Robert L. Kurtz W6PRO
#4 Santa Bella Rd.
Rolling Hills CA 90274

World of the Brass Pounders: Receive Morse Code the Easy Way

Microcomputing and amateur radio make an exciting combination, if you haven't already discovered it. This Morse code reader is an excellent example of what we mean.

A great feature of a personal-computing hobby is that it can be used to work with other hobbies. A case in point is this little program to decode and print out Morse code. I originally wrote this in machine language for my KIM system, where it took up less than one page, and finally decided to try it in BASIC.

Even though the program looks simple, it has some unusual surprises, such as self-adaptive adjustment for changes in code speed. In addition, the influence of changes in dash or dot length is weighted so that they must occur five or six times in succession before the computer decides that there has been a bona fide speed change. As a result, an occasional "bad" character will not mess up your copy; the printout is extremely stable and the copy is relatively foolproof. The program also detects the end of the word and prints out a space, if required.

The program's memory needs are minor—just a little over 1K of RAM is required. The program is written in Microsoft BASIC on the KIM computer, but should operate with any relatively fast BASIC. From a hardware standpoint, if your CRT or printer will go to 300 baud, this program will provide excellent copy of Morse code, up to 15 or 20 words per minute. If your terminal operates up to 1200 baud, it will follow the Morse transmissions to well over 30 words per minute.

```
1 REM MORSE CODE READER - WRITTEN BY R. KURTZ - W6PR0
2 RESTORE
3 PRINT CHR$(26):REM CAN DELETE - PUTS CURSOR AT TOP OF PAGE
5 DIM A$(100)
6 FOR N=1 TO 100:READ A$(N):NEXT N
10 A=PEEK(5888) AND 1
11 IF A=1 THEN 10
15 B=0
20 A=PEEK(5888) AND 1:B=B+10
30 IF A=1 THEN C=((5*C)+(2*B))/6:DO=2*DO:DA=2*DA:DO=DO+1:GOTO 100
40 IF B<(.5*C) THEN 20
50 DO=2*DO:DA=2*DA:DA=DA+1
60 A=PEEK(5888) AND 1:B=B+10
70 IF A=0 THEN GOTO 60
80 C=((4*C)+B)/5
100 B=0
110 A=PEEK(5888) AND 1
111 B=B+10
120 IF A=0 THEN GOTO 15
130 IF B<(.5*C) THEN GOTO 110
140 GOSUB 300
150 A=PEEK(5888) AND 1
151 B=B+10
160 IF A=0 THEN GOTO 15
170 IF B<(2*C) THEN GOTO 150
180 PRINT " ";
190 GOTO 10
300 DA=DA*2
310 D=DA+DO
330 IF D>100 THEN D=100
340 PRINT A$(D);
350 DA=0:DO=0
360 RETURN
400 DATA E,T,I,A,N,M,S,U,R,W,D,K,G,O,H,V,F,-,L,-,P,J,B,X,C
410 DATA Y,Z,Q,-,-,5,4,-,3,-,-,-,2,-,-,-,-,-,-,-,1,6,-,/,-
420 DATA -,-,-,-,7,-,-,-,8,-,9,0,-,-,-,-,-,-,-,-,-,-,-,-,?
430 DATA -,-,-,-,-,-,-,-,.,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-
```

Program listing.

Loading the Program

Input the program exactly as written, even though some of the instructions may seem redundant. It is written this way to save operating time—a very important consideration when you are dealing with fast-acting dots and dashes.

Lines 10, 20, 60, 110 and 150 instruct a PEEK to location 5888 (decimal). In the KIM computer, this is a peripheral input address at 1700 (hex). This location reads a total of eight input ports as an eight-bit word.

The AND 1 on lines 10, 20, 60, 110 and 150 assures that the computer is only reading the port to which the incoming Morse is connected. Obviously,

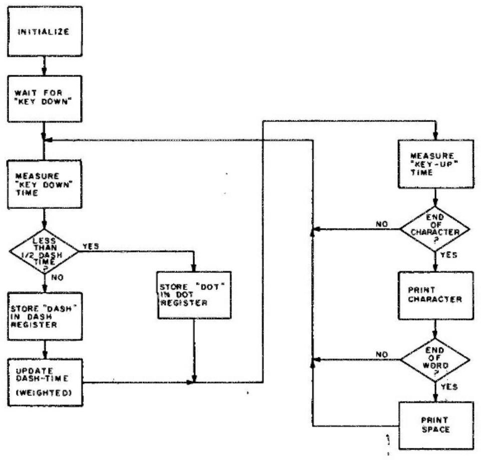

Fig. 1. Simplified flow diagram.

EXAMPLE: D = — ● ●
Initial conditions: Dot register = 0, Dash register = 0.
First period: Input a dash.
  2 times dash register = 2 x 0 = 0.
  2 times dot register = 2 x 0 = 0.
  Add 1 to dash register = 1 + 0 = 1.
    Summary: dash register = 1, dot register = 0.
Second period: Input a dot.
  2 times dash register = 2 x 1 = 2.
  2 times dot register = 2 x 0 = 0.
  Add 1 to dot register = 1 + 0 = 1.
    Summary: dash register = 2, dot register = 1
Third period: Input a dot.
  2 times dash register = 2 x 2 = 4.
  2 times dot register = 2 x 1 = 2.
  Add 1 to dot register = 1 + 2 = 3.
    Summary: dash register = 4, dot register = 3.
End of Character—determine lookup number for D.
  2 times dash register = 2 x 4 = 8.
  Add dot register and dash register = 8 + 3 = 11.
Answer: D = 11.

Example 1.

this must be changed to fit your particular system.

In addition, the program assumes that when a dot or a dash occurs, a logic 0 appears on the input port. This is in agreement with a "key down" shorting the input port to ground, and also in agreement with the hardware interface circuit described later. If your hookup provides a logic 1 during a dot and a dash, then lines 11, 30, 70, 120 and 160 must be changed so that all IF A=1 statements should read IF A=0, and vice versa.

Program Description

Fig. 1 is a simplified flow diagram of the program. The initialization routine (lines 1 through 6) sets the lookup table that will permit the printout of the proper character. The program then waits for a key down to occur (lines 10 and 11). The first part of the operating program (lines 20 through 80) measures the length of time that the key is down and compares this with the stored value for the length of a dash.

If the key is raised in less than one-half of the stored dash time, the computer writes a dot into the dot register (line 30) and goes to the second part of the operating program.

If the key remains down longer than one-half of the dash time, a dash is stored in the dash register (line 50), and the value of the dash time is updated with a one-to-four weighting (line 80). This is accomplished by multiplying the old value of the dash time by four, adding the new value, and then dividing by five. As a result, the stored value of the dash time cannot change drastically from character to character, and the copy is not susceptible to errors from erratic sending habits.

The second part of the program (lines 100 through 190) measures the length of time the key is up. If it's up less than one-half of the dash length, the program assumes that the character is not complete and no printout is provided (see lines 100 through 130). If the key is up longer, the program jumps to line 300, the print-character subroutine. If the key is up longer than twice the dash length, the program assumes that a word is complete and a "space" is printed (lines 170 and 180).

1. If the input signal is a dash:
   A. Double the values in the dot and dash registers.
   B. Add 1 to the dash register (see line 50).
2. If the input signal is a dot:
   A. Double the values in the dot and dash registers.
   B. Add 1 to the dot register (see line 30).
3. If the character is complete:
   A. Double the value in the dash register.
   B. Add the dash and dot registers to obtain the lookup number.
   C. Clear the dot and dash registers.

Table 1.

Lookup Table

The heart of the program is the algorithm that counts the dots and dashes and develops a number used to look up the actual character to be printed. In other words, each combination of dots and dashes in Morse code has a discrete number that commands a given character to be printed. This algorithm has three conditions as listed in Table 1.

Steps 1 and 2 keep repeating until the character is complete. When the program detects a key-up period longer than one-half of a dash length, it is assumed that the character is complete and step 3 is accomplished (lines 300 to 430). The manner in which the lookup number for the letter D is

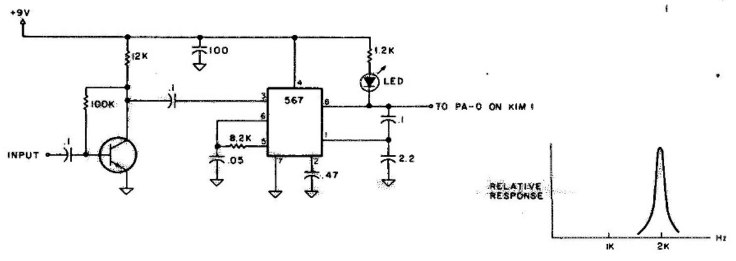

Fig. 2. Interface circuit.

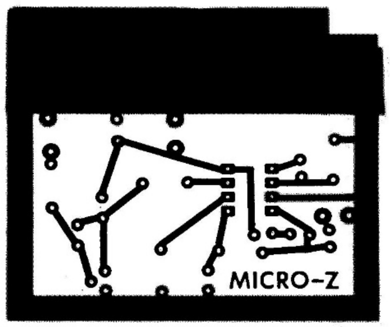

Fig. 3. Circuit board.

formed is shown in Example 1.

Interface Hardware

Fig. 2 shows a typical circuit for connecting your radio receiver to the computer. The NPN transistor is an R/C coupled audio amplifier connected to a type 567 phase-lock loop circuit. The free-running frequency of the phase-lock loop is set by the values of the capacitor and resistor connected to pins 5 and 6 of the 567, and is approximately 2000 Hz. The capacitors on pins 1 and 2 of the PLL adjust the bandwidth to about 100 Hz, and the LED serves as a tuning indicator—that is, it will start blinking when the signal is in the center of this narrow bandpass.

This circuit is compatible with the program, as written, in that the output signal goes to a logic 0 when a dot or a dash occurs. The circuit shown is not my original idea, but has appeared in numerous publications; you may have your own favorite circuit that you would like to use. As long as you maintain the same output logic (a 0 for a dot or dash), there will be no problem.

Incidentally, the circuit has another unique application. Since it is only activated by audio signals over a fairly narrow band, it can also be used to key an audio oscillator set to any frequency desired. When Morse CW comes in amidst a jumble of other signals, the phase-lock loop picks out the signal you want and keys the audio oscillator . . . and that is all you hear. A full-scale outline of the circuit board is shown in Fig. 3.

Adjusting the Program

One of the advantages of writing this program in BASIC is the ease with which the computation constants can be changed. For instance, you may wish to experiment with different algorithms to detect whether a key-down signal is a a dot or a dash . . . to take care of "swing-fisters." This can be accomplished easily by changing the factor in line 40 from (.5*C) to (.25*C) or (.75*C). By the same token, the constants in lines 111 and 151 can be changed to provide more leeway for the formation of characters and spaces.

One final note for KIM users: The 9K Microsoft BASIC is excellent and may be obtained from Micro-Z Co., Box 2426, Rolling Hills CA, for $100. In addition, a full kit of parts for the interface circuit, circuit board, Morse-code reader listings in both BASIC and machine language, and instructions on how to connect it to an audio oscillator are available from Micro-Z Co. for $16.50 postpaid. ■