UltraCoder V1.1

By Trevor Toms

This compiler for the ZX Spectrum offers as much compatibility as possible with interpreted Basic, within the limitations of small memory requirements, so that programs can be tested initially using the standard Basic in ROM, then subsequently compile to produce compact, efficient and fast code.

The main objectives in producing the compiler were:-

- allowing integer arithmetic throughout, since most games programs rarely need the power of 5-byte floating point precision.

- to offer additional commands suited to fast program execution.

- to allow quick development of new games by a small team of programmers.

The fact that you are reading this document means that you have been chosen to produce games using UltraCoder - in the short term, it will not be available on the commercial software market.

Phipps Associates reserve all rights on any programs produced using UltraCoder, and their express permission must be granted in writing if a program is intended for marketing through any other software house.

We sincerely hope that UltraCoder offers better facilities than any other compiler for the Spectrum, and that our concessionary royalties of 2.5% are lower than any other software house. (As a comparison, PSS require a **minimum** of 5% for the use of MCoder).

If you feel that UltraCoder could be improved in any way, then we would like to hear about it - several improvements are already being considered, and these are listed later on.

COMMANDS SUPPORTED BY ULTRACODER

The following list shows all commands, functions and operators that UltraCoder will accept. Some of these are additional commands which will be explained later.

BEEP	INVERSE	REM	LET	DRAW
CIRCLE	OUT	FOR	PAUSE	RETURN
INK	PRINT	NEXT	POKE	COPY
PAPER	LPRINT	STEP	PLOT	REM *b
BRIGHT	STOP	GOTO	RANDOMISE	REM *c
FLASH	BORDER	GOSUB	IF	REM *d
OVER	DIM	INPUT	CLS	REM *i
REM *k	REM *s	REM *t	CODE	INKEY$
SGN	ABS	PEEK	IN	USR
CHR$	NOT	RND	POINT	SCREEN$
ATTR	BIN	AT	TAB	

; , ' # = < > <> <= >= AND OR

COMMAND SYNTAX

Command formats are identical to that required by the interpreter (including the use of complex expressions to replace items - e.g. BORDER x*2+PEEK 23681).

Multi-statement lines are allowed, and these may additionally be used in conjunction with IF statements, e.g.:-

```
2370 IF x=5 THEN GOSUB 5000: LET x=0: PRINT "Hi there"
```

COMPILING A PROGRAM

A program is entered into the Spectrum in the usual manner. It can be tested in this way (see "Incompatibilities" below) then the compiler can be loaded by:-

```
CLEAR 59999
LOAD "ULTRACODER" CODE 60000
```

When the program is ready, compile it by entering:-

```
RANDOMIZE USR 60000
```

UltraCoder will respond with the message:-

```
Enter target address:
```

You should now enter the address in memory where you want the resulting code to be placed. Normally, this would be above RAMTOP but below UltraCoder, so that prior to compiling, you would use the "CLEAR xxxx" command to create sufficient space for your purposes.

If you wish to place the compiled code below the program area, then prior to compiling, you must move the program area up by using the method shown in Appendix A. This method creates a gap above the system variables but below the program, and UltraCoder will place all code there. Ensure that the CLEAR command is not used while this space is allocated. Once compiled, the program can be saved as bytes in the usual manner.

ULTRACODER MESSAGES

During compilation, UltraCoder reports on errors on the screen, showing an error letter code and the Basic line in question. At the end of compilation, a total error count is given and the complete program size shown. This size should be retained for use when saving the code as bytes. The starting address is as you have initially given to UltraCoder as the "target address".

Error codes:-

3	Array already dimensioned
A	Syntax unacceptable (usually with REM * commands)
B	BIN expression error
C	Nested expression error (" " " ")
K	User defined graphic character error (A-U)
L	Undefined line number
M	Multi-dimensioned array
N	Array not dimensioned
U	Undeclared data - should not normally occur.
X	Invalid item separator (usually a comma)

If the program contains errors, it will give spurious results if you attempt to run it.

It is not possible to predict in advance the amount of memory required for compilation, but on balance, the compiled program will occupy roughly the same amount of memory as the original Basic source program providing that none of the usual "memory saving" techniques have been adopted.

INCOMPATIBILITIES WITH INTERPRETED BASIC

The following list shows all areas where a difference occurs - study it carefully:-

- String variables not allowed (to be rectified in V2.Ø)

- Integer arithmetic means that maximum range of values for any variable is -32768<x<32767

- Expression evaluation has no priority, so brackets must be used to indicate the order of evaluation. This is also true for conditional expressions. See examples later.

- multi-dimensioned arrays are not allowed, although 26 arrays (A-Z) can be declared in the usual manner. The base subscript for arrays is 1 (as in the interpreter).

- variable names are significant in the first two letters only, so variables BALL and BAT are considered identical, while BL and BT are independent. Array names are independent of the simple variable of the same name.

- DRAW command does not accept "arc" parameters.

- CHR$ can only be used within PRINT and LPRINT commands.

- INPUT command cannot display variable data, although it can use a text prompt. E.G. INPUT "Enter level";sk is acceptable, but INPUT ("Enter level ";x);sk is not.

- GOTO and GOSUB may not use computed values (e.g. GOTO x*2)

All normal stream usage is acceptable (e.g. PRINT #1;AT 1,Ø), but Microdrives are not supported in V1.Ø.

ADDITIONAL COMMANDS

These commands extend the Spectrum capability to suit a compiled environment:-

REM *b	- check to see if the BREAK key is pressed and reports BREAK error code if so, otherwise execution continues.
REM *c attr,mask	- "colour wash". This command requires two parameters. The first identifies the value to be inserted into every attribute screen position. The second parameter gives a "mask" to indicate which values of the attribute character are to be altered. So, for example, REM *c 2,7 will change all ink on the screen to red. REM *c y,56 will change all paper to the colour code given by variable Y at runtime. You can additionally use this command to change the entire status of "bright" and "flash" if you wish.
REM *d nnnn	This is equivalent to the PAUSE Basic command, except that the pause is not terminated when a key is pressed. This command allows you to slow down a program which is operating too fast, without the delay being stopped by the player pressing a key interactively.

REM *s time,freq — This command drives the beeper directly, allowing sound effects to be produced by experiment. The first parameter gives the time required in 1/256ths of a second. The second gives a sliding frequency value, where a high figure gives a low tone. REM *c 256,1638 is almost equivalent to BEEP 1,Ø.

REM *i hh hh .. — Inline hex coding. The code is converted and included in the program at the appropriate place.

REM *k "x",1,2,3,4,5,6,7,8
Define user graphic character "x" with the eight rows that follow. All eight rows must be present and they may not be computed data.

REM *t n — Toggle compiler switch n. Compiler switches affect the way in which UltraCoder generates its runtime coding. All switches are initially set off (n=Ø) and can be set on by using the command once. If the command is issued again, the switch will be set off. In this way, portions of the program can be compiled in different ways. Current switches are:

REM *t 7 - switch on/off BREAK enable.
If on, the BREAK key is enabled at the start of every line, and when pressed, the line number will be shown in the report message. If off, BREAK is disabled, and a STOP command will jam the machine. Useful when testing.

REM *t 6 - switch on/off line tracing.
REM *t 7 must also be on before this switch has any effect. It forces the compiler to generate run-time code which displays each line number in the bottom right-hand corner of the screen. This also slows down execution quite considerably. Portions of a program can be traced by putting a "REM *t 6" command either side of the section to be monitored.

Enhancements to other commands:

- the BEEP command is implemented to use the floating point calculator in the ROM to evaluate the duration and pitch parameters. However, since all expressions must use integer arithmetic, the equivalent of BEEP Ø.Ø1,6 would be coded as BEEP 1/1ØØ,6 where the duration is written as one integer value divided by another. Any paranthesised sub-expressions will be treated as strictly integer, so for example BEEP (x*3/1Ø)/1ØØ,6 would evaluate "x*3/1Ø" in the normal integer manner before being passed to the floating point calculator.

- the expression RND on its own generates a random number between Ø and 32767. However, you can write RND*nnnn (nnnn can be any expression) to give a random number between Ø and (nnnn-1). So LET x=RND*6 will give x a value from Ø to 5. In order to allow compatible testing with the interpreter, always write in the form LET x=INT(RND*n). The compiler will ignore the INT function (since all arithmetic is integer anyway!) and give the identical result to the interpreter. You **must note**, however, that RND*n is **NOT** the same as n*RND.

- a RETURN command at the highest level (i.e. with no corresponding GOSUB) will cause a return to the standard ROM. E.g.:

```
1Ø FOR x=1 TO 2Ø: PRINT "Hello!": NEXT x
2Ø RETURN
```

This small program will print as directed, then return to the standard ROM for more commands.

CODING EFFICIENCIES

The following list shows some general guidelines to producing efficient and compact compiled code. Use them when speed or space become at a premium, otherwise write your code as normal.

- FOR/NEXT loops generate more coding than the equivalent LET/IF...THEN commands. This is purely because the compiler does not know to whereabouts it must jump back when generating the code. So, for example:-

```
1Ø FOR x=1 TO 23 STEP 2
2Ø PRINT "Line ";x
3Ø NEXT x
```

...is more ineffiecient (in memory requirements, but not necessarily speed) than...

```
1Ø LET x=1
2Ø PRINT "Line ";x
3Ø LET x=x+2: IF x<23 THEN GOTO 2Ø
```

- The expression PRINT CHR$ 65 is more efficient in memory than PRINT "A". The trade-off occurs when a string of more than two characters is to be printed. Example:-

```
10 PRINT CHR$ 65;CHR$ 66;      is efficient
10 PRINT "AB";                 is inefficient

10 PRINT CHR$ 65;CHR$ 66;CHR$ 67   is inefficient
10 PRINT "ABC"                     is efficient.
```

- GOTO and GOSUB generate three bytes of run-time code, and thus are very efficient. Use GOSUB frequently to replace the lack of DEF FN/FN expressions. A GOSUB is nearly always more efficient than two similar LET statements.

MISCELLANEOUS INFORMATION

1. Every compiled program has a basic overhead of approximately 750 bytes. Adding "REM *t 7" to a program causes an additional 6 bytes per line to be generated. "REM *t 6" causes another 3 bytes per line to be included.

2. String functions INKEY$ and SCREEN$ can only be used in conjunction with the CODE function, e.g.:

```
IF CODE SCREEN$(y,x)=65 THEN ...
```

CHR$ function can only be used within PRINT/LPRINT commands, so in order to print copy the character on screen position (0,0) to position (21,31), you would write:-

```
100 PRINT AT 21,31;CHR$ CODE SCREEN$(0,0);
```

Although it looks cumbersome, it generates very efficient code!

3. CIRCLE and BEEP commands use the ROM's floating point calculator and are thus intrinsicly slow. The BEEP command can be speeded up by using REM *s instead, but CIRCLE usage should be kept to a minimum.

4. As supplied, UltraCoder has room to compile a program which contains a combination of 512 program lines (not statements) and independent variables. An array is counted as one item. This should be sufficient for most purposes, but note that UltraCoder V1.0 has no memory overflow checks and thus it is important to be aware of this limitation in very large programs.

5. AND and OR act as Boolean operators, so that (21 AND 15) gives the result 5, while (21 OR 15) gives the result 31. In this way, you can avoid the "issue 3 Spectrum" problem associated with the IN function by masking out the lower 5 bits supplied by the keyboard:-

```
120 IF (IN 65278 AND BIN 00011111)<>31 THEN ....
```

6. The previous example shows a very important point: since expression evaluation is strictly "left-to-right", you must use brackets to surround priority expressions. This is particularly true when using AND or OR within IF statements. Consider the following:-

```
50 IF x>1 AND x<5 THEN GOTO 100
```

If x contains 0, then this would give a "true" result, since "x>1" gives the result 1, "1 AND x" gives the result 0, and finally "0<5" gives the result 1 (i.e. "true"), therefore the program would continue at line 100. You must always enclose such expressions within brackets to enforce the correct priority (c.f. line 1090 in BREAKOUT game) :-

```
50 IF (x>1) AND (x<5) THEN GOTO 100
```

EXAMPLES OF USE

The first program is a demonstration of the effect of the extended commands:-

```
10 FOR y=0 TO 7
20 REM *c y*8,BIN 00111000
30 FOR x=50 TO 250 STEP 12
40 REM *s 10,x
50 REM *b
60 NEXT x: NEXT y
70 RETURN
```

The second is a complete BREAKOUT game for the Spectrum! The program was originally written to run in normal inpterpreted Basic, and I have merely added and altered where necessary to suit the compiler (this explains the cumbersome line numbering). The resulting program generated just less than 3900 bytes of code! Not bad, eh?

```
  1 REM [illegible]
  2 REM [illegible]
 10 RANDOMIZE
 20 LET max=990
 30 DIM h(180)
 40 LET high=0
 50 PAPER 5
 60 REM #k"0",60,126,255,255,25
5,255,126,60
 100 IF CODE INKEY$ THEN GO TO 1
00
 101 BORDER 4: CLS : PRINT AT 8,
3; INVERSE 1;"** BREAKOUT **";AT
10,4;"© 1983 Phipps Associates"
AT 21,5;"Press any key to start

 102 LET y=0
 103 REM *cy,7
 104 IF CODE INKEY$ THEN GO TO 1
10
 105 FOR x=0 TO 400 STEP 4
 106 REM *s2,1280-x
 107 NEXT x: LET y=y+1: IF y>7 T
HEN GO TO 102
 108 GO TO 103
 110 CLS : PRINT TAB 9; INVERSE
1;"** BREAKOUT **"; INVERSE 0;''
'"Use the cursor keys to move you
r bat and control the ball. The
object of the game is to remove
all the bricks! Points are
scored according to the row
being knocked down."

 115 INPUT "Enter skill level (1
-15): ";sk
 120 IF sk<1 OR sk>15 THEN GO TO
 115
 130 CLS
 140 FOR i=1 TO 180: LET h(i)=1:
NEXT i
 150 LET x=0: FOR i=1 TO 10: LET
k=6+(2*i)
 160 FOR j=0 TO 17: PRINT INK x+
1;AT j,k;CHR$ 143;: LET x=(x=0):
NEXT j
 170 LET x=(x=0): NEXT i
 180 PRINT AT 18,0; PAPER 4;CHR$
6;CHR$ 6;AT 20,0;CHR$ 6;CHR$ 6;
PAPER 5;AT 21,1;"Score:";AT 21,
15;"High score:"
 190 LET s=0: GO SUB 6000: GO SU
B 5100
 280 LET bl=3
```

```
 600 LET bat=8
 610 LET y0=INT (RND*18)
 620 LET x0=6
 625 FOR j=0 TO 17: PRINT AT j,0
;CHR$ 32;CHR$ 32: NEXT j
 630 FOR j=7 TO 9: PRINT INK 3;A
T j,0;CHR$ 143: NEXT j
 640 LET dir=-INT (RND*3)-1
 645 PRINT PAPER 4; INK 0;AT 20,
0;bl;" balls left."
 647 PRINT #1;"Press any key to
start...";: PAUSE 0
 650 INPUT ""
1000 LET bp=0: IF x0>1 THEN GO T
O 1050
1010 IF ABS (y0-bat)>1 THEN GO T
O 3000
1015 GO SUB 7000
1020 IF dir=-2 THEN LET dir=INT
(RND*3)+1
1030 IF dir<0 THEN LET dir=2-(IN
T (RND*2)*(dir+2))
1040 GO TO 1200
1050 IF x0<31 THEN GO TO 1090
1060 GO SUB 7000: IF dir=2 THEN
LET dir=-INT (RND*3)-1
1070 IF dir>0 THEN LET dir=-2+(I
NT (RND*2)*(2-dir))
1080 GO TO 1200
1090 IF (x0<10) OR (x0>28) THEN
GO TO 1200
1095 IF (INT (x0/2)*2)<>x0 THEN
GO TO 1200
1097 LET n=(x0-8)/2
1100 IF h((n-1)*18+y0+1)=0 THEN
GO TO 1200
1110 GO SUB 7000: LET h((n-1)*18
+y0+1)=0
1120 LET s=s+n
1130 IF s=max THEN GO TO 3000
1135 GO SUB 6000
1140 IF ABS dir<>2 THEN GO TO 11
70
1150 LET dir=(-SGN dir)*(1+INT (
RND*3))
1160 GO TO 1200
1170 LET dir=(-SGN dir)*((ABS di
r-2)*INT (RND*2)+2)
1200 GO SUB 5500: IF y0>0 THEN G
O TO 1230
1210 IF ABS dir>1 THEN GO TO 125
0
1220 GO SUB 7000: LET dir=dir*3:
 GO TO 1250
1230 IF y0<17 THEN GO TO 1250
1240 IF ABS dir<3 THEN GO TO 125
0
1245 GO SUB 7000: LET dir=SGN di
r
1250 LET x1=x0+SGN dir
1260 LET y1=y0+(ABS dir-2)
1270 PRINT AT y0,x0;CHR$ 32; INK
 6;AT y1,x1;CHR$ 160
1280 REM *dsk
1290 LET x0=x1: LET y0=y1
1300 GO SUB 5500
1500 GO TO 1000
```

```
3000 BEEP .50,10
3010 LET bl=bl-1
3020 IF bl>0 THEN GO TO 600
3025 IF s<high THEN GO TO 3070
3030 LET high=s
3060 PRINT AT 21,0;" Score: ";s;
CHR$ 32;CHR$ 32; FLASH 1;" NEW H
IGH SCORE "; FLASH 0
3070 IF s<max THEN GO TO 3090
3080 PRINT AT 20,0; FLASH 1;" CO
NGRATULATIONS - Maximum score"
3090 PRINT #1;"Another game? ";
3095 LET x=CODE INKEY$: IF (x<>1
21) AND (x<>110) THEN GO TO 3095
3100 IF x=121 THEN GO TO 100
3120 RETURN
5000 RETURN
5500 LET x=CODE INKEY$
5510 LET p1=(x=54)-(x=55): LET b
l=bat+p1
5530 IF (bl<1) OR (bl>16) THEN L
ET p1=0
5540 IF p1=0 THEN RETURN
5550 PRINT AT bat-p1,0;CHR$ 32
5560 LET bat=bat+p1
5570 PRINT INK 3;AT bat+p1,0;CHR
$ 143
5580 RETURN
6000 PRINT AT 21,0;s;CHR$ 32
6010 RETURN
6100 PRINT AT 21,28;high
6110 RETURN
7000 IF bp=1 THEN RETURN
7010 BEEP .10,20
7020 LET bp=1
7030 RETURN
```

APPENDIX A - Program relocator

The following program relocates the entire program space up by an amount specified by the INPUT command. The actual routine that performs this task is:-

LD	HL,(PROG)	42,83,92
DEC	HL	43
LD	BC,nnnn	1,FN L(bytes),FN H(bytes)
CALL	1655h	205,85,22
RET		201

You can POKE this above RAMTOP and use it independently of the program below, by substituting the appropriate values into the third line (LD BC,nnnn).

```
  1 REM Program relocator
  2 REM ©1983 Phipps Associates
  3 REM By Trevor Toms
 10 DEF FN h(x)=INT (x/256)
 20 DEF FN l(x)=x-256*FN h(x)
 30 INPUT "How many bytes to be
reserved?  ";bytes
 40 LET address=PEEK 23635+256*
PEEK 23636
 100 DATA 42,83,92,43,1,FN l(byt
es),FN h(bytes),205,85,22,201
 120 LET ramtop=1+PEEK 23730+256
*PEEK 23731
 130 FOR x=ramtop TO ramtop+10:
READ n: POKE x,n: NEXT x
 140 RANDOMIZE USR ramtop
 200 PRINT "Task complete - ";by
tes;" bytes"'"reserved starting
at address"'address
```

APPENDIX B - Suggestions for V2.Ø

1. Strings to be implemented as arrays only. They will be DIMensioned in the usual manner, and string slicing would be allowed. The LEN function, however, would only return the length as DIMensioned.

2. Linking of several modules of compiled code to allow larger routines to be compiled. This would involve the use of "REM *j nnn" command, where nnnn gives the number of the module to be "jumped" into. At the start of compilation, you will specify which module number is being compiled (default is module Ø which contains the 75Ø-byte overhead).

3. Memory protection to safeguard against corruption by large programs.

4. Relocation of UltraCoder to the screen area to allow even larger programs!

5. Operator priorities to conform to Sinclair "standards".