MICROHOBBY 81, Page 26–27
Utility Routines

DIRECT READING FROM TAPE

Paco MARTIN

Surely you have wondered more than once if it was possible to know the content of a tape of programs without loading them into the computer's memory. Although with certain limitations, this routine fulfills this important task.

One of the biggest inconveniences that poor users who only have a cassette player to store any type of information and have a brand-new word processor "enjoy" is usually the terrible loss of time involved in loading a file that coincidentally it is usually quite long and of which we only know that it is called, for example, "Mechanics 1.7" which most of the time is of no use to us since, as often happens too often, we doubt if version 1.7 is the last or if it was a discarded version, etc, etc... and we lose our precious time during the loading of the evil file to finally realize that indeed, it was not the one we were looking for.

But then our beloved and confident editor in a fit of inspiration (already on the fifth, yes the fifth!, file laughing in his face) found the solution: why wait until the end to discover what the file contians instead of performing a "direct" reading of it? Said and done: we got to work in the research department and after an arduous task (the air conditioning is not working), we found the solution. Here it is: as expected, this routine works entirely in machine code, is 81 bytes long, is relocatable, and, despite the unbelievers, works even in the first 16K of memory.

Roughly speaking, we will tell you that the routine is responsible for printing all the (valid) ASCII codes on the screen as the tape is read until the screen is filled—at which point the process restarts. As we have thought of the possibility of inspecting by this method ANY type of program that is recorded at the normal speed of the ROM SAVE routine, that is, 1500 bauds, the routine checks if the loaded byte is ASCII: if you If you look at the listing you will realize that it does so by checking bit 7 (an ASCII code never has that bit assigned). In this way the routine tends to adjust itself (that's why the first byte of a text will often appear changed).

How does it work

Let us now explain in detail how the routine works:

We'll start by disabling interrupts (DI) and then create the return address (lines 40 to 50). This return address is used by the LOAD and SAVE ROM subroutines and its purpose is to reset the BORDER color, enable interrupts again and if BREAK has been pressed it will produce the subsequent report (in practice this is how we will get out of our program).

The next step is to assign the double DE register the initial screen address (position) (line 60).

Next, a check is made for the BREAK key (lines 70 to 80), returning immediately if it has been pressed.

The routine now prepares to load a “new byte”: register A is set to 1 (line 110) since there are eight bits that make up a byte. Register C takes the color of border 1 and register B is assigned a value according to the time required for the first bit since its role is that of a time counter (line 120).

A call is made to the ROM subroutine (line 130) whose function is to check the presence or not of an impulse in the EAR socket, keep a count of the time and check if BREAK is pressed. If there have been no pulses during the reading time (zeroes and ones are encoded as pulses of different durations) or if BREAK has been pressed, the subroutine returns with the CARRY FLAG set to zero.

When returning, a carry check is carried out: if this is zero then it jumps to BREAK where it is checked if this key has been pressed, if not, the program prepares for a new byte.

To form the value (0 or 1) of the new bit, register B is compared with the “intermediate time” (lines 150 to 160): if this is exceeded, the carry will take a value of one and vice versa. The new bit is then entered into register L (line 170).

Next, register B is assigned with the time constant for the next bit (line 180) and if there is more to be loaded, it jumps to LBITS (line 190).

This operation is done in a “cunning” way since if you remember, L was initially set to 1: with each successive rotation bit 7 passes to carry—and when it is set to 1 it means that the loading of the eight bits has been completed. Bits in register L.

Next, we check bit 7 of L (line 200) to ensure that the byte stored is, as far as possible, represents an ASCII code. If this is not the case (line 210) a new bit is added by jumping to LBITS.

If it passes the test, it is checked if it is a printable code (line 220 to 230) and if it is not, it is defined as if it were a space (line 250).

The next step is to form the address of the character graphic in HL (lines 260 to 310), preserve the D register in C to preserve the print position of the first byte, and then print it (lines 320 to 380) in the position corresponding screen position.

Again, the value of register D is recovered, the printing address (position) is increased and if we have not "filled" the corresponding third of the screen. a jump to NBYTE is made, thus repeating the process (lines 390 to 410).

Each time a third of the screen is filled, the address (position) of the new printing third is assigned (lines 420 to 490).

Practical use

As usual, it will be necessary to type the attached list (list 1) using the Universal Machine Code Loader. Once this is done, do a DUMP at address 40,000 and then save it on tape indicating 40,000 as the start address and 81 as the number of bytes.

The lucky possessors of an Assembler can use it to type in the accompanying Assembler listing.

A practical way to use the routine might be to start the tape, let it fill the screen with the text, stop the tape momentarily to examine it comfortably, and start the tape again.

As a marginal use of the routine we can discover the texts contained in the game programs without even having to load them to play. Unfortunately, very few programs today are recorded at standard loading speed.

One last warning: to initialize the routine you have to RANDOMIZE USR its load address. That is, if we place it at address 40000 with LOAD "" CODE 40000, we will then have to RANDOMIZE USR 40000. Enjoy it.

DESENSAMBLE DE LA RUTINA

```
10        ORG  40000
20 ;
30        DI
40        LD   HL,#53F
50        PUSH HL
60        LD   DE,#4000
70 LBYTE  LD   A,#7F
80        IN   A,(#FE)
90        RRA
100       RET  NC
110 NBYTE LD   L,1
120       LD   BC,#B201
130 LBITS CALL #5E3
140       JR   NC,LBYTE
150       LD   A,#CB
160       CP   B
170       RL   L
180       LD   B,#B0
190       JR   NC,LBITS
200       BIT  7,L
210       JR   NZ,LBITS
220       LD   A,L
230       CP   " "
240       JR   NC,OK
250       LD   L," "
260 OK    LD   H,0
270       ADD  HL,HL
280       ADD  HL,HL
290       ADD  HL,HL
300       LD   BC,15360
310       ADD  HL,BC
320       LD   C,D
330       LD   B,8
340 PCHAR LD   A,(HL)
350       LD   (DE),A
360       INC  HL
370       INC  D
380       DJNZ PCHAR
390       LD   D,C
400       INC  E
410       JR   NZ,NBYTE
420       LD   A,D
430       CP   #48
440       LD   D,#58
450       JR   Z,NBYTE
460       LD   D,#40
470       JR   NC,NBYTE
480       LD   D,#48
490       JR   NBYTE
```

LISTADO 1

```
1   F3213F05E51100403E7F  843
2   DBFE1FD02E010101B2CD  1144
3   E30530F03ECBB8CB1506  1199
4   B030F2CB7D20EE7DFE20  1475
5   30022E2026002929292901  290
6   003C094A06087E122314  356
7   10FA511C20CC7AFE4816  1081
8   5028C5164030C1164818  762
9   BD0000000000000000000  189
```