Using MCODER2

By Timothy Swenson

1. Introduction

MCODER2 is the primary compiler for Sinclair BASIC on the ZX81, written by David C. Threlfall in 1983 and published by Personal Software Services (PSS). David Threlfall went on to do a number of BASIC compilers for the Spectrum. MCODER2 is available from a number of sources on the Internet, including this link: http://www.zx81.de/english/soft_e.htm

How MCODER2 works is like this; MCODER2 is loaded into memory, then the BASIC program is loaded (meaning the code that is to be compiled), the compilation process takes place, and the compiled code can be saved.

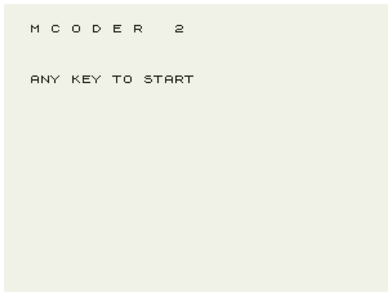
These days most ZX81 development is done with emulators, such as EightyOne for Windows or SZ81 for Linux. The code is written using text editors of the host OS. A utility, zxtext2p, is used to convert the code to a ZX81 binary file (.P) so that it can be loaded into an emulator and run.

The goal of this paper is the describe the process of using programs from zxtext2p and produce a compile version through MCODER2. I am not an expert on MCODER2, I have worked with it and decided to document the process. I have incorporated comments from others that have also used MCODER2. Some parts of this document are going to be conjecture about how MCODER2 works.

2. Getting Started

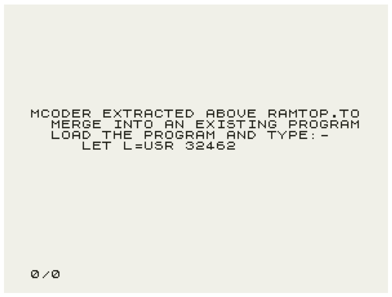
This section is a quick run down of what it takes to compile a BASIC program. It is assumed that the program has been written,tested, and saved as a .P file. The program could have been written directly into the emulator or through a tool like zxtext2p.

1. Load the MCODER2.P file into the emulator. Once MCODER2 starts, it will tell you to hit any key to continue. For opening screens of MCODER2, see image1 and 2, below:



```
M C O D E R   2

ANY KEY TO START
```

(Image 1)



```
MCODER EXTRACTED ABOVE RAMTOP.TO
   MERGE INTO AN EXISTING PROGRAM
   LOAD THE PROGRAM AND TYPE:-
       LET L=USR 32462

0/0
```

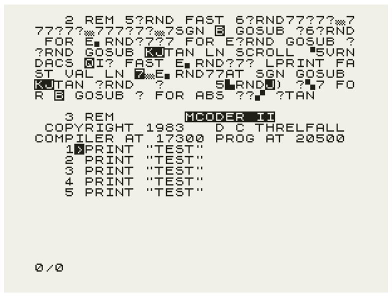
(Image 2)

2. Load your BASIC program from a .P file.

Be sure to use the load option that does not do a hard reset of the emulator. For EightyOne and ZX81, use the LOAD “” command and not the GUI to start the load process.

3. Enter “LET L=USR 32462”

This will load the REM statements needed by MCODER2 to the beginning of your code. If your program already has a line number 1 through 3, they will not be touched and a new line 1 to 3 will be added. I’m not sure how MCODER2 does this, but it does (see Image 3).

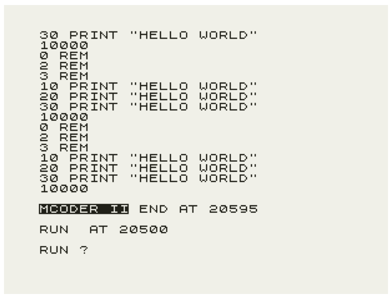


```
   2 REM 5?RND FAST 6?RND77???▒7
77???▒777777?▒7SGN B GOSUB ?6?RND
 FOR E▖RND?777 FOR E?RND GOSUB ?
?RND GOSUB KJTAN LN SCROLL ▘5VRN
DACS QI? FAST E▖RND??? LPRINT FA
ST VAL LN 7▒E▖RND77AT SGN GOSUB
KJTAN ?RND  ?      5▙RNDJ) ?▚7 FO
R B GOSUB ? FOR ABS ??▞ ?TAN

   3 REM          MCODER II
 COPYRIGHT 1983   D C THRELFALL
COMPILER AT 17300 PROG AT 20500
   1>PRINT "TEST"
   2 PRINT "TEST"
   3 PRINT "TEST"
   4 PRINT "TEST"
   5 PRINT "TEST"




0/0
```

(Image 3)

4. Enter “LET L=USR 17300”

This will start the compile process for your code. MCODER2 will list the lines of the program as it compiles and if there is an error, it will highlight the error with an inverse S next to the command it has problems with. The program can then be modified and the compilation process started again. The program is listed three times, it looks like MCODER2 is a three pass compiler. The compilation process is shown in image 4, below:

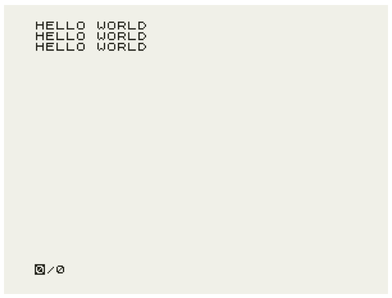


```
30 PRINT "HELLO WORLD"
10000
0 REM
2 REM
3 REM
10 PRINT "HELLO WORLD"
20 PRINT "HELLO WORLD"
30 PRINT "HELLO WORLD"
10000
0 REM
2 REM
3 REM
10 PRINT "HELLO WORLD"
20 PRINT "HELLO WORLD"
30 PRINT "HELLO WORLD"
10000

MCODER II END AT 20595

RUN  AT 20500

RUN ?
```

(Image 4)

5. Once the program is compiled, MCODER2 will tell you the address in memory of your code. The usual address is 20500.

6. Once the program is compiled, you will be given the option to run your code. MCODER2 will ask “RUN ?”. Enter Y or N to answer. If you entered Y, let your program continue until it ends, and then continue to the next step. Image 5 shows the example program after it has run.



```
HELLO WORLD
HELLO WORLD
HELLO WORLD




0/0
```

(Image 5)

7. Enter “LET L=USR 17281”

This will delete the source BASIC program. Since the original code is no longer needed, having it deleted from memory will free up more memory for the program to use.

8. Add the following line of BASIC:

```
10 LET L=USR 20500
```

This is how to start the compiled version of your code. When RUN is entered, this is the first line that will be executed. The runtime routines for MCODER2 are stored in the REM statements in lines 1-3 and these will not be executed by BASIC.

9. Use the SAVE keyword to save your program (depending on the emulator you are using).

10. LOAD your program and test that when you enter RUN, that the compiled version runs fine.

3. MCODER2 Definition

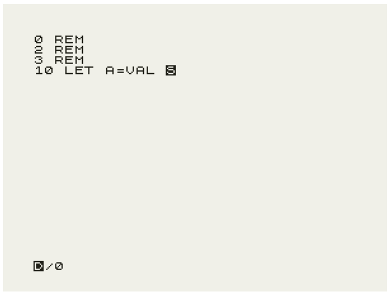
MCODER2 does not support the full ZX BASIC command set and will only compile a subset of those commands. These are also other limitations of MCODER2, as documented below:

1. Integer Only
   - Positive and negative values allowed. Negatives are preceded with a minus sign.

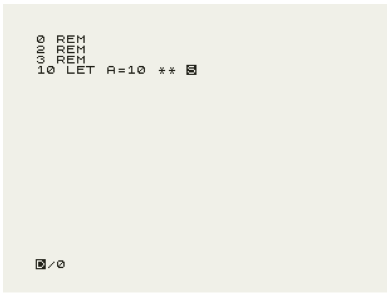
2. Arrays
   - No string arrays.
   - Only one dimensional numeric arrays.

3. For .. Next Loops
   - Only STEP 1 is allowed.
   - No negative steps.

4. Strings
   - Maximum length of 32 characters.
   - String slicing, A$(x TO y) allowed. A$ (TO x) or A$(x TO) is not allowed.



```
0 REM
2 REM
3 REM
10 LET A=VAL S







D/0
```



```
0 REM
2 REM
3 REM
10 LET A=10 ** S







D/0
```

(Example screenshots showing MCODER2 flagging an error)

Keywords Supported:

```
AND, ABS, CHR$, CLS, CLEAR, CODE, COPY, DIM, FAST, FOR, GOSUB, GOTO,
IF..THEN, INKEY$, INPUT, INT, LEN, LET, LPRINT, NEW, NEXT, OR, PAUSE,
PEEK, PLOT, POKE, PRINT, RAND, REM, RETURN, RND, SCROLL, SGN, SLOW,
SQR, STOP, TAB, UNPLOT, USR.
```

Notes on Keywords:

AND	- Only allowed in IF statements.
DIM	- Must have at least 2 times the array size in available bytes in spare memory.
FAST	- Does not return to SLOW mode during INPUT or PAUSE.
FOR	- Numbers must be less than 32767.
INPUT	- Numbers can have leading negative sign. Strings limited to 31 characters.
LEN	- No slicing of the string in the LEN statement.
OR	- Only used in IF statements. [need to confirm]
PAUSE	– PAUSE value must be positive and less than 32768. Pressing SHIFT-EDIT works like the BREAK key.
REM	- REM? Is a special code for MCODER2. It checks to see if the SHIFT-EDIT key has been hit and works like the BREAK key.
RETURN	– Can be used in the middle of the program to exit and return to BASIC.
RND	- Returns a number from 0 to 32767, which is different than the normal RND behavior. To get the normal behavior of RND, use LET X = USR 16550.
STOP	- Will stop compilation when found. Use LET L=USR 3292 as a STOP in the middle of the program.

4. Limitations

MCODER2 has a number of limitations, all listed above. Instead of letting those limitations inhibit your code, it is possible to find workarounds for a number of these limitations.

Two Dimensional Arrays

A number of other languages do not support two dimensional arrays, including the popular Small-C compiler (from which Z88DK comes from). It is fairly simple to utilize a one dimensional array and treat it as having two dimensions.

A two dimensional array is addressed by the variables, such as x and y, in this manner:

```
LET a = B[x,y]
```

To address a one dimensional array in the same manner, use the following:

```
LET a = B[x*MAX_X+y)
```

Where MAX_X is the x dimension for the two dimensional array. If an array is defined as:

```
DIM B[20,10]
```

Then MAX_X is 20, and the one dimensional array is defined at 20x10 or B[200].

VAL

The VAL keyword is not allowed by MCODER2. VAL converts a string to a number, or more generally, it treats a string as an arithmetic expression and evaluates it. In most cases, VAL is used to just convert a string to a number. If the string is comprised of a single character, then converting it to a number is a fairly simple process.

```
print "input a number"
input a$
print a$
gosub @vall
print "answer is :";
if BB = -1 then print "error"
if BB <> -1 then print BB
return

@vall:

let AA = code A$
let BB = AA - 28
if AA < 28 then let BB = -1
if AA > 37 then let BB = -1
return
```

STR$

The STR$ keyword is not allowed by MCODER2. STR$ converts a number to a string. This can be implemented in BASIC with a short routine.

```
let num = 12345
let a$ = ""
let x = 5
print "number is : ";num

@loop:

#   let y = 10 ** (x-1)

  let CC = 10
  for d = 1 to x-2
     let CC = CC * 10
  next d
  let y = CC
  print y
  let z = int(num/y)
  if z = 0 then goto @jump
```

```
          let a$ = a$+chr$(z+28)
          let num = num - int(z * y)
     @jump:
       let x = x - 1
       if x >0 then goto @loop

     print "string is : ";a$
```

Power

The power function (**) is not allowed by MCODER2. The power function takes a number (x) and raises it to the power of another number (y) and respresented by the expression x**y (or x^y). In the case where Y is 2, then X*X will do. For values greater than 2, then the routine below will provide the functionality of power.

```
print "input number"
input AA
print AA
print "input exponent"
input BB
print BB
gosub @power
print "result is ";
print CC

#stop
let l=usr 3292

@power:
let CC = AA
if BB = 0 then goto @pjmp1
for x = 1 to BB-1
  let CC = CC * AA
  print CC
next x
return
@pjmp1:
  let CC = 1
return
```

Non-Integers

It is possible to implement a form of floating point numbers, by shifting the decimal point on all of the numbers and keeping track of where the decimal point should be. In this situation, the numbers are integers, but, by convention, they are treated as floating point and printed out as floating point. With true floating point, the location of the decimal would change per number. This would be hard to track, so a simpler implementation is to set each number with a defined decimal point (ie. 2 decimal points for each number). Technically this is not floating point, but one can use the term “floating point” to mean numbers with a decimal point, or the more accurate but less ____ term, non-integer.

With this implementation, each number is considered to have 2 decimal places, and then be multiplied by 100 to get an integer. The number 3.14 would be stored as 314. The number 276 is really 2.76. The number 1 is 100. With the numbers stored as integers, they can be run through all of the math functions. Once through a function, the decimal point needs to be re-adjusted, depending on the mathematical operation that was performed. Below is the adjustment for each operation:

Add	Nothing
Subtract	Nothing
Multiply	Divide result by 100
Divide	Multiply result by 100

Here is an example:

3.14 x 1.76 = 5.5264   314 x 176 = 55264

To get the end if the first equation is 5.5264. Truncate it to 2 decimal points and it would be 5.52, then convert to our integer convention and it becomes 522. In the second equation the result is 55264. Divide by 100 and you get 522.64, then when truncated by the integer only math of MCODER2, the result is 552.

Remember that the maximum integer value that MCODER2 can handle is 64K or 65535. MCODER2 stores variables in 2 bytes or 16-bits. This scheme can be adapted for only 1 decimal point, and one would use 10 in place of the 100 listed above.

Using a number scheme like this may not be needed for every variable or number used in the program. To keep track of which variables are using this convention, it is suggested that the variables start with a unique letter, like F. So if using the variable A, and using this convention, the variable would be FA.

5. Cheat Sheet

This is a short list of all of the MCODER2 USR calls:

```
LET L=USR 32462
```
- Load MCODER2 into your program.

```
LET L=USR 17300
```
- Compile the program

```
LET L=USR 17281  (RAND USR 17281)
```
- Delete BASIC source code

```
LET L=USR 17287 (RAND USR 17287)
```
- Relocate compiled code in memory

```
LET L=USR 3292
```
- Used in place of STOP

```
LET X = USR 16550
```
- Normal behavior of RND