60,64,72,81
,91,96,108,121
90 DATA 128,144,162,182,193,217,243,256
```

Figure 10-1. Musical Scale Program

Note that the DATA statement in line 80 ends with a 256, which is outside of the designated range. The 256 is used as an end-of-data marker.

GAME CONTROLLER FUNCTIONS

Figure 10-2 is an illustration of the three controllers used with the Atari Personal Computers. The controllers can be attached directly to the Atari Personal Computer or to external mechanical devices so that outside events can be fed directly to the computer for processing and control purposes.

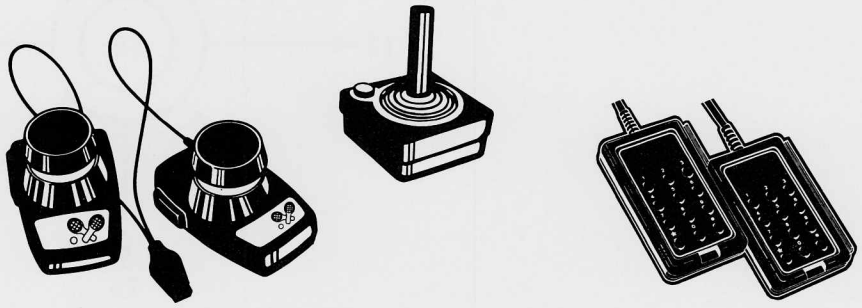


Figure 10-2. Game Controllers

PADDLE

Format: PADDLE(aexp)
Example: PRINT PADDLE(3)

This function returns the status of a particular numbered controller. The paddle controllers are numbered 0-7 from left to right. This function can be used with other functions or commands to "cause" further actions like sound, graphics controls, etc. For example, the statement **IF PADDLE(3) = 14 THEN PRINT "PADDLE ACTIVE."** Note that the PADDLE function returns a number between 1 and 228, with the number increasing in size as the knob on the controller is rotated counterclockwise (turned to the left).

PTRIG

Format: PTRIG(aexp)
Example: 100 IF PTRIG(4)=0 THEN PRINT "MISSILES FIRED!"

The PTRIG function returns a status of 0 if the trigger button of the designated controller is pressed. Otherwise, it returns a value of 1. The aexp must be a number between 0 and 7 as it designates the controller.

STICK

Format: STICK(aexp)
Example: 100 PRINT STICK(3)

This function works exactly the same way as the PADDLE command, but can be used with the joystick controller. The joystick controllers are numbered from 0-3 from left to right.

Controller 1 = STICK(0)
Controller 2 = STICK(1)
Controller 3 = STICK(2)
Controller 4 = STICK(3)

Figure 10-3 shows the numbers that will be returned when the joystick controller is moved in any direction.

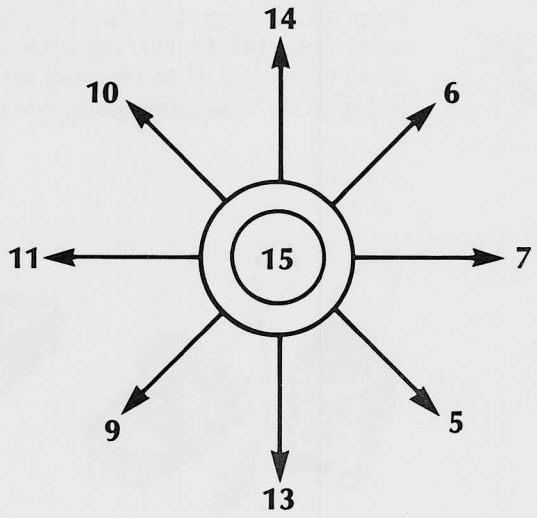


Figure 10-3. Joystick Controller Movement

STRIG

Format: STRIG(aexp)
Example: 100 IF STRIG(3)=0 THEN PRINT "FIRE TORPEDO"

The STRIG function works the same way as the PTRIG function. It can be used with both the joystick and keyboard controllers.

11

ADVANCED PROGRAMMING TECHNIQUES

This section includes hints on increasing programming efficiency, conserving memory, and combining machine language programs with Atari BASIC programs. This section does not include an instruction set for the 6502 microprocessor chip nor does it give instructions on programming in machine language. An additional purchase of the Atari Assembler Editor cartridge* and a careful study of Atari's Assembler Editor Manual are strongly recommended.

MEMORY CONSERVATION

These hints give ways of conserving memory. Some of these methods make programs less readable and harder to modify, but there are cases where this is necessary due to memory limitations.

1. In many small computers, eliminating blank spaces between words and characters as they are typed into the keyboard will save memory. This is not true of the ATARI Personal Computer System, which removes extra spaces. Statements are always displayed the same regardless of how many spaces were used on program entry. Spaces should be used (just as in typing on a conventional typewriter) between successive keywords and between keywords and variable names. Here is an example:

```
10 IF A = 5 THEN PRINT A
```

   Note the space between IF and A and between THEN and PRINT. In most cases, a statement will be interpreted correctly by the computer even if all spaces are left out, but this is not always true. Use conventional spacing.

2. Each new line number represents the beginning of what is called a new "logical line". Each logical line takes 6 bytes of "overhead", whether it is used to full capacity or not. Adding an additional BASIC statement by using a colon (:) to separate each pair of statements on the same line takes only 3 bytes.

*Available late 1980.

If you need to save memory, avoid programs like this:

```
10 X=Y+1
20 Y=Y+1
30 Z=X+Y
40 PRINT Z
50 GOTO 50
```

and consolidate lines like this:

```
10 X=X+1:Y=Y+1:Z=X+Y:PRINT Z:GOTO 10
```

This consolidation saves 12 bytes.

3. Variables and constants should be "managed" for savings, too. Each time a constant (4,5,16,3.14159, etc.) is used, it takes 7 bytes. Defining a new variable requires 8 bytes plus the length of the variable name (in characters). But each time it is used after being defined, it takes only 1 byte, regardless of its length. Thus, if a constant (such as 3.14159) is used more than once or twice in a program, it should be defined as a variable, and the variable name used throughout the program. For example:

```
10 PI=3.14159
20 PRINT "AREA OF A CIRCLE IS THE RADIUS
 SQUARED TIMES ";PI
```

4. Literal strings require 2 bytes overhead and 1 byte for each character (including all spaces) in the string.

5. String variables take 9 bytes each plus the length of the variable name (including spaces) plus the space eaten up by the DIM statement plus the size of the string itself (1 byte per character, including spaces) when it is defined. Obviously, the use of string variables is very costly in terms of RAM.

6. Definition of a new matrix requires 15 bytes plus the length of the matrix variable name plus the space needed for the DIM statement plus 6 times the size of the matrix (product of the number of rows and the number of columns). Thus, a 25 row by 4 column matrix would require 15 + approximately 3 (for variable name) + approximately 10 (for the DIM statement) + 6 times 100 (the matrix size), or about 630 bytes.

7. Each character after REM takes one byte of memory. Remarks are helpful to people trying to understand a program, but sometimes it is necessary to remove remark statements to save memory.

8. Subroutines can save memory because one subroutine and several short calls take less memory than duplicating the code several times. On the other hand, a subroutine that is only called once takes extra bytes for the GOSUB and RETURN statements.

9. Parentheses take one byte each. Extra parentheses are a good idea in some cases if they make an expression more understandable to the programmer. However, removing unnecessary parentheses and relying on operator precedence will same a few bytes.

PROGRAMMING IN MACHINE LANGUAGE

Machine language is written entirely in binary code. The ATARI Personal Computer contains a 6502 microprocessor and it is possible to call 6502 machine code subroutines from BASIC using the USR function. Short routines may then be entered into a program by hand assembly (if necessary).

Before it returns to BASIC, the assembly language routine must do a pull accumulator (**PLA**) instruction to remove the number (N) of input arguments off the stack. If this number is not 0, then all of the input arguments must be popped off the stack also using PLA. (See Figure 6-1).

The subroutine should end by placing the low byte of its result in location 212 (decimal), and then return to BASIC using an **RTS** (Return from Subroutine) instruction. The BASIC interpreter will convert the 2-byte binary number stored in locations 212 and 213 into an integer between 0 and 65535 in floating-point format to obtain the value returned by the USR function.

The ADR function may be used to pass data that is stored in arrays or strings to a subroutine in machine language. Use the ADR function to get the address of the array or string, and then use this address as one of the USR input arguments.

The following program, Hexcode Loader, provides the means of entering hexadecimal codes, converting each hexadecimal number to decimal, and storing the decimal number into an array. The array is then executed as an assembly language subroutine. (An array is used to allocate space in memory for the routine.)

1. To use this program, first enter it. After entering it, save this program on disk or cassette for future use.

```
10 GRAPHICS 0:PRINT "HEXCODE LOADER PROG
RAM":PRINT
20 REM STORES DECIMAL EQUIVALENTS IN ARR
AY A, OUTPUTS IN PRINTED 'DATA STATEMENT
S' AT
21 REM LINE NUMBER 1500.
30 REM USER THEN PLACES CURSOR ON PRINTE
D OUTPUT LINE, HITS "RETURN", AND ENTERS

31 REM REST OF BASIC PROGRAM INCLUDING U
SR STATEMENT.
40 DIM A(50),HEX$(5)
50 REM INPUT,CONVERSION,STORAGE OF DATA.

60 N=0:PRINT "ENTER 1 HEX CODE.  IF LAST
 ONE IS IN, ENTER 'DONE'.";
70 INPUT HEX$
80 IF HEX$="DONE" THEN N=999:GOTO 130
90 FOR I=1 TO LEN(HEX$)
100 IF HEX$(I,I)<="9" THEN N=N%16+VAL(HE
X$(I,I)):GOTO 120
110 N=N%16+ASC(HEX$(I,I))-ASC("A")+10
120 NEXT I
130 PRINT N:C=C+1
140 A(C)=N
150 IF N<>999 THEN GOTO 60
190 REM PRINT OUT DATA LINE AT 1500
200 GRAPHICS 0:PRINT "1500 DATA";
210 C=0
220 C=C+1
230 IF A(C)=999 THEN PRINT "999":STOP
240 PRINT A(C);",";
250 A(C)=0
260 GOTO 220
300 PRINT "PUT CORRECT NUMBER OF HEX BYT
ES IN LINE 1000.":STOP :REM TRAP LINE
999 REM ** EXECUTION MODULE **
1000 CLR :BYTES=0
1010 TRAP 300:DIM E$(1),E(INT(BYTES/6)+1
)
1030 FOR I=1 TO BYTES
1040 READ A:IF A>255 THEN GOTO 1060
1050 POKE ADR(E$)+I,A
1060 NEXT I
1070 REM BASIC PART OF USER'S PROGRAM FO
LLOWS
```

Figure 11-1. Hexcode Loader Input Program

2. Now add the BASIC language part of your program starting at line 1080 including the USR function that calls the machine language subroutine. (See example below.)

3. Count the total number of hex codes to be entered and enter this number on line 1000 when requested. If another number is already entered, simply replace it.

4. Run the program and enter the hexadecimal codes of the machine level subroutine pressing RETURN after each entry. After the last entry, type DONE and press RETURN.

5. Now the DATA line (1500) displays on the screen. It will not be entered into the program until the cursor is moved to the DATA line and RETURN is pressed.

6. Add a program line 5 GOTO 1000 to bypass the hexcode loader (or delete the hexcode loader through line 260). Now save the completed program by using CSAVE or SAVE. It is important to do this *before* executing the part of the program containing the USR call. A mistake in a machine language routine may cause the system to crash. If the system does hang up, press SYSTEM RESET. If the system doesn't respond, turn power off and on again, reload the program, and correct it.

Note: This method only works with *relocatable* machine language routines.

The following two sample programs can each be entered into the Hexcode Loader program. The first program prints NOTHING IS MOVING while the machine program changes the colors. The second sample program displays a BASIC graphics design, then changes colors.

```
1080 GRAPHICS 1+16
1090 FOR I=1 TO 6
1100 PRINT #6;"nothing is moving!"
1110 PRINT #6;"NOTHING IS MOVING!"
1120 PRINT #6;"nothing is moving!"
1130 PRINT #6;"NOTHING IS MOVING!"
1140 NEXT I
1150 Q=USR(ADR(E$)+1)
1160 FOR I=1 TO 25:NEXT I:GOTO 1150
```

After entering this program, check that line 1000 reads:

```
1000 CLR:BYTES = 21
```

Type RUN RETURN.

Now enter the hexadecimal codes as shown column by column.

68	2
A2	E8
0	E0
AC	3
C4	90
2	F5
BD	8C
C5	C7
2	2
9D	60
C4	

BYTES = 21

When completed, type DONE and press RETURN. Now place the cursor after the last entry (999) on the DATA line and press RETURN.

Now run the program by typing GOTO 1000 and pressing RETURN, or if line 5 has been added, type RUN RETURN. Press BREAK to stop program and delete line 5.

The second program, which follows, should be entered in place of the NOTHING IS MOVING program. Be sure to check the BYTES = _____ count in line 1000. Follow steps 2 through 6.

```
1080 GRAPHICS 7+16
1090 SETCOLOR 0,9,4
1100 SETCOLOR 1,9,8
1110 SETCOLOR 2,9,4
1120 CR=1
1130 FOR X=0 TO 159
1140 COLOR INT(CR)
1150 PLOT 80,0
1160 DRAWTO X,95
1170 CR=CR+0.125
1180 IF CR=4 THEN CR=1
1190 NEXT X
1200 X=USR(ADR(E$)+1)
1210 FOR I=1 TO 15:NEXT I
1220 GOTO 1200
```

Type RUN RETURN

Enter the hexadecimal codes for this program column by column.

68	2
A2	E8
0	E0
AC	2
C4	90
2	F5
BD	8C
C5	C6
2	2
9D	60
C4	

BYTES = 21

When completed, type DONE and press RETURN. Now place the cursor after the last entry (999) on the DATA line and press RETURN.

Now run the program by typing GOTO 1000 and pressing RETURN, or add line 5 GOTO 1000 and type RUN RETURN. Press BREAK to stop program and delete line 5.

Figure 11-2 illustrates an assembler subroutine used to rotate colors which might prove useful. It is included here for the information of the user.

Assembler Subroutine to Rotate Colors..

Address	Object Code	Line No.	Label	Mnemonic	Data	
		0100				Routine to rotate COLOR data
		0110				From one register to another.
		0120				4 colors are rotated.
		0130				
		0140				Operating system address
02C4		0150				COLOR 0 = $02C4
02C5		0160				COLOR 1 = $02C5
02C6		0170				COLOR 2 = $02C6
02C7		0175				COLOR 3 = $02C7
		0180				
		0190		* =	$6000	Machine program starting address*
6000	68	0200		PLA		Pop stack (See Chapter 4)
6001	A200	0210		LDX	#0	Zero the X register
6003	ACC402	0220		LDY	COLOR0	Save COLOR 0
6006	BDC502	0230	LOOP	LDA	COLOR1,X	
6009	9DC402	0240		STA	COLOR0,X	
600C	E8	0250		INX		Increment the X register (add one)
600D	E002	0260		CPX	#3	Compare contents of X register with 2
600F	90F5	0270		BCC	LOOP	Loop if X register contents are less than 2
6011	8CC602	0280		STY	COLOR3	Save COLOR 0 in COLOR 3
6014	60	0290		RTS		Return from machine level sub-routine
Assembler Prints This		This Portion is Source Information Programmer Enters Using Atari Assembler Cartridge				

# Indicates data (source)
* Routine is relocatable
$ Indicates a hexadecimal number

Figure 11-2. Assembler Subroutine To Rotate Colors

NOTES

APPENDIX A

ALPHABETICAL DIRECTORY OF BASIC RESERVED WORDS

Note: The period is mandatory after all abbreviated keywords.

RESERVED WORD:	ABBREVIATION:	BRIEF SUMMARY OF BASIC STATEMENT
ABS		Function returns absolute value (unsigned) of the variable or expression.
ADR		Function returns memory address of a string.
AND		Logical operator: Expression is true only if both subexpressions joined by **AND** are true.
ASC		String function returns the numeric value of a single string character.
ATN		Function returns the arctangent of a number or expression in radians or degrees.
BYE	**B.**	Exit from BASIC and return to the resident operating system or console processor.
CLOAD	**CLOA.**	Loads data from Program Recorder into RAM.
CHR$		String function returns a single string byte equivalent to a numeric value between 0 and 255 in ATASCII code.
CLOG		Function returns the base 10 logarithm of an expression.
CLOSE	**CL.**	I/O statement used to close a file at the conclusion of I/O operations.
CLR		The opposite of DIM: Undimensions all strings; matrices.
COLOR	**C.**	Chooses color register to be used in color graphics work.
COM		Same as DIM.
CONT	**CON.**	Continue. Causes a program to restart execution on the next line following use of the BREAK key or encountering a **STOP**.
COS		Function returns the cosine of the variable or expression (degrees or radians).
CSAVE		Outputs data from RAM to the Program Recorder for tape storage.

RESERVED WORD:	ABBREVIATION:	BRIEF SUMMARY OF BASIC STATEMENT
DATA	**D.**	Part of **READ/DATA** combination. Used to identify the succeeding items (which must be separated by commas) as individual data items.
DEG	**DE.**	Statement **DEG** tells computer to perform trigonometric functions in degrees instead of radians. (Default in radians.)
DIM	**DI.**	Reserves the specified amount of memory for matrix, array, or string. All string variables, arrays, matrices must be dimensioned with a DIM statement.
DOS	**DO.**	Reserved word for disk operators. Causes the menu to be displayed. (See *DOS Manual.*)
DRAWTO	**DR.**	Draws a straight line between a plotted point and specified point.
END		Stops program execution; closes files; turns off sounds. Program may be restarted using **CONT**. (Note: **END** may be used more than once in a program.)
ENTER	**E.**	I/O command used to store data or programs in untokenized (source) form.
EXP		Function returns e (2.7182818) raised to the specified power.
FOR	**F.**	Used with **NEXT** to establish **FOR/NEXT** loops. Introduces the range that the loop variable will operate in during the execution of loop.
FRE		Function returns the amount of remaining user memory (in bytes).
GET	**GE.**	Used mostly with disk operations to input a single byte of data.
GOSUB	**GOS.**	Branch to a subroutine beginning at the specified line number.
GOTO	**G.**	Unconditional branch to a specified line number.
GRAPHICS	**GR.**	Specifies which of the eight graphics modes is to be used. **GR.0** may be used to clear screen.
IF		Used to cause conditional branching or to execute another statement on the same line (only if the first expression is true).
INPUT	**I.**	Causes computer to ask for input from keyboard. Execution continues only when RETURN key is pressed after inputting data.
INT		Function returns the next lowest whole integer below the specified value. Rounding is always downward, even when number is negative.
LEN		String function returns the length of the specified string in bytes or characters (1 byte contains 1 character).

RESERVED WORD:	ABBREVIATION:	BRIEF SUMMARY OF BASIC STATEMENT
LET	**LE.**	Assigns a value to a specific variable name. LET is optional in Atari BASIC, and may be simply omitted.
LIST	**L.**	Display or otherwise output the program list.
LOAD	**LO.**	Input from disk, etc. into the computer.
LOCATE	**LOC.**	Graphics: Stores, in a specified variable, the value that controls a specified graphics point.
LOG		Function returns the natural logarithm of a number.
LPRINT	**LP.**	Command to line printer to print the specified message.
NEW		Erases all contents of user RAM.
NEXT	**N.**	Causes a **FOR/NEXT** loop to terminate or continue depending on the particular variables or expressions. All loops are executed at least once.
NOT		A "1" is returned only if the expression is NOT true. If it is true, a "0" is returned.
NOTE	**NO.**	See *DOS/FMS Manual*...used only in disk operations.
ON		Used with **GOTO** or **GOSUB** for branching purposes. Multiple branches to different line numbers are possible depending on the value of the **ON** variable or expression.
OPEN	**O.**	Opens the specified file for input of output operations.
OR		Logical operator used between two expressions. If either one is true, a "1" is evaluated. A "0" results only if both are false.
PADDLE		Function returns position of the paddle game controller.
PEEK		Function returns decimal form of contents of specified memory location (RAM or ROM).
PLOT	**PL.**	Causes a single point to be plotted at the X,Y location specified.
POINT	**P.**	Used with disk operations only.
POKE	**POK.**	Insert the specified byte into the specified memory location. May be used only with RAM. Don't try to POKE ROM or you'll get an error.
POP		Removes the loop variable from the **GOSUB** stack. Used when departure from the loop is made in other than normal manner.
POSITION	**POS.**	Sets the cursor to the specified screen position.
PRINT	**PR.** or **?**	I/O command causes output from the computer to the specified output device.

RESERVED WORD:	ABBREVIATION:	BRIEF SUMMARY OF BASIC STATEMENT
PTRIG		Function returns status of the trigger button on game controllers.
PUT	**PU.**	Causes output of a single byte of data from the computer to the specified device.
RAD		Specifies that information is in radians rather than degrees when using the trigonometric functions. Default is to **RAD**. (See **DEG**.)
READ	**REA.**	Read the next items in the **DATA** list and assign to specified variables.
REM	**R.** or **.** SPACE	Remarks. This statement does nothing, but comments may be printed within the program list for future reference by the programmer. Statements on a line that starts with **REM** are not executed.
RESTORE	**RES.**	Allows **DATA** to be **read** more than once.
RETURN	**RET.**	**RETURN** from subroutine to the statement immediately following the one in which **GOSUB** appeared.
RND		Function returns a random number between 0 and 1, but never 1.
RUN	**RU.**	Execute the program. Sets normal variables to 0, undims arrays and string.
SAVE	**S.**	I/O statement causes data or program to be recorded on disk under filespec provided with **SAVE**.
SETCOLOR	**SE.**	Store hue and luminance color data in a particular color register.
SGN		Function returns +1 if value is positive, 0 if zero, −1 if negative.
SIN		Function returns trigonometric sine of given value (**DEG** or **RAD**).
SOUND	**SO.**	Controls register, sound pitch, distortion, and volume of a tone or note.
SQR		Function returns the square root of the specified value.
STATUS	**ST.**	Calls status routine for specified device.
STEP		Used with **FOR/NEXT**. Determines quality to be skipped between each pair of loop variable values.
STICK		Function returns position of stick game controller.
STRIG		Function returns 1 if stick trigger button not pressed, 0 if pressed.
STOP	**STO.**	Causes execution to stop, but does not close files or turn off sounds.

RESERVED WORD:	ABBREVIATION:	BRIEF SUMMARY OF BASIC STATEMENT
STR$		Function returns a character string equal to numeric value given. For example: **STR$(65)** returns 65 as a string.
THEN		Used with **IF**: If expression is true, the **THEN** statements are executed. If the expression is false, control passes to next line.
TO		Used with **FOR** as in "FOR X = 1 TO 10". Separates the loop range expressions.
TRAP	**T.**	Takes control of program in case of an **INPUT** error and directs execution to a specified line number.
USR		Function returns results of a machine-language subroutine.
VAL		Function returns the equivalent numeric value of a string.
XIO	**X.**	General I/O statement used with disk operations (see *DOS/FMS Manual*) and in graphics work (Fill).

NOTES

APPENDIX B

ERROR MESSAGES

ERROR CODE NO.	ERROR CODE MESSAGE
2	**Memory insufficient** to store the statement or the new variable name or to DIM a new string variable.
3	**Value Error:** A value expected to be a positive integer is negative, a value expected to be within a specific range is not.
4	**Too Many Variables:** A maximum of 128 different variable names is allowed. (See **Variable Name Limit**.)
5	**String Length Error:** Attempted to store beyond the DIMensioned string length.
6	**Out of Data Error:** READ statement requires more data items than supplied by DATA statement(s).
7	**Number greater than 32767:** Value is not a positive integer or is greater than 32767.
8	**Input Statement Error:** Attempted to INPUT a non-numeric value into a numeric variable.
9	**Array or String DIM Error:** DIM size is greater than 32767 or an array/martix reference is out of the range of the dimensioned size, or the array/matrix or string has been already DIMensioned, or a reference has been made to an undimensioned array or string.
10	**Argument Stack Overflow:** There are too many GOSUBs or too large an expression.
11	**Floating Point Overflow/Underflow Error:** Attempted to divide by zero or refer to a number larger than 1×10^{98} or smaller than $1 \times 10 - {}^{99}$.
12	**Line Not Found:** A GOSUB, GOTO, or THEN referenced a non-existent line number.
13	**No Matching FOR Statement:** A NEXT was encountered without a previous FOR, or nested FOR/NEXT statements do not match properly. (Error is reported at the NEXT statement, not at FOR).
14	**Line Too Long Error:** The statement is too complex or too long for BASIC to handle.
15	**GOSUB or FOR Line Deleted:** A NEXT or RETURN statement was encountered and the corresponding FOR or GOSUB has been deleted since the last RUN.

ERROR CODE NO.	ERROR CODE MESSAGE
16	**RETURN Error:** A RETURN was encountered without a matching GOSUB.
17	**Garbage Error:** Execution of "garbage" (bad RAM bits) was attempted. This error code may indicate a hardware problem, but may also be the result of faulty use of POKE. Try typing NEW or powering down, then re-enter the program without any POKE commands.
18	**Invalid String Character:** String does not start with a valid character, or string in VAL statement is not a numeric string.
Note:	**The following are INPUT/OUTPUT errors that result during the use of disk drives, printers, or other accessory devices. Further information is provided with the auxiliary hardware.**
19	**LOAD program Too Long:** Insufficient memory remains to complete LOAD.
20	**Device Number Larger** than 7 or Equal to 0.
21	**LOAD File Error:** Attempted to LOAD a non-LOAD file.
128	**BREAK Abort:** User hit BREAK key during I/O operation.
129	**IOCB**[1] already open.
130	**Nonexistent Device** specified.
131	**IOCB Write Only.** READ command to a write-only device (Printer).
132	**Invalid Command:** The command is invalid for this device.
133	**Device or File not Open:** No OPEN specified for the device.
134	**Bad IOCB Number:** Illegal device number.
135	**IOCB Read Only Error:** WRITE command to a read-only device.
136	**EOF:** End of File read has been reached. (***NOTE:*** This message may occur when using cassette files.)
137	**Truncated Record:** Attempt to read a record longer than 256 characters.
138	**Device Timeout.** Device doesn't respond.
139	**Device NAK:** Garbage at serial port or bad disk drive.
140	**Serial bus** input framing error.
141	**Cursor out of range** for particular mode.
142	**Serial bus data frame overrun.**

[1]IOCB refers to Input/Output Control Block. The device number is the same as the IOCB number.

ERROR CODE NO.	ERROR CODE MESSAGE
143	**Serial bus data frame checksum error.**
144	**Device done error** (invalid "done" byte): Attempt to write on a write-protected diskette.
145	**Read after write compare error** (disk handler) or bad screen mode handler.
146	**Function not implemented** in handler.
147	**Insufficient RAM** for operating selected graphics mode.
160	**Drive number error.**
161	**Too many OPEN files** (no sector buffer available).
162	**Disk full** (no free sectors).
163	**Unrecoverable system data I/O error.**
164	**File number mismatch:** Links on disk are messed up.
165	**File name error.**
166	**POINT data length error.**
167	**File locked.**
168	**Command invalid** (special operation code).
169	**Directory full** (64 files).
170	**File not found.**
171	**POINT invalid.**

APPENDIX C

ATASCII CHARACTER SET

DECIMAL CODE	HEXADECIMAL CODE	CHARACTER
0	0	
1	1	
2	2	
3	3	
4	4	
5	5	
6	6	
7	7	
8	8	
9	9	
10	A	
11	B	
12	C	

DECIMAL CODE	HEXADECIMAL CODE	CHARACTER
13	D	
14	E	
15	F	
16	10	
17	11	
18	12	
19	13	
20	14	
21	15	
22	16	
23	17	
24	18	
25	19	

DECIMAL CODE	HEXADECIMAL CODE	CHARACTER
26	1A	
27	1B	
28	1C	
29	1D	
30	1E	
31	1F	
32	20	Space
33	21	!
34	22	”
35	23	#
36	24	$
37	25	%
38	26	&

DECIMAL CODE	HEXADECIMAL CODE	CHARACTER	DECIMAL CODE	HEXADECIMAL CODE	CHARACTER	DECIMAL CODE	HEXADECIMAL CODE	CHARACTER
39	27	'	55	37	7	71	47	G
40	28	(	56	38	8	72	48	H
41	29	)	57	39	9	73	49	I
42	2A	*	58	3A	:	74	4A	J
43	2B	+	59	3B	;	75	4B	K
44	2C	,	60	3C	<	76	4C	L
45	2D	-	61	3D	=	77	4D	M
46	2E	.	62	3E	>	78	4E	N
47	2F	/	63	3F	?	79	4F	O
48	30	0	64	40	@	80	50	P
49	31	1	65	41	A	81	51	Q
50	32	2	66	42	B	82	52	R
51	33	3	67	43	C	83	53	S
52	34	4	68	44	D	84	54	T
53	35	5	69	45	E	85	55	U
54	36	6	70	46	F	86	56	V

DECIMAL CODE	HEXADECIMAL CODE	CHARACTER	DECIMAL CODE	HEXADECIMAL CODE	CHARACTER	DECIMAL CODE	HEXADECIMAL CODE	CHARACTER
87	57	W	103	67	g	119	77	w
88	58	X	104	68	h	120	78	x
89	59	Y	105	69	i	121	79	y
90	5A	Z	106	6A	j	122	7A	z
91	5B	[	107	6B	k	123	7B	
92	5C	\	108	6C	l	124	7C	|
93	5D	]	109	6D	m	125	7D	
94	5E	^	110	6E	n	126	7E	
95	5F	_	111	6F	o	127	7F	
96	60		112	70	p	128	80	
97	61	a	113	71	q	129	81	
98	62	b	114	72	r	130	82	
99	63	c	115	73	s	131	83	
100	64	d	116	74	t	132	84	
101	65	e	117	75	u	133	85	
102	66	f	118	76	v	134	86	

DECIMAL CODE	HEXADECIMAL CODE	CHARACTER	DECIMAL CODE	HEXADECIMAL CODE	CHARACTER	DECIMAL CODE	HEXADECIMAL CODE	CHARACTER
135	87		151	97		167	A7	
136	88		152	98		168	A8	
137	89		153	99		169	A9	
138	8A		154	9A		170	AA	
139	8B		155	9B	(EOL) RETURN	171	AB	
140	8C		156	9C		172	AC	
141	8D		157	9D		173	AD	
142	8E		158	9E		174	AE	
143	8F		159	9F		175	AF	
144	90		160	A0		176	B0	
145	91		161	A1		177	B1	
146	92		162	A2		178	B2	
147	93		163	A3		179	B3	
148	94		164	A4		180	B4	
149	95		165	A5		181	B5	
150	96		166	A6		182	B6	

DECIMAL CODE	HEXADECIMAL CODE	CHARACTER	DECIMAL CODE	HEXADECIMAL CODE	CHARACTER	DECIMAL CODE	HEXADECIMAL CODE	CHARACTER
183	B7		199	C7		215	D7	
184	B8		200	C8		216	D8	
185	B9		201	C9		217	D9	
186	BA		202	CA		218	DA	
187	BB		203	CB		219	DB	
188	BC		204	CC		220	DC	
189	BD		205	CD		221	DD	
190	BE		206	CE		222	DE	
191	BF		207	CF		223	DF	
192	C0		208	D0		224	E0	
193	C1		209	D1		225	E1	
194	C2		210	D2		226	E2	
195	C3		211	D3		227	E3	
196	C4		212	D4		228	E4	
197	C5		213	D5		229	E5	
198	C6		214	D6		230	E6	

DECIMAL CODE	HEXADECIMAL CODE	CHARACTER	DECIMAL CODE	HEXADECIMAL CODE	CHARACTER	DECIMAL CODE	HEXADECIMAL CODE	CHARACTER
231	E7		240	F0		249	F9	
232	E8		241	F1		250	FA	
233	E9		242	F2		251	FB	
234	EA		243	F3		252	FC	
235	EB		244	F4		253	FD	 *(Buzzer)*
236	EC		245	F5		254	FE	 *(Delete character)*
237	ED		246	F6		255	FF	 *(Insert character)*
238	EE		247	F7				
239	EF		248	F8				

See Appendix H for a user program that performs decimal/hexadecimal conversion.

Notes:

1. ATASCII stands for "ATARI ASCII". Letters and numbers have the same values as those in ASCII, but some of the special characters are different.
2. Except as shown, characters from 128-255 are reverse colors of 1 to 127.
3. Add 32 to upper case code to get lower case code for same letter.
4. To get ATASCII code, tell computer (direct mode) to PRINT ASC ("_____") Fill blank with letter, character, or number of code. Must use the quotes!
5. On pages C-1 and C-3, the normal display keycaps are shown as white symbols on a black background; on pages C-4 and C-6 inverse keycap symbols are shown as black on a white background.

APPENDIX D

ATARI 400/800 MEMORY MAP

ADDRESS		CONTENTS
Decimal	**Hexadecimal**	
65535 57344	FFFF E000	OPERATING SYSTEM ROM
57343 55296	DFFF D800	FLOATING POINT ROM
55295 53248	D7FF D000	HARDWARE REGISTERS
53247 49152	CFFF C000	NOT USED
49151 40960	BFFF A000	CARTRIDGE SLOT A (may be RAM if no A or B cartridge)
40959 32768	9FFF 8000	CARTRIDGE SLOT B (may be RAM if no B cartridge) ← *RAMTOP (MSB)*
32767	7FFF	(7FFF if 32K system) DISPLAY DATA (size varies)
31755	7CIF	DISPLAY LIST (size varies) (7C1F if 32K system, (GRAPHICS 0) ← *OS MEMTOP*
		FREE RAM (size varies) ← *BASIC MEMTOP*
10880	2A80	BASIC program, buffers, tables, run-time stack. (2A80 if DOS, may vary) ← *OS MEMLO*, *BASIC LOMEM*
10879 9856	2A7F 2680	**DISK OPERATING SYSTEM (2A7F-700)** DISK I/O BUFFERS (current DOS)
9855 4864	267F 1300	DISK OPERATING SYSTEM RAM (current DOS)

ADDRESS		CONTENTS
Decimal	**Hexadecimal**	
4863 1792	12FF 700	FILE MANAGEMENT SYSTEM RAM (current DOS)
1791 1536	6FF 600	FREE RAM
1535 1406	5FF 57E	FLOATING POINT (used by BASIC)
1405 1152	57D 480	BASIC CARTRIDGE
1151 1021	47F 3FD	**OPERATING SYSTEM RAM (47F-200)** CASSETTE BUFFER
1020 1000	3FC 3E8	RESERVED
999 960	3E7 3C0	PRINTER BUFFER
959 832	3BF 340	IOCB's
831 512	33F 200	MISCELLANEOUS OS VARIABLES
511 256	1FF 100	HARDWARE STACK
255 212	FF D4	**PAGE ZERO** FLOATING POINT (used by BASIC)
211 210	D3 D2	BASIC or CARTRIDGE PROGRAM
209 208	D1 D0	FREE BASIC RAM
207 203	CF CB	FREE BASIC AND ASSEMBLER RAM
202 176 128	CA B0 80	FREE ASSEMBLER RAM ASSEMBLER ZERO PAGE BASIC ZERO PAGE
127 0	7F 0	OPERATING SYSTEM RAM

As the addresses for the top of RAM, OS, and BASIC and the ends of OS and BASIC vary according to the amount of memory, these addresses are indicated by pointers. The pointer addresses for each are defined in Appendix I.

APPENDIX E

DERIVED FUNCTIONS

Derived Functions	Derived Functions in Terms of Atari Functions
Secant	SEC(X) = 1/COS(X)
Cosecant	CSC(X) = 1/SIN(X)
Inverse Sine	ARCSIN(X) = ATN(X/SQR(-X*X + 1))
Inverse Cosine	ARCCOS(X) = – ATN(X/SQR(– X*X + 1) + CONSTANT
Inverse Secant	ARSEC(X) = ATN(SQR(X*X-1)) + (SGN(X-1)*CONSTANT
Inverse Cosecant	ARCCSC(X) = ATN(1/SQR(X*X-1)) + (SGN(X-1)*CONSTANT
Inverse Cotangent	ARCCOT(X) = ATN(X) + CONSTANT
Hyperbolic Sine	SINH(X) = (EXP(X)-EXP(-X))/2
Hyperbolic Cosine	COSH(X) = (EXP(X) + EXP(-X))/2
Hyperbolic Tangent	TANH(X) = -EXP(-X)/(EXP(X) + EXP(-X))*2 + 1
Hyperbolic Secant	SECH(X) = 2/(EXP(X) + EXP(-X))
Hyperbolic Cosecant	CSCH(X) = 2/(EXP(X)-EXP(-X))
Hyperbolic Cotangent	COTH(X) = EXP(-X)/(EXP(X)-EXP(-X))*2 + 1
Inverse Hyperbolic Sine	ARCSINH(X) = LOG(X + SQR(X*X + 1))
Inverse Hyperbolic Cosine	ARCCOSH(X) + LOG(X + SQR(X*X-1))
Inverse Hyperbolic Tangent	ARCTANH(X) = LOG((1 + X)/(1-X))/2
Inverse Hyperbolic Secant	ARCSECH(X) = LOG((SQR(-X*X + 1) + 1)/X)
Inverse Hyperbolic Cosecant	ARCCSCH(X) = LOG((SGN(X)*SQR(X*X + 1) + 1)/X)
Inverse Hyperbolic Cotangent	ARCCOTH(X) = LOG((X + 1)/(X-1))/2

Notes:

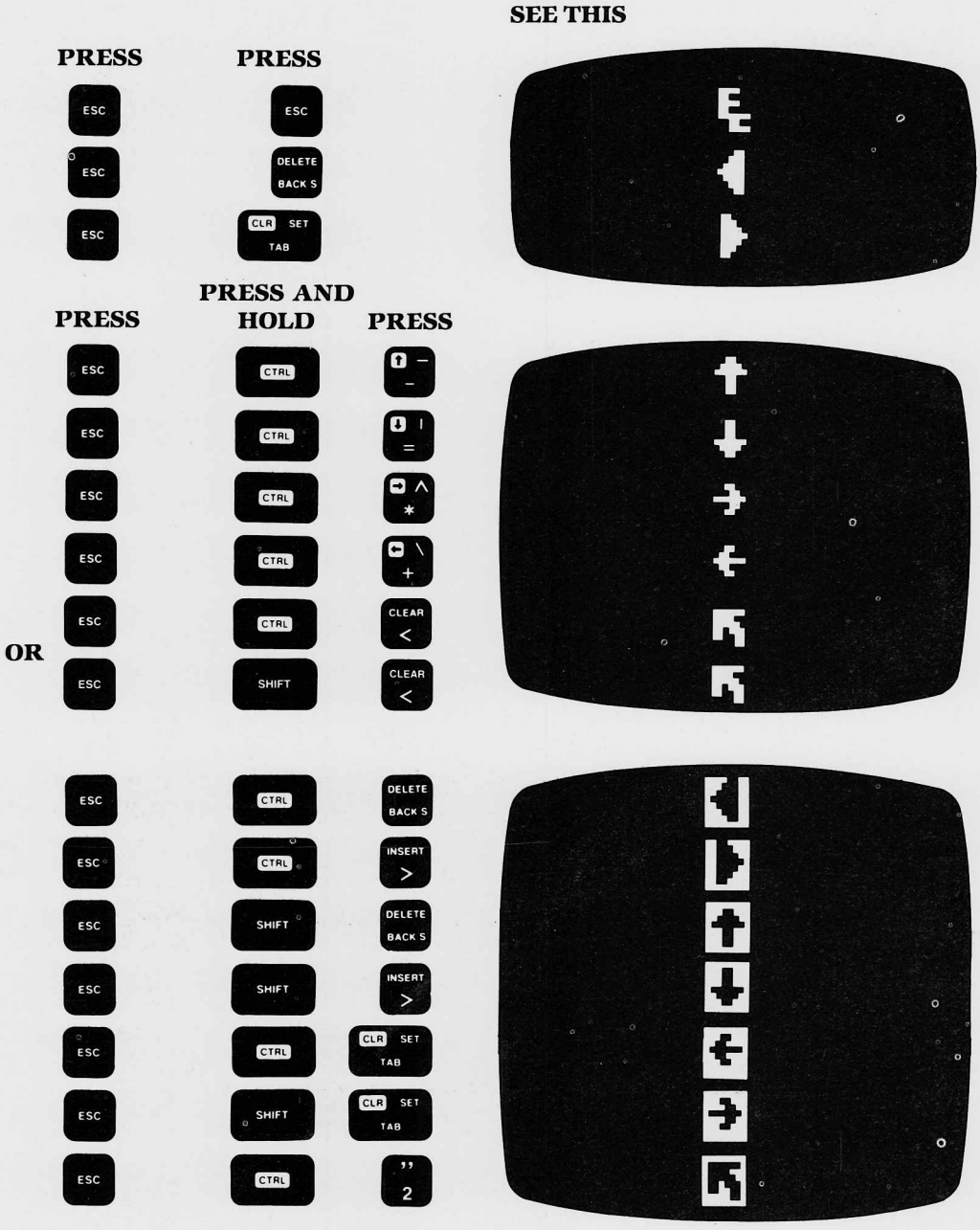
1. If in RAD (default) mode, constant = 1.57079633
   If in DEG mode, constant = 90.
2. In this chart, the variable X in parentheses represents the value or expression to be evaluated by the derived function. Obviously, any variable name is permissible, as long as it represents the number or expression to be evaluated.

NOTES

APPENDIX F

PRINTED VERSIONS OF CONTROL CHARACTERS

The cursor and screen control characters can be placed in a string in a program or used as a Direct mode statement by pressing the ESC key before entering the character from the keyboard. This causes the special symbols which are shown below to be displayed. (Refer to Section 1 - ESC Key.)



NOTES

APPENDIX G

GLOSSARY

Alphanumeric: The alphabetic letters A-Z, the numbers 0-9, and some symbols. (No punctuation marks or graphics symbols).

Array: A list of numerical values stored in a series of memory locations preceded by a DIM statement. May be referred to by use of an array variable, and its individual elements are referred to by subscripted variable names.

ATASCII: Stands for Atari American Standard Code for Information Interchange.

BASIC: High level programming language. Acronym for Beginner's All-purpose Symbolic Intruction Code. BASIC is always written using all capital letters. Developed by Mssrs. Kemeny and Kurtz at Dartmouth College in 1963.

Binary: A number system using the base two. Thus the only possible digits are 0 and 1, which may be used in a computer to represent true and false, on and off, etc.

Bit: Short for Binary Digit. A bit can be thought of as representing true or false, whether a circuit is on or off, or any other type of two-possibility concept. A bit is the smallest unit of data with which a computer can work.

Branch: Atari BASIC executes a program in order of line numbers. This execution sequence can be altered by the programmer, and the program can be told to skip over a certain number of lines or return to a line earlier in the program. This contrived change in execution sequence is called "branching".

Bug: A mistake or error usually in the program or "software".

Byte: Usually eight bits (enough to represent the decimal number 255 or 11111111 in binary notation). A byte of data can be used to represent an ATASCII character or a number in the range of 0 to 255.

Central Processing Unit (CPU): In microcomputers such as the Atari systems, these are also called microprocessors or MPU. At one time, the CPU was that portion of any computer that controlled the memory and peripherals. Now the CPU or MPU is usually found on a single integrated circuit or "chip" (in Atari's case a 6502 microprocessor chip).

Code: Instructions written in a language understood by a computer.

Command: An instruction to the computer that is executed immediately. A good example is the BASIC command RUN. (See **Statement**.)

Computer: Any device that can receive and then follow instructions to manipulate information. Both the instructions and the information may be varied from moment to moment. The distinction between a computer and a programmable calculator lies in the computer's ability to manipulate text as well as numbers. Most calculators can only handle numbers.

Concatenation: The process of joining two or more strings together to form one longer string.

Control Characters: Characters produced by holding down the key labeled CTRL while simultaneously pressing another key.

CRT: Abbreviation for "cathrode ray tube" (the tube used in a TV set). In practice, this is often used to describe the television receiver used to display computer output. Also called a "monitor".

Cursor: A square displayed on the TV monitor that shows where the next typed character will be displayed.

Data: Information of any kind.

Debug: The process of locating and correcting mistakes and errors in a program.

Default: A mode or condition "assumed" by the computer until it is told to do something else. For example, it will "default" to screen and keyboard unless told to use other I/O devices.

Digital: Information that can be represented by a collection of bits. Virtually all modern computers, especially microcomputers, use the digital approach.

Diskette: A small disk. A record/playback medium like tape, but made in the shape of a flat disk that is placed inside a stiff envelope for protection. The advantage of the disk over cassette or other tape for memory storage is that access to any part of the disk is virtually immediate. The Atari 800 Personal Computer System can control up to 4 diskette drive peripherals simultaneously. In this manual, disk and diskette are used interchangeably.

DOS: Abbreviation for "disk operating system". The software or programs which facilitate use of a disk-drive system. DOS is pronounced either "dee oh ess" or "doss".

Editing: Making corrections or changes in a program or data.

Execute: To do what a command or program specifies. To RUN a program or portion thereof.

Expression: A combination of variables, numbers, and operators (like +, -, etc.) that can be evaluated to a single quantity. The quantity may be a string or a number.

Format: To specify the form in which something is to appear.

Hard Copy: Printed output as opposed to temporary TV monitor display.

Hardware: The physical apparatus and electronics that make up a computer.

Increment: Increase in value (usually) by adding one. Used a lot for counting (as in counting the number of repetitions through a loop).

Initialize: Set to an initial or starting value. In Atari BASIC, all non-array variables are initialized to zero when the command RUN is given. Array and string elements are not initialized.

Input: Information transfer to the computer. Output is information transfer away from the computer. In this manual, input and output are always in relation to the computer.

Interactive: A system that responds quickly to the user, usually within a second or two. All personal computer systems are interactive.

Interface: The electronics used to allow two devices to communicate.

IOCB Input/Output Control Block. A block of data in RAM that tells the Operating System the information it needs to know for an I/O operation.

I/O: Short for input/output, I/O devices include the keyboard, TV monitor, program recorder, printer, and disk drives.

K: Stands for "kilo" meaning "times 1000". Thus 1 KByte is (approximately) 1000 bytes. (Actually 1024 bytes.) Also, the device type code for the Keyboard.

Keyword: A word that has meaning as an instruction or command in a computer language, and thus must not be used as a variable name or at the beginning of a variable name.

Language: A set of conventions specifying how to tell a computer what to do.

Memory: The part of a computer (usually RAM or ROM) that stores data or information.

Menu: A list of options from which the user may choose.

Microcomputer: A computer based on a microprocessor chip; in Atari's case, the 6502.

Monitor: The television receiver used to display computer output.

Null String: A string consisting of no characters whatever.

OS: Abbreviation for Operating System. This is actually a collection of programs to aid the user in controlling the computer. Pronounced "oh ess".

Output: See **I/O**.

Parallel: Two or more things happening simultaneously. A parallel interface, for example, controls a number of distinct electrical signals at the same time. Opposite of serial.

Peripheral: An I/O device. See **I/O**.

Pixel: Picture Element. One point on the screen display. Size depends on graphics mode being used.

Precedence: Rules that determine the priority in which operations are conducted, especially with regard to the arithmetical/logical operators.

Program: A sequence of instructions that describes a process. A program must be in the language that the particular computer can understand.

Prompt: A symbol that appears on the monitor screen that indicates the computer is ready to accept keyboard input. In Atari BASIC, this takes the form of the word "READY". A "?" is also used to prompt a user to enter (input) information or take other appropriate action.

RAM: Random Access Memory. The main memory in most computers. RAM is used to store both programs and data.

Random Number Generator: May be hardware (as is Atari's) or a program that provides a number whose value is difficult to predict. Used primarily for decision-making in game programs, etc.

Reserved Word: See **Keyword**.

ROM: Read Only Memory. In this type of solid-state electronic memory, information is stored by the manufacturer and it cannot be changed by the user. Programs such as the BASIC interpreter and other cartridges used with the Atari systems use ROM.

Save: To copy a program or data into some location other than RAM (for example, diskette or tape).

Screen: The TV screen. In Atari BASIC, a particular I/O device codes "S:"

Serial: The opposite of parallel. Things happening only one at a time in sequence. Example: A serial interface.

Software: As opposed to Hardware. Refers to programs and data.

Special Character: A character that can be displayed by a computer but is neither a letter nor a numeral. The Atari graphics symbols are special characters. So are punctuation marks, etc.

Statement: An instruction to the computer. See also **Command**. While all commands may be considered statements, all statements are certainly not commands. A statement contains a line number (deferred mode), a keyword, the value to be operated on, and the RETURN command.

String: A sequence of letters, numerals, and other characters. May be stored in a string variable. The string variable's name must end with a $.

Subroutine: A part of a program that can be executed by a special statement (GOSUB) in BASIC: This effectively gives a single statement the power of a whole program. The subroutine is a very powerful construct.

Variable: A variable may be thought of as a box in which a value may be stored. Such values are typically numbers and strings.

Window: A portion of the TV display devoted to a specific purpose such as for graphics or text.

APPENDIX H

USER PROGRAMS

This appendix contains programs and routines that demonstrate the diverse capabilities of the Atari Personal Computer System. Included in this appendix is a Decimal/Hexadecimal program for those users who write programs that require this type of conversion.

CHECKBOOK BALANCER

This is one of the "traditional" programs that every beginning computerist writes. It allows entry of outstanding checks and uncredited deposits as well as cleared checks and credited deposits.

```
10 DIM A$(30),MSG$(40),MSG1$(30),MSG2$(3
0),MSG3$(30),MSG4$(30),MSG5$(30),MSG6$(3
0)
20 OUTSTAND=0
30 GRAPHICS 0:? :? "     CHECKBOOK BALAN
CER":?
40 ? "You may make corrections at any ti
me  by entering a negative dollar value.
"
50 MSG1$="OLD CHECK -- STILL OUTSTANDING
"
60 MSG2$="OLD DEPOSIT -- NOT CREDITED
"
70 MSG3$="OLD CHECK -- JUST CLEARED
"
80 MSG4$="OLD DEPOSIT -- JUST CREDITED
"
90 MSG5$="NEW CHECK (OR SERVICE CHARGE)
"
100 MSG6$="NEW DEPOSIT (OR INTEREST)
 "
150 TRAP 150:? "Enter beginning balance
from your     checkbook";:INPUT YOURBAL
160 TRAP 160:? "Enter beginning balance
from your bankstatement";:INPUT BANKBAL
165 TRAP 40000
170 GOTO 190
180 CLOSE #1:? "PRINTER IS NOT OPERATION
AL."
185 ? "PLEASE CHECK CONNECTORS."
190 PERM=0
200 ? "Would you like a permanent record
 on  the printer";:INPUT A$
210 IF LEN(A$)=0 THEN 200
```

```
220 IF A$(1,1)="N" THEN 400
230 IF A$(1,1)<>"Y" THEN 200
240 TRAP 180
250 LPRINT :REM TEST PRINTER
260 PERM=1
280 LPRINT "YOUR BEGINNING BALANCE IS $"
;YOURBAL
290 LPRINT "BANK STATEMENT BEGINNING BAL
ANCE IS $";BANKBAL:LPRINT
400 TRAP 400:? :? "Choose one of the fol
lowing:"
410 ? "(1) ";MSG1$
415 ? "(2) ";MSG2$
420 ? "(3) ";MSG3$
425 ? "(4) ";MSG4$
430 ? "(5) ";MSG5$
435 ? "(6) ";MSG6$
440 ? "(7) DONE"
490 ?
500 INPUT N:IF N<1 OR N>7 THEN 400
505 TRAP 40000
510 ON N GOSUB 1000,2000,3000,4000,5000,
6000,7000
520 MSG$="NEW CHECKBOOK BALANCE IS
":AMOUNT=YOURBAL:GOSUB 8000
530 MSG$="NEW BANK STATEMENT BALANCE IS
":AMOUNT=BANKBAL:GOSUB 8000
540 MSG$="OUTSTANDING CHECKS-DEPOSITS=
":AMOUNT=OUTSTAND:GOSUB 8000
545 IF PERM THEN LPRINT
550 GOTO 400
1000 REM OLD CHECK -- STILL OUTSTANDING
1010 MSG$=MSG1$:GOSUB 8100
1020 OUTSTAND=OUTSTAND+AMOUNT
1030 RETURN
2000 REM OLD DEPOSIT -- STILL NOT CREDIT
ED
2010 MSG$=MSG2$:GOSUB 8100
2020 OUTSTAND=OUTSTAND-AMOUNT
2030 RETURN
3000 REM OLD CHECK -- JUST CLEARED
3010 MSG$=MSG3$:GOSUB 8100
3020 BANKBAL=BANKBAL-AMOUNT
3030 RETURN
4000 REM OLD DEPOSIT -- JUST CREDITED
4010 MSG$=MSG4$:GOSUB 8100
4020 BANKBAL=BANKBAL+AMOUNT
4030 RETURN
5000 REM NEW CHECK (OR SERVICE CHARGE) -
- JUST CLEARED
5010 MSG$=MSG5$:GOSUB 8100
5020 YOURBAL=YOURBAL-AMOUNT
```

```
5030 ? "IS NEW CHECK STILL OUTSTANDING";
:INPUT A$
5040 IF LEN(A$)=0 THEN 5030
5050 IF A$(1,1)<>"N" THEN 5060
5055 BANKBAL=BANKBAL-AMOUNT
5057 IF PERM THEN LPRINT "CHECK HAS CLEA
RED."
5058 RETURN
5060 IF A$(1,1)<>"Y" THEN 5030
5070 OUTSTAND=OUTSTAND+AMOUNT
5075 IF PERM THEN LPRINT "CHECK IS STILL
 OUTSTANDING."
5080 RETURN
6000 REM NEW DEPOSIT (OR INTEREST) -- JU
ST CREDITED
6010 MSG$=MSG6$:GOSUB 8100
6020 YOURBAL=YOURBAL+AMOUNT
6030 ? "HAS YOUR NEW DEPOSIT BEEN CREDIT
ED";:INPUT A$
6040 IF LEN(A$)=0 THEN 6030
6050 IF A$(1,1)<>"Y" THEN 6060
6052 BANKBAL=BANKBAL+AMOUNT
6053 IF PERM THEN LPRINT "DEPOSIT HAS BE
EN CREDITED."
6055 RETURN
6060 IF A$(1,1)<>"N" THEN 6030
6070 OUTSTAND=OUTSTAND-AMOUNT
6075 IF PERM THEN LPRINT "DEPOSIT HAS NO
T BEEN CREDITED."
6080 RETURN
7000 REM DONE
7010 ? "BANK'S BALANCE MINUS (OUTSTANDIN
G     CHECKS-DEPOSITS) SHOULD NOW EQUAL
YOURCHECKBOOK BALANCE."
7020 DIF=YOURBAL-(BANKBAL-OUTSTAND)
7030 IF DIF<>0 THEN 7040
7035 ? "IS $";BANKBAL;" THE ENDING BALAN
CE ON YOUR BANK STATEMENT";:INPUT A$
7036 IF LEN(A$)=0 THEN 7035
7037 IF A$(1,1)="Y" THEN ? "CONGRATULATI
ONS: YOUR CHECKBOOK       BALANCES!":END

7038 GOTO 7060
7040 IF DIF>0 THEN ? "YOUR CHECKBOOK TOT
AL IS $";DIF;" OVER YOUR BANK'S TOTAL. "
:GOTO 7060
7050 ? "YOUR CHECKBOOK TOTAL IS $";-DIF;
" UNDER YOUR BANK'S TOTAL."
7060 ? "WOULD YOU LIKE TO MAKE CORRECTIO
NS?"
7070 ? "REMEMBER, YOU CAN ENTER A NEGATI
VE    DOLLAR VALUE TO MAKE A CORRECTION.
"
```

```
7080 ? "ENTER Y OR N";:INPUT A$
7090 IF LEN(A$)=0 THEN END
7100 IF A$(1,1)="Y" THEN RETURN
7110 END
7999 REM MSG PRINTING ROUTINE
8000 ? MSG$;" $";AMOUNT
8010 IF PERM=1 THEN LPRINT MSG$;" $";AMO
UNT
8020 RETURN
8100 REM MSG PRINT & INPUT ROUTINE
8110 TRAP 8110:? "ENTER AMOUNT FOR ";MSG
$;:INPUT AMOUNT
8120 TRAP 40000
8130 IF PERM=1 THEN LPRINT MSG$;" $";AMO
UNT
8140 RETURN
```

BUBBLE SORT

This program uses the string comparison operator "<=" that orders strings according to the ATASCII values of the various characters. Since Atari BASIC does not have arrays of strings, all the strings used in this program are actually substrings of one large string. A bubble sort, though relatively slow if there are a lot of items to be stored, is easy to write, fairly short, and simpler to understand than more complex sorts.

```
10 DIM B$(1)
20 GRAPHICS 0:? :? "           STRING SO
RT":?
30 TRAP 30:? :? "Enter maximum strins le
nsth";:INPUT SLEN:SLEN1=SLEN-1
35 IF SLEN<1 OR INT(SLEN)<>SLEN THEN ? "
PLEASE ENTER A POSITIVE INTEGER > 0.":GO
TO 30
40 TRAP 40:? :? "Enter maximum number of
 entries."
41 ? "(Entries which are shorter than th
e   maximum will be padded with blanks.)
"
42 INPUT ENTRIES
45 IF ENTRIES<2 OR INT(ENTRIES)<>ENTRIES
 THEN ? "PLEASE ENTER A POSITIVE INTEGER
 > 1.":GOTO 40
47 TRAP 40000
50 DIM A$(SLEN*ENTRIES),TEMP$(SLEN)
60 ? :? "Enter strinss one at a time."
70 ? "Enter empty strins when done (just
 hitRETURN)."
75 ? :? "PLEASE STAND BY WHILE THE STRIN
GS ARE BEING CLEARED...";
80 FOR I=1 TO SLEN*ENTRIES:A$(I,I)=" ":N
EXT I
85 ? :?
90 I=1
100 FOR J=1 TO ENTRIES
110 ? "#";J;" ";:INPUT TEMP$
120 IF LEN(TEMP$)=0 THEN ENTRIES=J-1:GOT
O 190
130 A$(I,I+SLEN1)=TEMP$
140 I=I+SLEN
150 NEXT J
190 ? :? :? "PLEASE STAND BY WHILE THE S
TRINGS ARE BEING SORTED...";
200 GOSUB 1000:REM CALL SORT ROUTINE
202 ? :?
205 I=1
210 FOR K=1 TO ENTRIES
220 ? "#";K;" ";A$(I,I+SLEN1)
225 I=I+SLEN
230 NEXT K
240 TRAP 300:? :? "WOULD YOU LIKE A PRIN
```

```
TED COPY";:INPUT B$
250 IF B$(1,1)="Y" THEN 400
300 END
400 I=1:LPRINT :FOR K=1 TO ENTRIES
420 LPRINT "#";K;" ";A$(I,I+SLEN1)
430 I=I+SLEN:NEXT K:END
1000 REM STRING BUBBLE SORT ROUTINE
1010 REM INPUT: A$,SLEN,ENTRIES
1015 REM TEMP$ MUST HAVE A DIMENSION OF
SLEN.
1020 SLEN1=SLEN-1:MAX=SLEN*(ENTRIES-1)+1

1040 FOR I=1 TO MAX STEP SLEN
1050 DONE=1
1060 FOR K=1 TO MAX-I-SLEN1 STEP SLEN
1070 KSLEN1=K+SLEN1:KSLEN=K+SLEN:KSLENSL
EN1=KSLEN+SLEN1
1080 IF A$(K,KSLEN1)<=A$(KSLEN,KSLENSLEN
1) THEN GOTO 1110
1090 DONE=0
1100 TEMP$=A$(K,KSLEN1):A$(K,KSLEN1)=A$(
KSLEN,KSLENSLEN1):A$(KSLEN,KSLENSLEN1)=T
EMP$
1110 NEXT K
1120 IF DONE THEN RETURN
1130 NEXT I
1140 RETURN
```

TEXT MODES CHARACTER PRINT

This program prints the Atari characters in their default colors for text modes 0, 1, and 2. In entering this program, remember that the clear screen symbol "↰" is printed as "}".

```
1 DIM A$(1)
5 ? "}":REM CLEAR SCREEN
10 ? "GRAPHICS 0, 1, AND 2 (TEXT MODES)"

20 ? "DEMONSTRATION."
30 ? "DISPLAYS CHARACTER SETS FOR EACH M
ODE."
60 WAIT=1000:REM SUBROUTINE LINE NUMBER
70 CHBAS=756:REM CHARACTER BASE ADDRESS
80 UPPER=224:REM DEFAULT FOR CHBAS
90 LOWER=226:REM LOWER CASE LETTERS & GR
APHICS
95 GOSUB WAIT
100 FOR L=0 TO 2
112 REM USE E: FOR GRAPHICS 0
115 IF L=0 THEN OPEN #1,8,0,"E:":GOTO 11
8
116 REM USE S: FOR GRAPHICS 1 AND 2
117 OPEN #1,8,0,"S:"
118 GRAPHICS L
120 PRINT "GRAPHICS ";L
130 FOR J=0 TO 7:REM 8 LINES
140 FOR I=0 TO 31:REM 32 CHARS/LINE
150 K=32*J+I
155 REM DON'T DISPLAY "CLEAR SCREEN" OR
"RETURN"
160 IF K=ASC("}") OR K=155 THEN 180
165 IF L=0 THEN PUT #1,ASC(" "):REM ESCA
PE
170 PUT #1,K:REM DISPLAY CHARS
180 NEXT I
190 PRINT #1;" ":REM END OF LINE
200 IF L<>2 OR J<>3 THEN 240
210 REM SCREEN FULL
220 GOSUB WAIT
230 PRINT #1;"}":REM CLEAR SCREEN
240 NEXT J
250 GOSUB WAIT
265 PRINT "LOWER CASE AND GRAPHICS"
270 IF L<>0 THEN POKE CHBAS,LOWER:GOSUB
WAIT
275 CLOSE #1
280 NEXT L
300 GRAPHICS 0:END
1000 REM WAIT FOR "RETURN"
1010 PRINT "HIT RETURN TO CONTINUE";
1020 INPUT A$
1030 RETURN
```

LIGHT SHOW

This program demonstrates another aspect of Atari graphics. It uses graphics mode 7 for high resolution and the PLOT and DRAWTO statements to draw the lines. In line 20, the title will be more effective if it is entered in inverse video (use the Atari logo key).

```
10 FOR ST=1 TO 8:GRAPHICS 7
15 POKE 752,1
20 ? :? "     Atari's Special Light Show
":SETCOLOR 2,0,0
30 SETCOLOR 1,2*ST,8:COLOR 2
40 FOR DR=0 TO 80 STEP ST
50 PLOT 0,0:DRAWTO 100,DR
60 NEXT DR:FOR N=1 TO 800:NEXT N:NEXT ST

70 FOR N=1 TO 2000:NEXT N:GOTO 10
```

UNITED STATES FLAG

This program involves switching colors to set up the stripes. It uses graphics mode 7 plus 16 so that the display appears as a full-screen. Note the correspondence of the COLOR statements with the SETCOLOR statements. For fun and experimentation purposes, add a SOUND statement and use a READ/DATA combination to add "The Star Spangled Banner" after line 470. (Refer to Section 10.)

```
10 REM DRAW THE UNITED STATES FLAG
20 REM HIGH RESOLUTION 4-COLOR GRAPHICS,
 NO TEXT WINDOW
30 GRAPHICS 7+16
40 REM SETCOLOR 0 CORRESPONDS TO COLOR 1

50 SETCOLOR 0,4,4:RED=1
60 REM SETCOLOR 1 CORRESPONDS TO COLOR 2

70 SETCOLOR 1,0,14:WHITE=2
80 REM SETCOLOR 2 CORRESPONDS TO COLOR 3

90 BLUE=3:REM DEFAULTS TO BLUE
100 REM DRAW 13 RED & WHITE STRIPES
110 C=RED
120 FOR I=0 TO 12
130 COLOR C
140 REM EACH STRIPE HAS SEVERAL HORIZONT
AL LINES
150 FOR J=0 TO 6
160 PLOT 0,I%7+J
170 DRAWTO 159,I%7+J
180 NEXT J
190 REM SWITCH COLORS
200 C=C+1:IF C>WHITE THEN C=RED
210 NEXT I
300 REM DRAW BLUE RECTANGLE
310 COLOR BLUE
320 FOR I=0 TO 48
330 PLOT 0,I
340 DRAWTO 79,I
350 NEXT I
360 REM DRAW 9 ROWS OF WHITE STARS
370 COLOR WHITE
380 K=0:REM START WITH ROW OF 6 STARS
390 FOR I=0 TO 8
395 Y=4+I%5
400 FOR J=0 TO 4:REM 5 STARS IN A ROW
410 X=K+5+J%14:GOSUB 1000
420 NEXT J
430 IF K<>0 THEN K=0:GOTO 470
440 REM ADD 6TH STAR EVERY OTHER LINE
450 X=5+5%14:GOSUB 1000
460 K=7
470 NEXT I
500 REM IF KEY HIT THEN STOP
```

```
510 IF PEEK(764)=255 THEN 510
515 REM OPEN TEXT WINDOW WITHOUT CLEARIN
G SCREEN
520 GRAPHICS 7+32
525 REM CHANGE COLORS BACK
530 SETCOLOR 0,4,4:SETCOLOR 1,0,14
550 STOP
1000 REM DRAW 1 STAR CENTERED AT X,Y
1010 PLOT X-1,Y:DRAWTO X+1,Y
1020 PLOT X,Y-1:PLOT X,Y+1
1030 RETURN
```

SEAGULL OVER OCEAN

This program combines graphics and sounds. The sounds are not "pure" sounds, but simulate the roar of the ocean and the gull's "tweet". The graphics symbols used to simulate the gull could not be printed on the line printer. Enter the following characters in line 20.

20 BIRD$ = " V-- "

To get these symbols, use CTRL G, CTRL F, CTRL R, CTRL R.

```
10 DIM BIRD$(4)
20 BIRD$="    "
30 FLAG=1:ROW=10:COL=10
40 GRAPHICS 1:POKE 756,226:POKE 752,1
50 SETCOLOR 0,0,0:SETCOLOR 1,8,14
60 PRINT #6;"     the ocean"
70 R=INT(RND(0)*11)
80 POSITION 17,17
90 FOR T=0 TO 10
100 SOUND 0,T,8,4
110 FOR A=1 TO 50:NEXT A
120 IF RND(0)>0.8 THEN FOR D=10 TO 5 STE
P -1:SOUND 1,0,10,INT(RND(0)*10):NEXT D:
SOUND 1,0,0,0
130 GOSUB 200
140 NEXT T
150 FOR T=10 TO 0 STEP -1
160 SOUND 0,T,8,4
170 FOR A=1 TO 50:NEXT A
175 IF RND(0)>0.8 THEN FOR D=10 TO 5 STE
P -1:SOUND 1,D,10,8:NEXT D:SOUND 1,0,0,0

180 FOR H=1 TO 10:NEXT H
185 GOSUB 200
190 NEXT T
195 GOTO 70
200 GOSUB 300
210 POSITION COL,ROW
220 PRINT #6;BIRD$(FLAG,FLAG+1)
230 FLAG=FLAG+2:IF FLAG=5 THEN FLAG=1
240 RETURN
300 IF RND(0)>0.5 THEN RETURN
310 POSITION COL,ROW
320 PRINT #6;"  "
330 A=INT(RND(0)*3)-1
340 B=INT(RND(0)*3)-1
350 ROW=ROW+A
360 IF ROW=0 THEN ROW=1
370 IF ROW=20 THEN ROW=19
380 COL=COL+B
390 IF COL=0 THEN COL=1
400 IF COL>18 THEN COL=18
410 RETURN
```

VIDEO GRAFFITTI

This program requires a Joystick Controller for each player. Each joystick has one color associated with it. By maneuvering the joystick, different patterns are created on the screen. Note the use of the STICK and STRIG commands.

```
1 GRAPHICS 0
2 ? "VIDEO GRAFFITI"
5 REM X&Y ARRAYS HOLD COORDINATES
6 REM FOR UP TO 4 PLAYERS' POSITIONS.
7 REM COLR ARRAY HOLDS COLORS.
10 DIM A$(1),X(3),Y(3),COLR(3)
128 ? "USE JOYSTICKS TO DRAW PICTURES"
129 ? "PRESS BUTTONS TO CHANGE COLORS"
130 ? "INITIAL COLORS:"
131 ? "JOYSTICK 1 IS RED"
132 ? "JOYSTICK 2 IS WHITE"
133 ? "JOYSTICK 3 IS BLUE"
134 ? "JOYSTICK 4 IS BLACK (BACKGROUND)"

135 ? "BLACK LOCATION IS INDICATED BY A
BRIEFFLASH OF RED."
136 ? "IN GRAPHICS 8, JOYSTICKS 1 AND 3
ARE  WHITE AND 4 IS BLUE."
138 PRINT "HOW MANY PLAYERS (1-4)";
139 INPUT A$:IF LEN(A$)=0 THEN A$="1"
140 JOYMAX=VAL(A$)-1
145 IF JOYMAX<0 OR JOYMAX>=4 THEN 138
147 PRINT "GRAPHICS 3 (40X24), 5 (80X48)
,"
150 PRINT "7 (160X96), OR 8 (320X192)";
152 INPUT A$:IF LEN(A$)=0 THEN A$="3"
153 A=VAL(A$)
154 IF A=3 THEN XMAX=40:YMAX=24:GOTO 159

155 IF A=5 THEN XMAX=80:YMAX=48:GOTO 159

156 IF A=7 THEN XMAX=160:YMAX=96:GOTO 15
9
157 IF A=8 THEN XMAX=320:YMAX=192:GOTO 1
59
158 GOTO 147:REM A NOT VALID
159 GRAPHICS A+16
160 FOR I=0 TO JOYMAX:X(I)=XMAX/2+I:Y(I)
=YMAX/2+I:NEXT I:REM START NEAR CENTER O
F SCREEN
161 IF A<>8 THEN 166
162 FOR I=0 TO 2:COLR(I)=1:NEXT I
163 SETCOLOR 1,9,14:REM LT. BLUE
165 GOTO 180
166 FOR I=0 TO 2:COLR(I)=I+1:NEXT I
167 SETCOLOR 0,4,6:REM RED
168 SETCOLOR 1,0,14:REM WHITE
180 COLR(3)=0
295 FOR J=0 TO 3
```

```
300 FOR I=0 TO JOYMAX:REM CHECK JOYSTICK
S
305 REM CHECK TRIGGER
310 IF STRIG(I) THEN 321
311 IF A<>8 THEN 320
312 COLR(I)=COLR(I)+1:IF COLR(I)=2 THEN
COLR(I)=0:REM 2-COLOR MODE
313 GOTO 321
320 COLR(I)=COLR(I)+1:IF COLR(I)>=4 THEN
 COLR(I)=0:REM 4-COLOR MODE
321 IF J>0 THEN COLOR COLR(I):GOTO 325
322 IF COLR(I)=0 THEN COLOR 1:GOTO 325
323 COLOR 0:REM BLINK CURRENT SQUARE ON
AND OFF
325 PLOT X(I),Y(I)
330 JOYIN=STICK(I):REM READ JOYSTICK
340 IF JOYIN=15 THEN 530:REM NO MOVEMENT

342 COLOR COLR(I):REM MAKE SURE COLOR IS
 ON
344 PLOT X(I),Y(I)
350 IF JOYIN>=8 THEN 390
360 X(I)=X(I)+1:REM MOVE RIGHT
365 REM IF OUT OF RANGE THEN WRAPAROUND
370 IF X(I)>=XMAX THEN X(I)=0
380 GOTO 430
390 IF JOYIN>=12 THEN 430
400 X(I)=X(I)-1:REM MOVE LEFT
410 IF X(I)<0 THEN X(I)=XMAX-1
430 IF JOYIN<>5 AND JOYIN<>9 AND JOYIN<>
13 THEN 470
440 Y(I)=Y(I)+1:IF Y(I)>=YMAX THEN Y(I)=
0:REM MOVE DOWN
460 GOTO 500
470 IF JOYIN<>6 AND JOYIN<>10 AND JOYIN<
>14 THEN 500
480 Y(I)=Y(I)-1:IF Y(I)<0 THEN Y(I)=YMAX
-1:REM MOVE UP
500 PLOT X(I),Y(I)
530 NEXT I
535 NEXT J
540 GOTO 295
```

KEYBOARD CONTROLLER

This program alters registers on a chip called a PIA. To set these back to the default values in order to do further I/O, hit SYSTEM RESET or POKE PACTL,60. If this program is to be loaded from disk, use LOAD, not RUN and wait for the busy light on the disk drive to go out. Do not execute the program *before* this light goes out, otherwise the disk will continue to spin.

```
1 GRAPHICS 0
5 PRINT :PRINT "    KEYBOARD CONTROLLER
DEMO"
10 DIM ROW(3),I$(13),BUTTON$(1)
30 GOSUB 6000
40 FOR CNT=1 TO 4
60 POSITION 2,CNT*2+5:PRINT "CONTROLLER
# ";CNT;":";
70 NEXT CNT
80 FOR CNT=1 TO 4:GOSUB 7000:POSITION 19
,CNT+CNT+5:PRINT BUTTON$;:NEXT CNT
120 GOTO 80
6000 REM ** SET UP FOR CONTROLLERS **
6010 PORTA=54016:PORTB=54017:PACTL=54018
:PBCTL=54019
6020 POKE PACTL,48:POKE PORTA,255:POKE P
ACTL,52:POKE PORTA,221
6025 POKE PBCTL,48:POKE PORTB,255:POKE P
BCTL,52:POKE PORTB,221
6030 ROW(0)=238:ROW(1)=221:ROW(2)=187:RO
W(3)=119
6040 I$=" 123456789*0#"
6050 RETURN
7000 REM ** RETURN BUTTON$ WITH CHARACTE
R FOR BUTTON WHICH HAS BEEN PRESSED ON C
ONTROLLER CNT (1-4). **
7001 REM ** NOTE: A 1 WILL BE RETURNED I
F NO CONTROLLER IS CONNECTED. **
7002 REM ** A SPACE WILL BE RETURNED IF
THE CONTROLLER IS CONNECTED BUT NO KEY H
AS BEEN PRESSED. **
7003 PORT=PORTA:IF CNT>2 THEN PORT=PORTB

7005 P=1
7008 PAD=CNT+CNT-2
7010 FOR J=0 TO 3
7020 POKE PORT,ROW(J)
7030 FOR I=1 TO 10:NEXT I
7050 IF PADDLE(PAD+1)>10 THEN P=J+J+J+2:
GOTO 7090
7060 IF PADDLE(PAD)>10 THEN P=J+J+J+3:GO
TO 7090
7070 IF STRIG(CNT-1)=0 THEN P=J+J+J+4:GO
TO 7090
7080 NEXT J
7090 BUTTON$=I$(P,P)
7095 RETURN
```

TYPE-A-TUNE

This program assigns musical note values to the keys on the top row of the keyboard. Press only one key at a time.

KEY	MUSICAL VALUE
INSERT	B
CLEAR	B♭ (or A#)
0	A
9	A♭ (or G#)
8	G
7	F# (or G♭)
6	F
5	E
4	B♭ (or D#)
3	D
2	D♭ (or C#)
1	C

```
10 DIM CHORD(37),TUNE(12)
20 GRAPHICS 0:? :? "        TYPE-A-TUNE
PROGRAM"
25 ? :? "PRESS KEYS 1-9,0,(,) TO PRODUCE
 NOTES.";
27 ? "RELEASE ONE KEY BEFORE PRESSING TH
E   NEXT."
28 ? "OTHERWISE THERE MAY BE A DELAY."
30 FOR X=1 TO 37:READ A:CHORD(X)=A:NEXT
X
40 FOR X=1 TO 12:READ A:TUNE(X)=A:NEXT X

50 OPEN #1,4,0,"K:"
55 OLDCHR=-1
60 A=PEEK(764):IF A=255 THEN 60
63 IF A=OLDCHR THEN 100
65 OLDCHR=A
70 FOR X=1 TO 12:IF TUNE(X)=A THEN SOUND
 0,CHORD(X),10,8:GOTO 100
80 NEXT X
100 I=INT(PEEK(53775)/4):IF (I/2)=INT(I/
2) THEN 60
110 POKE 764,255:SOUND 0,0,0,0:OLDCHR=-1
:GOTO 60
200 DATA 243,230,217,204,193,182,173,162
,153,144,136,128,121,114,108,102,96,91,8
5,81,76,72,68,64,60
210 DATA 57,53,50,47,45,42,40,37,35,33,3
1,29
220 DATA 31,30,26,24,29,27,51,53,48,50,5
4,55
```

To play "Mary Had A Little Lamb" press the following keys:

5, 3, 1, 3, 5, 5, 5 3, 3, 3 5, 8, 8 5, 3, 1, 3, 5, 5 5, 5, 3, 3, 5, 3, 1

COMPUTER BLUES

This program generates random musical notes to "write" some very interesting melodies for the programmed bass.

```
1 GRAPHICS 0:? :? "          COMPUTER BLUE
S":?
2 PTR=1
3 THNOT=1
5 CHORD=1
6 PRINT "BASS TEMPO (1=FAST)";
7 INPUT TEMPO
8 GRAPHICS 2+16:GOSUB 2000
10 DIM BASE(3,4)
20 DIM LOW(3)
25 DIM LINE(16)
26 DIM JAM(3,7)
30 FOR X=1 TO 3
40 FOR Y=1 TO 4
50 READ A:BASE(X,Y)=A
60 NEXT Y
70 NEXT X
80 FOR X=1 TO 3:READ A:LOW(X)=A
90 NEXT X
95 FOR X=1 TO 16:READ A:LINE(X)=A:NEXT X

96 FOR X=1 TO 3
97 FOR Y=1 TO 7
98 READ A:JAM(X,Y)=A:NEXT Y:NEXT X
100 GOSUB 500
110 T=T+1
115 GOSUB 200
120 GOTO 100
200 REM PROCESS HIGH STUFF
205 IF RND(0)<0.25 THEN RETURN
210 IF RND(0)<0.5 THEN 250
220 NT=NT+1
230 IF NT>7 THEN NT=7
240 GOTO 260
250 NT=NT-1
255 IF NT<1 THEN NT=1
260 SOUND 2,JAM(CHORD,NT),10,NT%2
280 RETURN
500 REM PROCESS BASE STUFF
510 IF BASS=1 THEN 700
520 BOUR=BOUR+1
530 IF BOUR<>TEMPO THEN 535
531 BASS=1:BOUR=0
535 SOUND 0,LOW(CHORD),10,4
540 SOUND 1,BASE(CHORD,THNOT),10,4
550 RETURN
700 SOUND 0,0,0,0
710 SOUND 1,0,0,0
720 BOUR=BOUR+1
```

```
730 IF BOUR<>1 THEN 800
740 BOUR=0:BASS=0
750 THNOT=THNOT+1
760 IF THNOT<>5 THEN 800
765 THNOT=1
770 PTR=PTR+1
780 IF PTR=17 THEN PTR=1
790 CHORD=LINE(PTR)
800 RETURN
1000 DATA 162,144,136,144,121,108,102,10
8,108,96,91,96
1010 DATA 243,182,162
1020 DATA 1,1,1,1,2,2,2,2,1,1,1,1,3,2,1,
1
1030 DATA 60,50,47,42,40,33,29
1040 DATA 60,50,45,42,40,33,29
1050 DATA 81,68,64,57,53,45,40
2000 PRINT #6:PRINT #6:PRINT #6
2005 PRINT #6;"      Computer"
2006 PRINT #6
2010 PRINT #6;"        Blues"
2030 RETURN
```

DECIMAL/ HEXADECIMAL CONVERSION PROGRAM

This program can be typed in and used to convert hexadecimal numbers to decimal numbers and vice versa.

```
10 DIM A$(9),AD$(1)
20 GRAPHICS 0:? :? "     HEX NUMBER CONV
ERSIONS":?
30 ? :? "Enter 'D' for DEC to HEX conver
sion.":? "Enter 'H' for HEX to DEC conve
rsion.":INPUT A$
40 IF LEN(A$)=0 THEN 30
50 IF A$="H" THEN 300
60 IF A$<>"D" THEN 30
90 TRAP 90
100 ? :? "ENTER A DECIMAL NUMBER FROM 0
THROUGH 9999999999."
110 ? "DEC:";:INPUT N
120 IF N<0 OR N>=1E+10 THEN GOTO 100
130 I=9
140 TEMP=N:N=INT(N/16)
150 TEMP=TEMP-N*16
160 IF TEMP<10 THEN A$(I,I)=STR$(TEMP):G
OTO 180
170 A$(I,I)=CHR$(TEMP-10+ASC("A"))
180 IF N<>0 THEN I=I-1:GOTO 140
190 ? "HEX: ";A$(I,9):?
200 GOTO 110
300 TRAP 300
310 ? :? "ENTER A HEX NUMBER FROM 0 THRO
UGH     FFFFFFFF."
320 ? "HEX:";:INPUT A$
330 N=0
340 FOR I=1 TO LEN(A$)
345 AD$=A$(I,I):IF AD$<"0" THEN 300
350 IF A$(I,I)<="9" THEN N=N*16+VAL(AD$)
:GOTO 370
355 IF AD$<"A" THEN 300
357 IF AD$>"F" THEN 300
360 N=N*16+ASC(AD$)-ASC("A")+10
370 NEXT I
380 ? "DEC: ";N:?
390 GOTO 320
400 END
```

APPENDIX I

MEMORY LOCATIONS

Note: Many of these locations are of primary interest to expert programmers and are included here as a convenience. The labels given are used by Atari programmers to make programs more readable.

LABEL	DECIMAL LOCATION	HEXADECIMAL LOCATION	COMMENTS AND DESCRIPTION
APPMHI	14,15	DE	Highest location used by BASIC (LSB, MSB)
RTCLOK	18,19,20	12,13,14	TV frame counter (1/60 sec.) (LSB, NSB, MSB)
SOUNDR	65	41	Noisy I/O Flag (0 = quiet)
	77		Attract Mode Flag (128 = Attract mode)
LMARGIN, RMARGIN	82,83	52,53	Left, Right Margin (Defaults 2, 39)
ROWCRS	84	54	Current cursor row (graphics window).
COLCRS	85,86	55,56	Current cursor column (graphics window).
OLDROW	90	5A	Previous cursor row (graphics window).
OLDCOL	91,92	5B	Previous cursor column (graphics window).
	93	5C	Data under cursor (graphics window unless mode 0).
NEWROW	96	60	Cursor row to which DRAWTO will go.
NEWCOL	97,98	61,62	Cursor column to which DRAWTO goes.
RAMTOP	106	6A	Actual top of memory (number of pages).
LOMEM	128,129	80,81	BASIC low memory pointer.
MEMTOP	144,145	90,91	BASIC top of memory pointer.
STOPLN	186,187	BA,BB	Line number at which STOP or TRAP occurred (2-byte binary number).
ERRSAV	195	C3	Error number.
PTABW	201	C9	Print tab width (defaults to 10)
FR0	212,213	D4,D5	Low and high bytes of value to be returned to BASIC from USR function.

LABEL	DECIMAL LOCATION	HEXADECIMAL LOCATION	COMMENTS AND DESCRIPTIONS
RADFLG	251	FB	RAD/DEG flag (0 = radians, 6 = degrees).
LPENH	564	234	Light Pen* Horizontal value.
LPENV	565	235	Light Pen* Vertical value.
TXTROW	656	290	Cursor row (text window)
TXTCOL	657,658	291,292	Cursor column (text window)
COLOR0	708	2C4	Color Register 0
COLOR1	709	2C5	Color Register 1
COLOR2	710	2C6	Color Register 2
COLOR3	711	2C7	Color Register 3
COLOR4	712	2C8	Color Register 4
MEMTOP	741,742	2E5,2E6	OS top of available user memory pointer (LSB, MSB)
MEMLO	743,744	2E7,2E8	OS low memory pointer
CRSINH	752	2F0	Cursor inhibit (0 = cursor on, 1 = cursor off)
CHACT	755	2F3	Character mode register (4 = vertical reflect; 2 = normal; 1 = blank)
CHBAS	756	2F4	Character base register (defaults to 224) (224 = upper case, 226 = lower case characters)
ATACHR	763	2FB	Last ATASCII character.
CH	764	2FC	Last keyboard key pressed; internal code; (255 clears character).
FILDAT	765	2FD	Fill data for graphics Fill (XIO).
DSPFLG	766	2FE	Display Flag (1 = display control character).
SSFLAG	767	2FF	Start/Stop flag for paging (0 = normal listing) Set by CTRL 1.
HATABS	794	31A	Handler address table (3 bytes/handler)
IOCB	832	340	I/O control blocks (16 bytes/IOCB)
	1664-1791	680-6FE	Spare RAM
CONSOL	53279	D01F	Console switches (bit 2 = Option; bit 1 = Select; bit 0 = Start. POKE 53279, 0 before reading. 0 = switch pressed.)

* Future product.

LABEL	DECIMAL LOCATION	HEXADECIMAL LOCATION	COMMENTS AND DESCRIPTIONS
PORTA	54016	D300	PIA Port A Controller Jack I/O ports.
PORTB	54017	D301	PIA Port B Initialized to hex 3C.
PACTL	54018	D302	Port A Control Register (on Program Recorder 52 = ON, 60 = OFF).
PBCTL	54019	D303	Port B control register.
SKCTL	53775	D20F	Serial Port control register. Bit 2=0 (last key still pressed).

NOTES

INDEX

TABLE OF MODES AND SCREEN FORMATS

Gr. Mode	Mode Type	Horiz. (Columns)	Vert. (Rows) Split Screen	Vert. (Rows) Full Screen	#Of Colors	RAM Required (Bytes)
0	TEXT	40	-	24	2	993
1	TEXT	20	20	24	5	513
2	TEXT	20	10	12	5	261
3	GRAPHICS	40	20	24	4	273
4	GRAPHICS	80	40	48	2	537
5	GRAPHICS	80	40	48	4	1017
6	GRAPHICS	160	80	96	2	2025
7	GRAPHICS	160	80	96	4	3945
8	GRAPHICS	320	160	192	1/2	7900

MODE, SET COLOR, COLOR TABLE

Default Colors	Mode or Condition	SETCOLOR (aexp1) Color Register No.	Color (aexp)	DESCRIPTION AND COMMENTS
LIGHT BLUE DARK BLUE BLACK	MODE 0 and ALL TEXT WINDOWS	0 1 2 3 4	COLOR data actually determines character to be plotted	— Character luminance (same color as background) Background — Border
ORANGE LIGHT GREEN DARK BLUE RED BLACK	MODES 1 and 2 (Text Modes)	0 1 2 3 4	COLOR data actually determines character to be plotted	Character Character Character Character Background, Border
ORANGE LIGHT GREEN DARK BLUE BLACK	MODES 3, 5, and 7 (Four-color Modes)	0 1 2 3 4	1 2 3 - 0	Graphics point Graphics point Graphics point — Graphics point (background default), Border
ORANGE BLACK	MODES 4 and 6 (Two-color Modes)	0 1 2 3 0	1 - - - 	Graphics point — — — Graphics point (background default), Border
LIGHT GREEN DARK BLUE BLACK	MODE 8 (1 Color 2 Luminances)	0 1 2 3 4	- 1 0 - -	— Graphics point luminance (same color as background) Graphics point (background default) — Border

CONTROL GRAPHICS KEYBOARD

C015307 REV. 1