CRYSTAL RESEARCH LTD

Xtal SYSTEM 5.0 No. X5000048

A Xtaltron® Product

For Einstein Systems

40 MAGDALENE ROAD
TORQUAY
DEVON
ENGLAND
Tel.(0803) 27890

```
 SSSSSS  Y     Y   SSSSSS  TTTTTTT  EEEEEEE  M     M    5555555
S         Y   Y   S            T     E        MM   MM    5
S          Y Y    S            T     E        M M M M    5
 SSSSS      Y      SSSSS       T     EEEEE    M  M  M     SSSSS
      S     Y           S      T     E        M     M         5
      S     Y           S      T     E        M     M         5
SSSSSS      Y     SSSSSS       T     EEEEEEE  M     M    555555
```

SYSTEM ENHANCEMENTS PACKAGE FOR THE TATUNG EINSTEIN COMPUTER
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 3 7
(2)

* INTRODUCTION *

This Xtal SYSTEM 5 enhancement package provides a number of worthwhile improvements to the existing TATUNG EINSTEIN microcomputer system, and includes some new utility programs not found on the original system disc. Here is a short summary of the enhancements provided by the SYSTEM 5 package:

BASIC
- Extended floating-point precision, 6-16 significant figures.
- 80-Column card support.
- Strings of up to 32k in length, with ability to store complete screens in a single string.
- Structured loops, including REPEAT/UNTIL and WHILE/WEND, to add to the existing FOR/NEXT loops.
- BAUD rate setting command.
- Full screen dump to printer, including graphics.

DOS
- File handling speed 2-4 times faster.
- Support of 3" and 5.25" double-sided disc drives, for up to 400k disc capacity (386k under Xtal DOS).
- BACKUP.COM — Format/Backup utility, enhanced for double-sided drives.
- COPY.COM — File transfer/copy utility, some enhancements.
- XSM.COM — A straight-forward Z80 assembler, using ZILOG mnemonics, conditional assembly, source file INCLUDEs.
- XED.COM — Screen editor, supporting 40- and 80-columns.
- AUTO.COM — AUTOGEN auto-boot command-line generator.
- TIME.COM — Set and display real-time clock.

This manual is not designed to replace the existing EINSTEIN manuals for Xtal BASIC and Xtal DOS, rather to be used in conjunction with them, explaining the differences/improvements. All of the enhancements are designed to be compatible with the earlier version 4 BASIC and version 1 DOS.

We are always interested in comments and suggestions -- indeed, some of our most innovative ideas have resulted from customer 'wish lists', so do feel free to call or write to us about them.

****
* Xtal BASIC VERSION 5 FOR EINSTEIN *
****

Xtal BASIC 5 has been designed to be upward compatible with the existing version 4 that EINSTEIN users already know (and love, we hope!). It does, however, make several significant improvements which, we feel, will prove to be very worthwhile to the users, including the following:

1. 80/40/32-column screen handling and editing.
2. Floating-Point enhancements, including higher-precision arithmetic on all functions, including all transcendental functions.
3. String enhancements, in particular the allowing of string lengths of up to 32767 characters.
4. Inclusion of statements for structured loops, including WHILE/WEND and REPEAT/UNTIL.
5. The facility for dumping graphics to printer as well as text (in 32- and 40-column mode).
6. The alteration of serial port baud rates from within BASIC.

This has been done at the expense of about 2.5k of user space (you now have just under 41k to play with -- but then, nothing comes free these days!). In the main, the information found in the EINSTEIN manuals is still valid, the only differences being in these commands/functions:

CLEAR

The CLEAR command has not been changed, but this is included as a note that, in some programs which have used CLEAR N to set aside space for machine-code or data, it may be necessary to alter the value of N. This is not due to changes in BASIC, rather to changes in DOS, which starts 0.5K lower than in version 1.

CLS80

We already know CLS40 and CLS32, which allow clearing of the screen and setting of 40 and 32 columns. As we should expect, CLS80 allows us to clear to 80-column mode. If the 80-column card is not present, this command does nothing, i.e, the current screen mode will stay as it is. CLS by itself keeps the current screen mode while clearing the screen, again as we should expect.

FPI

The FPI command sets the overall floating-point displayed accuracy in significant figures. Note that this command has the side-effect of clearing all variables, and so should normally be used at the start of a program, or from direct mode. The FPI command is followed by an expression evaluating to an integer in the range 6 to 16. The appropriate internal working accuracy is then automatically set up by BASIC, and will subsequently apply to all functions and operators. Unlike most BASICs, this also applies to ALL transcendental functions, such as SQR, EXP, LOG, SIN, etc.
Example: FPI 9 Sets 9 significant figures, all internal storage for numbers in variables, etc being set to five bytes. The maximum capacity of the system is eight-byte storage, corresponding to a maximum accuracy of 16 digits.

SCRN$(255)

The SCRN$ function previously allowed arguments up to the number of lines on the screen (minus one!). This still applies, but we now also allow the argument 255, which represents the entire screen, allowing us to copy the whole screen to a string variable, or element of a string array, or even to a file, in a single command.
Example: A$(8)=SCRN$(255) copies the contents of the screen to the array element A$(8).

REPEAT/UNTIL

The REPEAT command indicates the start of a loop, which is terminated by an UNTIL statement. The REPEAT command appears by itself, and the UNTIL statement contains a conditional expression which, if true, causes execution to pass back to the statement directly following the REPEAT. If true, the loop terminates and execution continues with the statement following the UNTIL statement.

Example:

```
10 I=2                          is equivalent to:    10 I=2
20 REPEAT
30 PRINT I,                                          30 PRINT I,
40 I=I*I                                             40 I=I*I
50 UNTIL I>100000                                    50 IF I<=100000 THEN 30
60 END                                               60 END
```

When RUN, these both produce the output:
2 4 -16 256 65536

The big advantage of the REPEAT/UNTIL version is, of course, that the line number is not given, and cuts out the search for the line number. This does not matter in small programs, but can be very time-consuming in larger programs.

REPEAT/UNTIL loops are, like FOR/NEXT loops, always executed at least once, since the loop test is made at the end of the loop (as in the NEXT statement). Again, like FOR loops, they may be nested.

WHILE/WEND

As for REPEAT/UNTIL, the WHILE/WEND construct allows us to make structured loops, without resorting to specification of line numbers. In this case, however, the loop test is performed at the start of the loop, meaning that it is possible that the loop will not be executed at all. If the condition evaluated after WHILE is true, the body of the loop will be executed. Otherwise, the program is scanned forward for a matching WEND command, and execution continues from after this WEND command. Rewriting our previous example as for REPEAT/UNTIL we obtain this:

```
10 I=2
20 WHILE I<=100000
30 PRINT I,
40 I=I*I
50 WEND
60 END
```

It is perfectly reasonable to nest WHILE loops, just as for FOR and REPEAT loops. However, 'responsible' programming is called for, since scanning for WEND is done in the forward direction through the program.

Because the loop test is performed at the start, BASIC does not know where the end of the loop is (with NEXT and UNTIL, the start of their respective loops is held on the stack). Here is an example of 'irresponsible' programming to illustrate the point:

a.
```
10 I=2: GOTO 40
20 UNTIL I>100000
30 END
40 REPEAT
50 PRINT I,: I=I*I
60 GOTO 20
```

b.
```
10 I=2: GOTO 40
20 WEND
30 END
40 WHILE I<=100000
50 PRINT I,: I=I*I
60 GOTO 20
```

Program a., though horrible, will work just as the original REPEAT/UNTIL example. Program b., no more horrible, will work until I becomes greater than 100000, at which point a 'Loop Error' will occur, because the WEND cannot be found after the WHILE.

RST

This command was, in fact, available on earlier versions of Xtal BASIC, but was undocumented, because nobody could make up their mind exactly what features were to be covered by it! Well, here is the documentation anyway!

RST is a 'reset' command. Its purpose is to set defaults for various input/output commands, such as SEP, IOM, ZONE, etc. For instance, suppose a program uses SEP 0. On terminating, the separator is still zero. Then we run another program, which requires a comma separator. The programmer should remember to set SEP 44, otherwise there is a risk that the program will crash. There are quite a few of these examples, and it is not always convenient for them to be reset by termination of a program. In addition, certain operating system/MOS defaults may need to be reset. The RST command, then, resets the following:

a. ICOL 15,0: GCOL 15,0 (NB, the backdrop (BCOL) is NOT reset).
b. SPEED 255
c. ORIGIN 0,0
d. Reset all IOM flags to 1.
e. FMI 0,0
f. FPI 6 -- Does not apply to version 4.
g. SEP 44
h. ZONE 28,10
i. WIDTH 0
j. CLS 40
k. SPRITE OFF
l. Reset FDC and PSG
m. Restore default character set (reset all SHAPEs).
n. Reset TAB character to space.
o. Reset MOS BREAK vector at 0038H
p. Reset BASIC MOS jump vectors (shift-BREAK, X & Y commands).
q. Disable key, ADC and fire button interrupts (these are NOT normally used).
r. Set cursor ON.

Note that RST clears neither the program nor the variables.

Graphics Dump To Printer

A screen dump to printer can be produced by typing Ctrl-A, or by means of PRINT CHR$(1), but this is for text only, and produces output in the printer's own character font. However, Xtal BASIC 5 now allows us to dump graphics to the printer, from 40- and 32-column modes, by typing Ctrl-B, or by means of PRINT CHR$(2). This will print both text and graphics together if they are mixed on the screen, although the text will appear in the form as displayed on the EINSTEIN and not in the printer's own font.

We cannot of course guarantee that this will work with all printers run from the EINSTEIN, but we have verified that the following all work OK:
TATUNG TP80 EPSON MX80 FTIII STAR MICRONICS SR15.
We shall be interested to hear from anyone who has a printer supporting dot graphics other than those listed above, whether the graphics dump works correctly or otherwise. For those that do NOT produce the desired output, it may just be possible to provide a straightforward 'patch' to get it working!

BAUD

The BAUD command allows the setting of the baud rate and other facilities on the 8251 Programmable Communications Interface (PCI) chip used on the EINSTEIN. In previous versions, the baud rate could only be altered from MOS, or by means of OUT commands to the PCI and CTC ports.

Normally, the two transmit and receive rates must be supplied, as for the 'B' command in MOS, in the range 0 to 8, as follows:

```
0 --  75 baud        5 -- 1200 baud
1 -- 110 baud        6 -- 2400 baud
2 -- 150 baud        7 -- 4800 baud
3 -- 300 baud        8 -- 9600 baud
4 -- 600 baud
```

The transmit and receive baud rates must both be given, although they will usually be the same:

Example: BAUD 5,5 sets 1200 baud.

In addition, other options may be set up within the BAUD command, by means of an optional third parameter which, if used, sets the 8251 Mode and Control bytes, as follows:

```
MODE    Bits 0,1        00 -- SYNC mode,        01 -- ASYNC mode,
                                                    Clock 1x baud rate
                        10 -- 16x baud rate,    11 -- 64x baud rate
        Bits 2,3        00 -- 5 bits/char,      01 -- 6 bits/char
                        10 -- 7 bits/char,      11 -- 8 bits/char
        Bit 4            0 -- Parity disable,    1 -- Parity enable
        Bit 5            0 -- Odd Parity,        1 -- Even Parity
        Bits 6,7        00 -- Invalid,          01 -- 1 stop bit
                        10 -- 1 1/2 stop bits,  11 -- 2 stop bits
        Or, in SYNC mode (bits 0 & 1 set to 0):
        Bit 6            0 -- SYNDET output,     1 -- SYNDET input
        Bit 7            0 -- 2 SYNC characters, 1 -- 1 SYNC character

CONTROL Bit 0            0 -- Disable transmit,  1 -- Enable
        Bit 1           Set DTR output to opposite of bit value
        Bit 2            0 -- Disable RxRDY,     1 -- Enable RxRDY
        Bit 3           Send BREAK char, normally 0
        Bit 4           Error reset, resets any error conditions if 1
        Bit 5           Set RTS output to opposite of bit value
        Bit 6           Internal Reset, 1 -- Accept mode byte next
        Bit 7           Enter Hunt mode if 1 (SYNC mode only)
```

In each case, bit 7 is most-significant, and the mode byte is the most-significant of the two. The default setting is binary 1100111000100111, or &CE27, or 52775.

So the command BAUD 3,3,&CE27 sets the receive and transmit clocks to 300 baud, and the chip to 2 stop bits, no parity, 8-bit data, 16x baud rate factor, and enables Rx and Tx signals, together with the RTS handshaking signal for receiving. Note that we use the PCI in ASYNCHRONOUS mode.

More data on the 8251 chip can be gleaned from the appropriate INTEL CORPORATION data sheet.

****
* Xtal DOS VERSION 2 FOR EINSTEIN *
****

Xtal DOS 2 has been designed for Z80-based microcomputers, particularly for the TATUNG EINSTEIN computer, where it replaces the existing Xtal DOS version 1. In addition to overcoming a couple of minor 'bugs' in version 1, the main enhancements of Xtal DOS 2 lie in increased speed of disc access (in many cases up to a four-fold improvement!) and in support of double-sided and 80-track drives. A very worthwhile increase in storage capacity can be achieved, as the following list of possible configurations will show:

40-Track, Single-Sided (188k) -- The original 3" drive is of this type.
40-Track, Double-Sided (386k)
80-Track, Single-Sided (386k)
80-Track, Double-Sided (786k)

The disc supplied is in single-sided format, assuming that the user has a single-sided drive as drive 0. In addition, the default set-up under Xtal DOS 2 is for drives 0 and 1 to be single-sided, and drives 2 and 3 to be double-sided (all 40-Track), but this can be altered by the user quite easily.

COPYING SYSTEM TRACKS TO OTHER DISCS

We recommend users to upgrade the system tracks of their own discs to Xtal DOS 2. As for version 1, this is quite straightforward, and may be performed under MOS thus:

1. Put XTAL DOS 2 disc in drive 0.
2. Ensure that you are under MOS.
3. Type the command R100 1B00 and press <ENTER>.
4. Replace the disc in drive 0 with one requiring the upgraded system tracks.
5. Type the command W100 1B00 and press <ENTER>.
6. Repeat steps 4. and 5. as required.

NOTE: When making up discs to send/give to other users, remember that the system tracks must be BLANK, unless prior arrangement has been made with Crystal Research Ltd. This can be done very easily using the same instructions as above, but substituting the following for 3.:

3a. Type the command F100 1AFF FF and press <ENTER>.

CONFIGURING THE SYSTEM FOR COMBINATIONS OF DRIVES

As previously stated, the default setting is for drives 0 and 1 to be single-sided, and drives 2 and 3 to be double-sided. It is very easy to alter this configuration:

1. Put Xtal DOS 2 disc in drive 0.
2. Ensure that you are under MOS.
3. Type the command R100 1B00 and press <ENTER>.
4. Type the command T100 107 and press <ENTER>. You should see the display
   0100 00 E1 00 FB 00 FB 0C FF
   Bytes 0100 and 0101 give the system load address for the first sector (E100H), bytes 0102 and 0103 give the end address plus 1 for the system (FB00H), and bytes 0104 and 0105 give the execution address after loading (F900H). Byte 0106H is the one which we are interested in. The lower 4 bits specify whether drives are single-sided (0) or double-sided (1), while the upper 4 bits specify 40/80 tracks (0 for 40 tracks, 1 for 80 tracks). In this example, bits 2 and 3 are set, indicating that drives 2 and 3 are double-sided and that all drives are 40-track (bit 0 is the least-significant bit).

5. Type the command M106 and press <ENTER>. This will allow you to inspect and alter the drive configuration. Type a '.' and <ENTER> after your alteration.
   Example: Setting address 106H to CC produces the following:

0: 40T SS 1: 40T SS 2: 80T DS 3: 80T DS

6. Follow steps 4. and 5. of the previous section as necessary to update as many system discs as required.

BOOTING TO OTHER SYSTEMS

Although we recommend upgrading system tracks to version 2, it is recognised that users will often insert discs in drive 0 containing earlier systems, or insert a version 2 disc when an earlier system has been running. In either case, when a warm boot or shift-BREAK occurs, the system checks for a change of system and, if so, performs a RESET instead. A cold boot (ctrl-BREAK) will be unaffected by this, since it reloads the entire operating system anyway.

BACKUP VERSION 2

The combined backup/format utility is now released as BACKUP version 2, which has two major enhancements -- sector skewing is formatted into discs for greatly increased disc file transfer speed, and support has been given to double-sided and 80-Track drives, allowing formatting of four different types of discs. The decision on whether to format in a particular type depends upon the drive selected, whether it has been previously configured for single- or double-sided operation, etc. The actual drive configuration is displayed on the screen while BACKUP is being used.

The other changes from version 1 are that the 'track map' has been replaced by a 'current track' display, and the track and sector are both shown after any error has occurred.

Here is a table to show the relative capacities of the four types:

	40T SS	40T DS	80T SS	80T DS
Sides	1	2	1	2
Tracks	40	40	80	80
Sectors/track	10	20	10	20
Bytes/sector	512	512	512	512
Formatted Size	200k	400k	400k	800k
System tracks	2	1	2	1
System Size	10k	10k	10k	10k
Directory track	2	1	2	1
Directory Size	2k	4k	4k	4k
Directory Capacity	64 files	128 files	128	128
User file space	188k	386k	386k	786k

(Formatted Size - System Size - Directory Size)

In the backup section of BACKUP, the user may NOT backup from a single- to a double-sided drive, as the number of reserved tracks and the directory track sizes differ. If you wish to copy files from a single-sided disc onto a double-sided disc, or vice versa, use COPY instead.

COPY ENHANCEMENTS

COPY version 1.4 is provided with Xtal DOS 2, although only a few changes have been made from earlier versions. Bugs in earlier versions of COPY have been 'fixed', including the bug that prevented the sending of an EOF code at the end of a file to the serial port. In addition, the <U> and <L> options did not work properly on earlier versions of COPY.

Apart from the fact that COPY now puts up a 'start-up' message when it runs up, the only enhancement that has been made is to the <O> option, sometimes used when sending/receiving files via peripheral devices. Any key pressed at the receiving end while the transfer is taking place stops receiving and abandons the data received, except for the Ctrl-Z (End of File) combination, which stops the receiving end from taking any more data, but saves the data taken to disc. Note that, to act on these keys, the receiving end must be in the process of receiving data, since the keyboard is only checked as each character is received.

TIME

The program TIME.COM allows the user to set and read the current time of day, utilising the EINSTEIN real-time clock. To read the time, just type:

```
TIME <ENTER>
```

To set the time to say, 1427 and 35 seconds, type:

```
TIME 142735 <ENTER>        without spaces between the figures.
```

Trailing zeroes may be left out, e.g.,

```
TIME 1105        sets 11:05:00, and
TIME 21          sets 21:00:00.
```

In all cases, the current time will then be displayed on the console. If no time has been set, TIME tells you how long it has been since the system was switched on or RESET.

AUTOGEN

A feature of XtalDOS is that it allows for the insertion of a command line into the DOS monitor, so that this line will be executed whenever a cold boot is performed on the disc containing this command line. This can be used to create a 'turn-key' system, and is thus a most useful facility. AUTOGEN is a utility which inserts the desired command line into the DOS monitor, and is contained in the program AUTO.COM .

To insert an auto-boot command line, just type AUTO <ENTER>. The program asks for the name of the drive containing the disc to be updated.

The user is next prompted for the command line to be inserted and, after this has been typed in, the line is stored into XtalDOS on that disc, and the program terminates. On performing a COLD boot with this modified disc, the new command line will be executed before allowing the user to type anything in so that, for instance, in the following example, the directory will be displayed every time the system is rebooted from scratch (i.e, via switch-on, RESET, or Ctrl-BREAK).

```
Xtal AUTOGEN 2.1
Drive to use (0-3)? 0
Command line to be executed?
:DIR
```

NOTE that a WARM boot does NOT execute the command line.

AUTOGEN may be used again to change this command line, or to remove it (by just pressing <ENTER> for the command line).


```
*********************************
* XSM -- THE Xtal Z80 ASSEMBLER *
*********************************
```

XSM is an assembler for the Z80 microprocessor. It uses standard Zilog /Mostek mnemonics, supports conditional assembly and produces object code in Intel HEX format, suitable for loading with the HLOAD utility. XSM is a disc-based assembler, i.e. source code is read from disc, the resulting object code is written to disc and an optional print file can be written to disc. Alternatively, the print file can be redirected to the console. Inclusion of other source files is allowed for, making it possible to assemble very large programs (for example, the assembler was assembled by itself!).

1. USING THE ASSEMBLER

The assembler is initiated by typing:-

```
XSM p:filename.qr
```

This takes the file 'filename.ASM' and assembles it according to the parameters 'p' 'q' and 'r' and the options chosen. The parameters must be selected as follows:

p - the drive from which the source file, 'filename.ASM' is read. This should be in the range 0: to 3:, or not given if the default drive is to be used.

q - the drive on which the listing is to be written, 'filename.PRN'. 0-3, blank for default drive, X to the console, Y to the printer, N to suppress listing.

r - the drive on which the object code, 'filename.HEX' is to be written 0-3, or blank for default drive, N if not required.

For example, assembling the file TEST.ASM, assuming drive 0 is default:

```
XSM TEST.01
```

This expects the file TEST.ASM on drive 0, will write the listing file, TEST.PRN to drive 0 and the object code file, TEST.HEX to drive 1.

2. ASSEMBLER SYNTAX

a. Mnemonics

The assembler recognises all standard Z80 mnemonics as defined in the Z80 CPU Technical Manual. The following mnemonics are recognised in both forms shown:

```
ADC A,s          ADC s
ADD A,n          ADD n
ADD A,r          ADD r
ADD a,(HL)       ADD (HL)
ADD a,(IX+d)     ADD (IX+d)
ADD a,(IY+d)     ADD (IY+d)
```

In the current release, the IM instruction is handled in the forms IM0, IM1 and IM2, NOT as IM 0, etc.

Unlike some assemblers, such as ZEN, XSM does not allow operation codes or assembler directives to start from the first column of the screen -- only labels may start in the first column.

b. Labels
Labels may be up to 11 characters long. The first character of the label must be alphabetic (i.e. A-Z) or '_' or '$'. A colon, ':' following the label is optional.

c. Constants
XSM allows binary, octal, decimal, hexadecimal and ASCII constants according to the following conventions:

Binary - number formed from binary digits (0,1) and terminated by the character 'B'. Range 0000000000000000B to 1111111111111111B.

```
Example:        LD      HL,100011111011O1B
```

Octal - number formed from octal digits (0-7) and terminated by the character 'Q'. Range 000000Q to 177777Q.

```
Example:        LD      HL,23670Q
```

Decimal - number formed from decimal digits (0-9) and EITHER left unterminated or terminated by the character 'D'. Range: 0 to 65535.

```
Example:        LD      HL,32145
```

Hexadecimal - number formed from hexadecimal digits (0-9 and A-F) and terminated by the character 'H'. A hex number beginning with a letter must be preceded by a 0 to distinguish it from a label or register name. Range 0H to 0FFFFH.

```
Example:        LD      HL,0A3FH
```

ASCII - number formed from ASCII value of characters enclosed in single quotes. Range ' ' to '~' or 20H to 7EH including all alphanumerics and punctuation.

```
Example:        LD      A,'*'
```

The $ sign used as a constant represents the current value of the program counter.

d. Expression Evaluation
Operands in instructions and pseudo-ops may consist of arithmetic expressions which are evaluated at assembly time and the calculated value used as part of the object code. The following operators may be used to form expressions:

```
+       Addition
-       Subtraction (or unary Negation)
*       Multiplication
```

Expressions are evaluated left to right with no precedence. No parentheses are allowed in expressions, so some care is required to ensure that expressions are correctly formed.

e. Assembler Directives

DEFB (define byte)
The DEFB pseudo-op tells the assembler to reserve a byte or string of bytes with specific values. The bytes may be defined as constants, expressions as described earlier or as previously-defined symbols.

```
<label>:        DEFB    <item or list of items>

Example:
        DEFB    3,-5,34+LABEL,'A'
```

DEFS (define storage)
This tells the assembler to reserve a specified number of bytes for storage. Note that the data stored in the bytes is not defined.

DEFW (define word)
This is exactly the same as the DEFB Pseudo-op except that the data is stored in words (2 bytes each) rather than bytes.

END (end of assembly)
This directive is used to terminate assembly of a block of source code.

ENDIF (end of IF definition)
Used to terminate a block of conditionally assembled code. See IF directive for further details.

EQU (equate)
The EQU directive assigns a value to a label or symbol. The format is:

```
label:  EQU     <item>
```

Label is the name of the symbol and item is any of: a constant, an address, a label or an expression. Once the value of the symbol is assigned it is in effect for the entire source program.

IF (begin conditional assembly)
The IF, ELSE and ENDIF directives define a sequence of assembly language statements which are to be included or excluded during the assembly. The form is:

```
        IF      <expression>
        statement 1
        statement 2
          .....
        statement n
        ELSE                    (optional)
          .....                   "  "
          .....                   "  "
        ENDIF
```

Conditionals may not be nested. Values used in the expression must be defined before the IF statement. When the assembler finds an IF directive, the expression is evaluated. If the expression evaluates to a logical TRUE (non-zero) then the statements following, up to the ENDIF are included in the assembly. The optional ELSE directive allows alternate code to be generated when the opposite condition exists.

INCLUDE (file inclusion)
The INCLUDE directive temporarily stops assembling from the original source file, and carries on with a new source file. The format is:

```
INCLUDE filename
```

The filename is another .ASM file, which is then opened and read just as if its lines had been included in the original source file. This allows large programs to be split into more manageable chunks for easier editing, or for useful routines to be grouped together within their own source file. INCLUDEs may not be nested, but there is no limit to the number of INCLUDEs which may be placed within the original source file. Of course, the user must ensure that no END directive appears in the included file, otherwise assembly will stop at that point!

ORG (set origin)
The ORG directive tells the assembler to set the internal 'program counter' to the specified address, and then to generate code starting from that address. The format is:

```
ORG     <expression>
```

Several ORGs may be used in a source file, but at least one should normally be given, since code would otherwise be generated assuming an origin of 0000H. The normal procedure, for programs to start at 100H, would therefore be to insert the following line at the start of the source file (before any code-generating lines are encountered):

```
ORG     0100H
```

f. Comments.
Remarks or comments can include any printable ASCII characters, starting with a ';'. The comment may follow an opcode, operand or label or may exist on a line by itself.

3. ERROR MESSAGES

XSM generates a number of error messages while assembling to inform you of its progress. They fall into two categories: errors found while setting up files for assembly operation and those encountered while assembling the file. The first type of error message will be one of the following:

```
Source (.ASM) file not found.
Unable to open object (.HEX) file.
Unable to open list (.PRN) file.
```

The first message is generated when the filename specified in the command line cannot be found on the selected disc. The other messages occur while the output (HEX and PRN) files are being set up, usually the disc or directory is full. These errors are sent only to the console and the assembly is aborted.

The following errors can occur during the assembly process. The offending source line is displayed on the console, starting with a letter indicating the type of error message. If a listing file has been selected, the error appears in the listing as well as at the console. Errors are displayed even if no listing is being given.

I — IF directive error. Results from using ELSE or ENDIF without a matching IF, or if an attempt has been made to nest IFs.

R — Relative jump range error. The address specified in a relative jump instruction is out of the range -128 to +127.

F — Include file error. This occurs if the file specified by an INCLUDE is absent, or incorrectly specified.

O — Illegal opcode. The assembler cannot recognise the instruction.

M — Multiple definition. The symbol has already been defined.

S — Syntax error. This error message covers a multitude of sins. It is usually caused by misspelling or omission of a delimiter such as a comma or semicolon.

U — Undefined symbol. The symbol used in the instruction or expression has not been defined.

V — Value error. This error appears when the value of a symbol or expression exceeds the range allowable. For example, an index register displacement value must be in the range +127 to -128.

None of the above errors will cause assembly to stop, but to continue, listing any errors as they are found. However, there is one more error, which will cause assembly to abort:

SYMBOL TABLE OVERFLOW -- Means that there is no more space for symbols (i.e, labels and definitions) in memory. This is not likely to occur, since the source file would normally have to be very large indeed for the required number of symbols to be generated. If it does occur, it will happen on the first pass, before any other errors are found.

4. THE HLOAD UTILITY

XSM produces object files in the Intel standard 'HEX' format -- this is essentially a text file containing object code represented in hex as two ASCII characters per byte, organised in lines, each line showing the number of bytes in the line (usually 16, if generated by XSM), and a start address, of the first byte of this line. In order to produce executable files, the HLOAD utility is provided. This reads in a .HEX file, and produces a .COM file as output. HLOAD should be invoked thus:

HLOAD FILENAME where FILENAME is assumed to have the type .HEX . This will produce the file FILENAME.COM, which may subsequently be executed by typing FILENAME by itself in the usual way. An optional drive name may be given in the command, otherwise the default drive will be used. Either way, the .COM file is always written to the same drive as that from which the .HEX file was read.

Example: HLOAD 2:TEST loads the file TEST.HEX from drive 2, and then writes the file TEST.COM back to drive 2.

****
* XED -- THE Xtal SCREEN EDITOR *
****

XED is a general-purpose screen editor which is designed for operation with most Z80-based computers using CP/M or Xtal DOS. It is NOT a word-processor, since it does not support some of the facilities normally provided by such a package, but it is nevertheless quite easy to use and provides some powerful features. The version here for the TATUNG EINSTEIN automatically supports whichever screen format happens to be in use when XED is invoked.

1. RUNNING UP XED

There are two forms in which XED may be invoked from DOS:

a. XED by itself, or

b. XED <filename> where <filename> conforms to the usual rules, and may or may not exist. If it does not exist, then a file of that name will be created on the specified drive (or default drive if not given).

In either case, the editor comes up in edit mode, as indicated by a status-line display at the top of the screen. The status line gives the current line and column number, the current file name (if one has been given) and the mode.

Example: Suppose we wish to edit the file TEST.DOC, on drive 1. We type

XED 1:TEST.DOC

The editor runs up, displaying its start-up message and, after a few seconds, the first few lines of TEST.DOC appear on the screen, together with the status line:

Line 1 Col 0 TEST.DOC edit:

Note that the top line of a text file is line 1, but that the left-most column is column 0.

2. OVERALL OPERATION

Edit mode is the normal screen-editing mode, useful for entering text and general 'hacking about'. It is the mode that you will use most often. Command mode, on the other hand, is the one for loading, saving and inserting files of text, and making global searches and alterations within a file.

XED will work from either 40- or 80-column mode on the EINSTEIN, the screen being automatically configured to either mode, according to whichever mode you were in when running up XED. The top status line has already been described -- the rest of the screen displays up to 23 lines of text, at any one time showing a 'window' 40 or 80 columns wide. The maximum line length is 132 characters, and you may not enter characters beyond that length. If, however, you enter characters beyond the screen width, a 'horizontal scroll' of 20 characters occurs, this allowing you to continue typing on that line up to the 132 character maximum.

Clearly, it is still more convenient to work using the 80-column card, since there will then be less scrolling, but it does make 40-column editing a reasonable proposition, for those who do not yet have the 80-column card.

The INSERT and LITERAL switches

The INSERT switch is turned on from edit mode by means of the INS (shift-DEL) key, and is indicated by the word 'inst:' on the status line. This simply allows text entered on an existing line to be inserted into it rather than written over it, and moves any text right of the cursor to the right. In addition, when the <ENTER> key is pressed, rather than simply moving to the next line, a new line is inserted below that just typed. To turn off the INSERT switch, just press <INS> again.

The LITERAL switch is just a simple way of allowing the typing of control-characters into the text, these normally being reserved for editing facilities. It is turned on by means of a Ctrl-L from the keyboard, after which the status line will show 'lit:'. Typing any control character (e.g, Ctrl-A) will cause that character to appear as the appropriate letter/symbol with an up-arrow ('^') in front of it. Immediately after this, the status display will show 'edit:' again -- literal mode only lasts for one character, and another Ctrl-L will be needed to type in another.

3. EDIT MODE

In edit mode, a part of the text is displayed on the screen, with the screen cursor showing the current position in the file. Normal printable characters typed just appear in the text as typed (or inserted if the INSERT switch is on). However, there are also a number of useful editing functions provided by means of various control character combinations or special keys. Here are the screen editing functions available from the keyboard under edit mode:

<ESC> Enter Command Mode
The <ESC> key simply places you in command mode, with a prompt line formed at the bottom line of the screen, and the mode indicator at the top changing to 'cmnd:'. See section 4 for a full description of the commands available.

Arrow keys
As one would expect, the arrow keys just move the cursor in the appropriate direction. Note that the left and right arrow keys do not move onto another line from the ends of a line (for instance, the left arrow key will not move the cursor at all if it is already at the start of the line.

Ctrl-W Word Right
This key moves the cursor to the start of the next WORD in the line. It does not go onto the next line, but will stop at the end of the line.

Ctrl-Q Word Left
Moves the cursor one WORD to the left, to the start of that word. As for ctrl-W, this does not take the cursor off the line.

<ENTER> Newline
Moves to the start of the next line. In insert mode, it takes the characters right of the cursor with it to the next line, allowing you to split an existing line. Either way, the cursor ends up at column 0.

Ctrl-C Page Down
Moves to the next 'page', i.e, 22 lines down the text. This is handy for stepping quickly to a desired part of the text which may be fairly close by, but off the part of the text currently displayed.

Ctrl-R Page Up
Moves to the previous 'page', i.e, 22 lines back up the text.

Ctrl-I Tab
Moves the cursor to the next 'tab' point.

<DEL> Delete Character to Left
The DEL key deletes characters and moves the cursor to the left. If the cursor is at the end of a line, the last character of the line will be deleted, otherwise any characters to the right of the cursor will also be moved one place left.

Ctrl-<DEL> Delete Character at Cursor
The ctrl-<DEL> combination performs as for <DEL>, except that the cursor does not move. This has the effect of deleting the character under the cursor and moving the characters right of the cursor one place left.

<INS> Insert Mode
The <INS> key places you in INSERT mode. This means that non-control characters typed will be inserted into the line at the cursor, moving all characters right of the cursor one place right. All other control characters will behave as normal, except that the <INS> key acts as a toggle. So, to return to edit mode, just press <INS> again.

Ctrl-N Insert Line
Whether used in edit or insert mode, ctrl-N inserts a new line at the cursor, moving the rest of the screen down. It does not split the current line in any way. The cursor ends up at the start of the new inserted line.

Ctrl-P Join to Line Above
The ctrl-P combination joins the current line onto the end of the previous line. It does not matter where on the line the cursor starts, the cursor will end up at the join. If the resulting line would end up being too long (i.e, more than 132 characters, the message 'Line too long' will be shown at the top left of the screen, and the text left as it is.

Ctrl-U Clear to End of Line
The ctrl-U combination erases all characters from and to the right of the cursor.

Ctrl-X Delete Line
This is like ctrl-U, except that it deletes the whole line, bringing the rest of the screen below up one line. The cursor ends up back at the start of the line.

Ctrl-A Abandon Changes to Line
If characters have been typed on a line, or inserted/deleted, all of these changes may be aborted, and the line restored to its original state, as a copy of the original state of the line is kept while it is being edited. It does not always work, since the 'original state' is lost as soon as you move to another line, by up- or down-arrow, Ctrl-P or by <ENTER>. Note that it WILL work after ctrl-U and ctrl-X, however.

4. COMMAND MODE

You could get by using just the screen editing mode of XED, except for loading and saving your text file. Command mode does offer a number of useful extras, however, which can speed up the editing process, especially where lots of similar changes have to be made in a file.

Command mode is indicated by means of the 'cmnd:' indicator in the top line of the screen, together with a '*' prompt at the bottom left. The screen does not necessarily show the current position of the file (the screen cursor sits on the command line when waiting for a command), but the current text position is still normally shown on the status line and, in certain cases, such in as the Load or Insert commands, the screen is redrawn even though XED may still stay in command mode.

Each command consists of a letter, optionally preceded by a number in the range 0 to 32767. A '#' symbol is used to represent a huge number (i.e, 'as many as there are' or infinity, however you would like to visualise it! The command line terminates with the <ENTER> key, whereupon the line is executed.

In some commands, one or more strings of characters should be specified after the command letter. The end of such a string may be indicated by the <ENTER> key (which also terminates the line, of course), or by means of the <ESC> key, which displays as '$' when typed as part of a command. <ESC> therefore behaves as a delimiter in this case.

Several commands may be used in one line, using <ESC> delimiters where necessary, although these are only needed for commands which have following strings.
Example:
Ltest.doc$52GRfrom$with$V loads the file test.doc, if it exists, moves to the start of line 52, then finds the next occurrence of the string 'from' by the string 'with'. Finally, edit mode is entered, and the cursor is shown just after the word 'with' (all assuming, of course, that the 'from' string was found!).

In some cases, a prompt will appear just after the command, as follows:
Buffer not Saved -- Continue (Y/N)? and you should now press a key. If any key other than Y is pressed, the command will be cancelled. This 'Buffer not Saved' prompt only appears if additions/changes have been made to a file since last doing a Save to disc, and if the command does something drastic, such as loading another file, or quitting XED.

The following commands are available:

C -- Clear
Clears the memory buffer. Before doing this, if any alterations have been made since last saving the file, the 'Buffer not Saved' prompt will appear, otherwise, the buffer will be cleared right out.

nFstring -- Find
Finds the nth occurrence of <string> following the current cursor position. This is not restricted to the current line -- the search goes on through the file until found, or until the end of file is reached, whereupon the message 'Pattern Not Found' is displayed. Note that a question mark '?' in the search string will match any character in the text so that, for example, 'th?se' will match 'these' or 'those'.

nG -- Go
Moves the cursor to the start of line n. G by itself is 1G, and so moves to the start of the file. Similarly, #G moves to the end of the file. OG moves to the 'old line', i.e, the line that the cursor was on when the system entered command mode from edit mode. Although several commands may have been issued since then, XED remembers which line number it was on, so that OG moves to that line number (though it is possible that the information in it may have changed).

Ifilename -- Insert
Like Load, this command reads a file from disc but, in this case, inserts it in front of the current line. The cursor stays on the current line, which will now follow the last line of the inserted file.

nK -- Kill
Deletes n lines, including the current line. #K kills the whole buffer from the current line onward.

Lfilename -- Load
The Load command reads a file from disc as specified, and sets the cursor to line 1, column 0. If there is already a file in memory, the specified file overwrites it (but if any changes have been made since the last Save, a 'Buffer not Saved' prompt will be given). On completion of loading, the screen will display the first few lines of the file, to confirm the presence of the file in memory, but XED will stay in command mode, unless a 'V' command directly follows it.

nP -- Print
Prints n lines from the current line to the parallel printer port. #P prints the whole file starting from the current line.

Q -- Quit
Quits the program, returning to DOS. If changes have been made since doing a Save, the 'Buffer not Saved' prompt appears, to remind you.

nRstring1$string2 -- Replace
This command finds the next n occurrences of <string1> and replaces them with <string2>. As each occurrence is found, the line number and part or all of the line (depending on its length!) is displayed, so that you can see where the alterations have been made. Note the delimiter between string1 and string2 is NOT of course the '$' symbol, but the ESC key.

As for the Find command, question marks may be used within string1 and string2 to match any character. However, when performing the actual replacement, each '?' in string2 will take the character found at the appropriate position in string1.
Example: The command is: 2Rfrom ? to ?$to ? from ?
Here is the original line (the current column is zero):
Take me from A to B and from G to H now.
This becomes:
Take me to A from B and to G from H now.
The number of question marks in string2 must be less than or equal to that in string1, otherwise a 'Too Many '?'s ' error will be shown, and the command abandoned.

Sfilename -- Save
Saves the specified file to disc. The filename is, however, optional, and if a current file is already displayed, then that is where the memory buffer will be saved. The information is first written to a 'temporary file' of type '.$$$', which will be renamed to the desired type, after first renaming the original file (if it exists and is of the same name) to type '.BAK'. This then becomes a 'backup' file.

nT -- Tab length
Sets the tab length to n, where n is in the range 1 to 132. If n is specified outside this range, tabs are set to every 8 columns (this is also the default setting).

V -- View
Returns to edit mode, with the cursor at the last position set by previous commands, if any.

5. ERROR MESSAGES

A number of error messages may be encountered while using XED, most of which occur in command mode, although a few can also happen in edit mode. All error messages are displayed on the top line, in place of the status line, for a period of about 0.5 seconds, and are accompanied by a 'beep'. These error messages are as follows:

File Not Specified
A file name is required in the Load and Insert commands.

File Not Found
The file specified in a Load/Insert command could not be found on the specified disc.

Line Too Long
The line that would be formed by joining two lines together would be longer than 132 characters.

Line Chopped
In loading/inserting a file, one or more of the lines read was more than 132 characters long, and was therefore shortened to that length. This message appears for each such line encountered.

Pattern Not Found
The string searched for in a Find/Replace command could not be found.

Invalid Command
The command typed does not exist -- the valid command letters are:
C F G I K L P R S and V

Too Many ?'s
In a Replace command, the total number of '?' characters in the replacement string may not exceed the number in the string to replace.

Disc Write Error
In saving a file, either the disc or the directory has filled up! Try saving again, on another disc.

Memory Full
No more memory is available for adding/inserting characters. Save the current file, then type your extra text in another file!

6. COMMAND SUMMARY

Finally, here is a short summary of the facilities available under XED:

a. Edit Mode.

<ESC>	Enter Command Mode
<ENTER>	Newline
'→'	Cursor Right
'←'	Cursor Left
'↑'	Cursor Up
'↓'	Cursor Down
Ctrl-W	Word Right
Ctrl-Q	Word Left
Ctrl-C	Page Down
Ctrl-R	Page Up
Ctrl-I	Tab
<DEL>	Delete Character to Left
Ctrl-<DEL>	Delete Character at Cursor
<INS>	Insert Character
Ctrl-N	Insert Line
Ctrl-P	Join to Line Above
Ctrl-U	Clear Up to End of Line
Ctrl-X	Delete Line
Ctrl-A	Abandon Changes to Line

b. Command Mode.

C	Clear
nFstring	Find
nG	Go to line n
Ifilename	Insert file
nK	Kill n lines
Lfilename	Load file
nP	Print n lines
Q	Quit
nRstring1$string2	Replace
Sfilename	Save file
nT	Set Tab length
V	View (return to Edit mode)

CRYSTAL RESEARCH LTD

REG. IN ENGLAND No. 1629571

40 MAGDALENE ROAD
TORQUAY
DEVON
ENGLAND

Tel. (0803) 27890

VAT Reg. No. 313 3396 79

March 1986

SYSTEM 5 INFORMATION SHEET No.1

The following have all 'crawled out of the woodwork' over the past few weeks. Some of them have been spotted by us, but some have also been found by your good selves, and we should like to thank those of you who pointed them out.

To make alterations to any of the utility programs as detailed below, do the following, e.g, with XED:

From DOS, do LOAD XED.COM
Enter MOS, do 'M' commands as detailed. 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 M9E4 and, if it contains 08, change this to 00, and then 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!
Finally, re-enter DOS, and do SAVE 52 XED.COM .
We recommend that you do not touch the original System 5 disc, but make these alterations on your working copy.

1. XED (52 blocks)

a. Ctrl-chars show as ^@ under insert mode.
M09E4 (08) 00.

b. Mods to lines making them shorter sometimes still show old end on swapping pages.
M1ED3 (21) D5 21 00 00 CD 9A 21 C1 C3 15 24.

2. XSM (23 blocks)

a. Drive 3 cannot be selected.
M0359 (04) 05.

b. Bad Opcodes sometimes cause crash.
M09E9 (8A) FF.
C1712 1788 1713 NB. Block Copy command.
M1712 (25) 00.
M14CD (8B) 8C 17 60.
M07FF (00) 11 D1 14 C3 8E 07.
M0901 (D1) 13 17.
M0ADD (D1) 13 17.
M0B44 (D1) 13 17.
M0F3D (D1) 13 17.

c. Disc/Directory Full error when no .PRN file specified.
M02BF (58) 59 28 19 FE 4E D0.

d. The 'lines read' count is twice as big as it should be!
M124B (05) 6E 08.
M125F (46) 40.

3. XBAS (68 blocks)

a. Incorrect results when using angle limits in ELLIPSE/POLY commands.
M3409 (91) 90.

Now on 14/4/86

b. Screen Dump.

A large number of printers have been found to work with the printer graphics dump routine now supplied in Xtal BASIC 5.

The following alterations are, however, required to work the Taxan/Kaga KP-810 printer in this mode:

M3147 (A7) 51 44.
M4451 (ED) CD A7 31 41 3E 08 CF 9F C9.

DO NOT make these alterations unless your printer is of this type!

CRYSTAL RESEARCH LTD
REG. IN ENGLAND No. 1629571

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

SYSTEM 5 INFORMATION SHEET No.2 April 1986

The following have all 'crawled out of the woodwork' over the past few weeks. Some of them have been spotted by us, but some have also been found by your good selves, and we should like to thank those of you who pointed them out.

To make alterations to any of the utility programs as detailed below, do the following, e.g, with XED:

From DOS, do LOAD XED.COM
Enter MOS, do 'M' commands as detailed. 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 M23C2 and, if it contains E5, change this to C1, and then 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!
Finally, re-enter DOS, and do SAVE 52 XED.COM .
We recommend that you do not touch the original System 5 disc, but make these alterations on your working copy.

Previous sheets

Not all users will receive earlier update sheets, as all of the changes that were detailed in those sheets will have already been made to their system discs. If your serial is later than that shown, you do not need that sheet:

Sheet 1: All copies up to and including No.X5000067.

1. XED (52 blocks)

NOT DONE

XED does not work with version 1.1 MOS. This alteration allows it to work with all versions of MOS.
M23C2 (E5) C1 D1 D5 C5 55 ED 53 4A FB C9.

Note that you should have version 1.2 MOS or later, as version 1.1 does not work with the 80-column card. Contact TAITUNG (UK) Ltd to discuss replacement of the ROM.

2. XBAS (68 blocks)

Problems occur when trying to CREATE a file which was previously not found by an OPEN.
M2E3D (DF) DE.

NOW OK 14/4/86

3. Using WORDSTAR (WS.COM -- 71 blocks)

Some users have complained that the version of WORDSTAR released by IATUNG 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:

```
M0234 (02)  01 1D 00.
M0254 (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 5B 62 C7 01 09 00 ED B0 C9.
M46F3 (22)  CD 29 C6.
M4711 (21)  2A 64 C7.
```

```
                    X     X
                     X   X    t                    ll
                      X X     ttttttt   aaaaa     l
                       X      t             a    l
                      X X     t          aaaaaa   l
                     X   X    t         a     a  l
                    X     X    ttttt    aaaaaa  lll

                BBBBBB      AAA      SSSSS    III   CCCCC
                B     B    A   A    S     S    I    C     C
                B     B   A     A   S          I    C
                BBBBBB    AAAAAAA    SSSSS     I    C
                B     B   A     A         S    I    C
                B     B   A     A   S     S    I    C     C
                BBBBBB    A     A    SSSSS   III   CCCCC

 CCCCC    OOOOO   M     M  PPPPPP   III  L        EEEEEEE  RRRRRR
C     C  O     O  MM   MM  P     P   I   L        E        R     R
C        O     O  M M M M  P     P   I   L        E        R     R
C        O     O  M  M  M  PPPPPP    I   L        EEEEE    RRRRRR
C        O     O  M     M  P         I   L        E        R   R
C     C  O     O  M     M  P         I   L        E        R    R
 CCCCC    OOOOO   M     M  P        III  LLLLLLL  EEEEEEE  R     R

          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 *
************************************
```

I. 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,

0:XC 1:HANGMAN compiles the program HANGMAN.XBS on drive 1, where the compiler is resident on drive 0. 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,

```
0: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 'Branch 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.

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.