Monarch Data Systems

P.O. Box 207, Cochituate, Massachusetts 01778

Monarch Data Systems

ABC

A BASIC Compiler for Atari™ Computers

Reference Manual

ABC

A BASIC Compiler
for Atari Computer Systems

Version 1.0

Monarch Data Systems
P.O. Box 207, Cochituate, Massachusetts 01778

Contents

Section 1
Introduction

ABC is a software development tool designed to improve the performance of your Atari BASIC programs. It lets you enjoy the high speed and memory efficiency of compiled languages like FORTH and C, without leaving the familiar environment of your BASIC cartridge.

1.1 How Does It Work?

ABC stands for "A Basic Compiler." A compiler is a program that accepts source code (your BASIC program) and translates it into another form, in this case a compact pseudo-code or P-code. Once compiled, this P-code can be executed by another program called a run-time interpreter. Both a compiler and an interpreter are included on your ABC disk.

To compile a BASIC program with ABC, you must first SAVE the BASIC program on a disk. The ABC compiler reads your BASIC program off the disk and translates it into P-code, one line at a time. Then it permanently links a copy of the ABC interpreter to the P-code, and saves the compiled program as a binary disk file which can be loaded and executed like a machine-language program. The BASIC source file is unaffected.

The benefits of using ABC include:

Faster execution speed. ABC-compiled programs run from four to twelve times faster than the original BASIC version, depending on the source code. This makes it possible to use Atari BASIC for professional game development and other speed-critical applications.

Greater memory efficiency. The P-code produced by ABC is considerably more compact than tokenized BASIC. Numbers are stored in three bytes instead of the six required by the Atari floating-point routines. Additionally, the ABC interpreter requires only 4K of memory, about half that used by the BASIC cartridge. The result is a compiled program that requires much less memory overhead than the original BASIC version.

Non-cartridge environment. Compiled programs can be run without the BASIC cartridge! This allows access to the upper 8K of memory in a 48K system, which is normally de-selected by the cartridge.

Source code protection. ABC P-code is a compressed and encoded version of the original source program which is very difficult to understand without detailed knowledge of the ABC run-time interpreter. For this reason, a BASIC program processed by ABC cannot be listed or disassembled and is extremely hard to "break."

1.2 System Requirements

To use the ABC software, your Atari computer system must include a minimum of 40K memory and at least one disk drive. You must also have the Atari Disk Operating System, DOS 2.0S, to create your working copy of the ABC disk.

The memory required to run a compiled program depends on the size of the original BASIC source code, and may be little as 16K bytes.

1.3 Distributing Your Compiled Software

No royalties or licensing fees are required to distribute software processed by ABC. However, we do require that your software bears the following notice:

"Produced using copyrighted software products of Monarch Data Systems, Cochituate, MA 01778."

Display the notice prominently on either the program title screen or in the documentation provided with the product. Failure to reproduce this notice may constitute a copyright infringement.

1.4 Contents Of This Package

Your ABC package should contain:

-- A disk containing the ABC compiler, run-time interpreter and a relocation utility;
-- A label for your working copy of the ABC disk;
-- This reference manual, and;
-- A user registration form.

Please take the time to fill out and return the registration form. This will enable us to supply you with revisions and enhancements to the ABC system, and to keep you informed of new products as they become available.

1.5 Purpose Of This Manual

The ABC Reference Manual is intended to show you how to operate your ABC software. You should already be familiar with Atari BASIC programming and with Atari DOS 2.0S.

1.6 References

The following reference documents are published by Atari.

Atari BASIC Reference Manual (CO14722)
Atari Disk Operating System II Reference Manual (CO16347)
Atari Technical Reference Notes (CO16555)

Section 2
Disk Backup and Replacements

The ABC BASIC compiler disk is shipped on a copy protected disk to prevent unauthorized duplication. Although we dislike copy protected products ourselves, we are aware that there are some illegal copies of an early (buggy) version of our compiler.

To help lessen the blow to our legitimate users, we allow the purchase of a single backup disk for a reduced rate when ordered along with a completed user registration form. The price for that backup copy is printed on the form.

We also provide replacement diskettes for worn out copies when you ship back your original diskette along with a check or money order for $6.95.

We will notify our registered users of bugs that have been found and improvements in future releases.

Under no circumstances will we offer support to users that have not returned a completed registration form.

Section 3
Compiling A BASIC Program

A compiled program can only be as good as the original BASIC source. If your BASIC code has bugs in it, ABC will faithfully translate the bugs into P-code, resulting in incorrect operation at best and a total system crash at worst. Then you'll have to return to your BASIC source, track down the bugs and re-compile. Always make sure your BASIC program is working properly before you try to compile it.

Once you're satisfied with your source code, SAVE it onto a disk using a BASIC command in the form:

```
SAVE "D1:PROGRAM.BAS"
```

Naturally, you can specify a different drive number if you want, with any legal file name or extender. The ".BAS" extender is useful because it helps you tell the BASIC and compiled versions of your programs apart.

The ABC compiler is supplied as an AUTORUN.SYS file that will execute whenever your working copy is booted on Drive #1. Use the following procedure to compile a SAVEd Atari BASIC program:

a. Remove all cartridges from your computer. Turn on Drive #1 and wait for the "BUSY" light to go out.
b. Insert your ABC working disk into Drive #1 and turn on the computer.
c. The ABC title screen and copyright notice will appear on your TV set or monitor. Approximately one second later, the system will begin reading the ABC run-time interpreter into memory.
d. When the interpreter is loaded, the program will ask you to remove the ABC disk and insert the disk that contains your SAVED Atari BASIC program.

e. ABC will next request the name of your BASIC source program (the one you are compiling). Respond with a drive specifier (D1:, D2:, etc.) and the full file name, including any extensions. A default drive specifier of "D1:" and a ".BAS" extension will be provided if they are not supplied by you. If your filename has no extender, include a trailing period ("PROGRAM.") to prevent ABC from trying to add the ".BAS" extender.
f. You will now be asked to specify the name of the destination file. This file will eventually become the compiled version of your BASIC program. Again respond with a drive specifer and a full file name. The defaults are "D1:" with a ".CMP" extender.
g. ABC will immediately open a new disk file with the name you selected. It will then write a copy of the run-time interpreter into the new file. A temporary "scratch pad" file is also created on Drive #1 for ABC's own use. (NOTE: In single-drive systems, the destination file is always written on the same disk as the source file. Make sure the disk is not write-protected or you will get an error message.)
h. The compiler now begins to scan your original BASIC program. First, it displays the number of variables (symbols) used in your program, followed by the total number of program lines. Using this information, the compiler proceeds to convert each BASIC line into P-code. The progress of the compiler is indicated by displaying every 25th line number of the BASIC program, with intermediate lines represented by a single dot.
i. After a successful compilation, ABC will display a "Compilation Completed" message. You will then be offered a choice via the console switches of whether to re-run the compiler (START), reboot the system (OPTION) or return to DOS (SELECT). To run the program you just compiled, return to DOS by pressing the SELECT key and use DOS option "L" to load the destination file, which will begin executing automatically.

Section 4
BASIC Programming Considerations

To achieve the high speed and efficiency of the ABC system, it was necessary to place a few limitations on the Atari BASIC code that can be compiled. Most programmers will find that these "limitations" aren't very restricting at all -- in fact, they may actually help to improve your programs by making you explore alternative methods of problem-solving.

4.1 Integer Arithmetic

Each constant and variable in an Atari BASIC program is stored in floating-point format, using six bytes of binary-coded decimal. Whenever you RUN a BASIC program, these numbers must be translated back and forth from floating-point to "straight" binary so that they can be used by the Atari operating system ROM. This constant translating and the general "laziness" of floating-point operations are the main reasons for the notoriously slow speed of Atari BASIC.

ABC avoids the speed limitations of floating-point by using only integer (whole number) arithmetic. Values are stored as three bytes of "straight" binary, with a usable range of approximately -8 million to +8 million.

Most Atari programs do not need floating point arithmetic. Games, graphics and systems software rarely employ fractions or complex mathematical functions. As a result, you may find that many of your favorite BASIC programs can be compiled with little or no alteration. And because of ABC's wide usable number range, it's possible to simulate almost any floating-point function using simple integer operators.

4.2 Unsupported Functions

Because ABC does not employ floating-point math, it will not accept a BASIC program that contains any of the following functions:

ATN	CLOG	COS	EXP	LOG
RND	SIN	SQR		

4.3 Simulating Floating-Point Numbers

You can partially compensate for ABC's lack of floating-point by scaling all of your intermediate results. For example, if you multiply a number by 100 before performing a division, you will obtain two significant digits after the "imaginary" decimal point in your answer.

Suppose you need to divide 7 by 2, with an accuracy of two significant digits. In regular Atari BASIC, this would be coded as:

```
ANSWER = 7/2  (evaluates to 3.50)
```

In a BASIC program intended for compilation, you could use:

```
ANSWER = INT((7*100)/2)
```

which evaluates to 350 in both Atari and ABC-compiled BASIC.

This method is not intended as a substitute for the convenience of automatic floating-point. But it should satisfy the limited need for fractions in the majority of games and systems programs.

4.4 Simulating The RND() Function

At first it may not seem obvious why RND() is included on the list of unsupported functions. In Atari BASIC, RND() returns a value that is less than one and greater than or equal to zero. This value cannot be represented by a whole number, and therefore requires floating-point. So if you need a random number in your ABC program, you'll have to find a way to obtain it without using RND().

Fortunately, the hardware provides a simple way to simulate the RND() function. The Atari operating system is constantly storing a new random integer between 0 and 255 into memory location 53770. Almost any random value can be obtained by PEEKing this location and scaling the result appropriately.

To illustrate the technique, let's assign the memory address 53770 to the variable RANDOM:

```
RANDOM = 53770
```

Suppose your latest computer game needs a random value from 0 to 9, inclusive. You could obtain it with the following expression:

```
VALUE = INT(PEEK(RANDOM)*10/256)
```

To obtain a value from 0 to 99 you could use:

```
VALUE = INT(PEEK(RANDOM)*100/256)
```

In the event that you want a random value greater than 255, you will have to break up the number into groups of one or two decimal digits. If, for instance, you need a value between 0 and 999, you could get the "hundreds" digit with:

```
HUNDS = INT(PEEK(RANDOM)*10/256)
```

Now get the tens and ones digits together:

```
OTHERS = INT(PEEK(RANDOM)*100/256)
```

Combine the results:

```
VALUE = HUNDS*100+OTHERS
```

and the variable VALUE will contain a random number between 0 and 999.

4.5 Simulating Trigonometric Functions

The simplest way to simulate an Atari BASIC trig function is to prepare a look-up table. You can either enter the table values in a DATA statement, use integer approximations to calculate the values at run-time, or use Atari BASIC to compute the values once and fill an array with the results.

The essential trick is to convert each table element to a whole number by scaling it by an appropriate factor. If you need accuracy to two significant digits, you would multiply by 100; for three-digit accuracy, 1000, etc. Using the SIN() function as an example:

```
VALUE = INT(1000*SIN(X))
```

Then SIN(0) becomes 0, SIN(45) becomes 707 (normally 0.707) and SIN(90) becomes 1000 (normally 1).

Now, suppose you have prepared tables of scaled SIN and COS values in arrays S() and C(), respectively, and you want to draw a circle of radius R at center point X and Y. The following instructions will accomplish this:

```
100 FOR I = 0 TO 359
110 PLOT X+R*S(I)/1000,Y+R*C(I)/1000
120 NEXT I
```

To generate a trig table at run-time you can make use of the trigonometric identities:

```
sin(a+b) = sin(a)cos(b)+sin(b)cos(a)
cos(a+b) = cos(a)cos(b)-sin(a)sin(b)
```

By selecting the angle b as a constant and looking up its sine and cosine, you can iterate through all the angles by the increment of b and fill in an array with appropriately scaled values.

4.6 Limited Size Of Constants

Although the range of variables that can be handled ABC exceeds 16 million, it cannot compile a BASIC program that contains a constant larger than 65,535.

The blame again lies in the operating system. The Atari ROM routines that convert binary-coded decimal to "straight" binary only support numbers in the range from 0 to 65,535.

It's very easy to get around this limitation. As an example, suppose your program uses a variable BIGNUM with a value of 250,000. In regular BASIC, you would assign this value with the expression:

```
BIGNUM = 250000
```

ABC would disapprove of all those zeroes. But the expression:

```
BIGNUM = 250*1000
```

yields exactly the same result without making ABC unhappy.

Don't forget that the numbers in a DATA statement are not regarded as constants. So you can also use the expressions:

```
100 READ BIGNUM
110 DATA 250000
```

and still satisfy both BASIC and ABC.

4.7 Order Of Operations

ABC handles division operations differently than Atari BASIC. Consider the following example (constants are used as a convenience):

```
X = INT(5/3*2)
```

The BASIC cartridge would first divide 5 by 3 (yielding a result of 1.66), multiply by 2 (with a result of 3.32) and then apply the INT function to obtain a final value of 3. But because the ABC interpreter deals only with whole numbers, it treats all division operations as an implicit INT(x/y) function. This means that ABC would interpret the above expression as:

```
X = INT(INT(5/3)*2)
```

which evaluates to 2 instead of 3!

To make the above example work in both standard and compiled BASIC, all that is needed is a simple inversion of terms:

```
X = INT(5*2/3)
```

This technique yields the desired result (3) in either case.

Division is the only ABC operation that does not conform to Atari BASIC. Multiplication, addition and subtraction are performed in the normal manner.

4.8 Unsupported Arithmetic Operators

Only one arithmetic operator is not supported by the ABC compiler: the exponentiation operator "^." This operation is easily simulated (with greater speed) by using sequential multiplications.

4.9 Unsupported BASIC Statements

Once an Atari BASIC program has been translated into P-Code, it cannot be accessed by the BASIC cartridge. For this reason, compiled programs must not try to use the loading, saving and editing functions supported by the cartridge. In addition, because ABC does not employ floating-point math, the DEG and RAD statements have no meaning to the interpreter.

ABC will not compile a BASIC program that contains any of the following statements:

BYE	CLOAD	CONT	CSAVE	DEG
DOS	ENTER	LIST	LOAD	LPRINT
NEW	RAD	RUN	SAVE	

4.10 Break Key Handling

When you hit the BREAK key during the execution of a normal Atari BASIC program, the program STOPs at the current line number and returns to the cartridge for the READY prompt.

A compiled program has no cartridge to return to, so hitting the BREAK key does not stop the program unless the key was struck during an I/O operation. This forces an Error #128 (Break Key Abort) which, unless TRAPped, causes the program to terminate.

You can avoid problems with the BREAK key by disabling it with appropriate POKEs. Refer to the Atari Technical Reference Notes for more information on controlling the BREAK key.

4.11 Subroutines And FOR/NEXT Loops

When using ABC, it's important to keep track of how you exit subroutines and FOR/NEXT loops. In the following example:

```
100 FOR I = 1 TO 100
110 IF I = 50 THEN GOTO 130
120 NEXT I
130 PRINT "Loop aborted."
```

the lack of a POP statement would probably confuse ABC when the loop index reached 50. The correct method is:

```
110 IF I = 30 THEN POP : GOTO 130
```

This is good programming practice even in a non-ABC environment.

The ABC interpreter is designed to handle no more than 64 outstanding GOSUBs and/or FOR/NEXT loops simultaneously. If you manage to write a BASIC program that requires greater stack depth than this, congratulations!

4.12 Arrays And Strings

ABC does not use the same memory allocation method for arrays and strings as Atari BASIC. Consequently, programs that take advantage of BASIC's array and string allocation structure will not operate correctly when compiled. The ADR() function will, however, always return correct values.

Consult Section 6.3 of this manual for more information on ABC's memory allocation scheme for arrays and strings.

4.13 Timing Loops

BASIC programmers often use "do-nothing" FOR/NEXT loops to obtain time delays. These usually appear in the form:

```
100 FOR DELAY = 1 TO 100
110 NEXT DELAY
```

You will be in for a shock if you compile and run the above instructions. ABC will execute the loop so rapidly that the delay will seem to disappear!

The best way to write a controllable time delay for ABC is to use one of the Atari's built-in hardware timers. The operating system changes the value of memory location 20 every 1/60th of a second. By PEEKing this location in a FOR/NEXT loop, you can obtain precise time delays that will work correctly in both the BASIC and compiled versions of your software.

The following time-delay subroutine can be appended to any BASIC program:

```
1000 REM * ABC TIME DELAY SUBROUTINE
1010 REM * Set the value of variable JIFFIES equal to
1020 REM * the desired time delay in 60ths of a second.
1030 REM * Then perform a GOSUB 1000 to obtain delay.
1040 FOR DELAY = 1 TO JIFFIES
1050 TICK = PEEK(20)
1060 IF TICK = PEEK(20) THEN 1060
1070 NEXT DELAY
1080 RETURN
```

To get a 5-second time delay with this method, you could write:

```
100 REM * This is the body of your program.
110 JIFFIES = 60*5 : GOSUB 1000
120 REM * You just waited 5 seconds.
```

Section 5
Advanced Usage

The following information is included for advanced programmers who may want to alter the default properties of the ABC compiler. Software authors who wish to distribute their compiled programs should also read this section.

5.1 Changing The Load Address

The ABC compiler normally produces code that is loaded at memory address $2600 (hex notation). This default address is derived from the run-time interpreter that is automatically loaded by the compiler (see Section 3). You can obtain an alternative load address by choosing a different run-time interpreter when the ABC compiler is run.

Immediately after the ABC copyright message is displayed, the compiler scans the console switches for one second. If you press the OPTION key during this period, ABC will not proceed to load the $2600 interpreter. Instead, it will ask you for the name of one of the other interpreters included on your ABC disk. Respond with "INTERP.Xnn" where nn is the high byte of the load address in hex. For example, if you wanted a load address of $1F00, answer the prompt with "INTERP.X1F."

To find out which run-time interpreters are available on your ABC disk, enter DOS and use menu option "A" (directory) to examine the list of "INTERP" files. Contact Monarch Data Systems if you need an interpreter with a specific load address.

5.2 Generating Relocatable Code

When producing software for commercial distribution, it's a good idea to make the code relocatable to assure compatibility with different memory configurations. Your ABC disk includes a special utility called "MKRELO" that can be used to produce a compiled, fully relocatable version of your Atari BASIC programs.

The code-generating technique used by MKRELO is unusual. It requires that you compile your BASIC source program twice, using different load addresses. MKRELO then compares the two disk files and produces a third version of the program which can be loaded at any address.

The following procedure illustrates the proper use of MKRELO. It assumes that you have SAVEd a BASIC program called "GAME.BAS" on a source disk which also contains DOS 2.0S. Make sure there is plenty of free space on the source disk.

a. Boot your ABC working disk along with the default ($2600) interpreter as described in Section 3.
b. Respond to the prompt for the BASIC source filename ("GAME.BAS" or just "GAME" in this example).
c. Respond to the prompt for a destination filename, say "GAME.X26."
d. When the first compilation is completed, replace the ABC working disk into Drive #1. Press the START key and immediately press and hold the OPTION key until you receive the prompt for an interpreter filename. Respond with "INTERP.X1F."
e. After the compiler reads the $1F00 interpreter into RAM, replace the BASIC source disk and provide the source filename again ("GAME.BAS"). Then give ABC a destination filename that is different from the one used for the first compilation ("GAME.X1F," for instance).
f. When the second compilation ends, return to DOS by pressing the SELECT key. Re-insert the ABC disk and use DOS option "L" to load and automatically run the MKRELO program. Replace the program disk when the MKRELO title appears.

g. MKRELO will ask for the names of the two files created by the previous compilations. Respond with "GAME.X26" for the first prompt and "GAME.X1F" for the second prompt.
h. You will now be asked for the filename of the final, relocatable program. Respond with a suitable title (e.g.; "GAME.REL") and press RETURN.
i. MKRELO takes a while to finish because it compares the files one byte at a time. Once the process is completed, re-enter DOS by pressing the SELECT key. To load and run your relocatable program, use DOS option "L" and respond with the name of the file created by MKRELO.

Section 6
Technical Notes

This section provides various technical details about the ABC compiler and the P-code it produces.

6.1 Error Checking

Most program conditions that require monitoring are checked during run-time. However, one specific condition that is not checked is subscript values. Any negative or out-of-bounds subscript will cause the ABC interpreter to behave in an unpredictable manner. We decided not to check subscripts because it saves execution time, and it was assumed that your source programs would be debugged before compilation.

6.2 Low Memory Usage

The ABC run-time interpreter uses all page zero locations from $80 and $C2 hex, inclusive. The standard BASIC line number and error number locations are supported. However, other BASIC zero-page variables (such as the high address pointer and symbol table pointer) are not supported. Page six ($600-$6FF) is fully available for USR routines and other purposes.

6.3 Memory Allocation

Compiled programs initially set the OS variable APPMHI ($0E-0F) to the end of the loaded program module. During the course of program execution, the value of APPMHI is automatically adjusted upward for the following reasons:

Input Statement Buffer.
The first INPUT statement causes allocation of a 255-byte buffer.

GOSUB and FOR Stack.
The first GOSUB or FOR statement causes allocation of a 128-byte stack.

DIM Statements.
Each DIMensioned numeric array requires nine bytes of control information plus three additional bytes per array element. DIMensioned string variables require nine bytes of control information plus one byte for each string character.

Applications may allocate memory by adjusting APPMHI upward, but to be compatible with the BASIC cartridge you should work from MEMTOP ($2E5-2E6) downwards. It's also a good idea to execute all DIM statements, a loop or GOSUB and an INPUT statement before allocating memory to make sure there's enough room for ABC to work comfortably.

Section 7
Error Handling

7.1 Compilation Errors

Most of the error messages that can result from a compilation error are self-explanatory. However, there are two types of messages that require some explanation.

If your BASIC source program includes an illegal statement or function, the compiler will display a coded message number that indicates which type of statement or function caused the error. A list of error message numbers and their corresponding statement/function follows.

7.2 Illegal Statement Messages

Code #	Statement Name	Code #	Statement Name
4	LIST	5	ENTER
14	BYE	15	CONT
19	DEG	22	NEW
24	LOAD	25	SAVE
33	RAD	37	RUN
46	DOS	51	LPRINT
52	CSAVE	53	CLOAD

7.3 Illegal Function Messages

Code #	Function Name	Code #	Function Name
68	ATN	69	COS
71	SIN	72	RND
74	EXP	75	LOG
76	CLOG	77	SQR

7.4 Run-Time Errors And Program Termination

Only one type of message can result from a run-time error. This message displays a standard Atari BASIC error number along with the original BASIC line number that produced the P-code where the error occurred. You will also see a menu which allows you proceed in various ways by pressing a console key:

OPTION	Reboot entire system
SELECT	Return to DOS
START	Re-run the stopped program

The above menu will also appear if a BASIC END command is encountered, or if the interpreter runs out of instructions to execute.

Warranty Information

Monarch Data Systems warrants to the original purchaser that this Monarch Data Systems program diskette (not including the computer programs) shall be free from any defects in materials or workmanship for a period of 90 days from the original date of purchase. If a defect is discovered during this 90-day warranty period, and you have timely validated this warranty, Monarch Data Systems will repair or replace the diskette at Monarch Data System's option, providing that the diskette and proof of purchase are delivered or mailed, postage prepaid, to Monarch Data Systems.

This warranty shall not apply if the diskette:

-- Has been misused, or shows signs of excessive wear;
-- Has been damaged by the playback equipment, or;
-- If the purchaser causes or permits the diskette to be serviced or modified by anyone other than Monarch Data Systems.

Any applicable implied warranties, including warranties of merchantability or fitness, are hereby limited to 90 days from the original date of purchase. Consequential or incidental damages resulting from a breach of any applicable express or implied warranties are hereby excluded.

Notice

All Monarch Data Systems computer programs are distributed on an "as is" basis, without warranty of any kind. The entire risk as to the quality and performance of such programs lies with the purchaser. Should the programs prove defective following their purchase, the purchaser and not the manufacturer, distributor or retailer assumes the entire cost of all necessary servicing or repair.

Monarch Data Systems shall have no liability or responsibility to a purchaser, customer or any other person or entity with respect to any liability, loss or damage caused or alleged to have been caused directly or indirectly by computer programs sold through Monarch Data Systems. This includes but is not limited to any interruption of service, loss of business or anticipatory profits or consequential damages resulting from the use or operation of such computer programs.

The provisions of the forgoing warranty are subject to the laws of the state in which the diskette is purchased. Such laws may broaden the warranty protection available to the purchaser of the diskette.

SOFTWARE REPORT #2, November 1, 1983

ABC BASIC COMPILER REV 1.03

This note is intended to help you get the most out of the ABC BASIC Compiler by letting you know about existing problems and offering some helpful hints.

Items numbered 9 through 19 appeared in software report #1. Only the relevant items are repeated from that report, hence the strange numbering here. All known bugs in release 1.02 have been corrected in release 1.03. In addition, several improvements have been made and begin with item number 21 in list.

9. If you use an XIO 18,#6,0,0,"S:" which is a Fill function, you will find that, with ABC, you can no longer PLOT any more points. This is because the number after the filespec, in this case a zero, represents the read/write modes that are set on the file. The zero that most people use prevents further reading or writing to the screen. Atari BASIC doesn't complain because it does not use the standard OS output routine; but try to do a LOCATE after this kind of Fill and you'll see the problem. The solution is simple. Just use XIO 18,#6,12,0,"S:" and all will be fine.

14. The locations 186 and 187 used to hold the line number on errors can only be printed with the following code sequence. Changing the order of the expression will cause it to not work because locations 186 and 187 are right in the middle of the run-time value stack for ABC.

```
PRINT PEEK(186)+PEEK(187)*256
```

15. To make a compiled BASIC program run automatically when a disk is booted simple name the program "AUTORUN.SYS". Yes, it's that simple. If you want to sell your software, you'll have to get a license from ATARI to distribute DOS.

16. To have one compiled program run another the following code can be used:

```
OPEN #1,4,0,"D:FILENAME.EXT"
X = USR(5576)
```

This ONLY works with DOS 2.0. Thanks to Jerry White for this helpful hint.

17. Unfortunately, Atari software has a bug which prevents the RS-232 handler from operating if DOS is ever entered after the RS-232 handler is initialized. That is, once the AUTORUN.SYS file is executed, DOS, or more correctly DUP.SYS, when loaded, will destroy the handler.

The solution to this problem is to not allow DUP.SYS to load once the handler is initialized. The only way to do this and run a compiled program is to concatenate the compiled BASIC program to the AUTORUN.SYS file and load them together as one file. By keeping the name AUTORUN.SYS your program will run automatically when the disk is booted.

This is accomplished by first compiling the BASIC program in the normal manner. Then, type "DOS" to the BASIC cartridge. Now use the copy with append option to append the compiled file to the AUTORUN.SYS file as follows:

```
(COMPUTER TYPES)   SELECT ITEM OR RETURN FOR MENU
(YOU TYPE)         C
(COMPUTER TYPES)   COPY - FROM,TO?
(YOU TYPE)         NEWPGM.CMP,AUTORUN.SYS/A
```

Each time you recompile the program you will have to get a "fresh" copy of the AUTORUN.SYS file from you master diskette.

18. Sections 4.12 and 6.3 of the manual seem to cause confusion instead of clarifying the issue. What the manual was trying to say was that you cannot depend on ABC to allocate one string immediately after another in memory because of the string control bytes. So if you have "DIM A$(100),B$(100)" you cannot expect that the address of B$ to be exactly 100 more than the address of A$. Some programmers use this fact to allocate a player/missile graphics area. This will not work with ABC.

19. Use locations 14 and 15 instead of 144 and 145 to find the end of your BASIC program in memory. Locations 14 and 15 will work with both the cartridge and with ABC. Be sure to first DIMension all arrays and strings then add some slop (1000 bytes ought to be enough) to the address you find at 14 and 15.

Differences between release 1.02 and 1.03
(aside from all bugs in report #1 fixed).

21. The compiler now generates smaller code for variable references, saving three bytes per reference over release 1.02. Now, "crunching" your program by using variables instead of constants works great.

22. The GOSUB stack now allows up to 127 entries before overflowing. This number is up from 64 specified in section 4.11 of the manual. This change also means that now 254 bytes are allocated for the stack instead of 128 mentioned in section 6.3. Another byproduct of this change is that it should eliminate any error #10 you may have seen during compilations with release 1.02.

23. The disk is copy protected, but you can now load the compiler from any drive. Use the DOS 'L' command to load AUTORUN.SYS from the compiler disk and use the OPTION button to specify the <u>drive number</u> and name of the runtime package you want to use (see section 5.1 of the manual). This technique will allow you to have a double density drive as your default drive #1 as well as have the compiler run under other operating systems.

24. The temporary file needed for compilation is now placed on the same drive as the 'destination' file. This allows you to leave the compiler disk in drive #1 during a compilation.

25. The screen is now made 'attract proof' during compilation.

26. Spacing of most error messages has been corrected.

27. The correct number of 'dots' is displayed on the screen during compilation for those who noticed that there was one too many.

28. The 'UNSUPPORTED FUNCTION' message now displays the correct number shown in the manual. Release 1.02 showed a number that was 60 less in value.

29. The compiler still does not compile a program that has no variables.

30. Release 1.03 works on the new Atari 1050 drive; release 1.02 did not.

MORE HINTS

31. When looking through a BASIC program to determine whether it will work unmodified with ABC, check for all PEEKs and POKEs to addresses in the range of 128 to 255. These are the BASIC cartridge specific locations and are not supported by ABC (see note #19, for example).

32. Instead of using the code shown in section 4.13 for timing loops, you can use the following simpler code:

```
100 POKE 540,NJIFFIES              up to 255 in 1/60 seconds
110 IF PEEK(540)<>0 THEN 110       wait for time to elapse
```