ZZZIP COMPILER MANUAL

CONTENTS

1. INTRODUCTION

Zzzip is an Integer Basic Compiler. It converts a Basic program to machine code, so that it will run faster. The term "Integer" indicates that Zzzip achieves greater speed increases by treating the values of all constants and variables in the Basic program as integers (ignoring digits after a decimal point). There are therefore some limits to what you can do with Zzzip, but if you follow the instructions in this manual carefully, you will find that Zzzip is easy to use and the results can be quite astounding.

The obvious first question is "How much faster will compiled programs run?". The answer will depend on what sort of Basic program has been compiled. A program to sort random numbers into order might run nearly 50 times faster than Basic, while a program to sort strings might run only (!) 12 times faster. At the other end of the scale, a program which simply plotted dots or lines on the screen, without much calculation, would probably run only twice as fast as Basic (which is still a clear improvement). Much faster display methods are possible if you modify your Basic program to use the same methods as machine code programs (see the program BENCH.BAS on this tape for an example of fast display).

The programs on this tape are Zzzip (in three parts– ZIP, ZIPA, ZIPB) and BENCH.BAS, which contains some simple routines to demonstrate the speed

increase provided by Zzzip. You are recommended to compile BENCH.BAS first, to become familiar with the procedure for using Zzzip.

2. RUNNING Zzzip

Zzzip can be used equally well with tape or disk. Section 2.1 is written for tape use, but can be applied to disk use if you make allowance for the obvious differences (and note the points in section 2.2).

2.1 Tape systems

You should start from the Basic editor screen, with all other programs deleted. Press the START key to run Zzzip, which automatically loads in three sections. When ready Zzzip will prompt you to put in the tape containing the Basic program you wish to compile. Zzzip assumes you are using a remote control to your tape machine (if not, you will have to make use of the PAUSE key). Now type in the name of the Basic program (or simply press the ENTER key to load the first program found). Zzzip will load the Basic program into a reserved area of memory. Note that for ENTERPRISE 64 machines there is a limit of 16K bytes, and if the Basic program is too large, Zzzip will display "Memory Full".

Then Zzzip will prompt you to put in the tape on which you want to save the compiled version of your program, and to press the RECORD keys on your tape machine. Type in the name you wish to use for the compiled program and Zzzip will

start work. (See section 2.3 for more on program names). Zzzip makes four passes through the Basic program, and it displays the Basic line numbers so that you can monitor progress.

If Zzzip hits any problems during the first two passes (such as a Basic command that it can not handle), it will leave a message against the line number and continue with the next line until it has completed that pass. So you will have a list of any lines which have caused problems at this stage, and must check section 4 of these instructions for assistance in dealing with error messages.

Usually Zzzip will not hit any problems, and will continue to the third and fourth passes, during which it saves to tape a small Basic loader program and then the compiled program itself. There are some checks carried out during these passes, and if an error is found at this stage, Zzzip will display a message and stop immediately.

At the end Zzzip will ask if you want to compile any more programs. If so, Zzzip will return to the point at which the Basic source program is loaded.

2.2 Disk systems

First of all you will want to copy the programs ZIP, ZIPA, ZIPB and BENCH.BAS from tape to disk.

When entering the name of the Basic source program or naming the compiled program, you can include any of the normal additional information such

as drive identity or directory path. The source and compiled programs can be on different disks if required, and if you have only one drive the usual prompts to change disk will be displayed.

2.3 Program names

To ensure compatibility with disk systems, some rules are applied to program names.

If you do not supply a name for the compiled program (but just press the ENTER key), Zzzip will use the default name "Z".

The preferred format for names is a maximum of 8 characters for the main part of the name, followed by an optional suffix (full stop and up to 3 characters) to indicate the type of program. Zzzip will always limit the name of the compiled program to a maximum of 12 characters. If it finds a suffix, Zzzip will use that for the Basic loader program, and change it to ".Z" for the compiled program. If there is no suffix and the main part of the name is not longer than 8 characters, Zzzip will simply add the suffix ".Z" for the compiled program. A suggested convention for naming programs is :-

PROGNAME.BAS	(Basic version)
PROGNAME.ZIP	(Loader program)
PROGNAME.Z	(Compiled version)

3. RUNNING THE COMPILED PROGRAM

Compiled programs can be treated almost exactly as if they were Basic programs. The most important difference is that only one compiled program can be held in memory at a time and it must be program 0. You can use the RUN command in a program to cause automatic loading and running of another program (freely mixing Basic and compiled programs).

Each compiled program is preceded by a small Basic loader program, which loads the compiled program automatically and checks the size of the program, including space for variables etc. You might possibly get an "Insufficient memory" message if you have compiled a very large program. You can try resetting the machine and reloading the program, but if that does not help, you must modify the original program to reduce memory requirements.

The compiled program should now run with the same results as the original Basic program, but faster. You can use the STOP key to interrupt the program and the START key to restart it. Note that CONTINUE will not work, and you can not SAVE or LIST compiled programs.

If the unspeakable should happen, and your compiled program does not run correctly, consult section 4.11 for possible causes. However, first you should run the compiled program again, watching carefully to identify where things go wrong and thus narrow down the area of search in the Basic program.

4. DEALING WITH PROBLEMS

The only practical way to deal with problems is to go back and modify the original Basic program. Although Zzzip has been thoroughly tested, it is not impossible that you might find a bug (undocumented feature !), but you should still handle such a situation by modifying the Basic program, so that Zzzip compiles it correctly. Bear in mind also the possibility that an error may have occurred during loading of a program.

The following error messages may be generated by Zzzip:-

4.1 Memory full

While loading the Basic source program, this indicates that the Basic program is too long (limit for ENTERPRISE 64 is 16K bytes). Try the solutions suggested in section 4.2. At a later stage of compilation this message indicates that Zzzip has run out of space for storing variable names. Try reducing either the number of variables or the length of their names.

4.2 Too many labels

This means Zzzip has run out of space for storing labels (particularly line numbers). Try to reduce the number of lines by removing REM lines, by using multiple statements, or by splitting the program into two parts.

4.3 "....." not supported

Zzzip will indicate the Basic line number and the identity of the item it can not handle. The item may be a Command (see section 5) or the REF function, or it may be a variable used instead of a constant to define the size of an array or maximum length of a string (Zzzip reserves fixed areas of memory for these and does not therefore permit "dynamic dimensioning").

4.4 Syntax?

This probably indicates use of a variable with the same name as a built-in function, such as SIZE or VAL. This is permitted in some cases in Basic (although it prevents use of the corresponding built-in function), but Zzzip always assumes that you mean the function. Therefore it expects an operand in brackets, such as SIZE(X). Problems may also arise from using key-words, such as COLOUR or SCREEN, as variable names.

4.5 Garbage

This almost certainly indicates failure to load a program correctly.

4.6 Non-integer / out of range

Zzzip will indicate if a constant is not acceptable, either because it is not an integer or because it is outside the normally permitted range of -32767 to +32767.

4.7 Loop/block termination

Zzzip checks for correct termination of loops and blocks. Remember that Zzzip works through the Basic program in line order, which may not be the same order as for execution at run time. All types of loop and block can be nested but Zzzip expects opening and terminating statements to match each other in line order.

4.8 Reference not found

This indicates that reference is made to a location for which Zzzip has not generated a label. This may be due to a redundant line of Basic, containing a GOTO or other command which refers to a non-existent line. Another cause would be a statement such as RESTORE 200, where line 200 exists but does not contain DATA.

4.9 Identifier declared twice

This indicates that one or more of the variables or arrays in the Basic line have already been explicitly declared in a previous NUMERIC, STRING or DIM statement. This is most likely to occur wh local variables or arrays are explicitly declared within DEF blocks (see section 5.2). Check for any variable or array which has already been declared and change it to a unique name.

4.10 Warning – global variable

In order to assist in resolving problems with local variables in DEF lines or blocks (see section 5.2), Zzzip checks the variables used for passing parameters, such as X and Y in DEF FUNC(X,Y). Zzzip will give a warning message if a variable name has already been used in a previous line or if it has been explicitly declared in any other line. The message reminds you that the variable will be treated as global by Zzzip, but since this may not affect the running of the compiled program, the message is only a warning and compilation will continue.

4.11 Compiled program does not run correctly

The two most probable causes are the different handling of local/global variables in DEF blocks and the effects of using integer arithmetic. Section 5.2 describes the handling of variables in DEF blocks. The simple rules are to use unique names for any local variables and to avoid situations where a defined function contains an instruction to CALL itself.

Using integer arithmetic all digits after a decimal point are ignored, and all values must be in the range −32767 to +32767. These rules apply to intermediate stages of a calculation as well as to the final result. So the order in which arithmetic functions are performed can be important. For example, LET X=2/4*10 will give a zero result, and should be changed to LET X=2*10/4 to give 5 as the correct result.

Similarly you should change LET X=5*10000/2 into LET X=10000/2*5. These examples are obvious because they use constants. It may not be so obvious when you have a statement like LET X=A/B*C. In such a case you would need to consider the possible values of A, B and C, applying the rule that it is better to multiply before dividing, unless the values are so large that the intermediate result could exceed 32767.

Under certain circumstances it is permissible for a variable to have a value in the range 32768 to 65535, as the result of addition or subtraction. This feature should be used with caution, since it relies on ignoring the conventional use of the most significant bit as a sign bit. In practice it is permissible to write, for example, LET X=25000+25000, and then to use X in any of the following types of statement :-

```
LET X=X+Y   or  LET X=X-Y
POKE X,Y    or  LET Y=PEEK(X)
SPOKE S,X,Y or  LET Y = SPEEK(S,X)
CALL FUNCTION (X,Y,Z)
LET A = FUNCTION (X,Y,Z)
IF X = Y THEN... (but not < or >)
```

A practical use of this feature would be in handling memory addresses, but be sure to use only the types of statement listed above.

5. BASIC COMMANDS

Zzzip supports almost all commands which can normally be executed within a Basic program. As noted below, a few commands require care in their use or are not supported, but all others can be used freely.

5.1 ALLOCATE

The ALLOCATE command causes a Basic program to be moved upwards in memory, to make space for machine code patches. Compiled programs can not be moved and therefore a space of 255 bytes (from 1300H upwards) has been reserved for patches. Note that the size of this space is not affected by the ALLOCATE command. In practice more than 255 bytes can be used for patches, at the risk of encroaching on string-handling space, which is used from address 17FDH downwards. So you may be able to include patches up to about 1K bytes in a program which does not involve manipulation of long strings. The best advice is to compile the program and see if it works !

Note that compiled programs do not need the routine commonly used to fix a bug in the ALLOCATE command in version 2.0 of EXOS. If included, this routine should check for VERNUM = 2, so that it is disabled in compiled programs (in which the function VERNUM returns the value 1).

5.2 DEF

Zzzip will accept most DEF commands, to define either single lines or blocks as functions, but there are some limitations. The most important point is that Zzzip treats all variables as global. One reason for this is that Zzzip works through a Basic program in line number order, which will probably not be the order in which lines are executed at run time. Therefore it can not always determine from history whether a variable is supposed to be global or local. If you want to make variables local within a DEF block, the simple way is to use unique names. However you should note that this will not make Zzzip treat the variables as local to each run-time execution of a DEF block, and therefore you can not use DEF blocks recursively. To explain this point, suppose that a block headed by the statement DEF FUNC(X) contains the statement CALL FUNC(Y). Basic would preserve the local value of X each time the function called itself, and would subsequently restore these successive values of X as the function unwound itself. Zzzip would not preserve these local values, and would therefore not give the correct result in such a situation. So do not allow functions to CALL themselves.

An exception to the "globals only" rule is that if a variable within a DEF block has the same name as the block itself, it may be used in a LET statement and Zzzip will then treat it as a local variable. This enables a DEF block to return a value to the main program. Take as an example the following function :-

```
1000 DEF TEN
1010   LET TEN=10
1020 END DEF
```

Within the main program the statement PRINT TEN will now cause the number 10 to be printed. Line 1010 contains a simple LET statement and Zzzip will treat TEN here as a local variable. However using the name TEN in any other type of statement in line 1010 would cause a recursive CALL of the function TEN, and this would not work correctly in a compiled program.

Similarly you can use a local variable to return a value from a function which includes dummy variables for passing parameters. For example, LET DOUBLE = X*2 is permitted within a block headed by DEF DOUBLE(X), and PRINT DOUBLE(3) in the main program would cause 6 to be printed. Also simple function names can be passed as parameters to be used within another function, so that PRINT DOUBLE(TEN) in the main program would cause 20 to be printed. These examples will work correctly in a compiled program.

Note that Zzzip does not support the REF statement and will give an error message. You will have to find another way of getting the same results in your Basic program.

5.3 PROGRAM

This command has no effect in a compiled program and will be treated as REM by Zzzip.

5.4 RESTORE

If a line number is given (as in RESTORE 200), it must be a line containing DATA. Otherwise Zzzip will give an error message.

5.5 RUN

This command can be used to restart a compiled program (from the beginning or from a specified line number). It can also be used to load and run another program (Basic or compiled), but it is not possible to pass parameters from one program to another.

5.6 STOP

As a command within a program, STOP is treated the same as END, causing exit to Basic. You can use the STOP key to interrupt a compiled program, just as in Basic, but you will not be able to CONTINUE a compiled program from the point of interruption. You will have to START from the beginning again.

5.7 EXCEPTION handling

By its very nature an EXCEPTION means that something has gone wrong, and therefore it could be difficult to handle in a compiled program. Nevertheless Zzzip does attempt to provide a useful capability for EXCEPTION handling. There should not be any problem during compilation, as Zzzip accepts all associated commands, including CONTINUE and

RETRY. The potential problems arise at run time. Situations generated by a CAUSE EXCEPTION command should give no trouble, but if you try to CONTINUE after an EXCEPTION arising from another source, such as an error check in EXOS, then results are not guaranteed !

There are two particular limitations imposed by Zzzip. Only one level of EXCEPTION handling is supported, or in other words, nesting of WHEN EXCEPTION commands is not effective and any secondary exception will result in a return to Basic with an error message. Also the RETRY command is a bit different – instead of going back to the line in which the EXCEPTION occurred, the compiled program will go back to the last WHEN EXCEPTION command. Clearly you can arrange for this to give the same result by putting the WHEN EXCEPTION line immediately before the line to which you want to return (for example, an INPUT line).

5.8 CHAIN, IMAGE, PRINT USING, TRACE, TYPE

These commands are not supported by Zzzip and will cause an error message during compilation.

6. BUILT-IN FUNCTIONS

Zzzip supports all the built-in functions. although in some cases there are limitations on their use, and these are noted below. You may be surprised to see that Zzzip supports trigonometric functions such as COS and SIN, which are not really compatible with integer maths. However these functions are more difficult to use and should be left alone until you have gained some experience with Zzzip.

6.1 BIN(X)

For values of X which do not exceed 32767 (in other words 0 to 11111) this function can be used as normal, with a variable or a constant as the operand in brackets. For greater values the operand must be expressed as a constant, for example BIN(11010101), which Zzzip treats as a special case.

6.2 CEIL(X), INT(X), IP(X), ROUND(X,N), TRUNCATE(X,N)

Because fractions are ignored in integer maths. these functions have no practical effect and simply return the value X.

6.3 EPS(X)

This function always returns the value 1. which is the smallest unit of change in integer maths.

6.4 EXLINE

In a compiled program no record is kept of the number of the line being executed and therefore this function always returns the value 0.

6.5 FP(X)

Since fractions are ignored in integer maths, this function always returns the value 0.

6.6 FREE

This function returns the number of bytes available for use by the compiled program, as you would expect. However the value will appear to be negative if it exceeds 32767.

6.7 INF

This function always returns the value 32767, which is the highest value in signed 16 bit binary notation.

6.8 PI

This function always returns the value 3.

6.9 RGB

This function normally requires the primary colours to be defined as a series of values between 0 and 1. As a special case Zzzip will accept such values provided that they are expressed as constants in decimal form. That is to say RGB(0..4,1) is acceptable, but RGB(0.3/7.7/7) or RGB(X,Y,Z) would not be.

Note that Zzzip only uses the first digit after the decimal point, and therefore RGB(0..42..99) would be treated as RGB(0..4,.9). 6.10 RND and RND(X)

RND(X) operates exactly the same as in Basic. RND by itself always returns the value 0, but as a special case RND*X will give the same result as RND(X).

6.11 VERNUM and VER$

The VERNUM function always returns the value 1, to avoid confusion with versions 2.0 and 2.1 of EXOS (in integer maths 2.0 and 2.1 would both become 2). However the VER$ function will return a string which indicates the version of EXOS actually being used.

6.12 Trigonometric and Logarithmic functions

Functions such as COS and SIN are not really compatible with integer maths, because their values usually lie between 0 and 1. In order to make provision for these functions, scaling factors are used. These are described in detail for COS, as an example of the principle, and then a complete table is given for all affected functions.

In order to obtain useful reults, the value returned by COS(X) is automatically scaled up (multiplied) by a factor of 1000, so that it will be an integer in the range 0 to 1000 (instead of 0 to 1). You can then use this result in your program, but you must remember to divide by 1000 at some convenient

point to get the correct answer. Obviously the point at which you divide by 1000 will have to be chosen carefully, and must be after the value of COS(X) has been multiplied by some other value, to avoid reducing the result to zero. This extra step (to divide by 1000) must be added after normal testing of the Basic program but before compilation. The extra step would give the wrong result if tested in Basic.

If you are using DEGREES to define angles, the operand X in COS(X) can be the same as in the original Basic program. But if you are using RADIANS, then X must be scaled up (multiplied) by 1000 before COS(X) is used. Again the point in the program where this is done must be chosen carefully to ensure a meaningful value. It is no good letting the value of X fall below 1 and then multiplying by 1000, as the resulting value of X will always be zero !

Function	Input scaling factor	Output scaling factor
ACOS) ASIN) ATN)	1000	1 (degrees) or 1000 (radians)
COS) COT) CSC) SEC) SIN) TAN)	1 (degrees) or 1000 (radians)	1000
DEG	1000	1
RAD	1	1000
EXP	1000	1
LOG) LOG2) LOG10)	1	1000
COSH) SINH) TANH)	1000	1000
ANGLE(X,Y)	1	1 (degrees) or 1000 (radians)

Note that some trigonometric and logarithmic functions may give results greater than 32767 (particularly when scaled up by 1000). In such cases the output will automatically be limited to 32767. This applies also to the power function (^).

7. COPYING COMPILED PROGRAMS

Compiled programs can not be simply copied to tape using the SAVE command. One method of making further copies would of course be to compile the Basic program again. Another method, requiring only one tape machine, uses a facility which is included in the Basic loader program preceding each compiled program. Starting from the Basic editor screen with all previous programs deleted. proceed as follows :-

1) Insert the tape containing the compiled program and prepare to load.
2) Type in LOAD "XXX" (where XXX is the name of the program).
3) Insert the destination tape (on which you wish to make a copy) and prepare to save.
4) Type in SAVE "XXX"
5) Insert the tape containing the compiled program and prepare to load.
6) Type in RUN 100
7) The program will prompt you when you should insert the destination tape and prepare to save.
8) Then press the ENTER key and the compiled program will be written out to the destination tape.