Laser Compiler

Schneider CPC
464/664/6128

Amstrad CPC
464/664/6128

Amstrad CPC
464/664/6128

ocean

IQ

LASER BASIC COMPILER

by OASIS SOFTWARE

COPYRIGHT NOTICE

THIS MANUAL

Piracy has reached epidemic proportions and it is with regret that we are forced to reproduce this manual in a form which cannot be photocopied. Our apologies for any inconvenience this may cause to our genuine customers. A reward will be paid for information leading to the successful prosecution of parties infringing this Copyright Notice.

NOTE

This manual is essential for the use of Laser BASIC Compiler. For this reason we would warn customers to look after it very carefully, as separate manuals will not be issued under any circumstances whatsoever.

ENQUIRIES

If you have any queries on the use of Laser BASIC Compiler, please send them to us in a letter, ensuring you enclose the Enquiry Card printed on the last page of this manual. A new card will be returned to you with your reply. Please note that enquiries not accompanied by the card will not be answered.

CONTENTS

The Laser BASIC Compiler
by Vinay Sajip

INTRODUCTION

The Laser BASIC Compiler was written as a companion product to the Laser BASIC extension, but since it compiles a full integer subset of the resident Amstrad BASIC, it will also be useful to programmers not using Laser BASIC. The operation of the compiler has been designed to optimise the size of the programs which can be compiled. For this reason the compiler is loaded in two passes and the run-time libraries are transferred during compilation. This means that tape users will have to endure the inconvenience of a fair amount of tape swapping. We feel the benefits outweigh the drawbacks but would welcome any comments from you.

The code produced is exclusively integer and so none of the Amstrad's floating point features can be compiled. A full list of exceptions is given in a later section. The virtue of an integer compiler is that the code produced is very fast and very compact. The compiler produces pure Z80 code and not P-code - this again leads to faster execution.

If a Laser BASIC program is being compiled then the Laser BASIC run-time code will need to be linked with the compiled code and this procedure is outlined in a later section. This is the only form in which compiled Laser BASIC programs can be reproduced and sold. Under no circumstances whatsoever can any of the Laser BASIC interpreter itself be reproduced other than for back-up purposes.

The compiled code will execute RSX's (BAR commands) if they are loaded and logged on by the compiled code. If you intend to market software which executes RSX's then these will obviously need to be saved with your compiled code. If the code you are saving is covered by a copyright notice then be sure you have permission from the publishers. As previously mentioned, Oasis do not give permission for the Laser BASIC interpreter to be used in this manner, but since the run-time code is supplied with the package there is no need to do this anyway.

TAPE MAP

The distribution tape contains the following files:

SIDE A:	(i)	COMP. BAS	The BASIC loader for the compiler
	(ii)	COMP.BIN	The syntax pass
	(iii)	COMP2. BIN	The code generation pass
	(iv)	BRTS.BIN	The BASIC run-time code
	(v)	LASE.BIN	The Laser BASIC run-time code
	(vi)	XFR.BIN	The file transfer utility
SIDE B:	(i)	DEMO.BIN	The compiled demo program

OPERATING INSTRUCTIONS

1. The first step is to load the BASIC program you wish to compile and then re-save it as an ASCII file. If, for example, the program you wished to compile were called "DEMO" then you could type:

```
LOAD"DEMO
```

Now insert a new tape (disk users may wish to insert a different disk) and type:

```
SAVE"DEMO",A
```

Note the ',A' following the filename. This tells the system to save the file in ASCII format. You will notice that the ASCII file is larger than the normal BASIC file, so make sure the tape is long enough or that you have sufficient room on your destination disk. Disk users should note that the destination disk should also have enough space to temporarily store BRTS.BIN and LASE.BIN.

2. Reset the machine (press SHIFT/CTRL/ESC) and then execute the compiler using the following procedure:

Tape users: Insert the compiler tape and rewind to the beginning of Side A if you have not already done so and type:

```
RUN"
```

Disk users: Insert the compiler disk and type:

```
RUN"COMP
```

The compiler will now load and execute

3. The compiler now issues the prompt:

```
"Tape or Disk (T/D)"
```

Tape users should press "T" and disk users should press "D". If you have selected "D" for disk the compiler will now load BRTS.BIN and LASE.BIN and issue the prompt:

```
"Insert destination disk"
```

Insert the destination disk and press any key. BRTS.BIN and LASE.BIN will now be transferred.

NOTE: It is probably a good idea to use the same disk for the ASCII source file and the object output.

The compiler now prompts for the name of the source file which is to be compiled. In our example the file was called "DEMO".

Tape users: Insert the tape on which you saved the ASCII version of the "DEMO" and rewind to the start of the file. Now enter the filename which in this case is "DEMO".

Disk users: Insert the disk on which you saved the ASCII version of the "DEMO" and then enter the filename which in this case is "DEMO".

4. The compiler will now load the ASCII file from tape or disk and carry out a syntax checking pass. This also converts the source file into an internal format and checks the syntax. If any errors are found, an error message is printed with a rough indication of where the error occurred. All source lines are printed on the screen as they are read, so any error messages will appear immediately after the offending line. When an error is found, the program waits for a key press before continuing.

If any syntax errors occur, compilation is aborted and the compiler prompts for the name of a new source file. Enter another filename, or type CTRL-C (or CTRL-SHIFT-ESC) to reset the machine and modify the source program which caused the errors.

If no errors occurred, the compiler loads the code generation pass. This is on the compiler tape/disk and so the program prompts you to enter the correct tape/disc. The syntax and code generation passes are loaded separately to save memory and allow larger programs to be compiled. The drawback to this method is that if errors occur during the code generation pass then the compiler itself will need to be reloaded before a second attempt at compilation. We felt the gain in performance offset the inconvenience of use.

The compiler then begins the code generation pass.

Tape users: The first part of the code generation phase involves the loading and outputting of BRTS.BIN., the run-time code. During this phase you will need to alternate between the compiler tape (prompted by "press PLAY then any key") and the destination tape (prompted by "press REC and PLAY then any key").

Disk users: Insert the destination disk.

The code generation pass proper then begins and the object code is written to the destination tape/disk. If errors occur during this pass, compilation is aborted and the output file is abandoned. You will need to reload the compiler before another attempt. If compilation is successfully completed then the length of the compiled code is printed. Again, line numbers are printed to assist in debugging.

Tape users: If you are compiling a Laser BASIC program then the final phase of compilation involves the loading and outputting of LASE.BIN, the Laser BASIC run-time code. During this phase you will again need to alternate between the compiler tape (prompted by "press PLAY then any key") and the destination tape (prompted by "press REC and PLAY then any key").

Your program should now be compiled. The filename of the output file will be the same as the source file but will have an extension of .BIN. For our example the output file would be called "DEMO.BIN". The compiled program will always be at least 7k or so in length because the compiler's run-time code is always saved with it regardless of file size. Before moving on to see how the compiled program is executed we'll consider compile-time errors which may be generated during the compilation of your source file.

COMPILE TIME ERRORS

There are three types of error: usage, syntactic and semantic. Syntactic errors occur during the first pass and are mostly caused by missing operands, keywords or punctuation. Semantic errors are those which specify meaningless operations (e.g. multiplying two strings). These are normally caught in the second pass.

Syntactic Errors

Most of these occur because a symbol which was expected, was not found. These errors are signified by an indication of the symbol expected. What follows is not an exhaustive list, but is indicative of what might be found.

'(' EXPECTED
')' EXPECTED
'GOSUB' EXPECTED
'=' EXPECTED
'TO' EXPECTED
END OF STATEMENT EXPECTED
INTEGER EXPECTED
VARIABLE EXPECTED
IDENTIFIER EXPECTED
TOO MANY ARGUMENTS
TOO FEW ARGUMENTS
COMMAND EXPECTED
LINE NUMBER EXPECTED
BOOLEAN EXPRESSION EXPECTED

Other examples are due to errors in program constructs, e.g.

FOR/NEXT MISMATCH
BAD STATEMENT
BAD PRIMARY EXPRESSION
BAD SUBEXPRESSION

Still others are found when reading keywords or punctuation, e.g.

ILLEGAL CHARACTER

Usage Errors

There are errors due to memory, disk or tape errors, or errors due to incompatible file type, e.g.

INPUT FILE MUST BE ASCII – please try again
NEED MORE SPACE!! (compiler workspace exhausted)

Semantic Errors

There errors are usually due to type mismatches or bad array subscripting, e.g.

ILLEGAL OPERATION ON STRING ATTEMPTED
BAD ARRAY INDEX

Array index errors are caused by mismatches between the dimensions of an array declaration and its usage. The compiler makes no checks on subscript values, even if known at compile-time. The compiler does not regard the use of uninitialised variables as an error.

EXECUTING COMPILED PROGRAMS

Programs not using Laser BASIC Commands

If the program you have compiled does not contain any Laser BASIC extended commands then it can be executed as it is.

Tape users: Insert the tape containing the compiled code and rewind if you have not already done so. Then type RUN" and the code will load and execute.

Disk users: Insert the disk containing the compiled code and type RUN"filename (in our example RUN"DEMO). The code will load and execute. Note that if DEMO.BAS is on the same disk as DEMO.BIN, it will be executed and not the compiled version.

Programs using Laser BASIC Commands

If the compiler detects any Laser BASIC commands during the syntax checking, then a flag is set to tell the system that LASE.BIN (the Laser BASIC run-time code) needs to be linked to the compiled code. The run-time library is 12k in length and the whole library is linked. This means that compiled Laser BASIC programs are at least this size when compiled. In fact the run-time code is saved as part of the compiled code and the first thing the compiled demo does when executed is to block-move the library up to &7600 which is its current execution address.

If your program does use Laser BASIC commands then the compiler will automatically prompt you to put in the tape/disk which contains the LASE.BIN file (side A – straight after the BASIC run-time code).

In practise your master will probably contain two files because your program will probably load sprites at some stage (see the section on loading sprites into a compiled program). Let's consider the steps to make a final master of a compiled Laser BASIC program. Assume the program is called "DEMO" and uses a set of sprites saved as "DEMOSPR". We'll also assume that the sprites may have been saved using the Laser BASIC interpreter and therefore nominally reside from &6FFF downwards.

Tape users will use 4 tapes – the Laser Compiler tape, the tape with the source file (the program to be compiled), the tape with the sprite files and an empty tape for the final master. Disk users will probably use several disks in addition to the compiler distribution disk.

(i) The first step is to modify the program so that sprites are loaded to the higher address of the run-time code. This is achieved by replacing the SSPR,SMAX,&7000 by SSPR,SMAX,&7600 and then replacing all GSPR commands by MSPR commands (see the section on loading sprites from compiled programs).

(ii) The next step is to modify the use of the MEMORY command (see the section "RESTRICTIONS ON THE BASIC TO BE COMPILED" which details the use of the MEMORY command).

(iii) If you have used the RND function you would also need to modify the syntax (see the section "RESTRICTIONS ON THE BASIC TO BE COMPILED").

(iv) Once the above modifications have been carried out, the file is re-saved as an ASCII file and compiled. The compiled program is the first file on the master tape/disk. If your compiled program loads sprites then do not rewind the master tape but instead carry out step (v).

(v) Reset the machine by pressing SHIFT/CTRL/ESC. Load and execute Laser BASIC and then load the required sprites using GSPR (in our example A$="DEMOSPR":|GSPR,@A$). Now insert the master and save the sprites.

The tape/disk now contains the final master which can be executed by typing RUN"filename (in our case RUN"DEMO).

NOTE: Laser Basic RSX's take up a lot of memory during the compilation phase and whenever possible a GOSUB to a line containing the RSX will save space and allow larger programs to be compiled.

Running the Compiled Demo

On Side B of the distribution tape is a compiled version of the Laser BASIC demo. You may be surprised to find that the execution speeds do not seem to be much faster than their uncompiled equivalents. There are several reasons for this.

1. Laser BASIC is dedicated to fast graphics and since much of the processor time is spent executing graphics commands, compilation has little effect on programs which do not contain logic. Your own programs will probably contain a lot of logic and will therefore be noticeably speeded up.

2. Most of the animation is metred by synchronising to frame-flyback.

3. Much of the code is executing under interrupt.

NOTE: The compiled demonstration program on Side B of the distribution tape is exactly the same as that supplied with the Laser BASIC package except that the ISET commands were replaced by IGETs and the RND and MEMORY commands were modified.

EXAMPLE PROGRAMS

What you don't see is the enormous increase in the speeds with which the game logic is executed. In order to give you an idea of just how much faster program logic will run when compiled, here are two example programs for you to compile and run.

The sieve of Eratosthenes (approximate speed increase: 30 to 1)

```
100 REM
110 REM Sieve of Eratosthenes.
120 REM
130 DIM prime%(5000)
140 start=TIME
150 FOR i%=2 TO 2500
160 IF prime%(i%)=1 THEN 215
170 j%=i%+i%
180 WHILE j%<=5000
190 prime%(j%)=1
200 j%=j%+i%
210 WEND
215 REM target for GOTO
220 NEXT i%
230 calcstook=(TIME-start)/3
240 start=TIME
250 FOR i%=2 TO 5000
260 IF prime%(i%)=0 THEN PRINT i%,
270 NEXT i%
280 printstook=(TIME-start)/3
290 PRINT
300 PRINT:PRINT "Calculations took ";USING "###";calcstook;
310 PRINT " hundredths of a sec"
320 PRINT:PRINT "Printing took ";USING "###";printstook;
330 PRINT " hundredths of a sec"
340 END
```

Circle Drawing (approximate speed increase 5 to 1)

```
100 REM
110 REM Circle drawing demonstration.
120 REM
130 DEFINT a-z
140 CLS
150 WINDOW 1,80,23,25
160 INPUT "Circle centre ";xc,yc
170 INPUT "Circle radius ";radius
180 ORIGIN xc,yc
190 GOSUB 260
200 GOTO 160
210 REM
220 REM Draw the circle
230 REM The graphics origin is expected to be at the centre of the
240 REM circle and the variable "radius" to hold the radius.
250 REM
260 x=0:y=radius
270 delta=3-radius-radius
280 WHILE x<y
290 GOSUB 390
300 delta=delta+x+x+x+x+6
310 IF delta>=0 THEN delta=delta-y-y-y+4:y=y-1
320 x=x+1
330 WEND
340 IF x=y THEN GOSUB 390
350 RETURN
360 REM
370 REM Plot one point in each of the eight octants.
380 REM
390 PLOT x,y:PLOT -x,y:PLOT -x,-y:PLOT x,-y:PLOT y,x:PLOT -y,x
400 PLOT -y,-x: PLOT y,-x:RETURN
```

RESTRICTIONS ON THE BASIC TO BE COMPILED

The Laser BASIC Compiler is compatible with a fairly complete integer implementation of locomotive BASIC - none of the floating point functions are supported. Immediate commands are not supported and neither are commands and functions which relate to the BASIC interpreter's allocation of memory to program and variables. The MEMORY function is still supported, so that other programs (notably RSX's and Laser BASIC sprites) can reside in memory with the compiled user program.

Specific keyword exceptions are:

ATN	AUTO	CHAIN	CINT	CLEAR	CONT	COS	CREAL
DEFREAL	DEG	DELETE	EDIT	ERASE	ERR	ERL	EXP
FRE	HIMEM	LIST	LOG	LOG10	MERGE	NEW	PI
RAD	RENUM	RESUME	ROUND	SIN	TAN	TRON	TROFF

Laser BASIC exceptions are:

ADDR	ISET

There are also a number of commands and functions whose operation is slightly different when compiled:

LOAD BASIC files cannot be loaded. Binary files are loaded directly to their destination address if one is specified. This means that the 2k buffer in only required if a binary file is loaded without specifying the load address. For this reason you should always specify the load address if it is known. No checks are performed during loading so care must be taken not to overwrite any reserved areas of memory.

END This just executes a RET instruction and on compilation produces the same code as RETURN. This means that if your program terminates correctly with an END then control will return to command level.

FIX FIX will pass the compiler syntax checking but will have no effect.

INT INT will pass the compiler syntax checking but will have no effect.

CINT CINT will pass the compiler syntax checking but will have no effect.

MEMORY The MEMORY command dictates the highest memory location that the compiled code can use. The compiler will require space for dynamic variables and string processing. The default value for the top of memory when a program not using Laser BASIC is compiled will be &A4FF and the default value for the top of memory when a program using Laser BASIC is compiled will be

&75FF. Note that the MEMORY command and the Laser BASIC equivalent MSET should be used to set memory protection as high as sprites (when loaded) will allow before compilation. Make sure you take into account any dynamically allocated sprite space! In order to find where sprites start, load them into Laser BASIC using the GSPR command. The start of sprites can now be found by typing:

```
START=0:|DEEK,&7004,@START
```

START has now been set to the position that sprite data starts from when run under the interpreter. When sprites are loaded into compiled code, however, they start &600 higher. You should now type:

```
START=START+&600
```

START now points to the revised start address. If your program is going to dynamically create some sprites then you will need to calculate the memory they require (each sprite requires HGTxLEN bytes) and subtract this from START. Suppose in our example we were to dynamically create 3 sprites with dimensions 3x4, 5x7 and 6x8, you would type:

```
START=START-3*4-5*7-6*8:PRINT START
```

The only other memory consideration now is vertical scrolling with wrap. If you require to carry out any vertical scrolling then further space will need to be allocated by further reducing START. Buffer space is discussed in the Laser BASIC manual under the section on Sprite Utilities. If you're unsure as to your vertical scrolling requirement it's a good idea to reduce START by 256 bytes or so. The value that you now have for START should be the value that you use for MSET in the program to be compiled. Your program should also do a MEMORY to START-1, as soon as possible.

DEFINT DEFINT and DEFSTR are fully implemented. DEFREAL is not, however. It is, however, a good idea to start all your BASIC programs with the command DEFINT A-Z so that your interpreted program will execute in more or less the same way as your compiled program – see the section on integer arithmetic. Variables are assumed to be integers unless they end with a '$' or are declared using DEFSTR.

RND RND(N) will produce a value in the range 0 to N-1, e.g. PRINT RND(6) would print a value in the range 0 to 5.

TIME This works in a very similar manner to the normal TIME function, but this time the result is stored as a 16 bit number. This means that the clock will overflow and re-commence counting from zero every 3½ *minutes (approx). Also note that since integers are treated as 'two's complement' numbers (see Integer Arithmetic) the value will appear to go negative when TIME*=32768. If you use the TIME function be sure to use it with care.

USING PRINT USING works in exactly the same wasy as interpreted BASIC except that "." and fields following it are ignored.

GOTO This works in the same way as the normal GOTO but compiled programs should not goto lines containing 'WEND' or 'NEXT', instead a 'REM' line should be substituted, preceeding the line containing the 'WEND' or 'NEXT' statement. The 'REM' line is the line to GOTO. See example program The Sieve of Eratosthenes

UNT This has no effect since all numbers are treated as two's complement integers by default.

INPUT The full Amstrad editor is not implemented but the delete key is used to delete charaters which have been wrongly entered.

.5 Amstrad BASIC will round up remainders of .5 or more when carrying out integer arithmetic, whereas compiled BASIC always rounds remainders down (truncates).

SUM	Amstrad Integer Result	Compiled Result
7/8	1	0
4/8	1	0
3/8	0	0
15/8	2	1
12/8	2	1
11/8	1	1

In order to simulate compiled programs in interpreted BASIC you should replace all "/" (divisions) by "\" (backslashes), this will force Amstrad's BASIC to truncate in the same manner as compiled BASIC.

BREAKS Compiled programs do not poll the break key and so cannot be broken out of. ON BREAK GOSUB is not implemented and Laser BASIC's background execution cannot be terminated.

Floating Point Functions The compiler does not expect to find floating point functions such as COS (5) and these will therefore be treated as variables or arrays.

Statement	Action.
PRINT COS (5)	Will put 0 (COS probably hasn't been assigned), followed by 5.
PRINT COS(5)	Will give "END OF STATEMENT EXPECTED" error.

Arrays	Array arguments are not checked for range in the compiled code, whereas they are in interpreted BASIC. If an array were dimensioned with 100 elements, interpreted BASIC would generate an error if the 101st element were used whereas the compiled code would not. If an attempt is made to read from an element that is out of range the result will be garbage. If an attempt is made to assign a value to an element that is out of range then the results are unpredictable.
↑	Exponentiation is not supported.
DEFINT	DEFINT is only accepted in the form DEFINT A–Z. Individual variable names will cause an 'unexpected character' error.
SOUND	Parameters are not checked for range and the result of errors is unpredictable.

LOADING AND SAVING SPRITES FROM A COMPILED PROGRAM

When Laser BASIC saves a file of sprites it also saves some system variables which dictate the position they will be loaded back into with the GSPR command. This means that sprites saved from the Laser BASIC extended interpreter or the sprite generator would load back in to reside at &6FFF downward whereas the compiled program requires them to reside at &75FF downward. This can be overcome by setting sprite space to the higher address - using SSPR,SMAX,&7600 and then merging the sprites with an MSPR instead of a GSPR-

It is very important to note that SSPR,SMAX,&7000 should always be replaced by SSPR,SMAX,&7600 before compilation regardless of whether sprites are to be loaded and saved. Failure to do this may not cause the system to crash but will waste 1.5k of space.

If your program uses more than one GSPR command then you will need to repeat the SSPR,SMAX,&7600. The use of the MSPR command, however, is unaffected. Sprites saved from a compiled program (using PSPR) will need to be loaded back into the uncompiled program using the same technique, i.e. SSPR,SMAX,&7000 and then MSPR.

INTEGER ARITHMETIC

Arithmetic carried out by the compiler is exclusively integer. If the program you are compiling begins with a DEFINT A-Z then the compiled program will behave in more or less the same way as the interpreted program. If you are familiar with the potential pitfalls of integer arithmetic then move on to the next section, but if not, consider the following example.

```
PRINT 5*4/2,5/2*4
```

will produce

```
 10          10
```

when run under the interpreter.

The same statement would produce

10 8

when compiled. The reason for this lies in the fact that interpreted BASIC stores a floating point intermediate result when dividing (regardless of the DEFINT) whereas the compiled code uses an integer intermediate result.

`5*4/2` The 5 is first multiplied by the 4 to give 20, and then divided by 2 to give 10. This works under the interpreter or the compiler.

`5/2*4` The 5 is first divided by the 2 to give 2.5 and 2 in the interpreted and compiled programs respectively. These results are them multiplied by 4 to give 10 and 8 respectively.

If your compiled program does not behave in the same way as your interpreted program, this is almost invariably the cause and can usually be cured by re-ordering the terms of the arithmetic expression. As a general rule, carry out all division as the final part (to the right) of any sum.

Two's Complement Notation

The Laser Compiler uses 'two's complement' notation so if any expression is PRINTed it will yield a result in the range -32768 to 32767. Assignments are allowed with values in the range -32768 to 65535. If an expression is printed using HEX$ then it will be in the range 0 to &FFFF.

The Amstrad's resident BASIC does not allow integer variables to be assigned to values greated than 32767 if they are in a decimal BASE, i.e.

`X%=32768`	will generate an overflow
`X%=&8000`	however, will not generate an overflow
`X%=&8000:PRINT X%`	will print -32768 because 32768 is actually -32768 in two's complement notation.

Compiled BASIC's integer variables behave in more or less the same manner as the Amstrad's integer variables except that assignments to decimal values greater than 32767 are legal.

USE OF VARIABLES

There are a couple of small points to note concerning the compiler's use of variables.

(i) When the compiled program is first entered the variable space may contain garbage and so all variables should be assigned a value before being used. When a variable is first declared in BASIC it will be initialised to contain 0. If your compiled program does not execute in the same manner as your interpreted equivalent then this is a good area to start looking for problems.

(ii) Try to avoid using too many string constants as the compiler does not carry out a 'garbage collection'. If, for instance, B$ is to contain the same string as A$ then use B$=A$ rather than assigning B$ to the same string.

(iii) If an array is used in interpreted BASIC before it has been declared in a dimension statement then it is automatically set up as a 10 element array. Compiled programs, however, require that all arrays must be dimensioned before being used otherwise a BAD ARRAY INDEX error will be issued.

BACKING UP COMPILED PROGRAMS

Compiled programs begin at &40. The length of the code is reported at the end of compilation. In order to make a backup of the compiled code you will need to use the file transfer utility which tape users will find as the last file on side A of the distribution tape. To execute, just type RUN"XFR.BIN and the program will load and execute with the following prompts:

ENTER INPUT FILE	You should now type the name of the file you wish to back-up, insert the source tape/disk and then press ENTER.
INSERT DESTINATION TAPE/DISK	You should now insert the tape or disk that you wish the file to be saved to and then press any key.

RSX'S AND BACKGROUND ROMS

The compiler and its workspace occupy all the available Amstrad RAM – it is therefore not possible to have any RSX's present in the machine while the compiler runs.

Compiled programs are ordinary machine code programs so when they are run, the firmware resets the machine to its EMS (Early Morning Startup) state. Thus if any RSX's are used in the program, they must be loaded and logged on from within the program itself. The same applies to any background ROMs which may be used by the program. Laser BASIC uses the RST #30 and will therefore be incompatible with some RSX's.

RUN-TIME ERRORS

- Div by zero
- String too long
- Bad argument to function
- String space exhausted

All will cause a message to be displayed (default). If an ON ERROR is set, it is executed. Otherwise, pressing a key will cause the execution to continue.

N.B. RESUME will be compiled as a RET.

MEMORY MAP

COMPILER:

&40	→	&7BFF	Workspace
&7C00	→	&A700	Compiler code and data

COMPILED PROGRAMS:

&40	→	&1BFF	Run-time support code (approximate)
&1C00	→	?	User program code
?	→	??	User data
??	→	???	String and dynamic workspace
???	→	MEMORY–1	Final workspace for run-time code
MEMORY	→	ROM WS	Not used (could be Laser BASIC run-time code and sprites or progam-loaded RSXs)

COMPILED LASER BASIC PROGRAMS:

&40	→	&1BFF	Run-time support code (approximate)
&1C00	→	?	User program code
?	→	??	User data
??	→	???	String and dynamic workspace
???	→	MEMORY–1	Fixed workspace for run-time code
MEMORY	→	&75FF	Sprites and Laser BASIC workspace
&7600	→	&A4FF	Laser BASIC run-time code
&A500	→	ROM WS	Not used

COMPILATEUR LASER

par OASIS SOFTWARE

NOTE COPYRIGHT

LE PRESENT MANUEL

Les actes de contrefaçon ont atteint des proportions astronomiques et c'est avec regret que nous sommes dans l'obligation de produire ce manuel sous une forme qui ne peut être photocopiée. Nous prions nos véritables clients d'accepter toutes nos excuses pour le dérangement que cela peut occasionner. Il sera offert une récompense pour toute information conduisant à la poursuite, conclue avec succès, de parties enfreignant cette note copyright.

AVERTISSEMENT

Ce manuel est essentiel pour l'usage du **Compilateur** . Pour cette raison, nous tenons à avertir les clients d'en prendre extrêmement soin car, en aucun cas, il ne sera fourni un autre manuel.

TABLE DES MATIERES

Le compilateur Laser BASIC
par Vinay Sajip

INTRODUCTION

Le compilateur Laser BASIC fut écrit dans le but d'accompagner l'extension Laser BASIC, mais puisqu'il compile un sous-ensemble entier complet de l'Amstrad BASIC résident, il servira également aux programmeurs ne faisant pas appel à Laser BASIC. L'exploitation du compilateur a été conçue pour rendre optimum la taille des programmes qui peuvent être compilés. Pour cette raison, le compilateur se charge en deux défilements et les bibliothèques d'exécution sont transférées au cours de la compilation. Cela signifie que les utilisateurs de bande auront assez fréquement recours à l'opération fastidieuse de changement de bandes. Nous pensons qu'au vu des avantages et des inconvénients la balance penche plutôt du côté des avantages. Nous serions prêts à recueillir tout commentaire que vous pourriez faire à ce sujet.

Le code est produit exclusivement en nombres entiers si bien qu'aucune des fonctions en virgule flottante Amstrad ne peut être compilée. Vous trouverez plus loin une section donnant une liste complète des exceptions. L'avantage d'un compilateur en nombres entiers est que le code produit est très rapide et très compact. Le compilateur génére un code Z80 pur et non pas un code P, conduisant là encore à une exécution plus rapide.

Si l'on effectue la compilation d'un programme Laser BASIC le code Laser BASIC d'exécution devra être relié au code compilé, procédure décrite dans une section ultérieure. C'est la seule forme sous laquelle les programmes Laser BASIC compilés peuvent être reproduits et vendus. Le programme d'interprétation Laser BASIC lui-même ne peut être reproduit en aucun cas hormis à des fins de sauvegarde.

Le code compilé exécutera les RSXs (instructions BAR) si ceux-ci sont chargés et sont mis en contact avec le système. Si vous avez l'intention de commercialiser du logiciel qui exécute des RSXs, ceux-ci devront à l'évidence être sauvegardés avec votre code compilé. Si le code que vous sauvegardez est couvert par un avis copyright, assurez-vous que vous avez l'autorisation des éditeurs. Comme il a été mentionné auparavant, Oasis n'octroie pas d'autorisation permettant un tel usage, mais comme le code d'exécution est fourni en standard une telle démarche n'est de toute manière pas nécessaire.

TOPOGRAPHIE DE BANDE

la bande de distribution contient les fichiers suivant:

FACE A:	(i)	COMP.BAS	Le chargeur BASIC pour le compilateur
	(ii)	COMP.BIN	Le défilement syntaxique
	(iii)	COMP2.BIN	Le défilement de génération de code
	(iv)	BRTS.BIN	Le code d'exécution BASIC
	(v)	LASE.BIN	Le code d'exécution Laser BASIC
	(vi)	XFR.BIN	L'utilitaire de transfert de fichier
FACE B:	(i)	DEMO.BIN	Le programme de démonstration compilé

CONSIGNES D'EXPLOITATION

1. La première étape consiste à charger le programme BASIC que vous souhaitez compiler et ensuite le re-sauvegarder en tant que fichier ASCII. Si, par exemple, le programme que vous désirez compiler s'appelait "DEMO", vous pourriez taper:

   LOAD"DEMO

Introduisez maintenant une nouvelle bande (les utilisateurs de disques peuvent vouloir introduire un nouveau disque) et tapez:

SAVE"DEMO",A

Remarquez le ',A' à la suite du nom de fichier. Ce symbole a pour objet de commander au système la sauvegarde du fichier sous format ASCII. Vous constaterez que le fichier ASCII est plus grand que le fichier normal BASIC si bien que vous devez vous assurer que la bande est suffisamment longue ou que vous avez assez d'espace sur votre disque de destination. Les utilisateurs de disque doivent également veiller à ce que le disque de destination comporte suffisamment d'espace pour mettre temporairement en mémoire BRTS.BIN et LASE.BIN.

2. Remettez la machine à l'état initial (en appuyant sur SHIFT/CTRL/ESC) et exécutez ensuite le programme compilateur de la façon suivante:

Utilisateurs de bande: Introduisez la bande compilatrice et rebobinez jusqu'au début de la Face A si cela n'a pas déjà été fait et tapez:

RUN"

Utilisateurs de disque: Introduisez le disque compilateur et tapez:

RUN"COMP

Le compilateur sera alors chargé et exécuté.

3. Le compilateur émettra ensuite le message suivant:

"Tape or Disk (T/D)" ("Bande ou Disque (B/D)")

Les utilisateurs de bande doivent appuyer sur la touche "T" tandis que les utilisateurs de disque doivent appuyer sur la touche "D". Si vous avez sélectionné la touche "D" pour le disque, le compilateur chargera alors les codes BRTS.BIN et LASE.BIN puis émettra le message suivant:

"Insert destination disk" ("Introduisez le disque de destination")

Introduisez alors le disque de destination et appuyer sur n'importe quelle touche. BRTS.BIN et LASE.BIN seront désormais transférés.

REMARQUE: Il est sans doute bon d'utiliser le même disque pour le fichier source ASCII et le fichier sortie objet.

Le compilateur demande ensuite le nom du fichier source à compiler. Dans notre exemple, le fichier s'appelait "DEMO".

Utilisateurs de bande: Introduisez la bande sur laquelle vous avez sauvegardé la version ASCII de "DEMO" et rebobinez jusqu'au début du fichier. Introduisez maintenant le nom du fichier qui dans ce cas est "DEMO".

Utilisateurs de disque: Introduisez le disque sur lequel vous avez sauvegardé la version ASCII de "DEMO" puis introduisez le nom du fichier qui dans ce cas est "DEMO".

4. Le compilateur chargera alors le fichier ASCII se trouvant sur bande ou sur disque et effectuera un passage de vérification syntaxique. Ce passage convertit aussi le fichier source dans un format interne et vérifie la syntaxe. Si une erreur quelconque est détectée, un message d'erreur est affiché avec une indication grossière de l'endroit où l'erreur s'est produite. Toutes les lignes sources sont affichées sur l'écran telles qu'elles sont lues si bien que tout message d'erreur apparaîtra immédiatement après la ligne erronée. Lorsqu'une erreur est détectée, le programme attend une dépression de touche avant de continuer.

S'il se produit une erreur de syntaxe quelconque, l'exécution de la compilation est suspendue et le compilateur demande le nom d'un nouveau fichier source. Introduisez un autre nom de fichier, ou bien tapez CTRL-C (ou CTRL-SHIFT-ESC) pour remettre la machine à l'état initial et modifier le programme source qui a causé les erreurs.

S'il n'y aucune erreur, le compilateur chargera le défilement de génération de code qui se trouve sur la bande compilatrice/le disque compilateur. C'est pourquoi le programme vous demande d'introduire la bande/le disque correct(e). Les défilements syntaxique et de génération de code sont chargés séparément afin d'économiser de l'espace en mémoire et également pour permettre la compilation de plus gros programmes. Le désavantage de cette méthode est que s'il se produit des erreurs au cours du défilement de génération de code, le compilateur lui-même devra être rechargé avant de faire une deuxième tentative de compilation. Nous avons pensé que les avantages gagnés sur le plan de la performance compensent les inconvénients d'utilisation.

Le compilateur commence alors le défilement de génération de code.

Utilisateurs de bande: La première partie de la phase de génération de code comprend le chargement et la mise en sortie de BRTS.BIN, le code d'exécution. Au cours de cette phase vous devrez utiliser en bascule la bande compilatrice (demandée par le message "press PLAY then any key": "appuyez sur PLAY puis sur n'importe quelle touche") et la bande de destination (demandée par le message "press REC and PLAY then any key": "appuyez sur REC et sur PLAY puis sur n'importe quelle touche").

Utilisateurs de disque: Introduisez le disque de destination.

La génération de code proprement dite commence alors et le code objet est écrit sur la bande/le disque de destination. Si, au cours de ce défilement, il y a des erreurs il est mis un terme à la compilation et le fichier sortie est abandonné. Il vous faudra recharger le compilateur avant de faire une autre tentative. Si la compilation s'effectue bien, la longueur du code compilé sera alors affichée. Là encore, pour faciliter la mise au point les numéros de ligne sont affichés.

Utilisateurs de bande: Si vous effectuez la compilation d'un programme Laser BASIC, la phase finale de compilation comprend le chargement et la mise en sortie de LASE.BIN, le code d'exécution Laser BASIC. Au cours de cette phase, vous devrez encore utiliser en bascule la bande compilatrice (demandée par le message "press PLAY then any key": "appuyez sur PLAY puis sur n'importe quelle touche") et la bande de destination (demandée par "press REC and PLAY then any key": "appuyez sur REC et PLAY puis sur n'importe quelle touche").

Votre programme doit désormais être compilé. Le nom de fichier du fichier objet sera identique à celui du fichier source mais comprendra l'extension .BIN. Dans notre exemple, le fichier objet s'appellerait "DEMO.BIN". La longueur du programme compilé sera toujours d'à peu près 7k quelque soit la taille du fichier, ceci parce que le code d'exécution du compilateur est toujours sauvegardé avec le programme compilé. Avant d'examiner la manière dont le programme compilé est exécuté, nous nous attarderons sur la question des erreurs à la compilation pouvant être générées lors de la compilation de votre fichier source.

ERREURS A LA COMPILATION

Il existe trois types d'erreurs: les erreurs d'utilisation, les erreurs syntaxiques et les erreurs sémantiques. Les erreurs syntaxiques se produisent au premier passage; elles sont le plus souvent causées par des opérandes, des mots-clés ou une ponctuation manquants. Les erreurs sémantiques sont celles qui se rapportent aux opérations sans signification (par ex. la multiplication de deux chaînes). Elles sont normalement détectées au cours du deuxième passage.

Erreurs syntaxiques

La plupart se produisent parce qu'un symbole attendu est introuvable. Ces erreurs sont signifiées par une indication de symbole attendu. A titre d'indication, voici une liste non exhaustive de ce que l'on peut trouver:

'(' EXPECTED
')' EXPECTED
'GOSUB' EXPECTED
'=' EXPECTED
'TO' EXPECTED
END OF STATEMENT EXPECTED
INTEGER EXPECTED
VARIABLE EXPECTED
IDENTIFIER EXPECTED
TOO MANY ARGUMENTS
TOO FEW ARGUMENTS
COMMAND EXPECTED
LINE NUMBER EXPECTED
BOOLEAN EXPRESSION EXPECTED

On trouve d'autres messages à la suite d'erreurs dans les structures de programme, par exemple:

FOR/NEXT MISMATCH
BAD STATEMENT
BAD PRIMARY EXPRESSION
BAD SUBEXPRESSION

On en rencontre encore d'autres lors de la lecture de mots-clés ou de la ponctuation, par exemple:

ILLEGAL CHARACTER

Erreurs d'utilisation

Elles sont dues aux erreurs en mémoires, sur disque ou sur bande, ou bien au type de fichier incompatible, par exemple:

INPUT FILE MUST BE ASCII – please try again
NEED MORE SPACE!! (espace de travail du compilateur épuisé)

Erreurs sémantiques

Ces erreurs sont en général le résultat de discordance de frappe ou bien d'un mauvais indiçage de tableau, par exemple:

ILLEGAL OPERATION ON STRING ATTEMPTED
BAD ARRAY INDEX

Les erreurs d'indiçage de tableau sont dues à des discordances entre les dimensions d'une déclaration de tableau et son utilisation. Le compilateur n'effectue aucune vérification sur les valeurs d'indice, même si elles sont connues au moment de la compilation. Le compilateur ne considère pas l'utilisation de variables non-initialisées comme une erreur.

EXECUTION DE PROGRAMMES COMPILES

Programmes ne faisant pas appel à des instructions Laser BASIC

Si le programme que vous avez compilé ne contient aucune instruction étendue en Laser BASIC, il peut alors être exécuté de la façon suivante:

Utilisateurs de bandes: Introduisez la bande contenant le code compilé et rebobinez si vous ne l'avez pas déjà fait. Tapez alors RUN" et le code se chargera et sera exécuté.

Utilisateurs de disques: Introduisez le disque contenant le code compilé et tapez RUN" nom de fichier (dans notre cas RUN"DEMO). Le code se chargera et sera exécuté. Notez que si DEMO.BAS est sur le même disque que DEMO.BIN, celui-ci sera exécuté mais pas la version compilée.

Programmes faisant appel à des instructions Laser BASIC

Si, au cours de la phase de vérification syntaxique, le compilateur détecte une instruction en Laser BASIC, un indicateur apparaîtra pour signaler au système que LASE.BIN (le code d'exécution Laser BASIC) doit être relié au code compilé. La bibliothèque d'exécution est longue de 12k et la bibliothèque entière est reliée. Cela signifie que les programmes Laser BASIC compilés sont une fois compilés au moins de cette taille. En fait le code d'exécution est sauvegardé en tant que partie du code compilé et la première chose que le programme de démonstration effectue lorsqu'il est exécuté est un transfert de bloc de la bibliothèque jusqu'à &7600 qui est son adresse d'exécution actuelle.

Si votre programme utilise bien des instructions Laser BASIC, le compilateur vous demandera automatiquement de mettre la bande/le disque qui contient le fichier LASE.BIN (Face A – tout de suite après le code d'exécution BASIC).

En pratique votre programme principal contiendra sans doute deux fichiers car celui-ci, à un moment donné, chargera probablement des symboles graphiques (voir la section sur le chargement des symboles graphiques dans un programme compilé). Voyons maintenant les étapes à suivre pour faire un programme principal d'un programme Laser BASIC compilé. Supposons que le programme s'appelle "DEMO" et qu'il utilise un ensemble de symboles graphiques sauvegardés en tant que "DEMOSPR". Nous supposerons également que ces symboles graphiques ont pu être sauvegardés à l'aide de l'interpréteur Laser BASIC et que par conséquent ils résident nominalement à partir de &6FFF en descendant.

Les utilisateurs de bandes utiliseront 4 bandes – la bande compilatrice Laser, la bande avec le fichier source (le programme à compiler), la bande avec les fichiers de symboles graphiques et une bande vierge pour le programme principal final. Les utilisateurs de disques utiliseront probablement plusieurs disques en plus du disque compilateur de distribution.

(i) la première étape consiste à modifier le programme de sorte que les symboles graphiques soient chargés à une adresse plus élevée du code d'exécution. Cette opération s'effectue en remplaçant les adresses SSPR,SMAX,&7000 par SSPR,SMAX,&7600 et en remplaçant toutes les instructions GSPR par des instructions MSPR (voir la section sur le chargement des symboles graphiques à partir des programmes compilés).

(ii) L'étape suivante consiste à modifier l'utilisation de l'instruction MEMORY (voir la section "RESTRICTIONS SUR LE BASIC A COMPILER" qui explique l'utilisation de l'instruction MEMORY).

(iii) Si vous avez utilisé une fonctions RND, vous aurez également besoin de modifier la syntaxe (voir la section "RESTRICTIONS SUR LE BASIC A COMPILER").

(iv) Une fois les modifications ci-dessus effectuées, le fichier est à nouveau sauvegardé en tant que fichier ASCII et compilé. Le programme compilé est le premier fichier sur la bande/le disque d'exploitation. Si votre programme compilé charge des symboles graphiques, ne rebobinez pas la bande d'exploitation mais procédez par contre à l'étape (v).

(v) Remettez la machine à l'état initial en appuyant sur SHIFT/CTRL/ESC. Chargez et exécutez le Laser BASIC et chargez ensuite les symboles graphiques à l'aide de GSPR (dans notre exemple A$="DEMOSPR":|GSPR@A$). Introduisez maintenant la bande d'exploitation et sauvegardez les symboles graphiques.

La bande/disque contient maintenant le programme principal final qui peut être exécuté en tapant RUN"nom de fichier (dans notre cas RUN"DEMO).

REMARQUE: Les RSXs Laser BASIC utilisent beaucoup d'espace mémoire au cours de la phase de compilation et lorsque c'est possible le recours à un GOSUB à une ligne contenant le RSX économisera de l'espace et permettra la compilation de plus gros programmes.

Passage du programme de démonstration compilé

Sur la face B de la bande de distribution se trouve une version compilée du programme de démonstration Laser BASIC. Il se peut que vous soyez surpris par les vitesses d'exécution qui ne semblent pas être beaucoup plus rapide que leurs équivalents non-compilés. Il y a plusieurs raisons à cela.

1. Laser BASIC se spécialise dans le traitement graphique et comme la plupart du temps utilisé par le processeur est passé à l'exécution des instructions de graphique, la compilation a peu d'incidence sur les programmes n'ayant pas de logique. Vos propres programmes comprendront probablement beaucoup de logique et leurs vitesses seront donc visiblement augmentées.
2. La plupart de l'animation est mesurée par synchronisation au retour du faisceau électronique dans le cadre.
3. La plupart du code est exécuté sous interruption.

REMARQUE: Le programme de démonstration compilé se trouvant sur la face B de la bande de distribution est exactement le même que celui fourni avec l'ensemble Laser BASIC si ce n'est que les instructions ISET ont été remplacées par des IGETs et que les instructions RND et MEMORY ont été modifiées.

PROGRAMMES TYPES

Ce que vous ne voyez pas c'est la forte augmentation dans les vitesses avec lesquelles la logique de jeu est exécutée. Afin de vous donner une idée de la rapidité à laquelle la logique de programme est exécutée une fois compilée, voici deux programmes types à compiler et à passer par vous.

Le crible d'Erathostène (augmentation de vitesse approximative 30:1)

```
100 REM
110 REM Crible d'Erathostène
120 REM
130 DIM prime%(5000)
140 start=TIME
150 FOR i%=2 TO 2500
160 IF prime%(i%)=1 THEN 215
170 j%=i%+i%
180 WHILE j%<=5000
190 prime%(j%)=1
200 j%=j%+i%
210 WEND
215 REM cible pour GOTO
220 NEXT i%
230 calcstook=(TIME-start)/3
240 start=TIME
250 FOR i%=2 TO 5000
260 IF prime%(i%)=0 THEN PRINT i%,
270 NEXT i%
280 printstook=(TIME-start)/3
290 PRINT
300 PRINT:PRINT "Les calculs ont pris  "; USING "###";calcstook;
310 PRINT " centièmes de seconde"
320 PRINT:PRINT "L'impression a pris  "; USING "###";printstook;
330 PRINT " centièmes de seconde"
340 END
```

Dessin de cercle (augmentation de vitesse approximative 5:1)

```
100 REM
110 REM Démonstration de dessin de cercle
120 REM
130 DEFINT a-z
140 CLS
150 WINDOW 1,80,23,25
160 INPUT "Centre du cercle ";xc,yc
170 INPUT "Rayon du cercle ";radius
180 ORIGIN xc,yc
190 GOSUB 260
200 GOTO 160
210 REM
220 REM Dessiner le cercle
230 REM   Cette routine s'attend à ce que l'origine de la graphique soit au centre du cercle
240 REM   requis, et que la variable "rayon" contienne le rayon voulu.
250 REM
260 x=0:y=radius
270 delta=3-radius-radius
280 WHILE x<y
290 GOSUB 390
300 delta=delta+x+x+x+x+6
310 IF delta>=0 THEN delta=delta-y-y-y-y+4:y=y-1
320 x=x+1
330 WEND
340 IF x=y THEN GOSUB 390
350 RETURN
360 REM
370 REM Tracer un point dans chacun des huit octants
380 REM
390 PLOT x,y:PLOT -x,y:PLOT -x,-y:PLOT x,-y:PLOT y,x:PLOT -y,x:PLOT
    -y,-x:PLOT y,-x:RETURN
```

RESTRICTIONS SUR LE BASIC A COMPILER

Le compilateur Laser BASIC est compatible avec une assez complète version en nombres entiers de locomotive BASIC – aucune des fonctions en virgule flottante n'est prise en charge. Les instructions à opérande direct ne peuvent être reçues, pas plus que les instructions et les fonctions qui se rapportent à l'affectation de mémoire au programme et aux variables de l'interpréteur BASIC. La fonction MEMORY est toujours prise en charge, si bien que les autres programmes (en particulier les RSXs et les symboles graphiques Laser BASIC) peuvent résider en mémoire avec le programme de l'utilisateur compilé.

Les exceptions de mot-clé spécifique sont:

ATN	AUTO	CHAIN	CINT	CLEAR	CONT	COS	CREAL
DEFREAL	DEG	DELETE	EDIT	ERASE	ERR	ERL	EXP
FRE	HIMEM	LIST	LOG	LOG10	MERGE	NEW	PI
RAD	RENUM	RESUME	ROUND	SIN	TAN	TRON	TROFF

Les exceptions Laser BASIC sont:

ADDR ISET

Il existe aussi un certain nombre d'instructions et de fonctions dont l'exécution est quelque peu différente lorsqu'elles sont compilées:

LOAD Les fichiers BASIC ne peuvent pas être chargés. Les fichiers binaires sont chargés directement dans leur adresse de destination s'il en existe une de spécifier. Cela signifie que la mémoire tampon de 2k n'est requise que dans le cas où un fichier binaire est chargé sans spécifier l'adresse de chargement. Pour cette raison, si l'adresse de chargement est connue il faut la spécifier. Aucune vérification n'est effectuée au cours du chargement si bien qu'il faut prendre bien soin de ne pas recouvrir une zone de mémoire réservée.

END Cette fonction exécute simplement une instruction RET et à la compilation elle produit le même code que RETURN. Cela signifie que si votre programme s'achève correctement avec un END le contrôle retournera au niveau de commande.

FIX FIX passera l'opération de vérification syntaxique du compilateur mais n'aura aucun effet.

INT INT passera l'opération de vérification syntaxique du compilateur mais n'aura aucun effet.

CINT CINT passera l'opération de vérification syntaxique du compilateur mais n'aura aucun effet.

MEMORY L'instruction MEMORY prescrit l'emplacement mémoire le plus élevé que le code compilé peut utiliser. Le compilateur aura besoin d'espace pour les variables dynamiques et le traitement de chaînes. La valeur implicite pour le haut de la mémoire lorsqu'un programme ne faisant pas appel à Laser BASIC est compilé, est &A4FF et la valeur implicite pour le haut de la mémoire lorsqu'un programme faisant appel à Laser BASIC est compilé, est de &75FF. Remarquez que l'instruction MEMORY et l'équivalent Laser BASIC MSET doivent être utilisées pour établir une protection de la mémoire aussi élevée que les symboles graphiques (une fois chargées) le permettront avant la compilation. Assurez-vous que vous prenez en compte tout espace de symbole graphique affecté dynamiquement! Afin de savoir où commencent les symboles graphiques, chargez-les dans Laser BASIC à l'aide de l'instruction GSPR. Le début des symboles graphiques peut alors être trouvé en tapant:

START=0:|DEEK,&7004,@START

START a maintenant été positionné à l'endroit à partir duquel les données de symboles graphiques commencent lorsqu'elles sont passées à l'exécution de l'interpréteur. Cependant, lorsque les symboles graphiques sont chargés dans le code compilé, ils commencent à une adresse &600 plus élevée. Il vous faut alors taper:

START=START+&600

START désigne alors l'adresse de commencement révisée. Si votre programme va créer dynamiquement quelques symboles graphiques, il vous faudra calculer l'espace mémoire dont ils auront besoin (chaque symbole graphique a besoin de HGTxLEN octets) et le soustraire à START. Supposez que dans notre exemple, nous allions créer dynamiquement 3 octets de dimensions 3×4, 5×7 et 6×8, vous taperiez:

START=START–3*4–5*7–6*8:PRINT START

La seule autre considération de mémoire qui reste est le défilement vertical avec bouclage. S'il vous faut effectuer un défilement vertical, davantage d'espace devra êtra affecté en réduisant encore START. L'espace tampon est traité dans le manuel Laser BASIC à la section sur les utilitaires de symboles graphiques. Si vous n'êtes pas certain de votre besoin en défilement vertical il est bon de réduire START de 256 octets à peu près. La valeur que vous avez maintenant pour START devrait être celle que vous utilisez pour MSET dans le programme à compiler. Votre programme devrait aussi effectuer une MEMORY à START–1 dès que possible.

DEFINT DEFINT et DEFSTR sont totalement mis en oeuvre. Cependant, DEFREAL ne l'est pas. Il est pourtant bon de commencer tous vos programmes BASIC avec l'instruction DEFINT A–Z si bien que votre programme interprété sera exécuté plus ou moins de la même façon que votre programme compilé – voir la section sur le calcul en nombres entiers. Les variables sont supposées être des nombres entiers à moins qu'elles finissent avec un '$' ou qu'elles soient déclarées à l'aide de DEFSTR.

RND RND(N) générera une valeur entre 0 et N–1, par exemple PRINT RND(6) imprimera une valeur entre 0 et 5.

TIME Cette fonction opère d'une façon très semblable à la fonction TIME normale, mais cette fois le résultat est mémorisé comme chiffre à 16 bits. Cela signifie que l'horloge dépassera sa capacité et recommencera à partir de zéro toutes les 3 minutes et demi environ. Notez aussi que puisque les nombres entiers sont traités comme des chiffres en 'complément à deux' (voir calcul en nombres entiers), la valeur deviendra négative quand TIME=32768. Si vous utilisez la fonction TIME, soyez sûr de l'utiliser avec soin.

USING PRINT USING fonctionne exactement de la même façon que le BASIC interprété si ce n'est que "." et les champs sont ignorés.

GOTO Cette instruction opère de la même manière que le GOTO normal mais les programmes compilés ne doivent pas aller à des lignes contenant 'WEND' ou 'NEXT'; au lieu de cela une ligne 'REM' doit être ajoutée et celle-ci doit précéder la ligne contenant l'instruction 'WEND' ou 'NEXT'. La ligne 'REM' est la ligne de référence du GOTO. (Voir programme type 'Le crible d'Erathostène.)

UNT Ce code n'a aucun effet car tous les chiffres sont traités implicitement en tant que nombres entiers compléments à deux.

INPUT Le programme d'édition complet Amstrad n'est pas mis en oeuvre mais la touche d'effacement est utilisée pour effacer des caractères qui ont été introduits par erreur.

.5 En calcul arithmétique, Amstrad BASIC arrondira à l'unité supérieure les restes d'une valeur de .5 ou plus, tandis que le BASIC compilé arrondit toujours à l'unité inférieure (troncage).

ADDITION	Résultat entier Amstrad	Résultant compilé
7/8	1	0
4/8	1	0
3/8	0	0
15/8	2	1
12/8	2	1
11/8	1	1

Afin de simuler des programmes compilés en BASIC interprété, il vous suffit de remplacer tous les symboles "/" (divisiopns) par "\" (barres obliques inverses), opération qui forcera le BASIC Amstrad à tronquer de la même manière que le BASIC compilé.

INTERRUPTIONS Les programmes compilés n'appellent pas la touche Break (Interruption) et ne peuvent donc pas être interrompus. ON BREAK GOSUB n'est pas mis en oeuvre et il n'est pas possible de mettre fin à l'exécution de fond de Laser BASIC.

Fonctions à virgule flottante Le compilateur ne s'attend pas à trouver les fonctions à virgule flottante telles que COS (5) et celles-ci seront donc traitées comme des variables ou des tableaux.

Instruction	Intervention
PRINT COS (5)	Donnera 0 (COS n'a probablement pas reçue d'affectation), suivi de 5.
PRINT COS(5)	Donnera l'erreur "END OF STATEMENT EXPECTED" ("FIN D'INSTRUCTION ATTENDUE").

Tableaux Le contrôle des limites des arguments de tableaux n'est pas effectué dans le code compilé, tandis qu'il l'est dans le BASIC interprété. Si un tableau était défini par 100 éléments, le BASIC interprété générera un message d'erreur dans l'eventualité où un 101ème élément était utilisé tandis que le code compilé ne le ferait pas. Si la lecture à partir d'un élément qui se trouve hors des limites, est tentée, le résultat n'aura aucun sens. Si l'affectation d'une valeur à un élément qui se trouve hors des limites, est tentée, les résultats seront alors imprévisibles.

↑ L'élévation à la puissance n'est pas prise en charge.

DEFINT DEFINT n'est accepté que sous la forme DEFINT A-Z. Des noms de variable individuels causeront l'apparition du message d'erreur 'caractère inattendue'.

SON Le contrôle des limites des paramètres n'est pas effectué et le résultat des erreurs est imprévisible.

CHARGEMENT ET SAUVEGARDE DES SYMBOLES GRAPHIQUES A PARTIR D'UN PROGRAMME COMPILE

Lorsque le Laser BASIC sauvegarde un fichier de symboles graphiques, il sauvegarde aussi quelques variables de système qui dictent la position à laquelle elles seront rechargées avec l'instruction GSPR. Cela signifie que les symboles graphiques sauvegardés à partir de l'interpréteur étendu Laser BASIC ou du générateur de symboles graphiques seraient rechargés pour résider à l'adresse &6FFF vers le bas, tandis que le programme compilé exige qu'ils résident à &75FF vers le bas. Cette difficulté peut être surmontée en établissant l'espace de symboles graphiques à une adresse plus élevée – à l'aide de SSPR,SMAX,&7600 et en fusionnant les symboles graphiques avec un MSPR au lieu d'un GSPR.

Il est très important de noter que SSPR,SMAX,&7000 doivent toujours être remplacés par SSPR,SMAX,&7600 avant la compilation, que les symboles graphiques dussent être chargés et sauvegardés ou pas. Faute de quoi, le système ne tombera probablement pas en panne, mais 1,5k d'espace sera perdu.

Si votre programme utilise plus d'une instruction GSPR il vous faudra alors répéter le SSPR,SMAX,&7600. L'utilisation de l'instruction MSPR n'est cependant pas affectée. Les symboles graphiques sauvegardés à partir d'un programme compilé (à l'aide de PSPR) devront être rechargés dans le programme non compilé en utilisant la même technique, c'est-à-dire SSPR,SMAX,&7000 et puis MSPR.

CALCUL EN NOMBRES ENTIERS

Le calcul exécuté par le compilateur est uniquement en nombres entiers. Si le programme que vous compilez commence avec un DEFINT A–Z, le programme compilé se comportera alors plus ou moins de la même façon que le programme interprété. Si vous êtes accoutumé aux pièges que l'on peut rencontrer en calcul en nombres entiers, passez alors à la section suivante; sinon examinez l'exemple suivant:

```
PRINT 5*4/2,5/2*4
```

produira

```
10          10
```

en passage interpréteur.

La même instruction produirait

```
10          8
```

en mode compilé. Ceci s'explique par le fait que le BASIC interprété mémorise à la division un résultat intermédiaire en virgule flottante (quelque soit le DEFINT) tandis que le code compilé utilise un résultat intermédiaire entier.

5*4/2 Le 5 est d'abord multiplié par 4 pour donner 20 et ensuite divisé par 2 pour donner 10. Cette opération s'effectue ainsi avec l'interpréteur aussi bien qu'avec le compilateur.

5/2*4 Le 5 est d'abord divisé par 2 pour donner 2,5 et 2 respectivement, dans les programmes interprétés et compilés. Ces résultats sont ensuite multipliés par 4 pour donner respectivement 10 et 8.

Si votre programme compilé ne se comporte pas de la même manière que votre programme interprété, cela en sera presque certainement la cause et on peut en général y remédier en ré-ordonnant les termes de l'expression arithmétique. En règle générale, effectuez toutes les divisions comme partie finale (à droite) de toute addition.

Notation complément à deux

Le compilateur Laser fait appel à une notation 'complément à deux' de telle manière que si une expression est imprimée, elle conduit à un résultat entre –32768 et 32767. Des affectations sont possibles avec des valeurs entre –32768 et 65535. Si une expression est imprimée à l'aide d'HEX$ elle sera alors entre 0 et &FFFF.

Le BASIC résident d'Amstrad ne permet pas à des variables entières d'être affectées à des valeurs supérieures à 32767 si elles sont dans une BASE décimale, par exemple:

X%=32768	générera un dépassement de capacité
X%=&8000	ne générera par contre pas de dépassement de capacité
X%=&8000:PRINT X%	imprimera –32768 car 32768 est en fait –32768 en notation complément à deux.

Les variables entières du BASIC compilé se comportent plus ou moins de la même manière que les variables entières d'Amstrad si ce n'est que les affectations à des valeurs décimales supérieures à 32767 sont autorisées.

UTILISATION DE VARIABLES

Il existe quelques points de détail à noter en ce qui concerne l'utilisation de variables du compilateur.

(i) Lorsque le programme compilé est passé pour la première fois, l'espace des variables peut contenir des informations parasites si bien qu'avant d'être utilisées toutes les variables devraient avoir une valeur affectée à celles-ci. Lorsqu'en BASIC une variable est déclarée pour la première fois, elle sera initialisée pour contenir 0. Si votre programme compilé n'est pas exécuté de la même manière que votre équivalent interprété, c'est une zone propice à la recherche des problèmes.

(ii) Essayez d'éviter le recours à un trop grand nombre de constantes de chaîne car le compilateur n'effectue pas 'un programme récupérateur'. Si, par exemple, B$ doit contenir la même chaîne que A$, utilisez alors B$=A$ plutôt que d'affecter B$ à la même chaîne.

(iii) Si, avant d'être déclaré dans une instruction de dimension, un tableau est utilisé en BASIC interprété, il sera automatiquement initialisé comme tableau à 10 éléments. Les programmes compilés, cependant, exigent que tous les tableaux soient dimensionnés avant d'être utilisés, sinon une erreur BAD ARRAY INDEX sera signalée.

SAUVEGARDE DE PROGRAMMES COMPILES

Les programmes compilés commencent à &40. La longueur du code est rapportée à la fin de la compilation. Afin d'effectuer une sauvegarde de code compilé, il vous faudra employer l'utilitaire de transfert de fichier que les utilisateurs de bandes trouveront sur la face A de la bande de distribution, constituant le dernier fichier. Pour réaliser l'exécution, tapez simplement RUN"XFR.BIN et le programme se chargera et sera exécuté avec les demandes suivantes:

ENTER INPUT FILE	Vous devez alors taper le nom du fichier que vous désirez sauvegarder, introduire la bande/le disque source et puis appuyer sur ENTER.
INSERT DESTINATION TAPE/DISK	Vous devez alors introduire la bande ou le disque sur laquelle/lequel vous désirez que le fichier soit sauvegardé et appuyer ensuite sur n'importe quelle touche.

RSXs ET MEMOIRES MORTES DE FOND

Le compilateur et son espace de travail occupent toute la mémoire vive Amstrad disponible – il n'est par conséquent pas possible d'avoir un RSX quelconque présent dans la machine lorsque le compilateur marche.

Les programmes compilés sont en général des programmes code machine si bien que lorsqu'ils sont passés, la micrologique remet la machine à son état EMS (Démarrage de Bon Matin). Si donc un RSX est utilisé dans le programme, il doit être chargé et mis en contact avec le système au sein du programme lui-même. La même chose vaut pour toute mémoire morte de fond qui peut être utilisée dans le programme. Le Laser BASIC utilise le RST #30 et sera donc incompatible avec quelques RSXs.

ERREURS D'EXECUTION

– Div by zero	– Div par zéro
– String too long	– Chaîne trop longue
– Bad argument to function	– Mauvais argument de fonction
– String space exhausted	– Espace chaîne épuisé

Toutes ces erreurs produiront un message qui sera affiché (implicite). Si un ON ERROR est mis en place, il sera exécuté. Sinon, l'exécution sera continuée en appuyant sur une touche.

N.B. RESUME sera compilé en tant que RET.

TOPOGRAPHIE DE LA MEMOIRE

COMPILATEUR:

&40	→	&7BFF	Espace de travail
&7C00	→	&A700	Code et données de compilateur.

PROGRAMMES COMPILES:

&40	→	&1BFF	Code de soutien à l'exécution (approché)
&1C00	→	?	Code de programme de l'utilisateur
?	→	??	Données de l'utilisateur
??	→	???	Espace de travail dynamique et de chaîne
???	→	MEMORY-1	Espace de travail final pour code d'execution
MEMORY	→	ROM WS	Non utilisé (pourrait être code d'exécution et symboles graphiques Laser BASIC ou bien RSXs chargés sur programme)

PROGRAMMES LASER BASIC COMPILES:

&40	→	&1BFF	Code de soutien à l'exécution (approché)
&1C00	→	?	Code de programme de l'utilisateur
?	→	??	Données de l'utilisateur
??	→	???	Espace de travail dynamique et de chaîne
???	→	MEMORY-1	Espace de travail fixe pour code d'exécution
MEMORY	→	&75FF	Symboles graphiques et espace de travail Laser BASIC.
&7600	→	&A4FF	Code d'exécution Laser BASIC
&A500	→	ROM WS	Non utilisé.

COPYRIGHT-HINWEIS

DAS VORLIEGENDE HANDBUCH

Da unerlaubtes Kopieren epidemische Proportionen angenommen hat, bedauern wir, dieses Handbuch in einer Form vervielfältigen zu müssen, die nicht fotokopiert werden kann. Wir bedauern, wenn wir unseren echten Kunden hiermit Unbequemlichkeiten verursachen. Für Informationen, die zur erfolgreichen gerichtlichen Verfolgung von Parteien führen, die unseren Copyright-Hinweis verletzen, ist eine Belohnung ausgesetzt.

ANMERKUNG

Das vorliegende Handbuch ist für den Einsatz des Laser Kompilierers wesentlich. Aus diesem Grunde bitten wir, es sorgfältig zu behandeln, da unter keinen Umständen getrennte Handbücher geliefert werden.

INHALT

Der Laser BASIC Kompilierer
von Vinay Sajip

EINFÜHRUNG

Der Laser-BASIC Kompilierer wurde als Begleitprodukt für die Laser-BASIC-Erweiterung geschrieben. Da es jedoch einen vollen integreren Teilsatz der residenten Amstrad-BASIC kompiliert, ist er auch nützlich für Programmierer, die Laser BASIC nicht benutzen. Der Betrieb des Kompilierers wurde so ausgelegt, um die Länge der zu kompilierenden Programme zu optimieren. Aus diesem Grund folgt das Laden des Kompilierers in zwei Durchläufen, und während der Kompilierung werden die Betriebszeitbibliotheken umgespeichert. Dies bedeutet, daß sich Bandbenutzer mit der Unannehmlichkeit abfinden müssen, die Bänder ziemlich häufig zu wechseln. Wir glauben jedoch, daß die hieraus entstehenden Vorteile die Nachteile überwiegen, würden jedoch Ihr Kommentar hierzu begrüßen.

Der entstandene Code ist außschließlich integer, so daß keins der Fließkommamerkmale von Amstrad kompiliert werden kann. Eine genaue Liste der Ausnahmen finden Sie in einem späteren Abschnitt. Der Vorteil eines integren Kompilierers besteht daraus, daß der erzeugte Code sehr schnell und sehr kompakt ist. Der Kompilierer erzeugt reinen Z80 Code und nicht P-Code – was wiederum zu schnellerer Ausführung führt.

Wenn ein Laser Basic Programm kompiliert wird, ist der Laser BASIC Betriebszeitcode nicht mit dem kompilierten Code zu verketten, und dieses Verfahren wird in einem späteren Kapitel beschrieben. Es handelt sich um die einzige Form, in der kompilierte Laser BASIC Programme reproduziert und verkauft werden können. Der Laser BASIC Interpreter selbst kann unter keinen Umständen reproduziert werden, außer zu Unterstützungszwecken.

Der kompilierte Code führt RSX (BAR Befehle) aus, wenn sie vom kompilierten Code geladen oder registriert werden. Wenn Sie beabsichtigen, Software zu vertreiben, die RSX ausführt, sind diese offensichtlich mit Ihrem kompilierten Code zwischenzuspeichern. Wenn der Code, den Sie zwischenspeichern, durch Copyright-Hinweis geschützt wird, sorgen Sie dafür, daß Sie die Genehmigung von den Herausgebern haben. Wie bereits erwähnt, gibt Oasis keine Genehmigung dafür, daß der Laser BASIC Interpreter in dieser Weise benutzt wird, da jedoch der Betriebszeitcode mit dem Paket geliefert wird, besteht keine Notwendigkeit dafür.

BANDVERZEICHNIS

Das Verteilerband enthält folgende Dateien:

SEITE A:	(i)	COMP.BAS	Die BASIC-Ladedatei für den Kompilierer
	(ii)	COMP.BIN	Der Kompilierer selbst
	(iii)	COMP2.BIN	Der Codeerzeugungsdurchlauf
	(iv)	BRTS.BIN	Der BASIC-Betriebszeitcode
	(v)	LASE.BIN	Der Laser-BASIC-Betriebszeitcode
	(vi)	XFR.BIN	Das Dateienübertragungsdienstprogramm
SEITE B:	(i)	DEMO.BIN	Das kompilierte Demo-Programm

BEDIENUNGSANLEITUNG

1. Der erste Schritt besteht aus dem Laden des BASIC Programms, das Sie kompilieren wollen und dann erneuter Zwischenspeicherung als ASCII Datei. Wenn das Programm, das Sie kompilieren wollen z.B. "DEMO" hieße, würden Sie eingeben:

   LOAD"DEMO

   Jetzt stecken Sie das neue Band ein (Diskettenbenutzer setzen eine Diskette ein und geben ein:

SAVE"DEMO",A

Wir verweisen auf das ',A' nach der Dateibezeichnung. Dies befiehlt dem System, die Datei im ASCII Format zu speichern. Sie werden feststellen, daß die ASCII Datei länger als die normale BASIC Datei ist. Sie haben daher sicherzustellen, daß das Band lang genug ist oder daß Sie ausreichenden Platz auf Ihrer Bestimmungsdiskette haben. Diskettenbenutzer müssen weiterhin beachten, daß die Bestimmungsdiskette genügend Speicherplatz für die vorübergehende Speicherung von BRTS.BIN und LASE.BIN hat.

2. Maschine rückstellen (SHIFT/CTRL /ESC drücken) dann den Kompilierer mit folgendem Verfahren ausführen:

Bandbenutzer: Das Kompiliererband einlegen und auf den Anfang von Seite A zurückspulen, wenn Sie dies nicht bereits getan haben und eingeben:

RUN"

Diskettenbenutzer: Kompiliererdiskette einlegen und eingeben:

RUN"COMP

Jetzt lädt der Kompilierer und führt aus.

3. Jetzt gibt der Kompilierer die Aufforderung aus:

"Tape or Disk (T/D)" ("Band oder Diskette (T/D)")

Bandbenutzer müssen dann "T", Diskettenbenutzer "D" eintasten. Wurde "D" für Diskette eingegeben, lädt der Kompilierer jetzt BRTS.BIN und LASE.BIN und erteilt die Aufforderung:

"Insert destination disk" ("Bestimmungsdiskette einlegen")

Sodann die Bestimmungsdiskette einlegen und eine beliebige Taste betätigen. BRTS.BIN und LASE.BIN werden jetzt umgespeichert.

ANMERKUNG: Es wäre wahrscheinlich praktisch, die gleiche Diskette für die ASCII-Ursprungsdatei und die Objektausgabe zu verwenden.

Der Kompilierer fragt jetzt nach der Bezeichnung der Ursprungsdatei, die zu kompilieren ist. In unserem Beispiel heißt die Datei "DEMO".

Bandbenutzer: Band einlegen, auf dem Sie die ASCII Version der "DEMO" zwischengespeichert haben und zum Dateibeginn zurückspulen. Jetzt Dateibezeichnung eingeben, die in diesem Fall "DEMO" ist.

Diskettenbenutzer: Diskette einlegen, auf der Sie die ASCII Version von "DEMO" zwischengespeichert haben, dann Dateibezeichnung eingeben, in diesem Fall "DEMO"

4. Der Kompilierer lädt jetzt die ASCII Datei über Band oder Diskette und führt eine Syntaxkontrolle aus. Damit wird auch die Ursprungsdatei in ein internes Format umgewandelt und prüft die Syntax. Wenn Fehler gefunden werden, wird eine Fehlermeldung mit grobem Hinweis darauf, wo der Fehler entstand, ausgedruckt. Alle Ursprungszeilen werden auf dem Bildschirm gedruckt, wie sie gelesen wurden, so daß Fehlermeldungen direkt nach der fehlerhaften Zeile erscheinen. Wenn ein Fehler festgestellt wurde, wartet das Programme darauf, daß Sie eine Taste drücken, bevor es sich fortsetzt.

Wenn ein Syntaxfehler auftritt, wird die Kompilierung abgebrochen, und der Kompilierer fragt nach der Bezeichnung der neuen Ursprungsdatei. Eine andere Dateibezeichnung oder CTRL-C (oder CTRL-SHIFT-ESC) eingeben, um das Gerät rückzustellen und das Ursprungsprogramm zu ändern, das die Fehler verursachte.

Traten keine Fehler auf, lädt der Kompilierer den Codeerzeugungsdurchlauf. Dieser auf dem Kompiliererband oder der Kompiliererdiskette geschrieben. Deshalb erscheint ein Programmaufruf, das richtige Band oder die Diskette einzugeben. Syntax- und Codeerzeugungsdurchläufe werden separat geladen, um Speicherplatz zu sparen und das Kompilieren von umfangreicheren Programmen zu erlauben. Diese Methode hat nur den Nachteil, daß beim Auftreten von Fehlern während des Codeerzeugungsdurchlaufs der Kompilierer selbst neu geladen werden muß, bevor ein zweiter Versuch der Kompilierung erfolgt. Wir glaubten jedoch, daß die bessere Leistung diese Unannehmlichkeit während der Benutzung ausgleicht.

Der Kompilierer beginnt sodann den Codeerzeugungsdurchlauf.

Bandbenutzer: Der erste Teil des Codeerzeugungsdurchlaufs besteht aus dem Laden und der Ausgabe von BRTS.BIN, den Betriebszeitcode. Während dieser Zeit müssen Sie zwischen dem Kompiliererband (aufgefordert durch "press PLAY then any key": "PLAY betätigen, sodann eine beliebige Taste) und dem Zielband (aufgefordert durch "press REC and PLAY then any key": "REC und PLAY betätigen, sodann eine beliebige Taste") abwechselnd.

Diskettenbenutzer: Bestimmungsdiskette einlegen

Sodann beginnt der eigentliche Codeerzeugungsdurchlauf und der Objektcode wird auf Bestimmungsband/-diskette geschrieben. Wenn Fehler während dieses Durchgangs entstehen, wird die Kompilierung abgebrochen und die Ausgabedatei verlassen. In diesem Fall müssen Sie den Kompilierer erneut laden, um einen weiteren Versuch einzuleiten. Wird die Kompilierung erfolgreich beendet, erfolgt ein Ausdrucken der Länge des kompilierten Codes. Wiederum werden die Zeilenzahlen mitgedruckt, um bei der Fehlersuche zu helfen.

Bandbenutzer: Kompilieren Sie ein Laser-BASIC-Programm, besteht die letzte Phase der Kompilierung im Laden und der Ausgabe von LASE.BIN, dem Laser-BASIC-Betriebszeitcode. Während dieser Phase müssen Sie wieder zwischen dem Kompiliererband (aufgefordert durch "press PLAY then any key": PLAY betätigen, sodann eine beliebige Taste") und dem Bestimmungsband (aufgefordert durch "press REC and PLAY then any key": "REC und PLAY betätigen, sodann eine beliebige Taste") abwechselnd.

Jetzt sollte Ihr Programm kompiliert sein. Die Dateibezeichnung der Ausgabedatei ist dann die gleiche wie die Ursprungsdatei, hat jedoch eine Erweiterung von .BIN. In unserem Beispiel nennen wir die Ausgabedatei "DEMO.BIN". Das kompilierte Programm ist stets etwa 7 k lang, denn der Betriebszeitcode des Kompilierers wird stets damit zwischengespeichert, ungeachtet der Dateigröße. Bevor wir weitergehen, um festzustellen, wie das kompilierte Programm abläuft, betrachten wir zunächst die Kompiliererzeitfehler, die während der Kompilierung Ihrer Ursprungsdatei entstehen können.

KOMPILIERZEITFEHLER

Es gibt drei Fehlerarten: Benutzungs-, Syntax- und Semantikfehler. Syntaxfehler treten während des ersten Durchgangs auf und werden hauptsächlich durch fehlende Datenteile, Schlüsselworte oder Punktierung verursacht. Semantikfehler sind die, die bedeutungslose Fehler angeben (z.B. Multiplizieren von zwei Zeichenreihen). Sie werden normalerweise im zweiten Durchgang festgestellt.

Syntaxfehler

Die meisten Syntaxfehler treten auf, weil ein Symbol, das erwartet wurde, nicht gefunden wurde. Diese Fehler werden durch Angabe des erwarteten Symbols bezeichnet. Was folgt ist keine erschöpfende Liste aber bezeichnend für das, was gefunden werden kann:

'(' EXPECTED	INTEGER EXPECTED	LINE NUMBER EXPECTED
')' EXPECTED	VARIABLE EXPECTED	BOOLEAN EXPRESSION EXPECTED
'GOSUB' EXPECTED	IDENTIFIER EXPECTED	
'=' EXPECTED	TOO MANY ARGUMENTS	
'TO' EXPECTED	TOO FEW ARGUMENTS	
END OF STATEMENT EXPECTED	COMMAND EXPECTED	

Weitere Beispiele sind Fehler in der Programmzusammensetzung, z.B.

FOR/NEXT MISMATCH	BAD PRIMARY EXPRESSION
BAD STATEMENT	BAD SUBEXPRESSION

Weitere werden gefunden, wenn Schlüsselworte oder Punktierung gelesen werden z.B.

ILLEGAL CHARACTER

Benutzungsfehler

Es entstehen Fehler, bedingt durch Speicher-, Disketten- oder Bandfehler oder durch inkompatible Dateiart wie

INPUT FILE MUST BE ASCII – please try again
NEED MORE SPACE!! (Arbeitsplatz für den Kompilierer erschöpft)

Semantikfehler

Diese Fehler sind normalerweise bedingt durch Fehlpassung oder falschen Datenfeldfolgewert, d.h.

ILLEGAL OPERATION ON STRING ATTEMPTED
BAD ARRAY INDEX

Datenfeld-Indexfehler werden durch Fehlpassung zwischen den Maßen einer Datenfelderklärung und ihrer Benutzung verursacht. Der Kompilierer nimmt keine Kontrolle der Feldfolgewerte vor, selbst wenn diese bei Kompilierzeit bekannt sind. Der Kompilierer betrachtet den Einsatz von nicht initialisierten Variablen nicht als Fehler.

AUSFÜHRUNG VON KOMPILIERTEN PROGRAMMEN

Programme, die keine Laser BASIC Befehle benutzen

Wenn das Programm, das Sie kompiliert haben, keine erweiterten Laser BASIC Befehle enthält, kann es so ausgeführt werden, wie es ist.

Bandbenutzer: Das Band mit dem kompilierten Code einlegen und rückspulen, falls Sie dies nicht bereits getan haben. Dann RUN" eingeben, und der Code lädt sich und wird ausgeführt.

Diskettenbenutzer: Die Diskette einlegen, die den kompilierten Code enthält, und RUN" Dateibezeichnung (in unserem Beispiel RUN"DEMO) eingeben. Der Code lädt sich und führt das Programm aus. Es ist zu bemerken, daß falls DEMO.BAS sich auf der gleichen Diskette befindet wie DEMO.BIN, diese ausgeführt wird und nicht die kompilierte Version.

Programme, die Laser BASIC Befehle benutzen

Wenn der Kompilierer Laser BASIC Befehle während der Syntaxkontrolle findet, wird ein Zeichen gesetzt, um dem System zu sagen, daß LASE.BIN (der Betriebszeitcode für Laser BASIC) mit dem kompilierten Code zu verketten ist. Die Betriebszeitbibliothek ist 12k lang, und die gesamte Bibliothek wird verkettet. Dies bedeutet, daß kompilierte Laser BASIC Programme mindestens nach Kompilieren diese Größe aufweisen. Der Betriebszeitcode wird übrigens als Teil des kompilierten Code gespeichert, und das erste, was das kompilierte Demo tut, wenn es ausgeführt wird, ist die Bibliothek als Block auf &7600 zu bringen, ihre laufende Ausführungsadresse.

Wenn Ihr Programm keine Laser BASIC Befehle benutzt, bittet Sie der Kompilierer automatisch, das Band/die Diskette einzulegen, die die LASE.BIN enthält (Seite A – direkt nach dem BASIC Betriebszeitcode).

In der Praxis enthält Ihre Hauptdatei wahrscheinlich zwei Dateien, denn Ihr Programm lädt wahrscheinlich in einer bestimmten Stufe Sprites (siehe Absatz über das Laden von Sprites in ein kompiliertes Programm). Berücksichtigen wir einmal die Schritte, die erforderlich sind, um die endgültige Originalkopie eines kompilierten Laser BASIC Programms zu erstellen. Nehmen wir an, das Programm ist "DEMO" genannt und verwendet einen Spritesatz, der als "DEMOSPR" zwischengespeichert wird. Wir nehmen ferner an, daß die Sprites unter Verwendung des Laser BASIC Interpreter zwischengespeichert wurden und daher normalerweise ab &6FFF nach unten residieren.

Bandbenutzer benutzen 4 Bänder – das Laser Kompilierband, das Band mit dem Ursprungsprogramm (dem zu kompilierenden Programm), das Band mit den Spriteteilen und ein leeres Band für das endgültige Hauptprogramm. Diskettenbenutzer benutzen wahrscheinlich mehrere Disketten, abgesehen von der Kompilierer-Verteilerdiskette.

(i) Der erste Schritt besteht daraus, das Programm so zu ändern, daß die Sprites auf eine höhere Adresse des Betriebszeitcode geladen werden. Dies wird erzielt, indem die SSPR,SMAX,&7000 durch SSPR,SMAX,&7600 ersetzt werden und dann durch Ersatz aller GSPR Befehle durch MSPR Befehle (siehe Absatz über Laden der Sprites von kompilierten Programmen).

(ii) Der nächste Schritt besteht aus der Änderung des Einsatzes für den MEMORY Befehl (siehe Absatz "BESCHRÄNKUNGEN FÜR DIE ZU KOMPILIERENDE BASIC", der den Einsatz des MEMORY Befehls beschreibt).

(iii) Wenn Sie die RND Funktion benutzt haben, müssen Sie auch die Syntax ändern (siehe Absatz "BESCHRÄNKUNGEN DER ZU KOMPILIERENDEN BASIC").

(vi) Nachdem die vorstehenden Änderungen ausgeführt wurden, wird die Datei erneut als ASCII Datei zwischengespeichert und kompiliert. Das kompilierte Programm ist die erste Datei auf dem Hauptband/der Hauptdiskette. Wenn Ihr kompiliertes Programm Sprites lädt, ist das Hauptband nicht rückzuspulen, jedoch statt dessen Vorgang (v) auszuführen.

(v) Gerät durch Drücken von SHIFT/CTRL/ESC. rückstellen. Laser BASIC laden und ausführen, dann die gewünschten Sprites unter Verwendung von GSPR (in ihrem Beispiel A$="DEMOSPR":|GSPR,@A$) laden. Jetzt Hauptband oder -diskette einsetzen und die Sprites zwischenspeichern.

Das Band/die Diskette enthält jetzt das endgültige Hauptprogramm, das durch Eingabe von RUN"Dateibezeichnung (in unserem Fall RUN"DEMO) ausgeführt werden kann.

ANMERKUNG: RSX von Laser BASIC brauchen viel Speicherplatz während der Kompilierphase, und, wo möglich, spart ein GOSUB Befehl zu einer Zeile, die den RSX enthält, Platz und ermöglicht die Kompilierung von größeren Programmen.

Betrieb des kompilierten Demo

Auf Seite B des Verteilerbandes befindet sich eine kompilierte Version der Laser BASIC Demo. Sie sind wahrscheinlich erstaunt darüber festzustellen, daß die Ausführungsgeschwindigkeiten nicht viel kürzer sind als bei den unkompilierten Äquivalenten. Dafür gibt es mehrere Gründe.

1. Laser BASIC ist der schnellen Graphik gewidmet, und daher wird ein großer Teil der Prozessorzeit mit dem Ausführen der Graphikbefehle verbracht. Die Kompilierung hat wenig Wirkung auf Programme, die keine Logik enthalten. Ihre eigenen Programme enthalten wahrscheinlich viel Logik und werden daher merklich beschleunigt.
2. Der größte Teil der Bewegung wird durch Synchronisierung auf Bildrücklauf gemessen.
3. Ein großer Teil des Code wird bei Unterbrechung ausgeführt.

ANMERKUNG: Das kompilierte Demonstrationsprogramm auf Seite B des Verteilerbandes ist genau dasselbe wie das mit dem Laser BASIC Paket gelieferte, außer daß die ISET Befehle durch IGETS ersetzt und die RND und MEMORY Befehle geändert wurden.

BEISPIELPROGRAMME

Was Sie nicht sehen, ist die enorme Steigerung in der Geschwindigkeit, mit der die Spiellogik ausgeführt wird. Um Ihnen eine Vorstellung davon zu geben, wie viel schneller die Programmlogik nach Kompilierung läuft, geben wir Ihnen nachstehend zwei Beispielprogramme zum Kompilieren und Betreiben.

Das Sieb des Eratosthenes (ungefähre Geschwindigkeitssteigerung: 30:1)

```
100 REM
110 REM Sieb des Eratosthenes.
120 REM
130 DIM prime%(5000)
140 start=TIME
150 FOR i%=2 TO 2500
160 IF prime%(i%)=1 THEN 215
170 j%=i%+i%
180 WHILE j%<=5000
190 prime%(j%)=1
200 j%=j%+i%
210 WEND
215 REM ziel für GOTO
220 NEXT i%
230 calcstook=(TIME-start)/3
240 start=TIME
250 FOR i%=2 TO 5000
260 IF prime%(i%)=0 THEN PRINT i%,
270 NEXT i%
280 printstook=(TIME-start)/3
290 PRINT
300 PRINT:PRINT "Berechnungsdauer  ";USING "# # #"; calcstook;
310 PRINT " Hunderstel Sek"
320 PRINT:PRINT "Druckdauer  ";USING "# # #";printstook;
330 PRINT " Hunderstel Sek"
340 END
```

Kreiszeichnung (ungefähre Geschwindigkeitssteigerung 5:1)

```
100 REM
110 REM Demo Kreiszeichnung.
120 REM
130 DEFINT a-z
140 CLS
150 WINDOW 1,80,23,25
160 INPUT "Kreismitte ";xc,yc
170 INPUT "Kreisradius ";radius
180 ORIGIN xc, yc
190 GOSUB 260
200 GOTO 160
210 REM
220 REM Kreis zeichnen
230 REM   Diese Routine erwartet, daß der graphsiche Ausgangspunkt in der Kreismitte
240 REM   benötigt wird und die Variable "Radius", um den gewünschten Radius zu halten
250 REM
260 x=0:y=radius
270 delta=3-radius-radius
280 WHILE x<y
290 GOSUB 390
300 delta=delta+x+x+x+x+6
310 IF delta>=0 THEN delta=delta-y-y-y-y+4:y=y-1
320 x=x+1
330 WEND
340 IF x=y THEN GOSUB 390
350 RETURN
360 REM
370 REM Einen Punkt in jedem der acht Oktanten zeichnen.
380 REM
390 PLOT x,y:PLOT -x,y:PLOT -x,-y:PLOT x,-y:PLOT y,x:PLOT -y,x:PLOT -y,-x:PLOT
    y,-x:RETURN
```

BESCHRÄNKUNGEN FÜR DIE ZU KOMPILIERENDE BASIC

Der Laser BASIC Kompilierer ist kompatibel mit einer ziemlich kompletten integren Implementation Locomotive BASIC – keine der Fließkommafunktionen wird unterstützt. Sofortige Befehle werden nicht unterstützt, ebenfalls keine Befehle und Funktionen, die sich auf die Zuweisung des BASIC Interpreter von Speicherplatz für Programm und Variable beziehen. Die MEMORY Funktion wird noch unterstützt, so daß andere Programme (insbesondere RSX und Laser BASIC Sprites) mit dem kompilierten Benutzerprogramm im Speicher aufgenommen werden können.

Spezifische Schlüsselwortausnahmen sind:

ATN	AUTO	CHAIN	CINT	CLEAR	CONT	COS	CREAL
DEFREAL	DEG	DELETE	EDIT	ERASE	ERR	ERL	EXP
FRE	HIMEM	LIST	LOG	LOG10	MERGE	NEW	PI
RAD	RENUM	RESUME	ROUND	SIN	TAN	TRON	TROFF

Ausnahmen für Laser BASIC sind:

ADDR ISET

Es gibt ferner mehrere Befehle und Funktionen, deren Einsatz nach Kompilieren etwas anders ist:

LOAD: BASIC Dateien können nicht geladen werden. Binäre Dateien werden direkt in ihre Bestimmungsadresse geladen, wenn eine angegeben ist. Das bedeutet, daß der Zwischenspeicher von 2k nur erforderlich ist, wenn eine binäre Datei ohne Angabe der Ladeadresse geladen wird. Aus diesem Grund sollten Sie stets die Ladeadresse angeben, wenn sie bekannt ist. Während des Ladens werden keine Kontrollen vorgenommen, so daß Sie darauf achten müssen, keine reservierten Speicherbereiche zu überschreiben.

END Dieser führt nur einen RET Befehl aus und bei Kompiliervorgängen den gleichen Code wie RETURN. Das bedeutet, daß wenn Ihr Programm richtig mit END endet, die Steuerung zum Befehlsniveau zurückkehrt.

FIX FIX durchläuft die Syntaxkontrolle des Kompilierers, hat jedoch keine Wirkung.

INT INT durchläuft die Syntaxkontrolle des Kompilierers, hat jedoch keine Wirkung.

CINT CINT durchläuft die Syntaxkontrolle des Kompilierers, hat jedoch keine Wirkung.

MEMORY Der MEMORY Befehl diktiert die höchste Speicherposition, die der kompilierte Code benutzen kann. Der Kompilierer benötigt Platz für die dynamischen Variablen und die Reihenverarbeitung. Der Standardwert für die oberste Speicherposition, wenn ein Programm, das nicht Laser BASIC verwendet, kompiliert wird, ist &A4FF, und der Standardwert für die oberste Speicherposition, wenn ein Programm, das Laser BASIC verwendet, kompiliert wird, ist &75FF. Es ist zu bemerken, daß der MEMORY Befehl und der Äquivalentbefehl MSET von Laser BASIC zum Einrichten des Speicherschutzes in der gleichen Höhe wie die Sprites (falls geladen) benutzt werden sollte, was vor Kompilierung möglich ist. Achten Sie darauf, daß Sie alle dynamisch zugewiesene Spriteplätze berücksichtigen! Um festzustellen, wo Sprites beginnen, laden Sie diese unter Verwendung des Befehls GSPR in Laser BASIC. Der Beginn der Sprites kann jetzt durch folgende Eingabe festgestellt werden:

START=0:|DEEK,&7004,@START

Jetzt wurde der START auf die Position eingestellt, an der die Spritedaten beginnen, wenn sie mit Interpreter betrieben werden. Wenn Sprites in kompiliertem Code geladen werden, beginnen sie jedoch &600 höher. Jetzt sollten Sie eingeben:

START=START+&600

Der START weist jetzt auf die geänderte Startadresse. Wenn Ihr Programm dynamisch gewisse Sprites erzeugt, müssen Sie den Speicherplatz berechnen, den Sie benötigen (jeder Sprite benötigt HGTxLEN – Höhe x Länge – Bytes), und dieser ist vom START abzuziehen. In unserem Beispiel nehmen wir an, daß wir dynamisch 2 Sprites mit den Maßen 3x4, 5x7 und 6x8 erzeugen. Sie würden eingeben:

START=START–3*4–5*7–6*8:PRINT START

Die einzige weitere Speicherberücksichtigung ist jetzt senkrechter Durchlauf mit zyklischer Adressfolge. Wenn Sie einen senkrechten Durchlauf ausführen müssen, brauchen Sie weiteren Platz, wodurch sich START weiter reduziert. Zwischenspeicherplatz wird im Handbuch für Laser BASIC im Absatz über Sprite-Dienstprogramme besprochen. Wenn Sie sich nicht sicher sind, welchen senkrechten Durchlaufbedarf Sie haben, ist es gut, den START um etwa 256 Bytes zu reduzieren. Der Wert, den Sie jetzt für START haben, sollte der Wert sein, den Sie für MSET im zu kompilierenden Programm benutzen. Ihr Programm sollte auch sobald wie möglich ein MEMORY bis START-1 ausführen.

DEFINT DEFINT und DEFSTR werden voll implementiert, DEFREAL nicht. Es ist jedoch gut, alle Ihre BASIC Programme mit dem Befehl DEFINT A-Z zu beginnen, so daß Ihr interpretiertes Programm in mehr oder weniger der gleichen Weise ausgeführt wird wie Ihr kompiliertes Programme – siehe Absatz über integre Arithmetik. Die Variablen werden als integer angenommen, außer wenn sie mit einem '$' enden oder wenn erklärt wird, daß sie DEFSTR benutzen.

RND Durch RND(N) entsteht ein Wert im Bereich von bis N-1, d.h.
PRINT RND(6) würde einen Wert im Bereich von 0 bis 5 drucken.

TIME Dieser funktioniert sehr ähnlich wie die normale TIME Funktion, hierbei wird jedoch das Ergebnis als Zahl von 16 Bit gespeichert. Das bedeutet, daß die Uhr überfließt und (ca.) alle 3 1/2 Minuten erneut von 0 zu zählen beginnt. Es ist auch zu bemerken, daß der Wert anscheinend negativ wird, wenn TIME=32768, da die Integren als Zweierkomplementzahlen betrachtet werden (siehe integre Arithmetik). Falls Sie die TIME Funktion benutzen, sollte dies mit Vorsicht geschehen.

USING PRINT USING funktioniert in der gleichen Weise wie bei interpretierter BASIC, außer daß "." und die darauffolgenden Felder ignoriert werden.

GOTO Dieser funktioniert in der gleichen Weise wie der normale GOTO, jedoch sollten kompilierte Programme nicht zu Zeilen gehen, die 'WEND' oder 'NEXT' enthalten. Statt dessen ist eine 'REM' Zeile zu ersetzen, die der Zeile voraufgeht, die den Befehl 'WEND' oder 'NEXT' enthält. Die 'REM' Zeile ist die Zeile für GOTO. (Siehe Beispielprogramme 'Das Sieb des Eratosthenes'.)

UNT Dies hat keine Wirkung, da alle Zahlen als Standardannahme als zweier komplementäre Ganzzahlen behandelt werden.

INPUT Der vollständige Amstrad-Editor wird nicht implementiert. Die Löschtaste dient jedoch für das Löschen von falsch eingegebenen Zeichen.

.5 Amstrad BASIC rundet Reste von .5 oder mehr während der Durchführung von ganzzahligen Kalkulationen nach oben ab, während kompiliertes BASIC stets die Reste nach unten abrundet (abschneidet).

SUM	Amstrad ganzzahliges Ergebnis	Kompiliertes Ergebnis
7/8	1	0
4/8	1	0
3/8	0	0
15/8	2	1
12/8	2	1
11/8	1	1

Um kompilierte Programme in interpretiertem BASIC zu simulieren, müssen Sie alle "/" (Divisionen) durch "\" (Rückwärtsschrägstriche) ersetzen. Dies zwingt Armstrad BASIC auf gleiche Weise wie kompiliertes BASIC abzuschneiden.

UNTERBRECHUNGEN Kompilierte Programme unterliegen nicht der Unterbrechungstaste und können deshalb nicht unterbrochen werden. ON BREAK GOSUB wird nicht implementiert und die Hintergrundausführung von Laser BASIC kann nicht abgebrochen werden.

Gleitpunktfunktionen Der Kompilierer erwartet nicht das Auftreten von Gleitpunktfunktionen wie z.B. COS (5), diese werden deshalb als Variable oder Arrays behandelt.

Anweisung	Aktion
PRINT COS (5)	Schreibt 0 (COS wahrscheinlich nicht zugewiesen), danach 5.
PRINT COS(5)	Fehlermeldung "END OF STATEMENT EXPECTED" ("ANWEISUNGSENDE ERWARTET").

Arrays Parametergruppen werden im kompilierten Code nicht auf Bereich geprüft, während sie in BASIC interpretiert werden. Wird ein Array mit 100 Parameter reserviert, würde bei Verwendung des 101. Parameter BASIC einen Fehler erzeugen, dies ist beim kompilierten Code jedoch nicht der Fall. Wird es versucht, einen außerhalb des Bereichs liegenden Parameter zu lesen, ist das Resultat Müll. Wird es versucht, einem Parameter außerhalb des Bereichs einen Wert zuzuweisen, sind die Resultate nicht vorhersehbar.

↑ Eine Potentierung wird nicht unterstützt.

DEFINT DEFINT wird nur im Format DEFINT A-Z implementiert. Einzelne Variablenbezeichnungen verursachen den Fehler "unerwartetes Zeichen".

SOUND Eine Bereichsprüfung von Parametern findet nicht statt, das Resultat von Fehlern ist unvorhersehbar.

LADEN UND ZWISCHENSPEICHERN VON SPRITES VON EINEM KOMPILIERTEN PROGRAMM

Wenn Laser BASIC eine Sprite-Datei zwischenspeichert, speichert sie auch gewisse Systemvariablen zwischen, die die Position diktieren, in die sie mit dem Befehl GSPR zurückgeladen werden. Dies bedeutet, daß Sprites, die vom erweiterten Interpreter von Laser BASIC zwischengespeichert werden oder der Sprite-Generator zurückgeladen werden, so daß sie bei &6FFF nach unten im Speicher aufgenommen werden, wogegen sie beim kompilierten Programm bei &75FF nach unten aufgenommen werden. Das kann durch Einstellen des Sprite-Platzes auf eine höhere Adresse gelöst werden – unter Verwendung von SSPR, SMAX,&7600 und dann durch Verschmelzen der Sprites mit einem MSPR anstatt mit GSPR.

Es ist äußerst wichtig zu bemerken, daß SSPR,SMAX,&7000 stets durch SSPR,SMAX,&7600 vor Kompilierung zu ersetzen sind, ungeachtet dessen, ob die Sprites geladen oder zwischengespeichert werden sollen. Wenn dies nicht geschieht, kann im System ein Zusammenstoß erfolgen, der 1,5k Platz verschwendet.

Wenn Ihr Programm mehr als einen GSPR Befehl verwendet, müssen Sie SSPR,SMAX,&7600 wiederholen. Dies wirkt sich jedoch nicht auf den Befehl MSPR aus. Sprites, die von einem kompilierten Programm (unter Verwendung von PSPR) zwischengespeichert werden, sind in das unkompilierte Programm unter Verwendung des gleichen Verfahrens zurückzuladen, d.h. SSPR,SMAX,&7000 und dann MSPR.

INTEGRE ARITHMETIK

Die vom Kompilierer ausgeführte Arithmetik ist ausschließlich integer. Wenn das Programm, das Sie kompilieren, mit einem DEFINT A-Z beginnt, verhält sich das kompilierte Programm mehr oder weniger in der gleichen Weise wie das interpretierte Programm. Wenn Sie die möglichen Fallen der integren Arithmetik kennen, gehen Sie zum nächsten Abschnitt über, aber wenn nicht, sollte folgendes Beispiel berücksichtigt werden:

```
PRINT 5*4/2.5/2*4
```

damit entsteht

```
10          10
```

wenn dies mit Interpreter betrieben wird.

Der gleiche Befehl würde

```
10          8
```

erzeugen, wenn er kompiliert wird. Der Grund dafür liegt in der Tatsache, daß interpretierte BASIC ein Fließkomma-Zwischenergebnis speichert, wenn geteilt wird (unabhängig von DEFINT), wogegen der kompilierte Code ein integres Zwischenergebnis benutzt.

5*4/2 — Die 5 wird zuerst mit 4 multipliziert, so daß sich 20 ergibt und durch 2 geteilt, so daß sich 10 ergibt. Dies funktioniert mit Interpreter oder Kompilierer.

5/2*4 — Die 5 wird zuerst durch 2 geteilt, so daß sich 2,5 bzw. 2 im interpretierten bzw. kompilierten Programm ergibt. Diese Ergebnisse werden dann mit 4 multipliziert, so daß sich 10 bzw. 8 ergibt.

Wenn Ihr kompiliertes Programm sich nicht in der gleichen Weise verhält wie Ihr interpretiertes Programm, ist das fast immer die Ursache und kann gewöhnlich behoben werden, indem die Terms des arithmetischen Ausdrucks umgeordnet werden. Als allgemeine Regel sind alle Teilungen als letzter Teil (rechts von) jeder Summe durchzuführen.

Zweierkomplement-Bezeichnung

Der Laser Kompilierer benutzt eine 'Zweierkomplement'-Bezeichnung. Wenn daher eine Zeichenreihe gedruckt wird, entsteht ein Ergebnis im Bereich von –32768 bis 32767. Zuweisungen sind mit Werten im Bereich von –32768 bis 65535 zulässig. Wenn eine Zeichenreihe mit HEX$ gedruckt wird, liegt sie im Bereich von 0 bis &FFFF.

Die im Amstrad residente BASIC ermöglicht nicht die Zuweisung von integren Variablen für Werte von über 32767, wenn diese in dezimaler BASE gehalten sind. d.h.

X%=32768	erzeugt einen Überlauf
X%=&8000	erzeugt jedoch keine Überlauf
X%=&8000:PRINT X%	druckt –32768, weil 32768 eigentlich –32768 in der Zweierkomplement-Bezeichnung ist.

Die integren Variablen von kompilierter BASIC verhalten sich in mehr oder weniger der gleichen Weise wie die integren Variablen von Amstrad, außer daß Zuweisungen zu Dezimalwerten von über 32767 legal sind.

VERWENDUNG VON VARIABLEN

Es gibt einige kleine Punkte, die hinsichtlich des Einsatzes von Variablen durch den Kompilierer anzuführen sind.

(i) Wenn das kompilierte Programm zuerst eingegeben wird, kann der Variablenplatz Unrat enthalten, daher ist allen Variablen vor Gebrauch ein Wert zuzuweisen. Wenn eine Variable zuerst in BASIC erklärt wird, wird sie initialisiert, so daß sie 0 enthält. Wenn Ihr kompiliertes Programm nicht in der gleichen Weise ausgeführt wird wie Ihr interpretiertes Äquivalent, ist dies ein guter Bereich, um damit zu beginnen, nach Problemen zu suchen.

(ii) Versuchen Sie, nicht zu viele Reihenkonstanten zu verwenden, da der Kompilierer keine 'Unratsammlung' durchführt. Wenn z.B. B$ die gleiche Zeichenreihe enthalten soll wie A$, verwenden Sie lieber B$=A$, anstatt B$ der gleichen Zeichenreihe zuzuweisen.

(iii) Wenn ein Datenfeld in interpretierter BASIC benutzt wird, bevor es in einem Dimensionsbefehl erklärt wurde, wird es automatisch als Datenfeld von 10 Elementen eingerichtet. Bei kompilierten Programmen sind jedoch alle Datenfelder zu dimensionieren, bevor sie benutzt werden, sonst erscheint eine Fehlermeldung BAD ARRAY INDEX.

UNTERSTÜTZUNG VON KOMPILIERTEN PROGRAMMEN

Kompilierte Programme beginnen bei &40. Die Länge des Code wird am Ende der Kompilierung gemeldet. Um eine Unterstützung des kompilierten Code vorzunehmen, müssen Sie das Daten-Übertragungsdienstprogramm benutzen, das Bandbenutzer als die letzte Datei auf Seite A des Verteilerbandes finden. Zum Ausführen geben Sie nur ein RUN"XFR.BIN, worauf sich das Programm lädt und die folgenden Weckbefehle ausführt:

ENTER INPUT FILE	Jetzt sollten Sie den Namen der Datei, die Sie unterstützen wollen, eingeben, das Ursprungsband/die Diskette einsetzen und dann ENTER drücken.
INSERT DESTINATION TAPE/DISK	Jetzt sollten Sie das Band oder die Diskette einsetzen, auf der die Datei zwischengespeichert werden soll und dann jede beliebige Taste drücken.

RSX UND UNTERSTÜTZENDE ROMS

Der Kompilierer und sein Arbeitsplatz nehmen den gesamten Amstrad RAM-Platz ein – daher ist es nicht möglich, daß RSX in der Maschine vorhanden sind, während der Kompilierer betrieben wird.

Kompilierte Programme sind normale Maschinencode-Programme. Wenn sie daher betrieben werden, stellt die Firmware die Maschine auf ihren EMS-Zustand (Einschaltzustand am frühen Morgen) zurück. Wenn im Programm RSX verwendet werden, sind sie vom Programm selbst zu laden und zu erfassen. Das gleiche gilt für alle unterstützenden ROMs, die vom Programm eventuell benutzt werden. Laser BASIC verwendet das RST ≠30 und ist daher mit gewissen RSX inkompatibel.

BETRIEBSZEITFEHLER

– Div by zero	– Teilen durch Null
– String too long	– Datenreihe zu lang
– Bad argument to function	– Falsche Variable für Funktion
– String space exhausted	– Datenreihenplatz erschöpft

Bei allen ensteht die Anzeige einer Meldung (Standard). Wenn ON ERROR eingesetzt wird, wird ausgeführt. Sonst wird die Ausführung durch Drücken jeder beliebigen Taste fortgesetzt.

ACHTUNG: RESUME wird als RET kompiliert.

SPEICHERVERZEICHNIS

KOMPILIERER:

&40	→	&7BFF	Arbeitsplatz
&7C00	→	&A700	Kompilierercode und -daten

KOMPILIERTE PROGRAMME:

&40	→	&1BFF	Betriebszeit-Unterstützungscode (ungefähr)
&1C00	→	?	Benutzerprogrammcode
?	→	??	Benutzerdaten
??	→	???	Zeichenreihen und dynamischer Arbeitsplatz
???	→	MEMORY-1	Fester Arbeitsplatz für Betriebszeitcode
MEMORY	→	ROM WS	Nicht benutzt (kann Betriebszeitcode für Laser-BASIC und Sprites oder programmgeladene RSXs sein)

KOMPILIERTE LASER-BASIC-PROGRAMME:

&40	→	&1BFF	Betriebszeit-Unterstützungscode (ungefähr)
&1C00	→	?	Benutzerprogrammcode
?	→	??	Benutzerdaten
??	→	???	Zeichenreihen und dynamischer Arbeitsplatz
???	→	MEMORY-1	Fester Arbeitsplatz für Betriebszeitcode
MEMORY	→	&75FF	Sprites und Laser-BASIC-Arbeitsplatz
&7600	→	&A4FF	Laser-BASIC-Betriebszeitcode
&A500	→	ROM WS	Nicht benutzt

TECHNICAL ENQUIRY CARD

WE AT OASIS BELIEVE IN GIVING FULL TECHNICAL SUPPORT TO ALL OUR PRODUCTS. TO ASSIST US, PLEASE FILL IN THIS CARD AND RETURN TO THE ADDRESS BELOW. TELEPHONE ENQUIRIES, OR LETTERS NOT ACCOMPANIED BY THIS CARD CANNOT BE ANSWERED. A REPLACEMENT CARD WILL BE ENCLOSED WITH OUR REPLY. ALL OVERSEAS ENQUIRIES MUST BE SUBMITTED IN ENGLISH AND WILL BE ANSWERED AT OUR FIRST OPPORTUNITY:

MAKE AND MODEL OF COMPUTER

SOFTWARE PURCHASED

VERSION No.

PLACE AND DATE OF PURCHASE.

YOUR NAME.

ADDRESS.

TELEPHONE No.

AGE OCCUPATION.

PLEASE WRITE YOUR ENQUIRY HERE.

Please return to:-
OASIS SOFTWARE, 12 WALLISCOTE ROAD, WESTON-SUPER-MARE
AVON, ENGLAND. BS23 1UG.