LeoBAS

a set of BASIC extensions
for the Sharp PC-1500 / Radio Shack PC-2

second revision
by Ernst Mulder

Table of contents

What is LeoBAS

LeoBAS is a software extension to add new functionality to the Sharp PC-1500 and Radio Shack TRS-80 PC-2 Pocket Computers. It cannot be installed on just any of these without extra hardware. It can only work with a (writable) memory bank from &8800 to &9FFF bank switched on !PV. Some PC-1500 fan sites have developed or are in the process of developing modules for this purpose. The developer of the PockEmul emulator may someday provide a memory block in this range (I have had contact with the developer about this, no promises).

The PC-2 was introduced to me in a Tandy shop. As soon as I found out (in the shop) the PC-2 had PEEK, POKE and CALL commands my interest was awakened and with help of my father I could one day buy one. My quest to understanding my PC-2 and to start programming it in machine code started earlier on. Helped by excellent articles provided by Tandy's Bruce Elliot and of course the Technical Reference Manual I was able to buy.
I bought an ATARI SDRAM module and installed it inside my PC-2 using unlabelled thin wires from a transformer coil. I do not dare to open my PC-2 again for fear of breaking wires and not knowing anymore how to rewire them.

After some practice I did all of my machine language programming without even using MNEMONICS, I knew all machine codes by heart. In decimal because I found hexadecimal hard to cope with at the time. Stupid me, I just didn't know better. So, for me 154 was and still is the code for RTN.

Sometime in the early 1990's I wrote a PC-2 emulator for my Mac (running Mac OS). It still works today on an emulated 68k Mac using Sheepshaver. It was written in Pascal with competitive multitasking and without threads. Never dared to publish it because it obviously contained the pocket computer's copyrighted original ROM. My emulation was rudimentary, not as detailed as the emulated PC-2 in PockEmul.

At the time I had both a self-assembled Mac Plus and a self-assembled Apple][.I had set-up the PC-2 as a keyboard for the Apple][with wires hanging from its extension port to my Apple][. That way I was also able to copy the PC-2 ROM and my own work by simply slowly "typing" it to a file on the Apple][which I then transferred to my Mac using a serial connection.

Later I became member of the now defunct Dutch Sharp user club called SHACC. Great times. At this wonderful club I probably bought the above mentioned SDRAM module. I fondly remember their AREAD club magazine and their user days!

So, I spent ages figuring out stuff and this is (was) the end result.

In total wrote three Extension Blocks. In chronological order:

BASIC Extension Block I (memory range &9800 - &9FFF)

My initial set of BASIC extensions
New keyboard routine
New character set

BASIC Extension Block II (memory range &9000 - &97FF)

Extra commands that didn't fit in the first part

BASIC Extension Block III (memory range &8800 - &8FFF)

Mathematical constants
Plotter table for the new character set

Block I was the first one obviously. This block implements a couple of new commands I found useful, some to make programs shorter and a new keyboard routine and my character set. My keyboard routine especially really made my PC-2 more useful because I didn't like the fact that the original one didn't have auto-repeat.

I have decided to document my set of BASIC extensions in this manual. Nobody is probably going to use this extension set since the pocket computer is vintage, but I wanted to document it anyway. Some of the contents of this manual is actually from my original manual I once wrote on my Apple][(so I could try to sell this extension set on SHACC user days, actually sold one!). I actually needed a physical Apple][with a Super Serial Card and modem cable, a Mac and the wonderful Virtual][emulator to get the text out as a PDF.

One has to admire hardware emulation.

At the moment I am writing this (late 2022) the knowledge I had in the late 1980's has faded quite a bit. However, using the abundance of information that can be found on the internet nowadays plus what was left of my own documentation I regained quite a lot of it. I even fixed a bug in bank II that had escaped me at the time.

I'm actually a bit amazed of what I could accomplish when I was young. All BASIC tokens have their proper tokens for their specific address ranges on the !PV bank. Several commands that are also available in other official extensions such as ERL, ERN and SPACE$ have their special F0 token to makes that possible without conflicts.

Note that this set of extensions is probably not bug-free, so if you want to use it, it is at your own risk.

The keyboard routine and character set

Extension Block I & III

When cold booting the pocket computer with Extension Block I loaded, an alternative keyboard routine is automatically activated. If this new keyboard routine causes unforeseen problems, it can be EXITed using the EXIT command.

The new keyboard routine adds the following capabilities:

- It has AUTO-REPEAT.

- In REVERSE-mode it is now possible to use the up- and down-arrow keys to browse through the function key definitions. There are also three extra function keys available (one per block, three sets of [F1...F7]).

- The function keys are now also accessible by shifting the number keys. So SHIFT-[1...7] will execute [F1...F7].

- Special characters !,",#,$,% and & can now be accessed by simply pressing the function keys. To access the function keys themselves the SHIFT key needs to be pressed first.

- For people that want to be able to recognise unused memory (such as myself) the function SHIFT-CL will now replace all unused BASIC memory with the underscore character CHR$ 95. Not to be used when you use the memory after the BASIC program for other purposes such as machine language.

- Turing the machine off using the OFF key will cause an AUTO-POWER-OFF, in other words you won't lose any information. If you want to turn it off the regular way, use SHIFT-OFF.

- The lower-case characters shown when using SMALL mode are different than the regular lowercase. Created because I didn't really like the original lower-case. The new lowercase has a new range in the character table and are not ASCII compatible. If you want to input the regular lower-case characters simply use the SHIFT key. This function can be disabled, see chapter "System information".

By using a function which was, I think, meant for the Japanese version of this pocket computer it is possible to add 128 characters to the character set. When enabled the katakana symbols ka na (カナ) are visible on the pocket computer's LCD. The new characters are in the character range 128...255.

Due to the way the BASIC interpreter handles tokens (two-byte codes representing a BASIC command to safe space and speed up the interpreter) there are two areas of characters that behave differently.

Character range 128...223

These characters can be used in BASIC command lines without issues. They can't be typed however. Extension block I introduces a powerful command AWRITE that can directly write to the input buffer. This command can be used to write BASIC command lines containing characters in this range:

Example: AWRITE "10PRINT"+CHR$ 34+"H" *in PRO mode*
+CHR$ 192+"l"+CHR$ 193+"ne ↵
10PRINT"Hélène
(Press back-arrow and return to enter the line)
10 PRINT "Hélène

Character range 161...186, a subset of this range, is my own set of lower-case characters.

Character range 224...255

Bytes in this range are seen as BASIC tokens by the BASIC interpreter. They can therefore not be used in BASIC command lines. They can however be used in strings and using CHR$.

Example: A$=CHR$ 234 ↵
A$+A$ ↵
ΩΩ

Due to lack of space in Extension Blocks I and II the data needed to plot the characters in this character set is stored in Extension Block III. So, if block III isn't loaded the plotter will not be able to print these characters, in fact it will probably crash hopelessly.

Extension Block I
an alphabetical overview

This is an alphabetical overview of the commands in memory block &9800 - &9FFF.

ANGLE

This command produces the value of a stretched angle in the current angular mode.

Example:
```
DEGREE ↵
ANGLE ↵
180
```

ASKEY

This command gives the ASCII value of the key that is currently pressed and will return the value 0 when no keys are pressed. This is actually the short version of the regular command ASC INKEY$.

Example: *in PRO mode*
```
10 BEEP 1:IF ASKEY END
20 GOTO 10
```

AT ERROR

A variant on the regular command ON ERROR GOTO. When a running program encounters an error the program execution is continued after this command.

Example: *in PRO mode*
```
10 AT ERROR PRINT "Incorrect input!
20 INPUT A
30 PRINT "LN(A)=";LN A
40 GOTO 20
```

Please note that the AT command can sometimes cause issues when entering BASIC commands. The following line, for instance, is misinterpreted when entered:

Example: *in PRO mode*
```
10 IFATHENPRINT"Hello ↵
10 IF AT HENPRINT "Hello
```

Avoid this by helping out the BASIC interpreter and type it with a temporary extra character:

Example: *in PRO mode*
```
10 IFA:THENPRINT"Hello ↵
10 IF A:THEN PRINT "Hello
(Now simply remove the surplus ":" after the A)
10 IF A THEN PRINT "Hello
```

AWRITE

This really useful command does what can be seen as the opposite of the regular command AREAD. Instead of reading current value of the input buffer it actually writes its result *to* the input buffer. This result can then be used for further calculations. This can be used for several purposes.

The most common usage would be to create real function shortcuts to be used using the DEF key.

Example: 10 "C"AREAD X:AWRITE COS X *in PRO mode*

Using DEF-C the COS value of the input buffer is calculated and written back to the input buffer and can be used for further calculations.

Another use can be to get special characters and even tokens to the input buffer.

Example: AWRITE CHR$ 241+CHR$ 152 ↵

Press the back-arrow and you'll see the BASIC command LET.

Example: AWRITE"10DATA"+STR$ LN 2 ↵ *in PRO mode*

Press the back-arrow and ↵ and suddenly you have a BASIC line containing the value of LN 2 as a number.

BIN$

This function converts an 8-bit number into its 8-bit binary value as a string.

Example: BIN$ 42 ↵
00101010

BYTE$

This function converts an 8-bit number into its hexadecimal value as a string.

Example: BYTE$ 66 ↵
42

DCR

A command to decrease the value of one or more numerical variables. Non-existing variables (except for arrays) will be created. Useful for loops.

Example:
```
A=1,B=2,C=3 ↵
DCR A ↵
DCR A,B,C ↵
PRINT A;B;C ↵
-1 1 2
```

ERL

The ERL command shows the line number where the last BASIC command error occurred. The result will be 0 when the error has already been cleared.

Example:
```
10 AT ERROR PRINT "Error in line ";ERL :RESUME
```
in PRO mode

ERN

Belongs together with ERL. Its result is the last BASIC error itself and 0 when the error has been cleared.

Example:
```
10 AT ERROR PRINT "Error number ";ERN :RESUME
```
in PRO mode

EXIT

This command will exit a custom keyboard routine (such as the one installed by LeoBAS Extension Block I) and switch back to the original keyboard routine. This can be useful if you are using your own keyboard routine in a part of memory you want to use for something else. If you want to use the memory range &9800 - &9FFF for something else than LeoBAS you will need to EXIT first as well. After a cold start the LeoBAS keyboard routine will be loaded again.

FACT

Use FACT to calculate the factorial of a number. The number must be in the range $0 \leq \text{number} \leq 69$.

Example:
```
FACT 69 ↵
1.711224524E 98
```

FIX

Removes everything after the decimal point of a number. Similar to INT but different for negative numbers.

Example:
```
INT -2.5 ↵
-3
FIX -2.5 ↵
-2
```

FN

A function I'm particularly proud of. It can be used to parameterize programs by defining constants or often used formulae. Best explained by showing some examples.

Example: *in PRO mode*
```
100 PRINT "The surface is";FN "Surface
110 PRINT "The naam is ";FN "Name
120 END
.
.
.
900 "Height"6
901 "Width"7
902 "Name""Rob
903 "Surface"FN "Height"*FN "Width
```

Example: *in PRO mode*
```
10 "X"W OR Z
20 "Y"Z AND Y
30 "X XOR Y"(FN "X" AND NOT FN "Y") OR (NOT FN "X" AND FN "Y")
```

in RUN mode
```
FN "X XOR Y ↵
gives value of (W OR Z) XOR (Z AND Y)
```

FRAC

Sibling to INT, will remove everything before the decimal point of a number.

Example:
```
FRAC 2.42 ↵
0.42
```

GET

This command will wait for a key to be pressed. During this wait the machine can be switched off and AUTO-POWER-OFF will be operational as well. Depending on the kind of variable after GET either the ASCII value or the actual character.

Example:
```
10 WAIT 0:PRINT "Press any key...":WAIT
20 GET A$
30 PRINT "You pressed the";A$
```
in PRO mode

Example:
```
GET A ↵
(Now press the ´1´)
A ↵
49
```

HEXVAL

Will display the decimal value of the hexadecimal data found at the beginning of a string starting from the left and stopping at the first non-hexadecimal character. If the string is empty or does not start with a hexadecimal value the result will be 0. The value should be a 16-bit value.

Example:
```
HEXVAL "4000 is a hexadecimal number ↵
16384
```

Example:
```
HEXVAL "HELLO ↵
0
```

HOME

Does the same as CURSOR 0. One byte shorter in memory steps in a program and a little bit faster. I added this command simply because I had some spare bytes left to fill in this memory block.

ICR

Sibling to DCR. This command increases the value of one or more numerical variables. Non-existing variables (except for arrays) will be created. Useful for loops.

Example:
```
A=1,B=2,C=3 ↵
ICR A ↵
ICR A,B,C ↵
PRINT A;B;C ↵
3 3 4
```

ICURSOR

A command that can be used to implement an inverting cursor at the specified cursor position.

Example: 10 WAIT 0:PRINT "Press a key:? *in PRO mode*
20 ICURSOR 12:FOR I=1TO 20:NEXT I:A=ASKEY:IF A=0GOTO 20
30 WAIT :PRINT "You pressed:";A:END

INV

Inverts the whole screen, a column, individual dots or even a combination thereof.

Example: 10 INV *in PRO mode*
(Inverts the whole screen)

Example: 10 INV 5 *in PRO mode*
(Inverts column number 5)

Example: 10 INV 2,10 *in PRO mode*
(Inverts dot (2,10))

Example: 10 INV x1,y1,x2,y2,... *in PRO mode*
(Inverts several dots)

Example: 10 INV x1,y1,x2,y2,z *in PRO mode*
(Inverts several dots and column z)

Example: 10 PAUSE "** An Example ** *in PRO mode*
20 INV :INV :INV
30 FOR Y=0TO 6:FOR X=0TO 155
40 INV X,Y
50 NEXT X:NEXT Y
60 FOR Z=0 TO 155
70 INV Z
80 NEXT Z
90 GOTO 20

KEYWAIT

Similar to GET but without output to a variable. The command will wait for a key to be pressed. During this wait the machine can be switched off and AUTO-POWER-OFF will be operational as well.

Example: 10 WAIT 0:PRINT "Press any key... *in PRO mode*
20 KEYWAIT :END

PRES

Clears one or more dots on the screen.

Example:
```
10 PAUSE "No dot on the i
20 PRES 86,0
30 KEYWAIT :END
```
in PRO mode

PSET

Like PRES but this time to set one or more dots on the screen.

Example:
```
10 PAUSE "Dots on the a
20 PSET 73,0,75,0
30 KEYWAIT :END
```
in PRO mode

RCP

I saw this function (and the FACT function and the TEN and SQU functions found further on in this document) on the Sharp PC 1401 and thought they might be handy to have. The result of RCP will be the reciprocal value of the numerical non-zero argument.

Example:
```
RCP 2 ↵
0.5
```

RESUME

You might have encountered this function earlier in this document. It clears an error caught by ON ERROR GOTO or AT ERROR and will continue the program at the command directly after where the error occurred. Can be used *in RUN mode* as well to do the same after a program has stopped because of an error.

Example:
```
10 FOR A=0TO 10
20 PRINT "LN(A)=";LN A
30 NEXT A
40 END
```
in PRO mode

```
RUN ↵
ERROR 39 IN LINE 20
CL
RESUME ↵
(The program will continue)
```
in RUN mode

ROUND

Used to round of numbers after the first decimal.

Example: ROUND 5.4999 ↵
5

Example: ROUND 5.5 ↵
6

Example: ROUND -4.5 ↵
-4

Example: ROUND -4.50001 ↵
-5

SCROLL

Scrolls the screen to the left. What disappears on the left reappears on the right. Without an argument it will scroll the screen one column. With an argument it will scroll the specified number of columns. The scrolling speed can be specified using the WAIT command with a value in the range 1 ... 255.

Example: 10 WAIT 2:PRINT "Scrolling example *in PRO mode*
20 SCROLL 156: END

SHIFT

Like SCROLL but what disappears on the left doesn't come back at the right.

Example: 10 WAIT 1:PRINT "My mind is going! *in PRO mode*
20 SHIFT 95: END

SPACE$

Example: PRINT "5";SPACE$ 5;"Spaces ↵
5 Spaces

SQU

Quite easily implemented by a single jump to an existing ROM function this command can be used to square a number.

Example: SQU -5 ↵
25

STRING$

Create a string of specific length and of a specific character. The first argument is the length, the second the ASCII code of the character.

Example: STRING$ (5,65) ↵
AAAAA

TEN

Calculate 10 to the power of the argument. Produces an error (which is a bug) on arguments with a value of -100 or smaller. The actual result should be rounded to 0 for those negative arguments.

Example: TEN 4 ↵
10000

Example: TEN .5 ↵
3.16227766

TME$

Gives the value of STR$ (TIME * TEN 4).

Example: TME$ ↵
127235358
(If the timestamp of that particular moment
was January 27 at 23:53:58)

VMJ

A function meant for users using machine language programming. It can be used to look up addresses of vector-jumps of this pocket computer.

Example: VMJ &E0 ↵
52619

WORD$

WORD$ is the 16-bit equivalent of BYTE$ and will give the two-byte hexadecimal value of a 16-bit number.

Example: WORD$ VMJ &E0 ↵
CD8B

ZERO

Zeroes or clears the value of all the argument variables. Non-existing variables (except for arrays) will be created and emptied immediately. Handy for program initialisation. Can be used *in RUN mode* as well as in programs.

Example:
```
10 ZERO A,B,AA$,L$
20 ...
```
in PRO mode

Extension Block II
an alphabetical overview

This is an alphabetical overview of the commands in memory block &9000 - &97FF.

ADDRESS

The value of the 16-bit pointer at the address of the 16-bit argument.

Example: ADDRESS &C05C ↵
50820

BINVAL

Sibling to HEXVAL. Will display the binary value of the binary data found at the beginning of a string starting from the left and stopping at the first non-binary character. If the string is empty or does not start with a binary value the result will be 0. The value should be a 16-bit value.

Example: BINVAL "101010 is a binary number ↵
42

BUSY

If you do not want to show the BUSY symbol at the top left of your screen when running a program, you can now switch it on or off by entering BUSY OFF in your program. Switch it back on with BUSY ON.

CAP$

Turns all alphabetical characters in a string into capital characters. Works for regular lower case as well as for the new lowercase introduced by Extension Block I.

Example: CAPS$ "AbBa ↵
ABBA

CHAR$

Returns a specific character in a string. The first parameter is the string, the second is the character position as a number in the range 1...80. An empty string is returned if the number is greater than the length of the string.

Example: CHAR$ ("ABCD", 2) ↵
B

CLOCK$

This command shows the time in a neat manner using the character set of Extension Block I. The second example shows a nice clock one-liner using some LeoBAS commands.

Example: CLOCK$ ↵
20:08 [42]

Example: 200 "CLOCK"WAIT 0:BUSY OFF: *in PRO mode*
PRINT STRING$ (26,255): CURSOR 8:PRINT SPACE$ 10:
REPEAT CURSOR 9:PRINT CLOCK$:UNTIL

DATE

Returns the current day of the month.

Example: DATE ↵
30

DELETE

Deletes a range of BASIC command lines. All command lines present in the specified range are deleted. If there are no BASIC command lines in the specified range an error will be displayed.

Example: DELETE 100,200 *in PRO mode*
(Deletes line 100 up to and including line 200)

The first or the second argument may be omitted. Two examples:

Example: DELETE ,200 *in PRO mode*
(Deletes all lines up to an including 200)

Example: DELETE 200, *in PRO mode*
(Deletes line 200 and the lines following)

DNOT

Decimal NOT. A proper function to negate numbers. The standard NOT command sometimes does not suffice. This command turns non-zero real numbers into zero, and turns zero into -1.

Example: DNOT 0 ↵
-1

Example: DNOT 207422 ↵
0

Example: DNOT -42 ↵
0

DUMP$

With one argument this returns a string of 16 characters found at the specified address. With two arguments the length of the string can be specified to a maximum of 80 characters.

Example: DUMP$ &C34F↵
NEW0? :CHECK BRE

Example: DUMP$ (&9802, 7) ↵
EXT 1.1

FAST

Belongs with SLOW found further on in this document. SLOW slows down the pocket computer, useful for users that have increased the clock speed with a 5 MHz quartz crystal. SLOW, using an interrupt routine, essentially slows down the machine to half speed so that the plotter doesn't overspeed. FAST cancels this slowdown again.

INSTR

Returns the position of a string within another string. Returns zero when there is no match.

Example: INSTR ("ABCDE", "B") ↵
2

Example: INSTR ("Where is my bike", "my") ↵
10

LSB

Returns the Most Significant Byte of a two-byte number as a decimal value.

Example: LSB 12345 ↵
57

Example: BYTE$ LSB &9876
76

MLTPOKE

Handy command to poke more than one byte at a time. Can be used to poke numbers, text, addresses and hexadecimal values. Multiple arguments are possible. Handy in combination with DUMP$. Best to show an example.

Example: MLTPOKE &8000,#A12345, #T"HELLO",42,#H"5F5F

#A to specify a two-byte address
#T to specify a string of characters
#H to specify a string of hexadecimal numbers
no prefix for just a single byte

MSB

Returns the Least Significant Byte of a two-byte number as a decimal value.

Example: MSB 10940 ↵
42

NAME$

This command tries to find the name of the 16-bit argument interpreting it as a BASIC token. When no token is found the string "~" is returned.

Example: NAME$ &E280 ↵
DELETE

ODD

Returns whether a number in the range -32768 ... 32767 is odd or not.

Example: ODD 1 ↵
1

POL

Calculates the length of the vector specified by horizontal and vertical position. The first argument is the horizontal (real) position, the second argument is the vertical (imaginary) position.

Example: DEGREE ↵
POL (SQR 2,SQR 2) ↵
2

POL#

Calculates the angle of the vector specified by horizontal and vertical position. The first argument is the horizontal (real) position, the second argument is the vertical (imaginary) position.

Example: DEGREE ↵
POL# (SQR 2,SQR 2) ↵
45

POP

To keep track of GOSUB...RETURN, FOR...NEXT and REPEAT...UNTIL flow control commands the BASIC interpreter maintains a stack. Under certain circumstances (a.k.a. tricky programming) it might be useful to be able to manipulate that stack. By the way, REPEAT...UNTIL is a LeoBAS addition, described elsewhere in this document and it uses the same stack mechanism.

POP S Clears the BASIC stack completely.
POP N Clears the last FOR...NEXT or REPEAT...UNTIL stack entry.
POP R Clears the last GOSUB...RETURN stack entry.

POP S can only be used on its own.

POP N and POP R can be used in combination, so to escape two nested FOR...NEXT loops the command POP N,N can be used. As I wrote, useful for really tricky programming. Beware that S, N and R are switches here, not the names of variables.

The following example shows how it can be useful to clear a stack entry:

Example:
```
10 WAIT 0:PRINT "Press a key:?                                  in PRO mode
10 ZERO E: REPEAT A=ASKEY :IF DNOT AICR E:UNTIL E=10:
    ICURSOR 12:GOTO 10
20 POP N:WAIT :PRINT "You pressed:";A:END
```

This example is a more responsive version of the example code shown with the ICURSOR command earlier in this document. The UNTIL command is located after an IF and breaks the loop when the IF condition is true. The POP N is necessary after the break to clear the stack and prevent a stack overflow.

REC

Calculates the real part of the vector specified by length and angle. First argument is the length, the second argument is the angle.

Example:
```
DEGREE ↵
REC (1,60) ↵
0.5
```

REC#

Calculates the imaginary part of the vector specified by length and angle. First argument is the length, the second argument is the angle.

Example:
```
DEGREE ↵
REC# (1,60) ↵
0.8660254038E-01
```

RELOC

Moves the current BASIC program to a new location in memory. Simply type RELOC followed by the new memory location to move your program. You might for instance want to move your program up 100 bytes to be able to store a piece of machine code before it. Afterwards you can move your program 100 bytes back again. Remember that the current start of your BASIC program in memory is STATUS 2 - STATUS 1.

RENEW

If you have accidentally entered NEW and wish you hadn't, with RENEW you can get your program back, at least if you haven't entered a new one in the meantime.

Together with RELOC and knowledge of the size and locations of your BASIC programs you can use RENEW and RELOC together to have multiple BASIC programs in memory. Use NEW with the address where your program is stored (or RELOCated) and use RENEW to get it back.

REPEAT UNTIL

This is a variant on FOR...NEXT. It will repeat the code following REPEAT until the condition after UNTIL is true. When there is no condition after UNTIL, the condition is assumed as TRUE. The following example shows the complete character set.

Example: 10 WAIT 0: I=32 *in PRO mode*
20 REPEAT PRINT CHR$ I;:ICR I: IF I>57CURSOR 25:SHIFT 6
30 UNTIL I=256:KEYWAIT :END

REPEAT LOOP

A variant on REPEAT...UNTIL. Using REPEAT this way a numeric counter variable should be specified after LOOP. LOOP will decrease the value of the variable and will break the loop when the value of the variable is -1. The LOOP variable is an integer. If it isn't initially, it will be after the first loop step.

Example: 10 A=10:WAIT 0 *in PRO mode*
20 REPEAT PRINT A:LOOP A:END
(The value of A is now -1)

REPLACE$

The name of this function seems to indicate this is a sting-returning function. It isn't. It can be used to manipulate an alphanumeric variable. The first argument is the name of the variable, the second the position. The replacement string is specified after the = symbol.

Example:
```
A$="This is a great! ↵
REPLACE$ (A$, 9)="SUPER ↵
A$ ↵
This is SUPER!
```

REV$

Returns the reverse string of the string argument.

Example:
```
REV$ "olleH ↵
Hello
```

RGOTO
RGOSUB
RRESTORE

Relative versions of GOTO, GOSUB and RESTORE. RGOTO -10 will try to GOTO the BASIC line number 10 less than the current one.

To make looping the current line easier RGOTO without an argument will jump to the beginning of the current line.

SLOW

See FAST. Slows overclocked machines down to be able to use the pen plotter.

Extension Block III
an alphabetical overview

This is an alphabetical overview of the commands in memory block &8800 - &8FFF.

C#

Meaning:	The constant speed at which all electromagnetic radiation including light travels in a vacuum.
Value:	299792458

E#

Meaning:	The number e to the power of 1
Value:	2.718281828

E0#

Meaning:	The permittivity of vacuum (the ability of electrical fields to pass through a classical vacuum).
Value:	8.85418782E-12

Ee#

Meaning:	Elementary charge (the electric charge carried by a single proton).
Value:	1.6021892E-19

G#

Meaning:	The IAU gravitational constant.
Value:	6.672E-11

H#

Meaning:	Planck’s constant.
Value:	6.626176E-34

K#

Meaning:	Boltzmann’s constant.
Value:	1.380662E-23

Me#

Meaning:	The mass of an electron.
Value:	9.109534E-34

Mp#

Meaning:	The mass of a proton.
Value:	1.6726485E-27

Mn#

Meaning: The mass of a neutron.
Value: 1.6749543E-27

Mu#

Meaning: The mass of a muon.
Value: 1.883566E-28

NA#

Meaning: The Avogadro Constant.
Value: 6.022045E23

R#

Meaning: The gas constant.
Value: 8.31441

T0#

Meaning: The absolute zero temperature in degrees Celsius.
Value: -273.15

U0#

Meaning: Magnetic constant (vacuum permeability).
Value: 1.256637061E-06

Ue#

Meaning: Electron magnetic moment.
Value: 9.284832E-24

Up#

Meaning: Proton magnetic moment.
Value: 1.4106171E-26

Uu#

Meaning: Muon magnetic moment.
Value: 4.490474E-26

V0#

Meaning: The volume of 1 mol of a gas at NTP.
Value: 22.4136

System information

Used memory blocks:

Bank switched on !PV

Extension Block I:	&9800 - &9FFF
Extension Block II:	&9000 - &97FF
Extension Block III:	&8800 - &8FFF

Other information:

Keyboard table:	&9A80 - &9AFF
Character set table:	&9B00 - &9D7F
Plotter-table:	&89BE - &8D6A

System switches (Changing these will only work when the memory is writable):

LeoBAS lower-case on:	POKE &99E5, &60
LeoBAS lower-case off:	POKE &99E5, &20
CA memory-fill byte:	&99CA