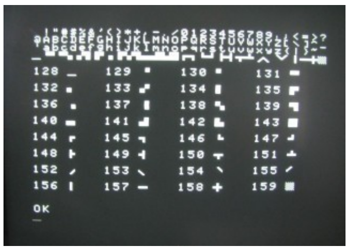
Tiny BASIC program examples

The file ExampleBas.hex contains 8 Tiny BASIC files for the TinyBasRV microcomputer. If we load them into the serial EEPROM memory 24256 or 24512, we get simple sample programs for an overview of what can be easily created with the microcomputer. The programs can also be used for testing functions and modifying the attached samples.
After writing the ExampleBas.hex file to memory, you can run these programs:

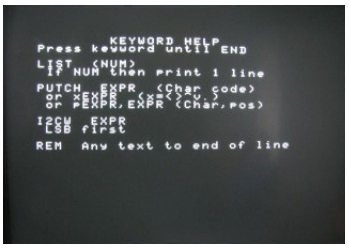
Program: **FILE 0**



```
 !"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_
`abcdefghijklmnopqrstuvwxyz{|}~
128 ▁    129 ▪    130 ▪    131 ▬
132 ▪    133      134 ▮    135
136 ▪    137 ▮    138      139
140 ▬    141      142      143 ■
144      145      146      147
148      149      150      151
152 /    153 \    154 \    155 /
156 |    157 —    158 +    159 ■

OK
_
```

Purpose of the program: **Example of character on the display**
Program description: The top 4 lines list all printable characters. The first 3 lines are characters corresponding to the ASCII table from value 32 to value 127. The fourth line shows semigraphic characters from value 128 to 159. The semigraphic characters are then repeated in a table where each character is assigned its value.

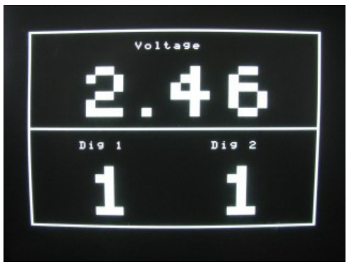
Program: **FILE 1**



```
         KEYWORD HELP
Press keyword until END
LIST  <NUM>
 If NUM then print 1 line
PUTCH  EXPR  <Char code>
 or xEXPR  <x=<>^v.>
 or PEXPR,EXPR <Char,pos>
I2CW  EXPR
 LSB first
REM  Any text to end of line
```

Purpose of the program: **Tiny BASIC command parameter help**
Program description: After entering the command, its parameters appear, possibly with a short comment. The program demonstrates the use of calling the GOSUB subroutine based on the calculated line number.

Program: **FILE 2**

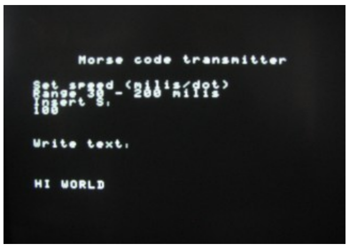


```
          Voltage
           2.46
   Dig 1           Dig 2
     1               1
```

Purpose of the program: **Analog input, digital inputs and graphics test**

Program description: Example of how to store fonts for digits in a 4x6 dot grid in variables in the @ field. A frame is then created on the screen, into which the analog value from the AIN input converted to voltage is written in the upper part. The status of the digital inputs D1 and D2 is displayed in the lower part. The AIN input can be used, for example, to measure the voltage on alkaline or lithium cells (AIN must always be at +).

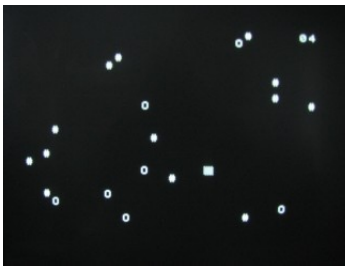
Program: **FILE 3**



```
        Morse code transmitter

Set speed (millis/dot)
Range 30 - 200 millis
Insert S.
100

Write text.

HI WORLD
```

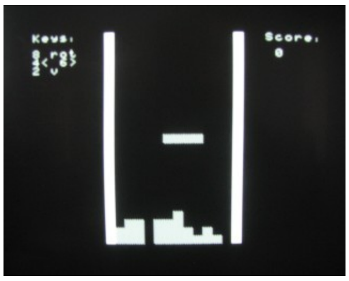
Purpose of the program: **Acoustic and optical Morse code transmitter**
Program description: The @ field first stores the Morse codes of the symbols in the form 1113, where 3 represents a comma and 1 represents a dot. According to the standard, a comma should be 3 times longer than a dot. The characters are encoded in the opposite way. The right digit is transmitted first and then the sequence continues to the left as long as there are digits in the code. The user can enter text up to 64 characters long and after pressing Enter the text is transmitted audibly. At the same time, the code is also signaled optically when an LED is connected to output D1. Initially, it is possible to enter the duration of the dot in milliseconds as a parameter.

Program: **FILE 4**



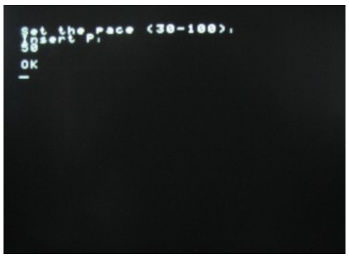
Purpose of the program: **Game "Little Snake"**
Program description: It shows how to generate random numbers (for a random position on the screen where new elements appear). It also shows the capabilities of the Tiny BASIC PUTCH and GETCH commands.
For simplicity, the snake remains the same size – one character. The task is to eat 100 pieces of food while avoiding poisonous characters. The snake crawls over the edge of the screen to the same row or column.

Program: **FILE 5**



Purpose of the program: **Mini TETRIS game**
Program description: The purpose was to show that even a simple microcomputer is a real computer. Because anything that can write and run TETRIS is really a computer. The GETCH and PUTCH commands are used. The shapes of the 7 basic game elements are encoded and stored in 4 variables in the @ array. Each shape is composed of 4 squares, whose position is determined relative to the basic position of the shape on the display. For example, a square one position to the right has the code 1, one position down has the code 32 (32 is the length of the line on the display). The rotation of the shapes is performed using minus operations and multiplication x32 followed by the modulo operation (overflow). The falling of the shapes gradually accelerates.

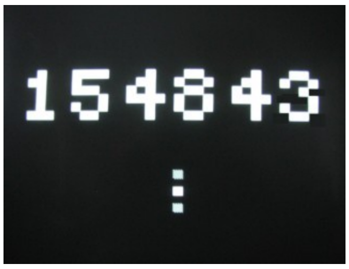
Program: **FILE 6**



```
Set the pace (30-100):
Insert P:
50
OK
_
```

Purpose of the program: **Sample tones and output to expander**
Program description: At the beginning of the program, a song with tones is stored in the @ field in the form:
- tone number (C = 1, D = 2, …), tone duration (1 = sixteenth note, … 16 = whole tone)
The program plays the song at the specified speed until it hits 0. At the same time, it sends a value to the possibly connected PCF8575 expander so that the LED corresponding to the number of the played tone can be lit. The program shows how it is possible to use GOTO with calculation of the target line using a variable.

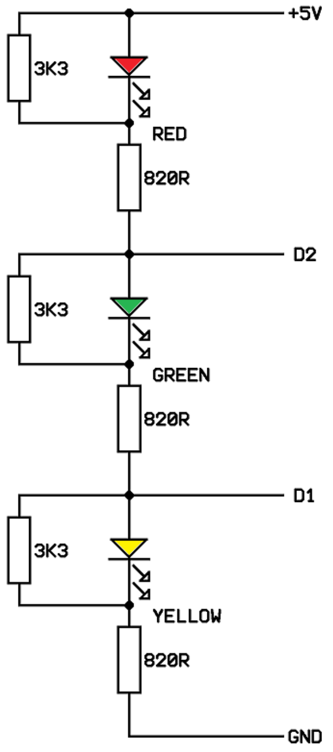
Program: **FILE 7**



```
154843
```

Purpose of the program: **Demonstration of accurate time and control of digital outputs**

Program description: Similar to the example in the FILE 1 program, a large font is used to display the time on the display (the user enters the start time after starting the program). The TIME command is used and it is tested whether the internal time value has already increased by 31914 (the number of ticks in 1 second). At the same time, a traffic light made of 3 LEDs is controlled via outputs D1 and D2 using the DOUT command. Since 2 outputs are not enough to directly control 3 LEDs, a slightly more complex traffic light connection was used:



The green LED, even though it is located at the bottom of the traffic light, must be electrically connected between the red and yellow.
Then the semaphore can be controlled using the following table:

Output D1	Output D2	LED green	LED yellow	LED red
0	0	OFF	OFF	ON
0	1	ON	OFF	OFF
1	0	OFF	ON	ON
1	1	OFF	ON	OFF

It is possible to ensure that all LEDs are turned off by switching both inputs to digital input mode using the DINP command.
The traffic light either flashes yellow after pressing the "b" character on the keyboard, or has normal operation (other keys).