OPTIMIZING COMPILER

for

Advan Language Designs

BASIC COMPILER

OPTIMIZING COMPILER

for

ADVAN LANGUAGE DESIGNS

BASIC COMPILER

WARNING

DISCLAIMER OF WARRANTY

THIS SOFTWARE AND MANUAL ARE SOLD "AS IS" AND WITHOUT WARRANTIES AS TO PERFORMANCE OR MERCHANTABILITY. THE SELLER'S SALESPERSONS MAY HAVE MADE STATEMENTS ABOUT THIS SOFTWARE. ANY SUCH STATEMENTS DO NOT CONSTITUTE WARRANTIES AND SHALL NOT BE RELIED ON BY THE BUYER IN DECIDING WHETHER TO PURCHASE THIS PROGRAM.

THIS PROGRAM IS SOLD WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES WHATSOEVER. BECAUSE OF THE DIVERSITY OF CONDITIONS AND HARDWARE UNDER WHICH THIS PROGRAM MAY BE USED, NO WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE IS OFFERED. THE USER IS ADVISED TO TEST THE PROGRAM THOROUGHLY BEFORE RELYING ON IT. THE USER MUST ASSUME THE ENTIRE RISK OF USING THE PROGRAM. ANY LIABILITY OF SELLER OR MANUFACTURER WILL BE LIMITED EXCLUSIVELY TO PRODUCT REPLACEMENT OR REFUND OF THE PURCHASE PRICE.

Optimizing Compiler for Advan BASIC

Introduction

You will find two Master disks for the Advan Optimizing Compiler in the center of this manual. Make sure that the write protect tabs on both disks are in place so that they can't be erased accidentally. You should put one away; it's your back up disk.

The compiler in Advan Basic generates fast, compact code, but even more speed is needed for some programs. The Optimizing Compiler speeds up the sections of program between FAST and FAST END, which are commands built into Advan BASIC. These FAST sections are usually speeded up 2 to 6 times faster than in Advan BASIC. However you don't get something for nothing. The length of these sections is also increased. The Optimizing Compiler compensates for this to some extent by optimizing and thus reducing the length of the remaining code by about 20 to 25 percent.

FAST and FAST END

The first step in using the Optimizing Compiler is to insert FAST and FAST END commands into the program. To do this you must load the program while in Advan Basic, insert these commands, and then save the program to a disk. You need to decide which sections of the code to speed up. For instance, it wouldn't make sense to speed up an INPUT command because its speed depends upon the time taken to type in the response.

Since loops are areas where a program usually spends most of its time, they should probably be the first regions to consider. For example, if you have a FOR loop you might place a FAST right in front of the FOR and FAST END right after the NEXT:

```
100 RTIME
110 FAST
120 FOR A=1 TO 30000
130  S=S+1
140 NEXT A
150 FAST END
160 PRINT S,TIME
```

If you use integers, Advan BASIC runs this program in about 15 seconds; the Optimizing Compiler reduces it to about 1.3 seconds. If you use real numbers, Advan BASIC takes about 32 seconds, and the Optimizing Compiler reduces that to about 16 seconds. ATARI BASIC needs 161 seconds to run the same program. A FOR loop at the end of a medium length program can be simulated by putting 100 REM statements at the beginning of the above program. In this case, ATARI BASIC needs 373 seconds for the loop, while the Advan times remain unchanged.

Rules governing the placement of FAST and FAST END:

(1) FAST And FAST END must be at the start of a line and there must be only one space after the line number. For example, 100 NEXT T:FAST END will give an error.

(2) If a FAST region of code (a section between FAST and FAST END) includes FOR, WHILE, REPEAT, ELSE, or IF DO, then it must also include the corresponding NEXT, WEND, UNTIL, or ENDIF. For example, the following programs will give the error message 'FAST PLACEMENT ERROR AT LINE 130':

```
100 FAST
110 FOR T=1 TO 7
120  S=S+T
130 FAST END
140 NEXT T
```

```
100 FAST
110 FOR T=1 TO 7
120  S=S+T
130 FAST END
140  C=C*T+1
150 FAST
160 NEXT T
170 FAST END
```

In the second program it is true that FOR and its accompanying NEXT are both in FAST sections of code, but they must be in the same FAST section.

(3) If a FAST region of code includes NEXT, WEND, UNTIL, or ENDIF it must also include the accompanying FOR, WHILE, REPEAT, or IF DO.

(4) If ON GOTO or ON GOSUB is in a FAST region, all the line numbers referred to in the command must be in FAST regions. However they do not have to be in the same FAST region as the ON command. Conversely, if ON GOTO or ON GOSUB is not in a FAST region, none of the referenced line numbers may be in FAST regions. For example:

```
100 FAST
110 ON T% GOSUB 200,300
120 FAST END
130 PRINT A$
140 END
190 FAST
200 A$="200":RETURN
300 A&="300":RETURN
310 FAST END
```

The above example will work even though the ON GOSUB is in a different FAST region than lines 200 and 300. If you did not include lines 190 and 310 you will get the error message 'FAST PLACEMENT ERROR AT LINE 110.'

(5) You can use GOTO or GOSUB from a FAST region to a non-FAST region and vice-versa. For example:

```
100 INPUT T
110 GOSUB 200:GOSUB 300
120 PRINT S,B:A@:END
130 FAST
200 S=T*T+1:RETURN
300 B=SQR(T):RETURN
400 SUB A@:? "DONE":SUBEND
500 FAST END
```

(6) The commands MACHINE and CODE may not be used in a FAST region.

(7) If a line number immediately follows THEN or ELSE, the line number must be in the same kind of region as the IF. Consider 100 IF A=T THEN 200. If line 100 is in a FAST region, then line 200 must also be in a FAST region. Consider 100 IF A=T THEN GOTO 200. The GOTO allows lines 100 and 200 to be in different kinds of regions.

(8) The line number referred to in a TRAP command must not be in a FAST region.

(9) The following commands run at the same speed whether or not they are in a FAST section: CLOSE, EOF, GET, GRAPHICS, INPUT, INPUTLINE, INSTR, LPRINT, LSCREEN, NOTE, OPEN, POINT, PRINT, PUT, READ. In addition, the functions ATAN, COS, EXP, LOG, SIN, SQR, TAN, and ^ will require about the same amount of time whether or not they are in a FAST section, unless you use the /F option (see p. 4).

Using the Optimizing Compiler

After you have placed FAST and FAST END you are ready to use the Optimizing Compiler. Insert the Optimizing Compiler Master disk into drive 1 and type:

```
EXEC OPCOM.COD
```

The program will load and ask you to insert the disk with the program and then enter the program name. At this time remove the Optimizing Compiler Master disk from drive 1 and insert the disk with the program you want (in this case, ALPHA). Your response is underlined.

```
Insert program disk & then
Enter program name
? ALPHA
```

The program then asks for the name of the compiled output file. You respond as follows:

```
Enter output file name
? ALPHA.COD
```

For maximum speed the screen will be blanked during the compile. Two work files are set up on the disk and then killed. Typically, they will take up 30 or fewer sectors.

When the compile is finished you will get the messages 'End Compile' and then 'INSERT BASIC DISK&RETURN'. The Optimizing Compiler is so large that part of the BASIC has to be removed and then reloaded after the compile. Remove the disk with the program, insert the Advan BASIC Master disk into drive 1, and then press RETURN. If the BASIC is on the program disk (i.e., you used FORMAT1.COD), just press RETURN. After the BASIC is reloaded the system will display Ready. You can now reinsert the disk with the program and execute the compiled code using the command EXEC ALPHA.COD.

If you have a 130XE you can put the program on the RAMDISK, or if you have a two drive system you can put the program disk in drive 2. For example:

```
EXEC OPCOM.COD

Insert program disk & then
Enter program name
?D2:BETA
Enter output file name
?D2:BETA.COD

End Compile
INSERT BASIC DISK&RETURN
Ready
```

/F Option

Advan BASIC generates code which uses the built-in ATARI floating point routines. The Optimizing Compiler can generate code which uses either the ATARI floating point routines or special high speed routines. There are special routines for floating point add, subtract, multiply, sin, cos, tan, atan, log, sqr, exp, and ^. These routines are used only in the FAST sections of code and will be about two to four times faster than the built-in ATARI routines. Using the high speed routines will increase the compiled program length since their code must be added to the end of the program. To keep the program as small as possible, only those routines actually used will be added to the end of the program. In other words if you don't use TAN, the code for it will not be added. Placing a /F at the end of the program name tells the compiler to add and use the special floating point routines. For example:

```
EXEC OPCOM.COD

Insert program disk & then
Enter program name
?GAMMA.BAS/F
Enter output file name
?GAMMA.COD
```

Other Compiler Options

Both the Advan BASIC compiler and the Optimizing Compiler provide jumps around function and subroutine definitions. Consider the following program:

```
100 A@:PRINT FNT
110 SUB A@:PRINT "A"
120 SUBEND
130 DEF FNT=5
140 END
```

When the above program is compiled, a jump to line 130 is inserted at the start of line 110 and a jump to 140 at the start of line 130. This allows the functions and subroutines to be placed anywhere in the program. Many programmers put all of their functions and subroutines at the end of the program and protect them with a GOTO around the entire group, or they put the END command before them. In either case the jumps around the functions and subroutines are not needed and just waste memory. You can

tell the Optimizing Compiler to eliminate these jumps by adding /J, /JF, or /FJ at the end of the program name. Of course if you use one of the options with an F then the special floating point routines will be added. The following sequence will compile the program named ALPHA and will not use jumps around functions and subroutines. It will, however, add the necessary special floating point routines:

EXEC OPCOM.COD

Insert program disk & then
Enter program name
?ALPHA/JF
Enter output file name
?ALPHA.COD

Another option you can specify when you compile is to remove the BASIC during program execution. Do this by following the program name with a space and then a 2. In most cases 800XL and XE owners won't need this option since it only gains about 3 or 4K of RAM. The rest of the BASIC is shifted to high RAM in these computers. 400/800 owners will gain about 17K with this option. For example:

EXEC OPCOM.COD

Insert program disk & then
Enter program name
?ALPHA/F 2
Enter output file name
?ALPHA.COD

This will compile the program named ALPHA and use the special floating point routines. When ALPHA.COD is executed the BASIC will be removed. At the completion of ALPHA.COD you will need to insert a disk with the BASIC and press RETURN.

Sample Program

As an example of using the Optimizing Compiler we will modify and compile the SIEVE.BAS program on the Advan BASIC Master disk. First insert this disk into drive 1 and type

LOAD SIEVE.BAS

Next type:

57 FAST
180 FAST END

This places FAST and FAST END into the program. Now remove the Master disk, insert one of your disks, and type:

SAVE SIEVE.FST

After the save is finished remove the disk, insert the Optimizing Compiler Master disk into drive 1, and type:

EXEC OPCOM.COD

The program will load and display:

```
Insert program disk & then
Enter program name
?
```

Remove the Optimizing Compiler Master disk, insert the disk with SIEVE.FST, and type:

SIEVE.FST

The display will read:

```
Enter output file name
?
```

Type SIEVE.COD

The screen will be blanked for about 30 seconds while the program is compiling. Then you will see:

```
End Compile
INSERT BASIC DISK&RETURN
```

If the disk with SIEVE.FST has the BASIC on it just press RETURN. Otherwise remove the disk, insert the Advan BASIC Master disk, and press RETURN. After the Ready appears reinsert the disk with the program and type:

EXEC SIEVE.COD

The program will load and ask if you wish to blank the screen. Type Y and press RETURN. The run should take a little less than two seconds and then it should print the time and the number of primes found.

Error Messages

If the input file cannot be found or the output file cannot be opened (i.e., write protected disk or file) you will get an error message saying 'NAME ERROR' and you will again be asked to give the names of input and output files.

Normally a program should be operational before you pass it through the Optimizing Compiler and so you shouldn't get syntax error messages. If this kind of error occurs, however, you will be given the line number and an error number. Unfortunately there wasn't room for written error messages. Appendix C in the Advan BASIC manual explains the error messages.

Probably the most common error message is 'FAST PLACEMENT ERROR AT LINE'. Several things cause this error and they are described in the section on FAST and FAST END (p. 2).

Reducing program size

In some cases you are more interested in reducing the length of a program rather than increasing its speed. For example, the program may be too long for the available memory or you may want to get more programs on a disk. In this case, don't insert any FAST commands into the program. Typically the Optimizing Compiler will reduce program length by about 20 to 25 percent. In addition, you might consider using the /J option.

SPECIAL NOTES

(1) While the program is executing FAST sections of code, you normally cannot use the BREAK key to stop the program. If you do press the BREAK key in a FAST section, the system will retain this and execute the break when the program leaves the FAST section.

(2) In FAST sections array subscripts and the values used in ON GOTO and ON GOSUB are not checked to verify that they are within bounds. This increases program speed, but it means that you can't use a TRAP command to check for these errors. If you need to check these bounds, you must do this as part of the program.

(3) If you get a runtime error while executing compiled code, the system will ask you to insert a disk with the program and then enter the program name. Normally the system then gives the error and the line number where the error occurred. For programs compiled with the Optimizing Compiler, the system still gives the error but the line number won't be correct because the program length has been changed.

Reporting Problems or Errors

If you should encounter a problem or error in the Optimizing Compiler or the manual, we would appreciate hearing about it. Please list the computer you were using and, if possible, a short example of a program which malfunctions. Send to:

Advan Language Designs
P.O. Box 159
Baldwin, Kansas 66006