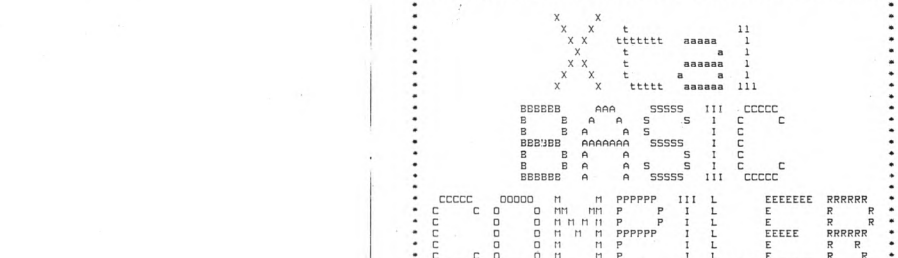

COMPILER PACKAGE FOR THE Xtal BASIC INTERPRETER
A Xtaltron (R) product

CONTENTS

This manual Printed & Published by:
Crystal Research Ltd.
40 Magdalene Road,
Torquay, Devon TQ1 4AF

Tel. (0803) 27890

ISBN No. 0 9506828 4 5

****
* Xtal BASIC COMPILER VERSION 5.10 *
****

1. INTRODUCTION

This is a brief introduction to the Xtal BASIC compiler, which has been designed to complement the now well-proven Xtal BASIC interpreter. Continual improvement to Xtal BASIC has meant that the compiler has tended to get pushed further and further back, to the point that we almost felt that it would never appear at all! Anyway, it is here now, it works, and has been tried and tested with a wide variety of Xtal BASIC software.

First, to explain the 'version' number! This is the first version of the compiler to be produced, but it is designed to support version 5 of the Xtal BASIC interpreter. It is intended that the compiler will parallel the future development of the interpreter, so that there will in future be a version of the compiler to support any new features or improvements provided in future versions of the interpreter.

INTERPRETER OR COMPILER?

It is as well at this point to explain the difference between a compiler and an interpreter, as this will serve to explain why we have developed a compiler in the first place.

Microprocessors work in their own 'low-level' language, machine-code, which is generally very primitive and long-winded for specifying tasks to be performed by the computer. Therefore, although we can write our programs in this machine-language (or in one directly related to it, that is, assembly language), we generally prefer to use a 'high-level' language, such as BASIC, FORTH, Pascal, FORTRAN, or one of many other languages available.

In order to operate in a high-level language, we must employ the services of a translator program, which will make our high-level program into a form understandable by the computer. Broadly speaking, the translator program can fall into two categories -- interpreter or compiler.

Both of these start with a source, or high-level, program. However, the interpreter does a continuous translation of the program and executes it while it does so. A compiler, on the other hand, translates the program once, into another program, known as the object program. This object program, generally now in machine-code, will then run as a program in its own right, without the help of another program. Thus, without the overhead of continuous translation while executing, a compiled program will run very much faster than an interpreted program. However, the object program may end up being a lot larger than the source program it replaces, depending upon how efficient the compiler is, and large programs may take a long time to compile, as the translation may be quite complicated (especially if the compiler uses special methods to optimise size as well as speed).

There is a form of compiler that lies between the two, which translates the source, not into an object program, but into an 'intermediate' program. This program then runs under a special 'slimmed down' interpreter, known as the run-time system. This system interprets the intermediate program rather faster than the usual interpreter would with the source program, but is obviously not as efficient as having an independent object program executing in machine-code. However, the intermediate program will be much more compact and, because of the simpler translation, compiling is very quick and straightforward.

Version 5.10 of the Xtal BASIC Compiler is of this latter form, and works in the following way:

1. The program XC.COM compiles a Xtal BASIC Source file (e.g, PROG.XBS) into an intermediate file (PROG.XBI)

2. The program XR.COM is then used to run this intermediate file.

Thus XC.COM is the compiler, and XR.COM is the intermediate interpreter or run-time system. While not operating at anything like machine-code speed, a run-time speed improvement of 1.5 to 5 times is possible, and the speed improvement tends to be greatest in very large programs.

The compiler is compatible with the same version of Xtal BASIC that it supports, and any program that runs under the interpreter should run under the compiler as well, with the following notable exceptions!

1. Programs using POKE/DOKE/PEEK/DEEK to locations within the BASIC interpreter itself. Clearly, these locations would not apply under the run-time system (note that PTR still works, where applicable). However, PEEKs/POKEs etc to .OBJ files or DOS/MOS scratch-pad locations should work fine.

2. The EVAL function is NOT supported.

3. The following commands do not have any effect under the run-time system (nor would it be sensible for them to do so!):

AUTO, DEL, LIST, MGE, NEW, REM, RENUM.

4. References to files may cause problems in that the .XBS default will now become .XBI under the compiler. Beware of any data files with a .XBS extension, which would now not be found, due to the fact that the system would be looking for a .XBI file!

5. Under release 1.0, the 'semi-chain' facility allowed under the interpreter will NOT work with the compiler, and the compiler will flag as errors the presence of any HOLD commands within programs. However, normal chaining between programs IS allowed.

II. USING THE Xtal BASIC COMPILER

1. Creating a Program.

It is assumed that the user is already a registered user of Xtal BASIC (you will only have been sold a copy of the compiler under that assumption, or that you will have purchased both interpreter and compiler at the same time!). Anyway, before a program can be compiled, it must first be entered, edited (and preferably tested) under the interpreter. Indeed, the normal development cycle for a program would be to design and test it using the interpreter (very convenient for debugging), and then to submit it to the compiler to bring it up to better efficiency. We do not describe the detail of editing/testing of your source program here, but instead refer you tc the BASIC manual(s) for a fuller description.

By way of an alternative, a source file may be entered and edited using a text editor or word processor program, in the form of a text (.ASC) file. However, such a file must then be LOADed under the Xtal BASIC interpreter and saved sgain as a Xtal BASIC Source (.XBS) file, before being compiled. This is because the compiler only recognises the compressed form of source file, and will not act on text alone.

2. Compiling a Program

Compiling a Xtal BASIC source file is quite straightforward. The compiler is XC.COM and let us suppose that the program to be compiled is called HANGMAN.XBS, on the default drive, together with XC.COM.

XC HANGMAN will then compile it, saving the result as HANGMAN.XBI on the same drive.

If it is desired to compile a program on a disc separate from the one containing XC.COM, just specify the drive name(s) in the command, e.g,

O:XC 1:HANGMAN compiles the program HANGMAN.XBS on drive 1, where the compiler is resident on drive O. The intermediate file HANGMAN.XBI will be created on drive 1. Note that the source and intermediate files must both be on the same drive.

Alternatively, XC may be invoked without specifying a file, in which case a '*' prompt appears. The file to be compiled can then be typed in or perhaps the disc can be swapped first, if the file(s) to be compiled are on a different disc from the one containing XC.COM , e.g,

```
O:XC
Xtal BASIC Compiler 5.10
*HANGMAN
Pass 1 Pass 2 Pass 3
Compiled OK
*
```

In this case, the prompt reappears on completion of the command, and either another command should then be typed, or else a warm boot can be performed by pressing <ENTER>.

XC is a three-pass compiler, which means that it makes three passes through the source file. However, the whole program is in memory at one time, making the compiling process much faster than it would be if the program were continuously read from disc. This limits source program size to about 40k, but this is no problem, since all source programs will have been created under the Xtal BASIC Interpreter anyway.

All being well, each pass should be shown as it takes place, followed by the message 'Compiled OK'. The .XBI file is then written back to disc, and control returns to DOS. Any error encountered is simply shown as such, and the compilation terminates straightaway, without making a .XBI file.

Pass 1 makes a list of line numbers used, marking the references within the program, and flags any undefined lines as errors (i.e, places which would have caused 'Sranch Errors' under the interpreter). It is sometimes possible for this pass to show up errors which may not have been noticed under the interpreter, since the routines containing the offending line numbers could be redundant (i.e, never used!).

Pass 2 next removes all unnecessary spaces and REM statements, and converts all numeric constants and variable names into tokens, to speed up their evaluation under the run-time system.

Finally, pass 3 rechecks all line number references (for example in GOTO/GOSUBs) and replaces them with relative jump offsets to their actual destinations. The line number table can then be dropped, the table of numeric constants is appended to the program, and the intermediate program thus formed is saved back to disc.

3. Running Compiled Programs.

To run your compiled program, use the run-time system program XR.COM. In the example from above, just do

XR HANGMAN and you will first be rewarded with the title display for the run-time system, followed by the start of the program itself.

As for the compiler, it is possible to specify the drives separately for the system and program, e.g,

2:XR 1:HANGMAN which loads XR.COM from drive 2 and then runs HANGMAN.XBI from drive 1.

4. Compiling and Running a Suite of Programs.

The above is all very well when compiling a single program, but it is sometimes necessary to modify this procedure for a suite of programs, in particular when variables are shared between two or more programs. In such a case, programs sharing variables must be compiled at the same time in a single command, for example:

XC ACCOUNTS ACCT1 ACCT2 ACCT3 compiles the four programs ACCOUNTS.XBS, ACCT1.XBS, etc., to form the programs ACCOUNTS.XBI, ACCT1.XBI, etc.

Note that it is only necessary to do this where the programs call each other by means of CHAIN commands -- if they invoke each other by means of RUN commands, they may be compiled by separate commands. Moreover, if you have too many files to fit on one command line, just invoke XC by itself and type in one or more files at each '*' prompt, until you have compiled them all.

III. ERROR MESSAGES

When compiling a program, there are several ways in which things may 'go wrong', and any one of the following errors may occur during compilation. Most of these errors are due to limitations on how the compiler uses tables to store constants line numbers and variables, but we have not found any problems with these limits so far, even with test programs of well over 30k in length.

1. Compiler Errors.

Source Name Needed
This simply means that the compiler has been invoked without specifying a valid source program.

No File
You have probably used the wrong name, or given the wrong drive for the source program.

Constant Table Full
The source program cannot contain more than 256 different floating-point constants. However, any number of integers in the range 0 to 65535 may be used.

Variable Table Full
The source program cannot contain more than 512 different simple variable names.

Array Table Full
As for constants and simple variables, the source program cannot contain more than 512 different array names.

Line Number Table Full
The maximum number of lines allowed in a single source program is 1024.

Disc Full
The intermediate file cannot be written back to disc, owing to lack of space.

Directory Full
The intermediate file cannot be closed, because the directory has no more space for directory entries.

Memory Full
An unlikely message, which will only happen if the source program is too large for the compiler to handle, or if the program grows too large during compilation. Again, this is very unlikely, since programs will generally get much smaller when compiled. The only way in which programs could get larger would be if very large numbers of single-character variable names are used (all variable names are compiled into two-byte tokens).

'HOLD' not allowed
'EVAL' not allowed
These two messages mean that the program for compilation contains either a HOLD command or EVAL function. As previously stated, these are not supported under version 5.10 of the compiler.

Reference Error(s)
This occurs if a line number referred to by e.g. GOTO/GOSUB is undefined. Pass 1 is used to make a list of line numbers and to mark references, so a list of bad references is then displayed and the message given.

Overflow Reference Error(s)
One or more line number references are larger than 65535.

Numeric Overflow
This occurs if a constant in the program is found to be too large, for example, 1.7E57. Note that the compiler converts all constants into tokens, apart from the single digits 0-9, which stay in their ASCII form. Floating-point constants are stored in a separate table appended onto the program, and are all calculated at compile time.

2. Run-time System Errors.

The errors which can occur while running the intermediate program are basically the same as those which would have occurred if the source program had been running under the interpreter. The only difference is that, because all line numbers have been stripped from the intermediate program, it is not possible to determine where the error has occurred! This makes it all the more important to test your programs under the interpreter first, before submitting it to the compiler.

The latest release of XSM, 1.05, overcomes a few bugs and makes some improvements, too. These are as follows:

1. Expressions.

The '/' operator is now allowed in expressions, which performs a 16-bit integer division, returning the integer part of the result (i.e, chopped, not rounded). Previously, only addition, subtraction or multiplication were allowed.

Example: Suppose the symbol XYZ has been defined as 73A6H:

```
        LD      A,XYZ/256       ; Puts high byte of symbol XYZ into A
```

The code generated would then be 3E 73.

2. Error Handling.

It was unfortunately very easy to 'fool' earlier releases of XSM, which tried too hard to make what it could of whatever was thrown at it! This led to a lot of 'howlers' being passed as correct code. The problem has been pretty well cleared up now, and an extra error code, 'W' for 'Warning' may be produced.

The 'W' error indicates that the assembler has found code other than a comment following what it thinks should be the end of an instruction. In many cases, there is no problem and the correct machine code has been generated (for instance, a common error is to type ':' instead of ';' to start a comment). Inspection of the error line as thrown up on the screen will then tell you whether editing and re-assembly is required.

3. Conditional Assembly.

There were some problems with the handling of IF, ELSE and ENDIF directives, which have now been cleared up. Additionally, the 'false' code (i.e, the code which is NOT generated as a result of the IF test) is not listed to the listing (or .PRN) file.

4. MCAL Routines.

An additional opcode has been defined for EINSTEIN owners, to allow the direct assembly of MCAL routines. This takes the form

```
        MCAL    ROUTINE         where ROUTINE is the routine number
```

required. For example, 9EH is the routine number to output a character from the A register to the screen, so that the following:

```
        LD      A,'H'
        MCAL    9EH             ; outputs 'H' to the screen.
```

Previously, this would have been required:

```
        LD      A,'H'
        RST     08H
        DEFB    9EH
```

5. Support of HITACHI 64180 Microprocessor.

Although this does not apply to EINSTEIN owners, many micro enthusiasts are investigating this relatively new microprocessor, which is really a 'super Z80' containing many hardware enhancements. It also implements some extra instructions, and XSM has been extended to allow them, without affecting the existing instructions supported. Obviously, on a Z80, although the 64180 instructions will generate code, they are undefined opcodes as far as the Z80 is concerned, and the results of using them will be equally undefined!

Full details can be found in the Hitachi document entitled 'HD64180 USER'S MANUAL' but, for completeness, here are the extra opcodes:

MLT	rp	Multiply registers in register pair rp, placing 16-bit result back in rp. For example, MLT BC multiplies B by C, placing the result back in BC. rp can be BC, DE, HL or SP.
TST	r	Test register r by ANDing it with register A, without affecting A. r can also be (HL) in this case.
TST	n	AND register A with number n (00-FF), again without affecting A.
TSTIO	n	AND contents of I/O port addressed by C with number n, again affecting only the flags.
IN0	r,(n)	Input from port n to register r. In both this and the OUT0 instruction, the top 8 address lines are zero during the I/O read/write.
OUT0	(n),r	Output register r to port n.
SLP		Enter SLEEP (low power consumption) mode.
OTIM OTDM OTIMR OTDMR		These are block output instructions, copying memory addressed by (HL) to the I/O port addressed by C. For OTIM and OTIMR, HL and C are then incremented, whereas for OTDM and OTDMR, HL and C are decremented. In all cases, B is then decremented and, in the cases of OTIMR and OTDMR, the sequence is repeated until B=0.

We hope that the above is of interest, even though it is of no use except when using the 64180 CPU.

CRYSTAL RESEARCH LTD
REG. IN ENGLAND No. 1629571

40 MAGDALENE ROAD
TORQUAY
DEVON
ENGLAND
Tel. (0803) 27890
VAT Reg. No. 313 3396 79

Using WORDSTAR (WS.COM -- 71 blocks)

Some users have complained that the version of WORDSTAR released by TATUNG on the EINSTEIN does not work under SYSTEM 5. On inspection, it was found that some patches had been made which made direct reference to DOS vectors, which was incorrect, as the vectors have moved. However, the pointers to those vectors are fixed, so that it is possible to make WORDSTAR work with past and future versions of Xtal DOS. In addition, the patches shown below remove or reduce some unnecessary delays inherent in the original WORDSTAR:

1. From DOS, do LOAD WS.COM
2. Enter MOS, and do 'M' commands as detailed below. Note that, for each alteration, we show the original value next to the address, in brackets (). If you should find that the new value is already present, it means that the alteration has already been made to your copy (lucky you!). So, for example, type M023A and, if it contains 0E, change this to 01, followed by 1D 00, do . and press <ENTER>. If you do find neither the old NOR the new value, check that you have the correct file in memory, and then contact us!
3. Finally, re-enter DOS, and do SAVE 71 WS.COM .

```
M023A (0E)  01 1D 00.
M025A (00)  01 15.
M028E (0A)  00 00.
M02AF (03)  02 06 10 18.
M4618 (22)  CD 1B C6 2A 64 C7.
M4683 (C3)  2A 62 C7 11 12 C6 CD 30 C6 21 09 C6 18 03
            21 12 C6 ED 58 62 C7 01 09 00 ED B0 C9.
M46F3 (22)  CD 29 C6.
M4711 (21)  2A 64 C7.
```


CRYSTAL RESEARCH LTD
REG. IN ENGLAND No. 1629571

40 MAGDALENE ROAD
TORQUAY
DEVON
ENGLAND
Tel. (0803) 27890
VAT Reg. No. 313 3396 79

June 1987

Dear Customer,

Please find enclosed updated copy of your SYSTEM 5 master disc, complete with Xtal BASIC Compiler. We have updated ALL of the files on the master disc and would ask that you take a backup copy straightaway, and discard any previous versions which we have supplied, as we can no longer guarantee them. This also applies to the DOS system area, as the DOS has now been updated to version 2.05.

We have included four BASIC programs for you to try with the compiler, in their .XBS form. These are as follows:

ERASTO.XBS — 'Sieve of Erastothenes', a bench-mark program which generates prime numbers. As supplied, it only prints out the total number of primes generated, so to actually display the primes (there are 1899 of them!), just remove the REM in front of the PRINT P, statement.

FRUIT.XBS — Fruit-machine game. Just like the arcade one-armed bandit version, except that there's no money to change hands!

HANGMAN.XBS — A simple, non-graphic version of the HANGMAN game, but it contains a fair library of words (some rather obscure!).

GRUFF.XBS — Find your way through the maze, trying to escape from the mad beast that wants his breakfast (i.e, you!).

We hope that you like the compiler -- if you do, please tell everyone you can -- if you don't, please tell us!

Yours faithfully,

Mr T F Brownen,
Managing Director,
Crystal Research Ltd.

Directors: T. F. Brownen, G. M. Brownen, A. J. Cornish, B.Sc.

```
100 27F 12
0100 C3 1A E5 18 48 25 02 3F 00 00 00 00 00 00 00 00 00 00 C.e.H%.?..........
0112 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..................
0124 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..................
0136 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 63 ..................c
0148 73 09 E2 00 00 06 25 B8 CA 16 E5 D3 24 C7 00 00 00 00 s.b...%8J.eS$G....
015A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..................
016C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..................
017E 00 00 00 3A 45 FB CB 57 28 0A 3E 0A D3 48 3E 60 D3 49 ...:E{KW(.>.SH>'SI
0190 CF D1 CF 9B C8 3E 00 00 00 00 00 00 00 00 00 00 3E OQO.H>..........>
01A2 0D CD A8 E2 3E 0A E5 D5 C5 F5 5F 0E 02 CD 05 00 F1 C1 .M(b>.eUEu_..M..qA
01B4 D1 E1 C9 CD 2D E4 AF 32 96 E2 11 76 E2 0E 0F 18 39 0E QaIM-d/2.b.vb...9.
01C6 11 11 76 E2 0D FE E2 C0 F5 CD 8A E3 4E 6F 20 46 69 6C ..vbM^b@uM.cNo Fil
01D8 E5 F1 C3 D5 E5 0E 13 18 1D 11 76 E2 0E 14 D5 C5 CD 05 eqCUe.....vb..UEM.
01EA 00 C1 D1 B7 C9 11 76 E2 0E 15 18 F0 11 76 E2 CD DD E2 .A@7I.vb...p.vbMJb
01FC 0E 16 CD E6 E2 3E 0A E5 D5 C5 F5 5F 0E 0B CD E6 E2 C8 0E ...Mfb<I...._..Mfb
020E 01 CD E6 E2 FE 1B 37 C8 B7 C9 0E 19 18 CA 11 80 00 0E .Mfb^.7H7I...J....
0220 1A 18 C3 C5 0E 1B CD E6 E2 FD 21 00 00 2B 06 08 23 4E ..CE..Mfb}!..+..#N
0232 CB 01 38 02 FD 23 1B 7A B3 28 04 10 F3 18 ED FD E5 E1 K.8.}#.z3(..s.m}ea
0244 C1 C9 0E 1F CD E6 E2 23 23 7E D6 02 C9 1E FF 0E 20 18 AI..Mfb&#~V.I... .
0256 CA 0E 26 18 C6 47 04 05 21 01 00 28 03 29 10 FD EB 0E J.&.FG..!..(.).}k.
0268 25 18 B6 CD 51 E3 87 87 87 87 47 3A 04 00 E6 0F B0 32 %.6MQc....G:..f.02
027A 04 00 C9 06 00 18 ..I...
>
```

E:56 3E C1 32 F4 E7 C3 1A E5
LDA, 01 LD(FFF4), A JP E51A

F5 F0

```
T E200 E37F 12
E200 C3 1A E5 18 48 25 05 3F 03 4D 4F 53 00 00 00 00 00 00 C.e.H%.?.MOS......
E212 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..................
E224 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..................
E236 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 63 ..................c
E248 73 0C E2 09 E2 06 25 B8 CA 16 E5 D3 24 C7 00 00 00 00 s.b.b.%8J.eS$G....
E25A 00 00 00 00 00 00 B2 E2 B8 3A 19 00 00 00 FD E3 EB E2 ......2b8:....}ckb
E26C 9A E4 EB E2 26 08 08 00 EA E9 01 4D 4F 53 20 20 20 20 .dkb&...j1.MOS
E27E 20 20 20 20 00 00 00 57 28 0A 3E 0A D3 48 3E 60 D3 49       ...W(.>.SH>'SI
E290 CF D1 CF 9B C8 3E 00 00 00 00 00 00 00 00 00 00 3E OQO.H>..........>
E2A2 0D CD A8 E2 3E 0A E5 D5 C5 F5 5F 0E 02 CD 05 00 F1 C1 .M(b>.eUEu_..M..qA
E2B4 D1 E1 C9 CD 2D E4 AF 32 96 E2 11 76 E2 0E 0F 18 39 0E QaIM-d/2.b.vb...9.
E2C6 11 11 76 E2 CD FE E2 C0 F5 CD 8A E3 4E 6F 20 46 69 6C ..vbM^b@uM.cNo Fil
E2D8 E5 F1 C3 D5 E5 0E 13 18 1D 11 76 E2 0E 14 D5 C5 CD 05 eqCUe.....vb..UEM.
E2EA 00 C1 D1 B7 C9 11 76 E2 0E 15 18 F0 11 76 E2 CD DD E2 .A@7I.vb...p.vbMJb
E2FC 0E 16 CD E6 E2 3C C9 0E 17 18 DF 0E 0B CD E6 E2 C8 0E ..Mfb<I...._..Mfb.
E30E 01 CD E6 E2 FE 1B 37 C8 B7 C9 0E 19 18 CA 11 80 00 0E .Mfb^.7H7I...J....
E320 1A 18 C3 C5 0E 1B CD E6 E2 FD 21 00 00 2B 06 08 23 4E ..CE..Mfb}!..+..#N
E332 CB 01 38 02 FD 23 1B 7A B3 28 04 10 F3 18 ED FD E5 E1 K.8.}#.z3(..s.m}ea
E344 C1 C9 0E 1F CD E6 E2 23 23 7E D6 02 C9 1E FF 0E 20 18 AI..Mfb&#~V.I... .
E356 CA 0E 26 18 C6 47 04 05 21 01 00 28 03 29 10 FD EB 0E J.&.FG..!..(.).}k.
E368 25 18 B6 CD 51 E3 87 87 87 87 47 3A 04 00 E6 0F B0 32 %.6MQc....G:..f.02
E37A 04 00 C9 06 00 18 ..I...
>
```

E2.05 SIL DISC PATCH