Morse Tutor on the ZX81

By Paul Newman G4INP

The coming of the cheap microcomputer into ham radio opens up many avenues of fun and education. One of the first and most obvious uses is that of morse tutor. The *ZX81* however, is not blessed with a sound facility such as the BEEP-command of the Spectrum. The *ZX81* will, however, produce a sound of sorts and the tutor described here uses this fact. Facilities available with this program are as follows:—

a) Speed variable from around 6 to 35 wpm in practical terms.
b) Delay between characters can be varied to aid learning.
c) Teach-mode; characters can be selected from the RSGB-recommended groupings and can be demonstrated at low speed.
d) Exam mode; the program selects a random grouping and sends 20 characters at the selected speed.
e) On-screen checking of received copy.
f) Save; the program can be saved with the last lesson set, to resume where you left off.

The morse is sent out of the *ZX81's* 'mic' line and may be monitored by a small earpiece or amplifier. Alternatively the TV sound may be turned up (the video may require slight detuning) and the morse heard that way. The picture collapses when CW is sent since it will only work in FAST mode. Don't try SLOW — it simply won't work. The program has been kept as simple as possible but should contain all you need for an effective tutor.

Program construction

The data storage in line 0 is made first as follows:
enter **1 REM** followed by the comments up to and including **PEEK TO TAN** — which are tokens. Add 36 dots to the **REM** at this point. Use the program below to **POKE** the data in table 1 into the space reserved by the dots.

Table 1: Data Values

16556	2	16557	4
16558	8	16559	16
16560	3	16561	7
16562	15	16563	6
16564	12	16565	24
16566	5	16567	9
16568	17	16569	14
16570	30	16571	21
16572	13	16573	22
16574	11	16575	10
16576	18	16577	27
16578	19	16579	20
16580	25	16581	29
16582	62	16583	60
16584	56	16585	48
16586	32	16587	33
16588	35	16589	39
16590	47	16591	63

```
2 FOR J = 16556 TO 16591
3 PRINT J,
4 INPUT N
5 POKE J, N
6 PRINT N
7 SCROLL
8 NEXT J
```

Enter one value (viz 2 is the first) and press **NEWLINE.** Enter each value in turn from the table. When complete, check thoroughly. Next add using **EDIT** the characters at the end of the **REM** (viz EISH... etc), ending with 5 dots. Then **POKE** in directly the last five values as follows: **POKE 16628,12; POKE 16629,1; POKE 16630,1; POKE 16631,4; POKE 16632,1.** Check the **REM** thoroughly and when satisfied enter **POKE 16510,0** as a direct command. Save the line for safety. It now appears as line zero to prevent accidental erasure.

Next, enter the rest of the program as given. **Table 2** gives the **CODES** of the characters in the string **'O'**, line 1365. Read across the table. **Fig. 2** shows the screen display for option selection, the black square shows the group selected. **Fig. 3** shows the screen checking.

Table 2: Characters in 0$

0	4
1	7
2	10
3	13
4	17
5	21
6	26
7	31
8	36
9	36
.	26
,	36

The program

Some lines in the program require a little explanation and this follows:

lines 1-19. Array **'P'** is for the CW character data. Array **'A'** is for character data in checking output. The variables **S,D,F,LS** are speed, delay, first in group and last in group.

lines 20-280 form the sending sequence. **line 30** — a random number between the first and last pointers to the group (1 and 4 indicate E I S H for example) is generated and line **40** loads the character data into array **'P'. Line 50** loads the corresponding characters for printing.

line 70 onwards perform the sending. Morse characters are held as decimal numbers with bits set according to the dot/dash pattern. **Lines 90 and 100** divide the value by 2 and find the remainder, ie the bit value. **Line 120** is the loop-control which will vary according to the value of **'E3'.** It will either be 10 or 10 + 20 = 30 ie. dot or dash. The construction of this line is correct despite its odd appearance. Your ZX81 manual will explain its operation.

line 130 is the **USR** call to the three machine code bytes **'PEEK TO TAN'** which are simply a way of making a click at the 'mic' socket. Repeated calls to this generate the buzz.

lines 155-160. This is a 'do-nothing'