

# Datasoft's Atari Basic Compiler

Converted to html by [Isaac Davis](#)

# THE BASIC COMPILER

For the ATARI 400/800® Computer

By  
Phillip Dennis  
® Special Software Systems

Copyright © 1983

**Datasoft Inc.**  
PERSONAL COMPUTER SOFTWARE

9421 Winnetka Avenue  
Chatsworth, CA 91311

ALL RIGHTS RESERVED

ATARI is a registered trademark of ATARI Inc.  
Percom is a registered trademark of Percom Data Co., Inc.  
DATASOFT is a registered trademark of Datasoft Inc.

## TABLE OF CONTENTS

<u>I. OVERVIEW</u>	1
<u>The Compiler Program Diskette</u>	1
<u>Requirements</u>	2
<u>II. HOW TO COMPILE A PROGRAM</u>	2
<u>Single-Drive System</u>	2
<u>Two-Drive System</u>	2
<u>THE COMPILER AND DOS OPTIONS</u>	3
<u>Which to Use, COMPILER or DOS?</u>	3
<u>COMPILER OPTION</u>	3
<u>BASIC FILE and OBJECT FILE</u>	4
<u>Integer or Floating Point Arithmetic</u>	4
<u>HOW THE COMPILER WORKS</u>	4
<u>PASS 1</u>	4
<u>THE SYSTEM RESET Key</u>	5
<u>PASS 2</u>	5
<u>PASS 3</u>	6
<u>PASS 4</u>	6
<u>The SYSTEM RESET Key</u>	6
<u>Line Reference Map</u>	7
<u>Run Program</u>	8
<u>1. Decimal Address Run</u>	8
<u>2. Rerun Program From Beginning</u>	8
<u>3. DOS Control</u>	8
<u>DOS</u>	9
<u>DOS OPTION</u>	9
<u>Transferring Compiler Support Files</u>	9
<u>Loading the BASIC COMPILER From DOS</u>	10
<u>III. TECHNICAL NOTES</u>	10
<u>Integer/Floating Point Arithmetic Option</u>	10
<u>Compiler Files</u>	11
<u>Disk Space Requirements</u>	13
<u>Assembler Program Control Statements</u>	14

<u>IV. COMPILER ERRORS</u>	14
<u>System Errors</u>	14
<u>Programming Errors</u>	16
<u>Compile-time Errors</u>	16
<u>Assembler System Error</u>	19
<u>Run-time Errors</u>	19
<u>TABLE OF RUN-TIME ERRORS</u>	23
<u>V. OPTIMIZING YOUR BASIC PROGRAM</u>	24
<u>Timing Considerations</u>	24
<u>How to Produce Smaller Compiled Programs</u>	24
<u>VI. DIFFERENCES BETWEEN COMPILED AND INTERPRETED BASIC</u>	27
<u>Commands Not Recognized by the Compiler</u>	27
<u>String Handling</u>	27
<u>FOR Loops</u>	28
<u>GOTO and GOSUB Variable Line Numbers</u>	28
<u>DATA Statements</u>	29
<u>Memory Between VSEC and VEND</u>	29
<u>GLOSSARY</u>	30
<u>APPENDIX A</u>	31
<u>APPENDIX B</u>	32
<u>APPENDIX C</u>	33
<u>COMMERCIAL SALE OF COMPILED PROGRAMS</u>	34

# ATARI BASIC COMPILER

## I. OVERVIEW

DATASOFT proudly presents the first compiler for the ATARI computer! The BASIC COMPILER is a useful tool for those who want the speed of a machine language program and the ease of BASIC programming. You have the advantage of writing your program in BASIC and compiling it into machine language to run at lightning speed. Knowledge of assembly or machine language is NOT required.

The ATARI BASIC COMPILER is a four-pass compiler which generates 6502 machine language. During Pass 1 the Compiler translates your BASIC program into assembler files. Passes 2, 3, and 4 utilize an assembler to assemble the files generated in Pass 1 into machine language. All assembler files and the final compiled program are saved on your disk.

As an added benefit to the advanced assembly language programmer, the Compiler produces DATASM compatible assembler files which can be utilized with other assembler programs.

This manual contains an index for quick reference, as well as a glossary to help familiarize you with the use of some words.

### **The Compiler Program Diskette**

The program diskette contains the following files:

DUP.SYS

DOS.SYS

AUTORUN.SYS-Title Page Program

BASCOMP.OBJ-The BASIC COMPILER

ASM.OBJ-The Assembler

SYSEQU.ABC-Run-time Library Equate file

SYSLIB.FP-Floating Point Run-time Library

SYSLIB.INT-Integer Run-time Library

The Compiler program (BASCOMP.OBJ), assembler program (ASM.OBJ), and the support files (run-time libraries) are described in Part III. Please do not remove the write protect tab from the BASIC COMPILER diskette..

### **Requirements**

ATARI 400/800 computer-48K

ATARI BASIC cartridge

ATARI 810 or Percom disk drive (one or more)

diskette with your BASIC program (DOS.SYS, DUP.SYS optional)

Printer (optional)

**NOTE:** The BASIC COMPILER requires that your BASIC program be structured within certain constraints. Please see Section III "TECHNICAL NOTES" and Section IV "COMPILER ERRORS" before attempting to compile your program. By reviewing this information and revising your program as needed, you will minimize "Compiler errors" and the necessity to interrupt the compilation process to make changes.

## II. HOW TO COMPILE A PROGRAM

### Single-Drive System

Remove the BASIC cartridge from the computer and turn the computer off. Close the door on your disk drive and turn it on. After the red busy light on your drive goes off, insert the BASIC COMPILER program diskette and close the drive door. Turn on the computer. (NOTE: If you forget to remove the BASIC cartridge the program will not boot and the message "REMOVE CARTRIDGES AND REBOOT" will be displayed.)

### Two-Drive System

If you have a two-drive system you will need to make a copy of the BASIC COMPILER program diskette before you begin.

Follow the instructions for a single-drive system, but in addition to inserting the BASIC COMPILER diskette into drive #1, insert a blank, formatted disk into drive #2. Turn on the computer. The program disk in drive #1 will boot and the title page will be displayed. Select the DOS option on the title page. After the DOS menu appears on the screen utilize "J" on the menu to duplicate the entire program diskette. (NOTE: The Compiler program (BASCOMP.OBJ) will not be functional-on the duplicate copy.) After the BASIC COMPILER program diskette is duplicated, reboot the system. After the title page appears, remove the BASIC COMPILER disk from drive #1 and replace it with the duplicate program disk. Insert the disk containing your BASIC program (BASIC program work disk) to be compiled in drive #2.

## THE COMPILER AND DOS OPTIONS

After the disk is booted the title page will be displayed with the following options:

COMPILER or DOS?

### Which to Use, COMPILER or DOS?

If you have a single-drive system the COMPILER option requires you to swap the BASIC COMPILER program disk with your BASIC program work disk (containing your BASIC Program to be compiled) five times. (NOTE: If you have a two-drive system all disk swapping is eliminated. You may proceed directly to the COMPILER OPTION instructions below.)

If you are familiar with using DOS-the disk operating system (and disk maintenance) which allows you to save and load files or programs-you should select the DOS option described on page 9. Utilizing DOS will allow you to copy any or all of the Compiler support files and the assembler program from the BASIC COMPILER program disk to your disk, thus eliminating most or all disk swaps.

## COMPILER OPTION

Press C and the Compiler program will load automatically. After the program is loaded, you will be

prompted to insert your BASIC program work disk. You will then be prompted to type your BASIC program file name. To see a catalog of the disk' in drive #1 before you type the file name, press **RETURN**.

## **BASIC FILE and OBJECT FILE**

Type in the name of the BASIC file on your disk you want to compile and press **RETURN**. You will then be prompted to type in the object file which is the name you want to give the compiled program. It is suggested you use the same name as your BASIC program but with a different extension such as ".OBJ" (for object file).

If you try to compile a non-BASIC file you will be prompted "FILE NOT BASIC" and reprompted to type in a BASIC file name

NOTE: On a two-drive system if your BASIC file and object file are in drive #2, you must specify "D2:".

## **Integer or Floating Point Arithmetic?**

After typing in the object file name, you will be asked if you want the Compiler to use integer or floating point arithmetic. Pressing **F RETURN** will instruct the program to use floating point arithmetic (the same as the BASIC cartridge). Answering with an **I RETURN** will instruct the program to use 16-bit integer arithmetic.

Such a program will run 5 to 20 times faster than the original BASIC program. However, some BASIC statements perform differently when the integer option is used. These differences are explained in Part III.

## **HOW THE COMPILER WORKS**

### **PASS 1**

After either the integer or floating point options are selected, the Compiler assembles each line of your BASIC program, creates assembler files (ASSEM.SG1, SG2, etc.), writes them on your disk (in drive #1), and displays a list of the BASIC line numbers to be compiled. (The SG1, SG2, etc. file name extensions refer to file segments which are numbered consecutively as they are written to your disk.) For example:

```
< BC > COMPILING LINE 110
< BC > COMPILING LINE 120
< BC > COMPILING LINE 130
etc.
```

After the assembler files have been generated the Compiler looks for the assembler (ASM.OBJ) and, if you have not transferred this file to your BASIC work disk, will display the message:

```
< BC > END-OF-PASS1
```

INSERT COMPILER DISK INTO DRIVE #1  
THEN PRESS **RETURN**

Remove your BASIC work disk and reinsert the Compiler program disk and press **RETURN**.

During, or at the end of, each pass the Compiler looks for particular files to execute the next step in the compilation process. Therefore, you will either be prompted to insert the BASIC COMPILER program diskette or to "insert disk with file..." Remember, the amount of prompts and the required disk swapping will depend on whether or not you transfer any Compiler files to your BASIC work disk.

Note: To avoid disk swapping you may transfer the necessary files from the BASIC COMPILER program disk to your own by utilizing the DOS Option at the beginning of the program. **See Page 9 for information on transferring the Compiler files.**

### **The SYSTEM RESET Key**

Pressing this key during PASS 1 will return the program to the beginning of PASS 1.

### **PASS 2**

After completion of PASS 1, the Compiler will load and execute the assembler program (ASM.OBJ) which translates the assembler files to executable machine language. The assembly process requires three passes which are numbered PASS-2, PASS-3, and PASS-4. As the assembler completes each of its passes it will update the screen with:

< BC > PASS-2, etc.

The assembly process may require more time for longer programs. To let you know that your computer and the assembler are operating, a cursor will be flashing in the upper left corner of the screen.

### **PASS 3**

At the end of PASS 2 the Compiler looks for the assembler files (ASSEM.SG1, SG2, etc.). Since these files were written to your BASIC work disk the screen will display the file name and prompt you to insert the correct disk as shown below.

< BC > PASS-2

Can't find file->D:ASSEM.SG1  
Please Insert Correct Disk

PRESS ANY KEY TO CONTINUE

After pressing any non-function key PASS 3 will begin., PASS 3 requires the SYSEQU.ABC file and you will be prompted to insert the disk (BASIC COMPILER program disk) which contains this file.

(NOTE: Utilizing the DOS option to transfer this file to your BASIC work disk will eliminate this and the following disk swap.)

## PASS 4

PASS 4 requires the assembler files again. Therefore, at the end of PASS 3 you will be prompted to insert the disk which contains this file (your BASIC work disk).

### The **SYSTEM RESET** Key

Pressing this key during Passes 2, 3, or 4 will abort the assembly process and return control to DOS.

After PASS 4 is completed your BASIC program will be compiled into machine language and saved to your disk in auto-runnable form with the file name you initially selected. The screen will now display three options:

**P**rint line map

**R**un program

**D**os

### Line Reference Map

The reference map provides a useful tool for the programmer who may need to know where a BASIC line number resides in the compiled program. You can also use the reference map to determine the size of the compiled program. (This is done by finding the address of the Compiler generated line number "99999." The address of this line is the last memory location used by the compiled program.)

Press **P** if you want to see the reference map. You will then be prompted to select where you want the map printed, either to your printer or screen, or written to your disk.

To:

**P**rinter

**S**creen

**D**isk

Press the appropriate letter and **RETURN** for the desired option. If you select the Screen option, press **CTRL 1** to start and stop the scrolling on the display.

### Save to Disk

If you want to save the reference map to your disk, press D. You will be prompted for the drive number and file name as shown:

DEVICE:FILENAME?->

Type in the drive # which contains the disk you want the map -written to, and then specify the file name you want to give the reference map. Press **RETURN** and the map will be saved to your disk under your specified file name.

After the map has been displayed on your screen, printed out, or saved to disk the "Select Option" prompt will reappear.

## Run Program

Press **R** to run your compiled program. After the program is finished and no errors have occurred, the message

BASIC exit

will be displayed. The run-time library (floating point or integer, depending upon your initial selection in Pass 1) is now in control and will display the message:

?Run address>

You may now execute one of three options: 1) rerun program at a specific decimal address, 2) rerun the entire program, or 3) yield control to DOS. If you select the last option, make sure the disk in drive #1 contains the file DUP.SYS.

### 1. Decimal Address Rerun

By typing a specific decimal address of one of the BASIC line numbers in your compiled program after the prompt "?Run address>," you may rerun the program at that address. The method for determining memory addresses is described under Section IV "Run-Time Errors."

### 2. Rerun Program From Beginning

Press **RETURN** to rerun the entire compiled program.

### 3. DOS Control

Alternatively, typing "DOS" will yield control to DOS. If you type "DOS" be sure you have the file "DUP.SYS" on the disk presently in drive #1.

All the compiled programs generated by the BASIC COMPILER are saved in "auto-runnable" form. This means that if you load the program using the BINARY LOAD function ("L" on the DOS menu), it will run automatically after it is loaded.

## DOS

You may go directly to DOS after PASS 4 is complete by pressing **D**.

## DOS OPTION

Press **D** to display the DOS menu. To see a directory of the Compiler program diskette, execute the "A" option. Your diskette will contain the following Compiler-related files.

DUP.SYS  
 DOS.SYS  
 BASCOMP.OBJ  
 ASM.OBJ  
 SYSEQU.ABC  
 SYSLIB.FP  
 SYSLIB.INT

### **Transferring Compiler Support Files**

As mentioned previously, you may want to transfer some or all of the Compiler support files and the assembler program to your diskette which contains your BASIC program to reduce disk swapping. However, you may not want to transfer all of these files as they will occupy too much disk space if you want to compile large BASIC programs.

For a single drive system SYSEQU.ABC is the only compiler support file you really need to transfer. The presence of this file on your disk will eliminate all disk swapping except for two required during Passes 1 and 2.

To eliminate all disk swapping you may transfer the system run-time libraries (SYSLIB.FP, SYSLIB.INT), the system equate file (SYSEQU.ABC), and, the assembler (ASM.OBJ) to your diskette. The Compiler (BASCOMP.OBJ) is copy-protected and can be transferred but it will be inoperable on the duplicated disk.

To transfer any of the above files to your disk on a single-drive system use "O" (DUPLICATE FILE) on the DOS menu. In a two drive system, use "C" (COPY FILE). To duplicate the entire disk, use "J" (DUPLICATE DISK).

If you transfer only some or none of these files, and if the Compiler or assembler cannot locate the files they need to execute a particular phase of the compilation process, you will be prompted to insert the disk containing the required file. For example, if the assembler file (ASM.OBJ) was not copied to your BASIC work disk you will be prompted with::

Can't find file->D:ASM.OBJ Please Insert Correct Disk

PRESS ANY KEY TO CONTINUE

Please see Section III for a more detailed discussion on the Compiler files before you begin transferring them.

### **Loading the BASIC COMPILER From DOS**

You may also load the BASIC COMPILER program directly from DOS by selecting the "L" option (BINARY LOAD) on the DOS menu. After pressing **L RETURN** you will be prompted with:

LOAD FROM WHAT FILE?

Type in BASCOMP.OBJ (the Compiler) and press **RETURN**. The Compiler will be automatically loaded. From this point forward follow the COMPILER Option instructions beginning with "BASIC FILE and OBJECT FILE" (Section II, on page 4).

### III. TECHNICAL NOTES

#### Integer/Floating Point Arithmetic Option

For the ATARI, floating point arithmetic is the standard method for performing all arithmetic functions. If you select integer arithmetic the Compiler program will insert a copy of the integer run-time package into the compiled program. This package performs identically to the floating point run-time package except any calls to the transcendental functions will produce a run-time error. These functions include:

SQR, COS, SIN, CLOG, LOG, EXP, and ATAN.

The RND (random) function also operates differently with integer arithmetic. In the integer option

RND(X)

will return a random integer between 0 and x-1 inclusive. For example:

RND(4)

will return a 0, 1, 2, or a 3. If you select the integer option also be sure that your program does not contain any fractional constants (e.g., 0.25). Such constants will be converted to integers resulting in a compiled program that will not perform the same as the original BASIC program.

Finally, even though a program compiled using the integer option can only generate numbers in the range from -32768 to 32767, constants embedded in your program for performing important PEEKs and POKEs will still generate the correct address. For example:

A=PEEK(53279)

will correctly read the console buttons **OPTION, SELECT, and START**. However:

PRINT 53279

will print as -12257, which is 53279-65536.

#### Compiler Files

The BASIC COMPILER program consists of two groups of files including the Compiler program segments and the Compiler support files.

All of the Compiler programs are identified by the .OBJ filename extension. The files are:

BASCOMP.OBJ -The Compiler (PASS 1)

ASM.OBJ -The Assembler (PASS 2, 3, and 4)

The Compiler support files include:

SYSEQU.ABC -The Run-time Library Equate File (used by the assembler)

SYSLIB.FP -The Floating Point Run-time Library

SYSLIB.INT -The Integer Run-time Library

ASSEM.SG1, -These are the actual assembler files in

ASSEM.SG2, DATASM compatible mnemonics. The SG1,  
etc. SG2, etc. extensions represent segments 1, 2. etc

Note: The Compiler and assembler programs assume that the first three support files are located on a diskette in drive #1. Also, the assembler files (ASSEM.SGI, SG2. etc.) are always written to drive #1, i.e. they always have the device specification D1:.

### SUGGESTED DRIVE LOCATIONS OF COMPILER FILES FOR A SINGLE OR MULTIPLE-DRIVE SYSTEM

File	Drive Number
SYSLIB.INT	1
SYSLIB.FP	1
SYSEQU.ABC	1
ASSEM.SG1,SG2, etc.	1
ASM.OBJ	1, 2, 3, or 4
BASIC program to be compiled	1, 2, 3, or 4
Object file	1, 2, 3, or 4

For a two-drive system an optimal diskette configuration is listed below.

File	Drive Number
ASM.OBJ	1 or 2
BASIC program to be compiled	2
Object file	2

```

SYSEQU.ABC  1
SYSLIB.INT  1
SYSLIB.FP   1

```

## Disk Space Requirements

The assembler files (ASSEM.SG1, SG2, etc.) require approximately five times as much disk storage as the BASIC program. The final executable file requires approximately the same amount of storage as the BASIC file. Therefore, you should make sure that the diskette you are using with the Compiler has at least as many free sectors equal to six times the size of the BASIC program which you want to compile. The largest BASIC program which can be compiled on a single drive system is 100 sectors. **6 x 100 sectors + 100 sectors (original program) = 700 sectors.**

The following data from an actual compilation of a 123-line BASIC program demonstrates the disk space requirements.

BASIC source file size:	42 sectors
ASSEM.SG1:	104 sectors
ASSEM.SG2:	81 sectors
object file size (compiled program including run-time):	89 sectors
TOTAL:	316 sectors

## Assembler Program Control Statements

The programmer who understands assembly language and wants to use or modify any assembler files (ASSEM.SG1, SG2, etc.) created by the Compiler should note that the following assembler statements are recognized by the assembler program (ASM.OBJ).

Program Control Statements:

.END ends the assembly

.FILE chains two or more files in an assembly

Symbol and Data Definition Statements:

= defines a symbol

.BYTE defines byte oriented data in memory

WORD defines address constants

.DBYTE defines word oriented (16 bits) data in memory

Also, the high byte of symbols is denoted by the greater than sign ( > ) and the low byte of a symbol with the less than ( < ) sign.

## IV. COMPILER ERRORS

Errors that occur during the compilation of a BASIC program are "compile-time" errors. They may be due to errors in the BASIC program which is being compiled or due to ATARI system errors. Errors that occur while running a compiled program are "run-time" errors. If an error occurs an error message will be displayed and you may proceed as described.

### System Errors

If an ATARI system error number is displayed, consult your BASIC or ATARI manual for a description. The following system errors, including BASIC programming errors, may also occur during PASS 1.

```
SYSTEM ERROR-CAN'T RUN ASSEMBLER  
COMPILATION ABORTED  
PRESS RETURN TO CONTINUE
```

Cause: The Compiler has encountered a bad ASSEMBLER file (D:ASM.OBJ). This usually means you have a damaged disk.

Recovery: Restore your Compiler disk from a backup copy.

```
BAD FREE TEMP  
COMPILATION ABORTED  
PRESS RETURN TO CONTINUE
```

Cause: There is an internal Compiler inconsistency. The Compiler has attempted to deallocate storage from the temporary results stack when no storage was actually allocated.

Recovery: Reboot the Compiler and try to recompile your program.

```
BAD INPUT FROM BASIC  
FILE COMPILATION ABORTED  
PRESS RETURN TO CONTINUE
```

Cause: The Compiler has encountered an unexpected character in the BASIC program. This can be indicative of a damaged file.

Recovery: Try to execute the program from BASIC. If the program appears to execute properly, perform

the following steps.

```
LIST "D:TEMP.TXT
NEW
ENTER "D:TEMP.TXT"
SAVE "D:XXX.BAS"
```

Now rerun the Compiler.

If any of the above errors occur your only option is to press **RETURN** to continue. The Compiler will relinquish control to DOS. Therefore, you should make sure your diskette in drive #1 contains a copy of DUP.SYS and DOS.SYS before pressing **RETURN**.

### Programming Errors

When the Compiler encounters one of the programming errors listed below, it will display the error message followed by the line:

```
SKIPPING STATEMENT
CONTINUE OR ABORT (C/A)?
```

Pressing **C** permits PASS1 to continue, thus allowing all possible programming errors in the BASIC program to be detected. The Compiler will skip the statement containing the error and continue the compilation at the next statement in the BASIC program. At the end of PASS 1, if any errors occurred the following message will be displayed.

```
X ERROR(S) DETECTED
```

X is the total number of errors which were detected. At this point, the Compiler will stop, requiring you to reboot the system. Turn the computer off, insert your BASIC cartridge and correct the errors. Recompile your program.

If you press **A RETURN** in response to the CONTINUE or ABORT question, control will be returned to DOS.

### Compile-time Errors

```
ILLEGALLY PLACED STATEMENT
```

Cause: The Compiler has encountered a non-DATA statement after DATA statements.

Recovery: Move the non-DATA statement(s) to lines numbered lower than all DATA statements.

### ILLEGAL NEXT

Cause: A NEXT is trying to increment a loop variable which does not match the variable in the corresponding FOR statement.

Example:

Incorrect: FOR 1=1 TO 10:NEXT J

Correct: FOR 1=1 TO 10:NEXT I

Recovery: Correct the NEXT statement.

### DYNAMIC DIM NOT ALLOWED

Cause: A DIM statement must use constants instead of variables to allocate array and string storage.

Example:

DIM X(A), Y(2\*A), Z(A+3)

Recovery: Replace the DIM statement with one which uses constants. For example, if A equals 2 in the above statement make the replacement such that the statement reads DIM X(2), Y (4), Z (5).

### NEXT WITHOUT FOR

Cause: The BASIC program contains a NEXT statement without a matching FOR statement.

Recovery: Remove the NEXT or insert the appropriate FOR.

### RE-DIMENSION ERROR

Cause: A string or array appears in more than one DIM statement. The second will be ignored.

Recovery: Remove the extra DIM statement.

### SYNTAX ERROR

Cause: The BASIC program contains a misspelled BASIC word, is missing a comma, or quote marks,

etc.

Recovery: Correct the syntax error using BASIC and recompile.

### CAN'T COMPILE STATEMENT

Cause: The BASIC program contains statements such as "100 LIST" not supported by the Compiler.

Recovery: Remove the inappropriate statements.

### UNDIMENSIONED ARRAY

Cause: The Compiler has encountered a statement containing a doubly subscripted array (e.g. A(8,12)) before its DIM or COM statement.

Recovery: Move the array's DIM statement to a line number lower than all lines which reference the array.

### UNDEFINED LINE NUMBERS

Undefined line numbers in your BASIC program are trapped during PASS 3 by the assembler. For example, if your BASIC program contains the statement:

```
100 GOTO 1000
```

and there is no line numbered 1000, then the assembler will respond by displaying the incorrect assembler instruction and the line number which is undefined. For this example, the assembler would display the message:

```
->JMP L1000
```

```
< BC > System error
```

```
< BC > Ref: Line #- > 1000
```

```
< BC > Unresolved line number  
Continue (Y/N)?
```

JMP L1000 is the assembler instruction and the line number is 1000. The assembler will ask you if you want to continue. Pressing **Y** for "yes" will resume the assembly process and permit the program to check for other possible undefined line numbers. You should not try to run a program which contains undefined

line numbers. Running such a program will cause your computer to stop if it reaches a statement containing an undefined line number. If you do not want to continue the assembly after an unresolved line number, then just press **N** for "no." Control will be returned to DOS.

### GOTO/GOSUB VAR or EXP

**Cause:** The BASIC program contains a GOTO/GOSUB to a variable line number statement. Statements such as GOTO 1000 + X are not supported by the Compiler.

**Recovery:** Replace the GOTO/GOSUS statement by the appropriate GOTO or GOSUB statement. For example:

```
ON X GOTO 1001, 1002....
```

### Assembler System Error

In addition to the normal ATARI system errors the following system error can occur during PASSes 2, 3, and 4.

System error: 255

**Cause #1:** The assembler has encountered a reference which is not defined in the system equate file. This usually means that the file SYSEQU.ABC has been damaged.

**Cause #2:** The assembler cannot find the next assembler file (ASSEM.SG1, etc.). This usually means that your assembler files have been destroyed since they were created by PASS #1. This should never occur, since a damaged file would normally be trapped as an ATARI system error. It is also possible that the Compiler itself (BASCOMP.OBJ) is damaged.

**Recovery:** Rerun the Compiler using different diskettes and restore SYSEQU.ABC. If the Compiler does appear to be damaged, you will need to contact DATASOFT for a replacement disk.

### Run-time Errors

Running a compiled program may produce the following runtime errors. These errors are identical to the ATARI BASIC error numbers. Any error number not listed is not produced by the compiled version. The ATARI BASIC TRAP statement works identically in the compiled version. Executing a PEEK(195) will return the type of error encountered. If a TRAP statement has not been executed or the trapping mechanism is reset (as in TRAP 40000), then the run time package will print the address of the inappropriate instruction. You may then resume at a given address when the run time package prompts:

?Run address>

This address should be a decimal address corresponding to the address of a BASIC line number as shown in the line reference map. Instead of typing an address you can also enter one of the following options.

- Type **RETURN** to rerun the program.
- Type **DOS RETURN** to abort the program completely and return control to DOS.
- Type **C RETURN** to continue running the program beginning from the line where the error occurred.

### Example of Run-time Error

The following discussion shows how to determine the line number at which a run-time error occurs. To determine the line number you need a listing of the BASIC program and a listing of the line reference map. Consider the following BASIC program as an example.

```

100 REM
110 REM TEST RUN-TIME ERROR->
120 REM
130 REM PROGRAM WILL GET AN
140 REM ERROR 11 WHEN I=0
150 REM
160 FOR I=10 TO 0 STEP -1
170 PRINT 10/I
180 NEXT I
190 END

```

When run from BASIC the program produces the following results.

```

1
1.11111111
1.25
1.42857142
1.66666666
2
2.5
3.33333333
5
10

```

```

ERROR- 11 IN LINE 170

```

```

READY

```

Compiling the program will produce the following line reference map.

LINE # 100 = 12811  
 LINE # 110 = 12811  
 LINE # 120 = 12811  
 LINE # 130 = 12811  
 LINE # 140 = 12811  
 LINE # 150 = 12811  
 LINE # 160 = 12811  
 LINE # 170 = 12825  
 LINE # 180 = 12849  
 LINE # 190 = 12882  
 LINE # 99999 = 12918

This map shows that the compiled code for line 170 resides at memory addresses 12825 to 12848 inclusive.

When the compiled program is executed the following display should result.

```

1
1.11111111
1.25
1.42857142
1.66666666
2
2.5
3.33333333
5
10
  
```

```

ERROR- 11
Trace:
12840
?Run address>
  
```

The compiled program reports an error 11 and then shows a trace of addresses which show the sequence of subroutine calls which led to the error. In this case no subroutines were called so the trace just shows the address 12840. Since this address is between the start and end address for line number 170 (as shown in the line reference map) we conclude that line 170 is where the error occurred.

## TABLE OF RUN-TIME ERRORS

<u>Error Number</u>	<u>Definition</u>
06	Out of data
11	Arithmetic error (OVERFLOW OR DIVIDE BY ZERO)

18	Invalid string character
128	Break key abort
129	IOCB already open
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	End of file
137	Truncated record
138	Device timeout
139	Device NAK
140	Serial bus error
141	Cursor out of range
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 files open
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

## V. OPTIMIZING YOUR BASIC PROGRAM

### Timing Considerations

Many programming applications require timing loops, either to provide synchronization or small pauses during presentations. Since compiled programs run faster, you should change your timing parameters using the following information as a guide.

If you compile using the standard floating point package your program will run approximately three times faster. You should therefore make your timing loops about three times larger.

If you compile using the integer package, your program will run approximately 15 times faster. Your timing loops should, therefore, be about 15 times bigger.

As a quantitative example, the following FOR NEXT loop will produce the indicated time delays.

```
100 FOR I=1 TO N:NEXT I
```

Program environment time delay seconds:

Uncompiled BASIC 0.0025 \* N

Compiled FP 0.000975 \* N

Compiled INT 0.000195 \* N

An alternative to delay loops is to use the system countdown timers. Such delays will be the same whether interpreted or compiled.

### How to Produce Smaller Compiled Programs

The ATARI BASIC Compiler was designed to produce the fastest possible machine code. Consequently, to achieve greater speed, the code produced is typically longer than the uncompiled BASIC program. The size increase is usually 20% larger, excluding the run-time package. For example:

```
100 X = A + B
```

is compiled into:

```
L100
```

```

LDX # < A
LDY # > A
JSR LD0; LOAD FP REG ZERO
LDX # < B
LDY # > B
JSR ADD; ADD TO B
LDX # < X
LDY # > X
JSR STO; STORE REG ZERO AT

```

which is a total of 21 bytes. The original BASIC line requires 10 bytes.

The statement:

```
100 GOSUB 2000
```

is compiled into:

```

L100
JSR L2000

```

which takes only 3 bytes.

Therefore, the key to producing compact code is to use subroutines liberally. Instead of repeating the statement:

```
...X = A + B
```

throughout your program (which requires 21 bytes per use), replace it by a subroutine. Then each additional usage of "X = A + B" will only require 3 bytes. If you maximize the use of subroutines you will have a compiled program smaller and faster than the uncompiled BASIC program.

Other BASIC statements which produce large compiled blocks include the following.

Substring expressions as in:

$X(I,J)$  etc.

Array references as in:

$X(I) = I$

FOR NEXT loops with a variable step size. For example:

```
FOR I = 1 TO 100 STEP X
```

```
"
```

```
"
```

```
"
```

```
Next I
```

The NEXT I statement compiles into:

```
LDX# < I
LDY # > I
JSR LD0; LOAD FP REG ZERO
LDX # < X
LDY # > X
JSR ADD; ADD STEP SIZE
LDX # < I
LDY # > I
JSR STO; SAVE LOOP VARIABLE
LDA X
BMI *+13
LDX # < N100
LDY # > N100
JSR GT
BCS *+16
BCC *+11
LDX # < N100
LDY # > N100
JSR LE
BCS *+5
JMP A
```

The variable step size check requires 25 bytes. If the step size were a constant, only 9 bytes would be

needed.

## VI. DIFFERENCES BETWEEN COMPILED AND INTERPRETED

### BASIC Commands Not Recognized by the Compiler

BYE  
CONT  
CLOAD  
CSAVE  
DOS  
ENTER  
LIST  
LOAD  
NEW  
SAVE

In addition:

RUN "FILESPEC"

is not recognized by the Compiler. RUN without a file name causes the compiled program to be rerun from the beginning.

### String Handling

The Compiler handles strings differently than the interpreter. This difference will not be noticed if sub-string assignments are not used as in:

A\$(I,J) = B\$

The Compiler uses an end of line (EOL) byte (155) to indicate an end of string. Thus, the assignment above will insert an extra EOL byte into the A\$ sub-string. If your program uses sub-strings you should re-DIMension your strings to make room for this additional byte.

As an example, suppose you want to store 12 character names in a string variable called NAMES\$ and you want to store 10 names. You should DIMension NAMES\$ to 130(i.e., 13[the string length + 1] x 10).

An added drawback of this feature is that a compiled program cannot print CHR\$(155) since the print routine quits when it sees an EOL byte. If you have to send the EOL character (i.e. CHR\$(155)) to a device, you must use PUT as in:

```
PUT #1,155 or PRINT #1
```

If you must insert substrings use:

```
FOR J=1 TO LEN (B$)
POKE ADR(A$)+I+J-1, PEEK(ADR(B$)+J-1)
NEXT J
```

instead of A\$(I,I+LEN(B\$)-1)=B\$.

## **FOR Loops**

FOR loops can have only one NEXT. For example:

```
FOR I=0 TO 10: IF X(I)=Y THEN NEXT I
PRINT X(I):NEXT I
```

is not allowed. If such a construct is in your program, you will encounter the following compile-time error message when the Compiler reaches the second "NEXT I":

```
NEXT WITHOUT FOR IN LINE XXXX
SKIPPING TO NEXT LINE
```

## **GOTO and GOSUB Variable Line Numbers**

These variable line numbers are not supported by the Compiler. For example:

```
GOSUB 1000+X
```

or

## GOTO SORT

is not permitted. The solution to the first type of GOSUB is to use:

```
ON X GOSUB 1000,1001.....
```

The solution to the second is to use the correct line number for the variable SORT.

## DATA Statements

DATA statements must always be the last statements in your program. Any executable statements located after a DATA statement will result in the compile-time error:

```
ILLEGALLY PLACED STATEMENT IN LINE XXXX
```

Also be certain to place an END, STOP, or GOTO statement before the first DATA statement. Failure to do so could result in the program trying to "execute" the DATA with unpredictable results!

## Memory Between VSEC and VEND

When a compiled program is run, all memory between VSEC and VEND (see Appendix A) is cleared to zeros, including strings. Therefore, to be safe you should initialize strings to the null string. For example, (A\$="")

## GLOSSARY

- assembler- A program which converts an assembler file to machine executable machine-code.
- assembler files- The files produced by PASS 1 of the Compiler, called ASSEM.SG(n) in this system. All assembler files are given the file name ASSEM. with the extension SG1, SG2, etc. for each successive segment.
- object file- The final output of the Compiler which is a machine-executable program.
- run-time package- A collection of machine code routines utilized by the compiled BASIC program as it is executed.
- run-time library- See run-time package.
- source file- The BASIC program which is to be compiled.
- run-time library equate file- Memory location directory for the floating point and integer run-time libraries. The equate file contains the address numbers within the run-time libraries of each particular operational function (such as multiplication, addition) performed by these files.
- machine language (code)- Language understood directly by the computer without the use of an interpreter.

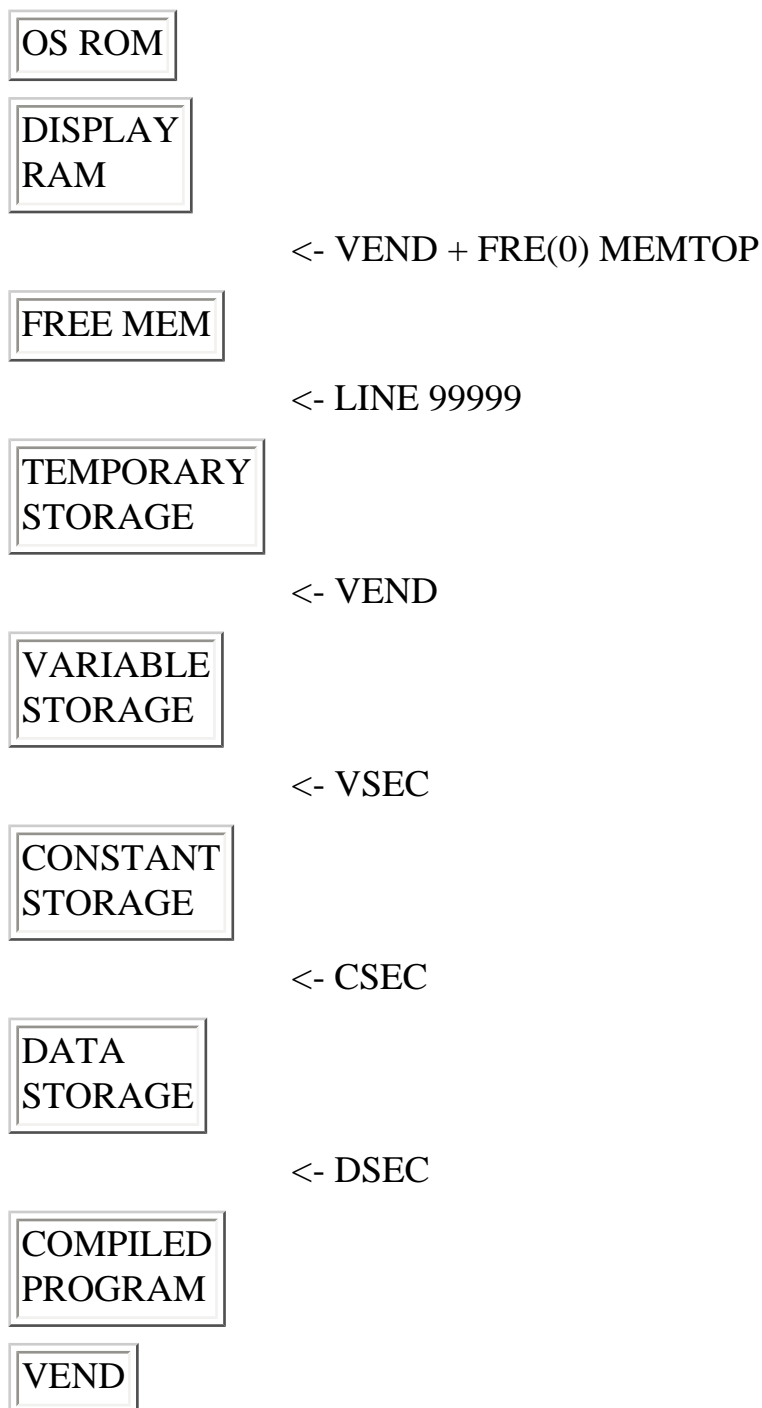
## APPENDIX A

### ATARI BASIC COMPILER Memory Map

The system library loads at \$2400; this address was chosen so that a compiled program would not overlay the RS-232 handler routines. The user code starts at \$3200 and proceeds upward.

The following diagram outlines the memory configuration at run-time of a program compiled with the ATARI BASIC Compiler.

\$FFFF



\$3206

VSEC

\$3204

CSEC

\$3202

DSEC

\$3200

RUN TIME  
PACKAGE

\$2400

DOS, SYSTEM  
WORKSPACE

\$0

**Notes:**

The pointers DSEC, CSEC, VSEC, and VEND are located at addresses \$3200, \$3202, \$3204, and, \$3206.

**APPENDIX B****Run-time Library Memory Usage**

## ZERO PAGE

HEX	DEC	DESCRIPTION
\$80	128	REGISTER SAVE AREA
\$81	129	REGISTER SAVE AREA
\$82,83	130,131	GENERAL USE POINTER
\$84	132	CURRENT COLOR FOR PLOTS
\$85	133	IOCB FOR CURRENT I/O
\$86	134	COMMAND NUMBER FOR XIO CALL
\$88,89	136,137	POINTER TO NEXT DATA STATEMENT
\$8C, 8D	140,141	STRING POINTER 1
\$90,91	144,145	STRING POINTER 2
\$92,93	146,147	STRING COUNTER 2
\$94,95	148,149	ADDRESS FOR USR CALL
\$BA, BB	186,187	STOP ADDRESS OF ERROR
\$C3	195	ERROR NUMBER
\$C9	201	PRINT TAB WIDTH
\$D4-D9	212-217	PSEUDO REGISTER 0
\$E0-E5	224-229	PSEUDO REGISTER 1
\$F2	242	FLOATING POINT USAGE

\$F3, F4	243,244	POINTER TO INPUT BUFFER
\$FB	251	RADIAN/DEGREE FLAG (0=RAD, 6=DEG)
\$FC,FD	252,253	POINTER TO FLOATING POINT NUMBER

## NON ZERO PAGE

\$480-\$4FF	1152-1279	LINE INPUT BUFFER & FILE NAME STORAGE
\$500-\$57F	1280-1407	FLOATING POINT BUFFER

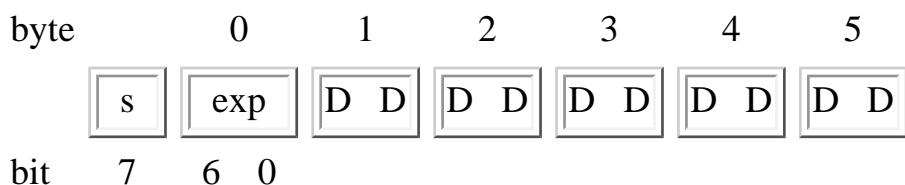
## APPENDIX C

### Internal Numeric Representation

This appendix describes the internal numeric representation used by the floating point and integer run-time packages.

#### FLOATING POINT FORMAT

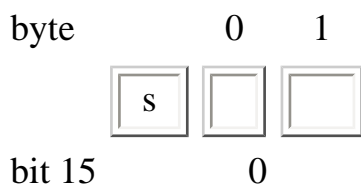
Floating point numbers are stored using the ATARI OS floating point format. Each floating point number is stored in six consecutive bytes. The sign of the number and a 64 excess power of 100 are stored in the first byte. The following five bytes contain BCD (binary coded decimal) digits, two per byte. This gives 10 digit floating point precision.



#### INTEGER FORMAT

Integers are 16 bits and stored in two consecutive bytes in memory. The bytes are stored in order of the most significant byte to the least significant byte. This is the opposite of the order in which the 6502 processor addresses bytes. This order was chosen to present a uniform location of the sign bit to the Compiler and run-time libraries, thus allowing the Compiler to produce code which is independent of the arithmetic option.

Integer representation:



## **COMMERCIAL SALE OF COMPILED PROGRAMS**

No royalty fees are required to sell programs compiled with the ATARI BASIC COMPILER. We do require that you place the following notice in your program documentation:

This program was compiled using DATASOFT'S BASIC COMPILER for the Atari.