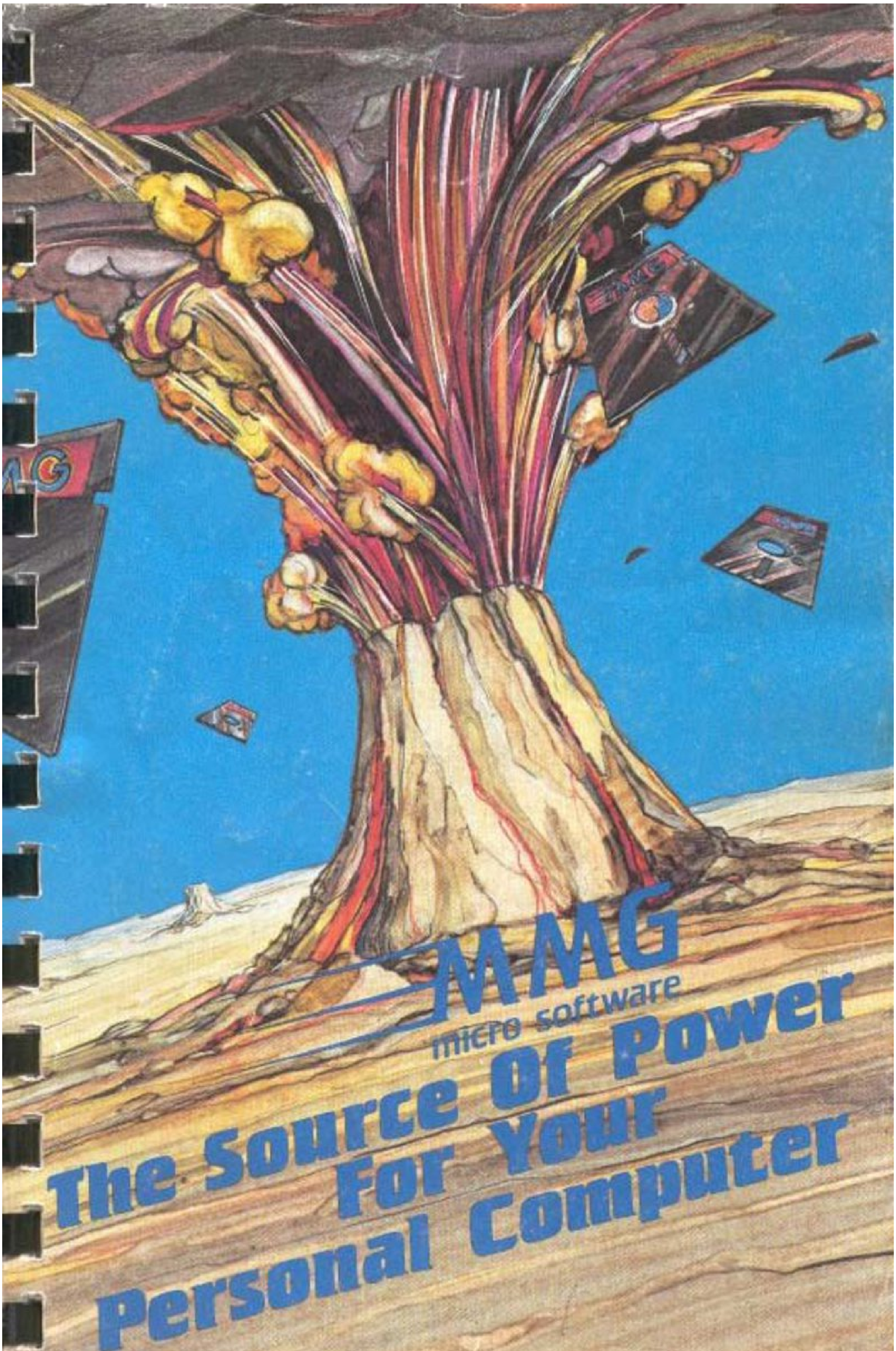

MMG
micro software

The Source Of Power For Your Personal Computer

MMG BASIC COMPILER

by Special Software System

TABLE OF CONTENTS

MMG BASIC COMPILER

INRODUCTION

Congratulations! You have in your hands one of the most powerful and versatile tools available for your ATARI computer, the MMG BASIC COMPILER. As you know BASIC is an interpreted language, and each time you run a BASIC program, your ATARI converts your program into machine language, the language of ones and zeros. However, as each line is translated and executed, the machine language code which is generated is thrown away, so each time you run your program, the translation must be done again. This results in a program which is easy to write and change, but is very slowly executed. Until now if you wanted FAST programs, you had to learn to program in assembly language.

Now you can write lightning-fast programs without having to learn a new language, with the MMG BASIC COMPILER. Write and debug your programs just as you always have, and then convert them to the fastest possible code, true machine language. No knowledge of assembly or machine language is required! Since the MMG BASIC COMPILER offers the option of using either integer or floating point arithmetic, it is well suited for both entertainment and business applications. In fact, several arcade-type games currently on the market were originally written in BASIC, and compiled using the MMG BASIC COMPILER for speed.

The MMG BASIC COMPILER is a three-pass compiler which generates true 6502 machine language code. PASS 1 converts your BASIC program into assembly language files on your disk, The next two passes assemble these files into machine language, which is then saved on your disk as a file which can be loaded and run from DOS or which can be named AUTORUN.SYS, and will then automatically run whenever that disk is booted.

Since the assembly language files produced can be stored on your disk, the advanced assembly language programmer can utilize these with other assembler files, or modify them as appropriate.

GETTING STARTED

Your MMG BASIC COMPILER disk contains the following files:

DOS.SYS
DUP.SYS
AUTORUN.SYS - Displays the title screen
CMP.OBJ - The MMG BASIC COMPILER itself, used in PASS I
ASM.OBJ - The assembler, used in PASSes 2 anti 3
SYSEQU.ABC - The System Equates library
SYSLIB.FP - The Floating Point library
SYSLIB.INT - The Integer library

The master MMG BASIC COMPILER disk has a write protect tab in place on it, and this should never be removed. Removing this write protect tab voids your warranty!

To begin using your MMG BASIC COMPILER check to be sure that all cartridges have been removed from your ATARI computer. Turn on your disk drive, and when the busy light goes out, place the MMG BASIC COMPILER disk in your drive, and securely close the door of your drive. Turn on your TV or monitor, and then turn on your ATARI computer. XL computers owners must hold down the OPTION key while turning on their computers. The program will boot, and you'll see the title page appear on your screen shortly. If you forgot to remove the BASIC cartridge or if you forgot to hold down the OPTION key of your XL computer, the message "REMOVE CARTRIDGES AND REBOOT" will appear instead of the title page, and you'll need to reboot your system.

FOR USE WITH ONE DISK DRIVE

Press the letter "D" for DOS when the title page appears. After the DOS 2.0S menu appears, remove the MMG BASIC COMPILER master disk from your drive, and insert a new disk. Format this disk using option F, and then copy all of the system files from your master disk to this new disk. This disk will not be functional without the master disk, but should be used during compilations. After you have made your backup copy, place the original master disk in your drive, and reboot the system.

FOR USE WITH TWO DISK DRIVES

After the title page appears, choose the letter "D" for DOS, and wait until the familiar DOS 2.0S menu appears. Place a blank, formatted disk in drive #2, or format one at this time. Then use the "C" option of DOS to copy each of the files from your master MMG BASIC COMPILER disk in drive #1 to the copy in drive #2.

This disk will not be functional without the master disk, but will be used during the compilation. When the copy is complete, transfer the BASIC program you want to compile onto this disk, and then reboot the system. When the title page reappears, replace your master disk with a blank, formatted disk, and place the copy you just made into drive #2.

COMPILING YOUR PROGRAM

Please note that there are several restrictions on your BASIC program (although far fewer than with any other compiler). If you have a problem compiling your program, or if your program fails to work properly once compiled, please read the remainder of this manual to discover the problem and the solution.

If you have only one disk drive, it will be necessary for you to periodically swap your backup master disk with the disk containing your BASIC program. The MMG BASIC COMPILER will prompt you each time this is necessary. Depending on the size of your BASIC program, a few of these swaps can be avoided by copying the appropriate SYSLIB file (.INT or .FP) onto the disk with your BASIC program, along with SYSEQU.ABC and ASM.OBJ. However, if your BASIC program is too large, there may not be enough room on the disk for these files, your BASIC program, and the assembler files produced, in which case, an ERROR 162-DISK FULL will occur, and you'll need to begin again. For those with two disk drives, all disk swapping has been eliminated.

At the title screen simply press "C" to load the MMG BASIC COMPILER. When it is loaded, you will be prompted to insert your BASIC program work disk, and then to type your BASIC program file name. If you want a directory of the disk in drive #1 before answering, just type a RETURN here. Otherwise, type the entire name of the BASIC program you want to compile, and press RETURN. For example, you might type:

```
D:MYPROG.BAS or D2:GAME
```

followed by a RETURN. You will then be asked for the name of the object file, which is the name you want to give the machine language program the MMG BASIC COMPILER will create for you. A good convention to follow is to name your BASIC program with an extension of .BAS, and your object file with an extension of .OBJ, but any name will do. Be sure, however, that it's different from your BASIC program' Examples are:

```
D:MYPROG.OBJ or D2:GAME.OBJ
```

followed by a RETURN. If the program you name as your BASIC program is, in fact not a BASIC program, then the MMG BASIC COMPILER will tell you "FILE NOT BASIC" and give you a chance to enter a different name.

After you enter both file names, you will be asked whether to use integer or floating point arithmetic. If you want integer, press I and RETURN; if you want floating point, choose F and RETURN. Maximum speed of execution is obtained by choosing the integer option, but this limits the arithmetic to be used. Please read the remainder of this manual for a further discussion of these choices. Once you have choosen integer or floating point arithmetic, your program will begin to be compiled.

HOW THE MMG BASIC COMPILER WORKS
FIRST PASS

As each line of your BASIC program is compiled, the MMG BASIC COMPILER prints that line number to the screen, so that you can follow the course of the compilation. At the same time, it is creating disk files named ASSEM.SGA ASSEM.SGB ASSEM.SGC, and so on. When the first pass is completed, an END-OF-PASS1 message will be written on the screen, and your program will have been entirely converted from BASIC to assembly language. At this point, we've half completed our conversion to machine language.

At this time, the message:

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

will be displayed for single drive systems. Do as instructed to continue the compilation. These and further prompts are displayed only when the MMG BASIC COMPILER cannot find the appropriate file. With two drives, you'll never see these prompts. Even with one drive, if you've transferred the files from your master disk to the disk containing your BASIC program, you won't need to swap disks either.

If the SYSTEM RESET key is pressed during the first pass of the compiler, the program will return to the beginning of PASS 1.

PASS 2 and PASS 3

After compilation to assembly language, the MMG BASIC COMPILER will load the assembler to convert these files to executable machine language. This process takes two passes, PASS 2 and PASS 3. The stage of this conversion will be printed to the screen, and the cursor will blink in the upper left corner of the screen to let you know that the assembly is proceeding.

At any time the assembler can't find a file it needs, it will print a message similar to the following to the screen:

```
/ASM/ PASS_2

Can't find file ->D:ASSEM.SGA
Please insert Correct Disk

PRESS ANY KEY TO CONTINUE
```

When you have inserted the appropriate disk, press any non-function key to continue the assembly.

Pressing the SYSTEM RESET key at any time past PASS 1 aborts the assembly process and returns control to DOS. After PASS 3 is complete, your BASIC program has been translated to machine language and saved to your disk in runnable form with the name that you originally selected. The screen will then display your three options:

Print line map

Run program

DOS

LINE REFERENCE MAP

The line reference map is a tool for programmers who want to know where the machine language code from a particular line of their BASIC program resides in memory. In addition, the line number 99999 has the address of the last memory location used by the compiled program, so it's easy to determine exactly how large the final version is. Press P if you want to see the reference map.

You may then choose to see it on your screen or printer, or to have it written to your disk. The prompt looKs like this:

```
To:
 Printer
 Screen
 Disk
```

Press the appropriate letter and RETURN. Remember that CTRL-1 will start and stop the scrolling on the display if the screen option is chosen. lf you choose disk, you will be prompted for the filename as follows:

```
DEVICE:FILENAME?->
```

Type the drive number, and give the map a name, such as:

```
D2:GAME.MAP
```

and press RETURN, and the map will be written to the disk. Once the map has been written to the screen, printer or disk, the "Select Option" prompt will reappear.

RUN PROGRAN

To run your newly compiled program, simply type R and RETURN. When it is finished running, the message "BASIC exit" will be displayed, and then the message:

```
?Run address>
```

will appear. You may now rerun the program from a specific decimal address, by typing the address and pressing RETURN, or you may rerun the entire program by simply pressing RETURN, or you can type DOS and press RETURN to return to DOS.

Note that your compiled program can also be run from DOS. Simply choose the L option of DOS, and your program will automatically start when it has completely loaded. Alternatively, you may name your compiled program AUTORUN.SYS, and if you have the DOS files on that disk, simply turning on your computer with that disk in place will cause your program to load and run, without BASIC. You may go directly to DOS after your program has been compiled by typing D from the "Select Option" prompt.

USING DOS

To save time and reduce the number of disk swaps that you'll need to do, you can transfer a number of the MMG BASIC COMPILER support files and the assembler to the disk containing your BASIC program. The only caution, mentioned above, is that if your BASIC program is large, there may not be enough room on the disk for the assembly language files and your BASIC program with these support files. The files to transfer are SYSLIB.INT if you are using integer arithmetic er SYSLIB.FP if you choose floating point, SYSEQU.ABC, and the assembler, ASM.OBJ (see page 9).

To transfer these using one disk drive, from the DOS menu choose option O (DUPLICATE FILE), and follow the prompts. Using two disk drives, it's faster to use option C (COPY FILE).

LOADING THE MMG BASIC COMPILER FROM DOS

The MMG BASIC COMPILER may be loaded using the L option of DOS. At the prompt:

```
LOAD FROM WHAT FILE?
```

type CMP.OBJ, and press RETURN. From this point on, follow the same instructions as for booting the MMG BASIC COMPILER disk.

TECHNICAL NOTES
INTEGER ARITHMETIC OPTION

In your ATARI BASIC program, all arithmetic is done using the floating point system of the ATARI. That is, numbers such as 1,245,645 or 1.2543 or 0.4689 are permitted. Integer arithmutic permits only integers, that is, whole numbers, between -32768 and 32767. Therefore, if you select integer arithmetic at compiling time, the MMG BASIC COMPILER inserts a copy of the integer run-time library into the compiled program. Since integer arithmetic by definition cannot support very large or fractional numbers, you should be aware that there are a number of BASIC commands which cannot be correctly executed using integer arithmetic. These include COS, SIN, CLOG, LOG, EXP and ATAN, all of which produce fractional numbers.

Furthermore, the BASIC command RND(0) produces a random number between 0 and 1. Integer arithmetic cannot support this range, since in integer arithmetic, the number must be either 0 or 1. To solve this Problem, the MMG BASIC COMPILER introduces a new function for random numbers. Simply insert any integer (or a variable which evaluates to an integer) within the parenthesis following the RND call, and the integer package of the MMG BASIC COMPILER will return a random integer between 0 and that number minus one. For instance, to produce a random number between 0 and 125, the statement:

```
X=RND(126)
```

will place this number in a variable called X.

Since integer arithmetic can only handle (cleverly enough) integers, please be sure that your BASIC program contains no fractional numbers, particularly constants such as 1.5, or 3.14159, or 0.25. Although your compiled program will run, these numbers will be converted to integers, and the results of your program will not be the same as the results of your BASIC program.

Note that the square root of a number, obtained with the BASIC command SQR(#), frequently results in a fractional answer. If you choose integer arithmetic, the answer obtained using this function will be the largest integer which, when squared, is equal to or less than your original number. This may NOT be an exact square root. For instance, in BASIC, the square root of 6 is 2.449499..., and this number squared is very close to 6. Using integer arithmetic, the square root of 6 becomes 2, and squaring this yields 4, a far cry from 6.

Finally, although integer arithmetic can only generate numbers between -32768 and 32767, there is a case in which you may use numbers outside of this range. This occurs with memory addresses for PEEKs and POKEs. For instance, to move player #0 harizontally, you may still POKE his horizontal Position as follows:

```
POKE 53248,XPOS
```

Similarly, to determine if any of the console buttons OPTION, SELECT or START have been pressed, you may still use the statement:

```
X=PEEK(53279)
```

You may not, however, correctly use a statement such as:

```
PRINT 53279
```

since this will produce -12257 (53279-65536).

It is important to point out that all of the limitations discussed in this section are not limitations of the MMG BASIC COMPILER, but rather are limitations of integer arithmetic. They can be avoided by simply choosing the floating point option at the time of compiling our BASIC program, if any of these functions are critical to the correct functioning of your program.

THE USE OF COM VARIABLES

The ATARI offers two types of statements to be used for dimensioning variables and strings. The first, and by far the most common, is the DIM statement. However, it also supports the COM statement and so does the MMG BASIC COMPILER.

The COM statement is idenlical in use to the DIM statement. For example, to reserve space 200 characters lang for A$, either

```
DIM A$(200) or COM A$(200)
```

would work. The major difference between these two statements comes into play when programs are chained together; that is, when the first program has a statement in it like:

```
RUN "D:PROGRAM2"
```

this case, of course, any variables dimensioned using the DIM statement would be cleared before the second program began execution.

Sometimes, however, it is useful to retain the values of a number of variables from program to program. In this case, simply include the same COM statements in both programs, and the values assigned in the first program to these variables will be retained in the second. For instance, if you want A$ and the array B to retain their values in PROGRAM2, put the following line in both programs:

```
10 COM A$(500),B(25)
```

COMPILER FILES

If you only have one disk drive, obviously all files must reside in drive #I, and the program will prompt you to swap disks at the appropriate times. If you have two disk drives, the MMG BASIC COMPILER will search both drives to find the files that it needs. However, the assembler working files will always be written to drive #1. Therefore, to maximize the use of your disk space, your BASIC and object Code files, as well as all of the system files and compiler files, should reside on drive #2, reserving drive #1 exclusively for the assembler files.

The assembler files (ASSEM.SGA, ASSEM.SGB, etc.) require approyimately five times as much disk storage as the BASIC program, although the final object code file, your runnable machine language program, should be roughly the same size as your BASIC program (not counting the run-time library). Therefore, you'll need a disk with five times as many free sectors as your BASIC program, so the largest program you can compile using only one disk drive is a little less than 120 sectors, although this depends on the nature of the program. Much larger programs have successfully been compiled on a single drive.

To maximize disk utilization, the MMG BASIC COMPILER has incorporated an optional command for retaining or deleting the assembler files as the program is assembled. If your BASIC program does NOT contain a LIST statement, then the assembler files will be deleted during PASS 3, while your object code file is being written. In other words, PASS 3 will make room on an otherwise full disk for your object code, by erasing the assembly language source code files as it is finished with each of them. If you don't need this extra space and want to retain the assembly language source code files, just include the LIST command anywhere within your BASIC program. The LIST command was chosen because it is a command which has no place in a machine language program; listing such a program will produce only garbage on the screen, instead of the normal BASIC code. A recommended approach is to add the line:

```
        32767 LIST
```

to the end of your program.

ASSEMBLY LANGUAGE PSEUDO-OPS

If you want to modify the assembly language files produced by the MMG BASIC COMPILER, you should be aware of the following statements recognized by the assembler (ASM.OBJ):

```
.END - ends the assembly
.FILE - chains two or more files in an assembly
= - defines a symbol
.BYTE - defines bytes of data stored in memory
.WORD - defines address constants
.DBYTE -defines word oriented data in memory
>-defines the high byte of a number
<-defines the low byte of a number
```

COMPILER ERRORS

These are errors that occur during the compilation of a BASIC program, due either to errors in the BASIC program itsef or due to ATARI system errors. These are to be cantrasted with errors that occur while running a compiled program, which are called run-time errors.

SYSTEM ERRORS

All system errors will be displayed with the standard ATARI error number. Please consult your BASIC or ATARI manual for a more complete description of these. In addition to these, the following system errors may also occur:

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

This means that the MMG BASIC COMPILER has encountered a bad ASM.OBJ file. To solve this Problem, recopy this file from your master disk or a backup.

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

This means that there is an internal compiler inconsistency. To solve this, reboot and recompile.

```
BAD INPUT FRON BASIC FILE
TOKEN = xxx
COMPILATION ADORTED
PRESS RETURN TO CONTINUE
```

This means that the MMG BASIC COMPILER has encountered an unexpected character in the BASIC program. This may be caused by a damaged disk. Try to execute the program from BASIC. If it works correctly, perform the following procedure:

```
LIST "D:TEMP"
NEW
ENTER "D:TEMP"
SAVE "D:FILENAME"
```

and reboot the compiler to try again.

If any of these errors occur, place any disk containing the file DUP.SYS in drive #1, and press RETURN, to relinquish control to DOS.

PROGRAMMING ERRORS

If your BASIC program has any of several errors in it (discussed below), the MMG BASIC COMPILER will display the error message, and then the lines:

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

If you press C, the compilation continues, thus allowing all possible programming errors in the BASIC program to be detected. The MMG BASIC COMPILER simply skips the statement with the error, and continues the compilation at the next BASIC statement. After compiling the whole program, the message:

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

will be displayed if any errors were detected, where X is the total number of errors. Since your program won't run anyway, the compiler stops here. Fix the BASIC program and then reboot the MMG BASIC COMPILER to recompile the corrected program.

If you press A in response to the CONTINUE OR ABORT question, you will return to the DOS menu.

COMPILE TIME ERRORS

ILLEGALLY PLACED STATEMENT

Cause: The compiler has encountered a COM Statement after non-COM statements.
Solution: Move all COM statements to lines numbered lower than all other non-COM statements.

ILLEGAL NEXT

Cause: A NEXT is trying to increment a loop variable which does not match the variable in the corresponding FOR statement, such as FOR I=1 TO 10:NEXT J.
Solution: Correct the loop variable.

DYNAMIC DIN NOT ALLOWED

Cause: A DIN or COM Statement must use constants, not variables, to allocate string and array storage. Statements such as DIM X(A) or DIM A$(X) are not allowed.
Solution: Replace the variables with constants.

NEXT WITHOUT FOR

Cause: The compiler has encountered a NEXT statement without a matching FOR Statement.
Solution: Remove the NEXT or insert an appropriate FOR.

RE-DIMENSION ERROR

Cause: A string or array is dimensioned more than once.
Solution: Dimension each string orarra, only once.

SYNTAX ERROR

Cause: There is a misspelling, a missing comma, orother error in your BASIC program.
Solution: Correct your BASIC program and recompile.

CAN'T COMPILE STATEMENT

Cause: The BASIC program contains a Statement not supported by the compiler, such as LOAD.
Solution: Remove such statements.

UNDIMENSIONED ARRAY

Cause: The compiler has encountered a statement containing an array or string before ist DIM or COM statement.
Solution: Move the DIM or COM statement to a line number lower than all lines which reference the array or string.

UNDEFINED LINE NUMBERS

Undefined line numbers are detected during PASS 3 by the assembler For example, if your BASIC program contains the following line:

```
100 GOTO 1000
```

and there is no line 1000 in your BASIC program, then the assembler will respond by displaying the incorrect assembler instruction and the line number which is undefined. For this example, the display would read:

```
    -->JMP L1000

    /ASM/ SYSTEM ERROR
    /ASM/ REF: LINE #->1000
    /ASHI UNRESOLVED LINE NUMBER
    CONTINUE (Y/N)?
```

JMP L1000 is the assembler instruction and the line number is 1000. The assembler is asking you if you want to continue the assembly. The first time you compile your program, you should continue so as to find any other errors, so type Y. The compiled program will not run correctly with these errors, so be sure to correct them and recompile before attempting to run.

GOTO/GOSUB VAR OR EXP

Cause: Your BASIC program contains a statement of the form:

GOTO A or GOSUB 1000+X

Solution: Replace these statements by the appropriate GOTO or GOSUB. Frequently, you can replace such statements by such lines:

```
ON A GOTO 1000,2000,3000
          or
ON X GOSUB 1000,1010,1020,1030
```

ASSEMBLER SYSTEM ERROR

In addition to the normal ATARI system errors, you may see the message:

```
SYSTEM ERROR: 255
```

during assembly. There are two possible causes for this error:
Cause #1: A reference is not defined in the system equates file, usually because of a damaged file SYSEQU.ABC.
Cause #2: The assembler cannot find the next assembly language source code file. This usually means that the files have been damaged since they were created during PASS 1, or that the compiler itself has been damaged.
Solution: Rerun your compilation after recopying all system files from either a backup or your master MMG BASIC COMPILER disk.

RUN TIME ERRORS

Running a compiled program may produce anyof the standard ATARI errors as runtime errors. In your compiled program, the ATARI BASIC command TRAP will work just like it does in BASIC, to assist you in debugging your program. Furthermore, PEEK(195) will return the type of error encountered, just like it does in BASIC. If an error is encountered which is not TRAPped, or if the TRAP has been reset like TRAP 40000, then the run time package in your compiled program will print the address of the incorrect instruction, and will allow you to resume execution at a given address when the run time package prompts you with:

```
?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 already discussed. Instead of typing an address, you can also enter one of the following three options:

Type RETURN to rerun the program from the beginning.

Type DOS and RETURN to return control to DOS.
Type C and RETURN to continue running the program beginning from the line where the error cccurred.

AN EXAMPLE OF A RUN TIME ERROR

The following discussion shows how to determine the line number at which a run time error occurs. You should have a listing of your BASIC program and a copy of the line reference map. We'll use the following program as an example:

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

When this program is run from BASIC, the following output is produced:

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

ERROR- 11 IN LINE 170

READY
```

The line reference map produced following the compilation of this program looks like:

```
LINE # 100 = 12811
LINE # 110 = 12811
LINE # 120 = 12811
LINE # 130 = 12811
LINE # 140 = 12111
LINE # 150 = 12811
LINE # 160 = 12811
LINE # 170 = 12825
LINE # 180 = 12849
LINE # 190 = 12882
LINE # 99999 = 12918
```

and shows that the compiled code for line 170, for example, lies between memory locations 12825 and 12848, inclusive.
Now, when we run the compiled program is executed, the following display is seen:

```
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 teils you that an error 11 was detected, 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 from our reference map, we know that line 170 contains the problem,

TABLE OF RUN TIME ERRORS

ERROR NUMBER	DEFINITION
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
168	Drive number error
161	Ton many files apen
162	Disk full
163	Unrecoverable system data I/O error
164	File number mismatch
165	File number error
166	POINT data length error
167	File locked
168	Command invalid
169	Directory full
170	File not found
171	POINT invalid

OPTIMIZING YOUR BASIC PROGRAM

TIMING CONSIDERATIONS

Many programs require timing loops, either to provide syrhronization or small pauses during portions of the program. Since compiled programs run much faster, you should change your timing parameters using the following information as a guide. Using the floating point package, your compiled program will run about three times faster than your original BASIC program, whereas il you use the integer package the difference in speed is approximately 15 fold. A better way to implement delays than using timing loops is to use the ATARI real-time clock, as in the following example.

```
100 DELAY=10:GOSUB 500
110 ...
110 ...


111 TIME=PEEK(20)+DELAY
510 IF PEEK(20)<>TIME THEN GOTO
510
520 RETURN
```

Locations 18, 19, and 20 are the ATARI real-time clock. Location 20 is updated once each jiffy (1/60 of a second). When location 20 goes from 255 to 0, location 19 is incremented by one, so location 19 counts one unit for about each 4.25 minutes. Location 18 is incremented once for each full cycIe of location 19, and so it counts one beat for about each 1083 minutes. By writing the subroutine at line 500 - 520 above to use any of these three locations, delays of virtually any duration are possible, and will be the same in BASIC or in machine language, in either integer or floating point arithmetic.

HOW TO PRODUCE SMALLER COMPILED PROGRAMS

The MMG BASIC COMPILER was designed to produce the fastest possible machine code, as opposed to the shortest, Most routines in your BASIC program will take up substantially more space, in Ihe machine language program than in the original BASIC program. One type of statement which does not take up more room in the final program is the GOSUB statement. Therefore, if you use subroutines for everything which is repetitious in your BASIC program, you'll dramatically cut down the size of the final machine language program produced.

BASIC instructions which take up large blocks of space in the compiled program include mathematical calculations such as X*Y+Z, substring expressions, such as A$(I,J), references to arrays, such as X(I), and FOR NEXT loops with a variable for the step function, such as FOR I=1 TO 100 STEP B, which takes almost 3 times as much space in the compiled program as the same statement with STEP 2, for example.

USE WITH DOUBLE DENSITY

If you have at least one true double density disk drive, you can considerably increase the size of a BASIC program to compile. This program DOES NOT support the ATARI 1050 density-and-a-half, only true double density drives. To operate in double density, simply boot up as described above, and select D from the title page, to go to DOS. Once the DOS menu appears, use the L option of DOS to load and run a program called:

CHANGE

This program will change the density of your drive(s). Just follow the prompts of the program, and turn your master disk over at the time indicated by the CHANGE program. You will then be in double density, and can load the compiler using the L option of DOS, loading CMP.OBJ from the double density side of the master disk.

COMMANDS NOT RECOGNICED BY THE COMPILER

Several BASIC keywords are not recognized by the MMG BASIC COMPILER, for obvious reasons. These are:

```
CONT
CLOAD
CSAVE
ENTER
LOAD
NEW
SAVE
```

In addition, FOR loops can have only one NEXT statememt. Finally, GOTO, GOSUB, and RESTORE cannot be followed by a variable, but must be to constant line numbers.

MEMORY MAP

The system library loads at $2400, and the user code starts at $3200 and procedes upward. The following diagram outlines the memory configuration at run time of a program compiled with the MMG BASIC COMPILER.

```
$FFFF________________
     :              :
     :   OS ROM     :
     :______________:
     :              :
     :   DISPLAY    :
     :     RAM      :
     :______________:<-VEND+FRE(0)=MEMTOP
     :              :
     :   FREE MEM   :
     :______________:<-LINE 99999
     :              :
     :  TEMPORARY   :
     :   STORAGE    :
     :______________:<-VEND
     :              :
     :STRING STORAGE:
     :______________:<-SSEC
     :   NUMERIC    :
     :     STORAGE 1   :
     :_____________ :<-VSEC
     :   CONSTANT   :
     :   STORAGE    :
     :______________:<-CSE
     :              :
     : DATA STORAGE :
     :______________:<-DSEC
     :              :
     :   COMPILED   :
     :   PROGRAM    :
     :______________:<-$320A
     :              :
     :     VEND     :
     :______________:<-$3208
     :              :
     :     SSEC     :
     :______________:<-$3206
     :              :
     :     VSEC     :
     :______________:<-$3204
     :              :
     :     CSEC     :
     :______________:<-$3202
     :              :
     :     DSEC     :
     :______________:<-$3200
     :              :
     :   RUN TIME   :
     :   PACKAGE    :
$2400:______________:
     :              :
     : DOS & SYSTEM :
     :  WORKSPACE   :
   $0:______________:
```

The compiled program entry paint is at $2400.

RUN TIME LIBRARY MEMORY USAGE

ZERO PAGE

HEX	DEC.	DESCRIPTION
$88	128	REGISTER SAVE AREA
$EI	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	148,141	STRING POINTER 1
$90,$91	144,145	STRING POINTER 2
$92,$93	146,147	ADDRESS FOR USR CALL
$96,$97	150,151	TRAP VECTOR
$98	152	TAB COUNTER
$99	153	GENERAL USE COUNTER
$9A	154	STACK POINTER SAVE
$9B	155	GENERAL USE FLAG BYTE
$BA,$BB	106,187	STOP ADDRESS OF ERROR
$C3	195	ERROR NUMBER
$C9	291	PRINT TAB WIDTH
$D4-$D9	212-217	PSEUDO REGISTER 0
111-115	214-111	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
$508-$57F	1288-1407	FLOATING POINT BUFFER

INTERNAL NUMERIC REPRESENTATION.

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 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.

COMMERCIAL SALE OF COMPILEID PROGRAMS

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

This program was compiled using the MMG BASIC COMPILER for the ATARI.

MMG Micro Software
PO Box 131
Marlboro, NJ 07746