SandBox_3D

The Last HP-41 Plug-in Module?

User's Manual and Quick Reference Guide

Compiled by Ángel M. Martin

December 2003

Sandbox 3D – The last HP-41 Plug-in Module?

1. Introduction.

The "Sandbox_3D" started as a rescue project, aimed to bring back to life many of the MCODE functions and routines published in old calculator user groups journals. Its name doesn't only refer to the silicon content of the IC's, but also to the character of experimental ground, employed by the author as a vehicle to learn MCODE while compiling a sensible collection of functions worth grouping together and thus adding a permanent value to the HP-41's legacy.

This module is mainly a tribute to the original authors of the many routines gathered on it, some of them learning their way into MCODE and willing to share their discoveries with all the user's community. Some others with a skilful command of the MCODE art, and yet gracious enough to also make their work available to the world. This module simply had to be done, for all these wonderful contributions had but gone forgotten undeservedly!

A very important part of the functions comes from Ken Emery's "*MCODE For Beginners*" book, which to the author's knowledge remains the single one and only published work for MCODE programming, and thus an obliged reference to any compilation like this. They were the seed around which the different sections developed, in an archaeological search that has now concluded.

Soon enough a few themes emerged from the diverse collection of functions: housekeeping utilities, alpha functions, MLDL functions, math functions, and even fun stuff. This classification allowed the author the opportunity to write a few new routines to round up the contents of the different sections, as well as seeking for a few key missing functions from other modules (most importantly the Hyperbolics from the *AECROM* module, published by RedShift Software).

The nature of the 41 module architecture soon forced compromises in the choice of functions and the internal distribution of these. It was quickly apparent that with a limitation of 64 functions per page in the FAT, there were too many of them to fit even in an 8k module. The author has borrowed the concept of *multi-function*, originally used in the HEPAX modules, to partially go around such a limitation. Note however that while the SandBox' multifunction implementation allows exceeding the function limit, it isn't as powerful as the original, as only one of them is programmable.

Many functions will be for sure quickly recognized by many 41 user, while a few others will come as a fresh surprise to some. All in all, the author hopes that this compilation, faulty as it might be, represents an interesting and valuable contribution to the rich and wonderful legacy of the HP-41, arguably the best calculator system ever produced. Enjoy!

2. Five sections in two pages.

The SandBox is divided in five sections, as follows:

2.1. "Sandbox 3D": or the General Utilities section.

This section comprises many popular routines never before published as MCODE functions, or not grouped together in the way they are here. Their scope ranges from the all-popular Key Assignment utilities (KALNG?, KACLR, KAPCK, LKAON, LKAOFF), to innovative additions to the extended memory manipulation (CALLXM, CLEM, RSTCHK, XMROOM, ARCLCHR), not forgetting the extended flag control (FS?S, FC?S, SFX, CFX, TOGF, STOF, RCLF).

Three multi-functions are also included in this section:
- **BIT56**, grouping many bit manipulation functions that work on all the 56 bits of the registers;
- **ST<>ST**, for many stack register exchanges (Y<>Z, Y<>T, etc.) and
- **XTRABOX**, for a group of miscellaneous functions.

A common characteristic of these three is their catalogues, which list the names of all the sub-functions and then return to the command index prompt. This is a usability enhancement not present in the HEPAX module, and while a full control of the listing speed and stopping isn't implemented, a partial speed control is possible. The catalogues are always accessed with index zero as answer to the prompts.

On a class by itself is CSST, to display a continuous SST listing of a FOCAL program, regardless whether it'd be private or not, and both in RAM or ROM memory. Speed control and other subtleties make it a unique function in all senses.

2.2. "Window Nut": The Alpha and Display section.

The alpha registers and the display constitute the subject of the 27 functions grouped within this section. Some *classic* functions (like XTOAL, A>RG, RG>A, VIEWA) and a few new ones taking advantage of the Halfnut display capability and extended character set (aVIEW, CHRSET, CTRST, CTRST?, CNT+, CNT-), sorely missing in the basic machine and any of its subsequently developed extensions. A few functions for sub-string extraction and character conversion borrow their names and functionality from their BASIC counterparts (LEFT$, RIGHT$, MID$, LOW$, UPR$).

Question: is a 1-line display enough? (Valid) Answer: it is for many more things that we give it credit for these days (of inflated *fatware* and wondrous PC's)

2.3 "Math Functions"; Semi-advanced and complementary Mathematics.

The original purpose of a calculator is probably that of performing mathematical operations upon numbers. Even with all its extensions, the 41 system doesn't comprise the most powerful math set available in a calculator (such award will probably go to the HP-15C, a masterpiece on its own – see appendix). The SandBox doesn't change that fact either, but at least corrects some historical inequities and brings a few functions to the 41 platform that should have always been there: simple complex arithmetic (Z+, Z-, Z*, Z/, 1/Z), hyperbolic functions (SINH, COSH, TANH and their inverses, all from the AECROM), Stack and Registers SORT, Recall Math, and in a class by itself, a super-fast quadratic equation root finder (QROOT).

Another set of handy routines include determining whether a number is prime, finding its smallest divisor if not (PRIME?); showing fractions from decimal numbers (DFRAC); and base conversion functions (T>BS for generic decimal to any base, and H>D, D>H for hex to and from decimal).

2.4 "Hacker Lab"; or where things can get dangerous.

This is the section where MLDL functions are included, as well as a few others for RAM manipulation (like the wonderful RAMEDIT, RCLB, STOB, X<>B, aNRCL). The ubiquitous *Code* and *Decode* (NNN>HEX, HEX>NNN), direct entry of NNN's (HEXIN, HXENTRY), and other essential functions to work with q-ROM and MLDL devices. Of particular interest to MLDL owners would be CHKROM and SUMROM, to verify and calculate the checksum word, and XQ>XR, to convert global XEQ into XROM calls (taken from the RAMBOX, published by W&W GmbH).

2.5. "Playground"; or the fun stuff at last.

A handful of functions to provide alternative BEEP tones (CLAXON, RASP), get the calculator to buzz on CPU activity (BUZZON, NOBUZZ), or a challenging High-Rollers game (ROLLERS) for those precious relaxing moments...

3. The functions in detail.

The remaining sections of this document describe the usage and utilization of the functions included in the SandBox Module. While some are very intuitive to use, others require a little elaboration as to their input parameters or control options, which should be covered here. Reference to the original author or publication is always given, for additional information that can (and should) also be consulted.

3.1. Housekeeping Utilities.

Sorted by the following functional groups, loosely defined:-

3.1.1. BIT56 Multi-function.

Multifunction grouping the following bit-manipulation sub-functions:

Index	Function	Author	Description
000	**SUBCAT**	Ángel Martin	Lists the sub-functions names with indexes.
001	**X+Y**	Gordon Pegue	Bitwise addition of values stored in X and Y; Result left in X
002	**Y-X**	Gordon Pegue	Bitwise subtraction of values in X and Y, Result left in X
003	**AND**	Gordon Pegue	Logical AND of values in X and Y, Result left in X
004	**OR**	Gordon Pegue	Logical OR of values in X and Y, Result left in X
005	**NOT**	Gordon Pegue	Logical NOT of values in X and Y, Result left in X
006	**RXR**	Gordon Pegue	Rotate right 56-bit field in X one digit (4 bits)
007	**RXL**	Gordon Pegue	Rotate Left 56-bit field in X one digit (4 bits)
008	**BRXL**	Gordon Pegue	Rotate Left 56-bit field in X one bit w/ wraparound
009	**RLN**	Gordon Pegue	Rotate Left 56-bit field in Y N digits, w/ N is in X
010	**RRN**	Gordon Pegue	Rotate Right 56-bit field in Y N digits, w/ N in X

Although this function is programmable, when in a program there's no choice for index, and the function **X+Y** will always be executed.

3.1.2. Catalogs and direct access to functions.

BLCAT	**[Block Catalog]**	Author: VM Electronics	Source: HEPAX Module

Lists the first function of every non-empty ROM block (i.e. Page), starting with Page 3 in the 41 CX or Page 5 in the other models (C/CV/BlankNut). The listing will be printed if a printer is connected and user flag 15 is enabled.

No input values are necessary. The displaying can be halted while any key (other than R/S or ON) is being depressed, resuming its normal speed when it is released again.

ROMCAT	**[ROM CATalog]**	Author: J.D.Dodin	Source: Au Fond de la HP-41

Lists the functions on the module which XROM number is in X. Once the module is finished, the listing continues with all the other modules plugged in on pages with higher number than the first one.

XROM	**[Xeq ROM]**	Author: Clifford Stern	Source: PPCJ V12 N3 p37

A very special prompting function. Allows direct entry of any function included in a plug-in module, by introducing its XROM number first and then the function number. For example, to call the function "XTRABOX" you'd input XROM 08,35.

This allows access to ROM header functions, such us "*–Sandbox 3d*", (XROM 08,00). Note that while XROM is not programmable, the function called can be entered into a program, thus it isn't necessary that the ROM be present to introduce its corresponding functions.

CSST	**[Continuous SST]**	Author: Phi Trinh	Source: PPCJ V9 N7 p49

Sequentially displays the program steps of the program pointed at by the Program Counter (PC). It's equivalent to using the SST key multiple times, and thus its name. The delay between lines shown can be adjusted by pressing any keyboard key, see the original source for further details. To use it, position first the PC at the target location (using GTO or similar).

3.1.3. Buffers and Key Assignment functionality.

BLNG?	**[Buffer Length Finder]**	Author: W&W GmbH	Source: RAMBOX ROM

Returns the length in registers of the buffer which id# is provided in X. Buffers are created by different modules (CCD, Advantage, Plotter, etc) for temporary or permanent data storage, and it's beyond the scope of this manual to provide further details on their creation and properties.

The following table (necessarily incomplete) lists some of the buffers known:

Buffer id#	Module / Eprom	Reason
1	David Assembler	MCODE Labels already existing
2	David Assembler	MCODE Labels referred to
3	Eramco RSU-1B	ASCII file pointers
4	Eramco RSU-1A	Data File Pointers
5	CCD Module, Advantage	Seed, Word Size, Matrix Name
6	Extended IL (Skwid)	Accessory ID of current device
7	Extended IL (Skwid)	Print Cols, number & width
10	Time Module	Alarms information
11	Plotter Module	Data and barcode parameters
12	IL Development, CMT-200	IL buffer and monitoring
13	CMT-300	Status Info
14	Advantage	INTEG & SOLVE scratch
15 (*)	Mainframe	Key Assignments

(*) KA area isn't really a buffer.

KACLR	**[Clear Key Assignments]**	Author: HaJo David	Source: PPCJ V12 N4 p24

Clears all key assignments presently configured on the USER keyboard. Very similar to CLKEYS function of the X-Functions module, but with added functionality: it requires the literal string "OK" in the alpha register to perform the clearing. If the string "OKALL" is found, then not only the KA registers but all the buffers will be cleared as well.

KALNG?	**[A Registers size finder]**	Author: W&W GmbH	Source: RAMBOX ROM

Returns the length in registers of the Key Assignment area in RAM memory. It requires no input values. (Note that this cannot be done with BLNG? above, using 15 in X).

KAPCK	**[Pack Key Assignments]**	Author: HaJo David	Source: PPCJ V12 N4 p24

Packs the key assignments registers area of the 41 RAM memory. This can recover some registers held up for key assignments by the calculator but not being used, which frequently occurs after de-assigning keys.

LKAOFF	**[Suspends Local KA]**	Author: Ross Cooling	Source: PPCJ V13 N2 p37
LKAON	**[Reactivates Local KA]**	Author: Ross Cooling	Source: PPCJ V13 N2 p37

LKAOFF Suspends the local key assignment, that is those in the first two rows un-shifted (A-J), plus the first row shifted (a-e). This permits the usage of these keys as local labels within a program, and thus not being overwritten by their global assignment.

LKAON Reactivates the Key assignments suspended by LKAOFF. These two functions should be used together to temporarily suspend and then reactivate the local assignments.

3.1.4. Extended User Flags control.

FS?S	**[Is Flag Set and Set]**	Author: Ken Emery	Source: PPCJ V11 N6 p11
FC?S	**[Is Flag Clear and Set]**	Author: Ken Emery	Source: PPCJ V11 N6 p11

Analogous to the mainframe functions FC?C and FS?C, only that the final action is to leave the tested flag enabled instead of disabled. Like them, the execution skips one program line if false.

SFX	**[Set Flag by X]**	Author: Michael Katz	Source: HPX V1 N6 p7
CFX	**[Clear Flag by X]**	Author: Michael Katz	Source: HPX V1 N6 p7

Sets or clears the user flag which number is in the integer part of the value in X. Contrary to the mainframe built-in functions, they are not limited to the first 30 User flags (0-29).

TOGF	**[Toggle Flag]**	Author: Ken Emery	Source: PPCJ V11 N6 p11

Toggles the status of the user flag which number is in the integer part of the value in X. Also allows for any flag to be toggled (0-55)

STOF	**[STO Flags]**	Author: Hajo David	Source: PPCJ V12 N5 p44
RCLF	**[RCL Flags]**	Author: Hajo David	Source: PPCJ V12 N5 p44

Identical to the STOFLAG and RCLFLAG functions on the X-Functions module. Stores or recalls the status of the first 43 user flags into a control string in X. This string can be stored into any data register for subsequent use with the complementary function.

3.1.5. Expanded Extended Memory control.

CALLXM	**[Call program in EM]**	Author: Ross Wentworth	Source: PPCJ V12 N3 p48

Transfers program execution to a program in Extended Memory which global label name is stored in Alpha. The program can be anywhere in EM, but its entire length must be contained within a single XM module (or the XM included in the XF/M module). All GTO's must also be precompiled before hand, or the execution will fail.

CLEM	**[Clear EM]**	Author: Hakan Thorngren	Source: PPCJ V13 N2 p14

Clears ALL Extended Memory. No input parameter is required.

XMROOM	**[Extended Memory ROOM]**	Author: Clifford Stern	Source: PPCJ V12 N3 p38

Returns the number of available registers in Extended Memory. Identical to the CX function EMROOM. No input parameter is required.

RSTCHK	**[Reset Checksum]**	Author: Hakan Thorngren	Source: PPCJ V13 N2 p14

Resets the checksum byte of a program file in Extended memory. Use it when this byte has been corrupted and the "CHECKSUM ERR" message is shown when trying to load a program from Extended Memory to main RAM. Requires the program file name in Alpha as input value.

ARCLCHR	**[ARCL Characters]**	Author: Hakan Thorngren	Source: PPCJ V13 N7 p19

Appends to alpha the number of characters specified in X from the current ASCII file, starting from the current pointer position (determined by SEEKPT or SEEPTA). Similar to but much more flexible than ARCLREC, in the X-Functions module.

3.1.5. Size and location Finders.

BLNG?	**[Buffer Length Finder]**	Author: W&W GmbH	Source: RAMBOX ROM

See description under section 3.1.3. above.

CRTN?	**[Curtain location finder]**	Author: Unknown	Source: MMEPROM

Returns to X the absolute address of the curtain (i.e. separation between program and data registers). No input value is required. The general equation is:

- *Total Registers = Data Regs + Program Regs*,

Where: Total Regs=512 on the CV and CX models.

DREG?	**[Data Registers Finder]**	Author: Ken Emery	Source: MCODE for beginners

Returns to X the number of Data registers allocated by SIZE. Equivalent to the SIZE? Function of the X-Functions module. No input value is required.

FREG?	**[Free Registers Finder]**	Author: Ken Emery	Source: MCODE for beginners

Returns to X the number of available (free) program registers in Main Memory. No input value is required.

SigmaRG?	**[Statistical Regs Finder]**	Author: Unknown	Source: MMEPROM

Returns to X the register number of the current location of the Statistical Registers, which by default is 11. No input value is required.

FLNG?	**[Disk File Length]**		Author: Unknown	Source: MMEPROM

Returns to X the length in registers of the (primary) mass storage file which name is specified in Alpha. If no HP-IL is present on the system an error message will be shown.

3.1.6. Other housekeeping functions.

NNRCL	**[Non-normalized RCL]**	Author: Sid Kelly	Source: PPCJ V12 N10 p6

Recalls to the X register the value of the data register which number is input in X, without normalization. Not valid to access the Alpha and Status registers (see "aNRCL" below)

CLRSAF	**[Clear ALL Rgs and Flags]**	Author: Gordon Pegue	Source: PPCJ V12 N3 p40

Almost a Memory Lost, but without losing any of the program or extended memory contents. Use it for a quick clear ALL action (Registers, Stack, Alpha, Flags), which requires no input values.

REPLX	**[Stack Replicate X]**	Author: J. D. Dodin	Source: Au Fond de la HP 41

Replicates the value in X to all stack registers (Y, Z, and T). Like using ENTER^ three times.

SKIPN	**[Skip N program lines]**	Author: Erik Blake	Source: PPCJ V11 N6 p32

In a running program, unconditionally skips as many program lines as the value in X. Use in combination with a conditional test to skip as many lines as desired, instead of the standard single line.

GTEND	**[Go to .END.]**	Author: Ken Emery	Source: MCODE for beginners

Sends the program counter to the permanent .END. in program memory (the position of the Curtain).

ST>Sigma	**[Stack to Stat Regs]**	Author: Zengrange Ltd.	Source: ZENROM manual

Stores the values in the stack registers into the statistical registers.

3.1.7. ST<>ST Multifunction.

Another multifunction, this time grouping together diverse stack and alpha register exchange sub-functions, as follows;-

Index	Function	Author	Description
000	**SUBCAT**	Ángel Martin	Lists the sub-functions names with indexes.
001	**Y<>Z**	Ken Emery	Exchanges X and Y registers
002	**Y<>T**	Ángel Martin	Exchanges Y and T stack registers
003	**Y<>L**	Ángel Martin	Exchanges Y and L stack registers
004	**Z<>T**	Ángel Martin	Exchanges Z and T stack registers
005	**Z<>L**	Ángel Martin	Exchanges Z and L stack registers
006	**T<>L**	Ángel Martin	Exchanges T and L stack registers
007	**M<>N**	Ángel Martin	Exchanges M and N alpha registers
008	**M<>O**	Ángel Martin	Exchanges M and O alpha registers
009	**N<>O**	Ángel Martin	Exchanges N and O alpha registers
010	**N<>P**	Ángel Martin	Exchanges N and P alpha registers

No input parameters are needed, and pretty much self-explanatory. Although this function is programmable, when in a program there's no choice for index, and the function **Y<>Z** will always be executed by default. (one at least, better than none).

3.1.8. XTRABOX Multifunction.

Multifunction grouping miscellaneous housekeeping routines. As with the previous case, the default function when inserted as a program line will be POPADR. The contents is as follows:-

Index	Function	Author	Description
000	**SUBCAT**	Ángel Martin	Lists the sub-functions names with indexes.
001	**POPADR**	Hakan Thorngren	Pops last return address from return stack
002	**X>ROM**	VM Electronics	Fetches ROM word at given address (in NNN)
003	**ROM>X**	VM Electronics	Writes ROM word at given address (in NNN)
004	**PGCOPY**	Ángel Martin	Copies ROM pages between addresses (in NNN)
005	**BUZZON**	Andres Meyer	Sets buzzer on CPU activity
006	**NOBUZZ**	Ángel Martin	Clears buzzer on CPU activity
007	**LASTRG**	Eramco System	Returns last available register
008	**X>$**	VM Electronics	Numeric value to Alphabetic NNN
009	**AVOGADR**	Ángel Martin	Returns Avogadro's number in X
010	**eCHARGE**	Simon Bradshow	Returns electron charge in X
011	**ABS**	W&W GmbH	Alpha Back Space

From these, **PGCOPY** requires a little further explanation. It takes their input parameters from the Y register (origin page) and the X register (destination page), as binary NNN's that contain the page number as absolute addresses in their address fields. See the description of **HEX>NNN** further down to find out how to produce such NNN's from their equivalent HEX codes in Alpha.

AVOGADR and **eCHARGE** are two "constant" functions, similar to the mainframe's **PI**. They return the Avogadro's number (6,02214199 E23 mol^{-1}) and the electron's Charge (1,6021892 E-19), useful in chemistry and physics problems.

We'll defer any discussion on BUZZON and NOBUZZ until the last section of the manual, where they'll be seen also as independent functions.

All the multifunction catalogs will return to the command prompt after the sub-function listing is completed, which should have jogged the user's memory for the index required.

Appendix 1.- Function overlap tables.

Function	HP-41	CX	SandBox
SF	*0-29*	Ö	*0-55*
CF	*0-29*	Ö	*0-55*
TOGF			*0-55*
FS?	Ö	Ö	Ö
FC?	Ö	Ö	Ö
FC?C	Ö	Ö	Ö
FS?C	Ö	Ö	Ö
FC?S			Ö
FC?S			Ö
STOF		Ö	Ö
RCLF		Ö	Ö

Function	CX	CCD	SandBox
CLEM			Ö
EMROOM	Ö		Ö
ARCLCHR			Ö
RSTCHK			Ö
CALLXM			Ö
sREG?	Ö		Ö
B?		Ö	
BLNG?			Ö
GETB		Ö	
SAVEB		Ö	
KALNG?			Ö
KAPCK			Ö
KACLR	Ö		Ö
CLBUFS		Ö	Ö
GETK		Ö	
SAVEK		Ö	
MRGK		Ö	

Flag control and X-Memory Functions.-

Function	HP-41	CX	SandBox
X=Y?	Ö	Ö	Ö
X#Y?	Ö	Ö	Ö
X>Y?	Ö	Ö	Ö
X>=Y?			Ö
X<Y?	Ö	Ö	Ö
X<=Y?	Ö	Ö	Ö
X=0?	Ö	Ö	Ö
X#0?	Ö	Ö	Ö
X>0?	Ö	Ö	Ö
X>=0?			Ö
X<0?	Ö	Ö	Ö
X<=0?	Ö	Ö	Ö
X=1?			Ö
X=Y?Z?			Ö
X=NN?		Ö	Ö
X#NN?		Ö	Ö
X<NN?		Ö	Ö
X<=NN?		Ö	Ö
X>NN?		Ö	Ö
X>=NN?		Ö	Ö

Value Comparison functions.-

## 3.2. Alpha and Display Functions.	

3.2.1. Alpha Utilities.

aVIEW	**[Lower Case AVIEW]**	Author: Ángel Martin	Source: Sandbox Project

Displays the contents of the alpha registers M and N using lower case characters available in the Halfnut models. No changes to the actual contents in alpha are made, only to the displayed string. It leaves unaltered the non-alphabet letters (i.e. numbers and other special chars). Supports a maximum of 12 chars in the displayed string. No other input parameters are required.

Note that this will obviously not work on the Fullnut models, where the Sandbox header functions will not be shown as complete, lacking the lower case characters after "e".

A>RG	**[Alpha to Registers]**	Author: Ken Emery	Source: MCODE for beginners
RG>A	**[Registers to Alpha]**	Author: Ken Emery	Source: MCODE for beginners

A>RG Stores the contents of the four alpha registers into a block of 4 consecutive data registers, beginning at the value in X.

Inverse to the previous function, RG>A restores the values from the data register block into the Alpha registers. Use these two functions together to temporarily store the alpha registers while being used in intermediate steps.

A>ST	**[Alpha to Stack]**	Author: Ángel Martin	Source: Sandbox Project
ST>A	**[Stack to Alpha]**	Author: Ángel Martin	Source: Sandbox Project

A>ST Stores the content of the Alpha registers into the four stack registers.
ST>A Restores the Alpha registers for the values in the Stack, inverse of the previous one.

No normalization is made. These functions should be used together to temporarily store Alpha while this is being used in intermediate steps/

AINT	**[Append Integer Part]**	Author: Frits Ferwerda	Source: ML ROM

Appends to Alpha the integer part of the value in X. Similar to "AIP" in the Advantage Module, or to "ARCLI" in the CCD Module.

ARCLCHR	**[ARCL Characters]**	Author: Hakan Thorngren;	Source: PPCJ V13 N7 p19

See description in 3.1.5. above.

AREV	**[Alpha Reverse]**	Author: Frans de Vries	Sourc: DF V10 N8 p8

Reverses the alpha string, building its mirror image. Two executions return the original string. No other input parameter is required.

ASUB	**[Alpha Substitute]**	Author: Zengrange Ltd.	Source: ZENROM manual

Substitutes the character which position is given in Y with the character which code is stored in. Equivalent to the function "YTOAX" from the Ext-IO module, only with opposite field descriptions (really "XTOAY").

CLAX	**(CLA from Comma)**	Author: Ángel Martin	Source: Sandbox Project

Clears the Alpha register from the position of a comma character and to the right. If more than one comma character is found, the rightmost is used to define the beginning of the clearing field.

LADEL	**[Left Alpha Delete]**	Author: Ross Cooling	Source: PPCJ V12 N2 p16
RADEL	**[Right Alpha Delete]**	Author: Ross Cooling	Source: PPCJ V12 N2 p16

LADEL deletes the leftmost character in alpha. No changes are made to the X register.
RADEL deletes the leftmost character in Alpha. No changes to X are made.

LEFT$	**[Left Substring]**	Author: Ross Cooling	Source: PPCJ N13 N2 p8
MID$	**[Mid Substring]**	Author: Ross Cooling	Source: PPCJ V12 N2 p29
RIGHT$	**[Right Substring]**	Author: Ross Cooling	Source: PPCJ N13 N2 p8

LEFT$ replaces Alpha with the substring defined by the leftmost "n" characters, where "n" is given in the X register.

MID$ replaces Alpha with a substring defined by the values in X and Y, as follows: X is the beginning of the string from the left, Y is the length of the substring.

RIGHT$ replaces Alpha with a substring defined by the rightmost "n" characters, where "n" is given in the X register.

LOW$	**[Alpha Lower Case]**	Author: Ángel Martin	Source: Sandbox project
UPR$	**[Alpha Upper Case]**	Author: Mark Power	Source: DF V8 N1 p10

LOW$ replaces all upper-case characters found in Alpha by their Lower-case equivalents. Does not alter non-alphabet letters like numbers or other special chars.

UPR$ replaces all lower-case characters found in Alpha by their upper-case equivalents. It also removes inverse video and underline bits from the byte values of the characters, if present.

RATOX	**[Right Alpha to X]**	Author: Ross Cooling	Source: PPCJ V12 N12 p10

Sends to X the code corresponding to the rightmost character in alpha, and deletes it from the alpha string**.**

It is the symmetrical function of ATOX from the X-Functions module - which could've also been called LATOX. In fact. similar functions exist in the Ext-IO and HP-IL Dev. Plug-in modules, named XTOAR and XTOAL respectively. The new name given here to this one is to avoid name duplication. (See Table-1 in the appendix section for a complete list of the Alpha Functions present in ROM's).

REMZER	**[Remove Leading Zeroes]**	Author: Ross Cooling	Source: PPCJ V12 N12 p6

Some functions (like some versions of CODE) return to Alpha a string right-padded with zero characters. REMZER removes the leading zeroes present in such Alpha strings, i.e. all the rightmost zero characters if present.

VIEWA	**[View Alpha non-stop]**	Author: Ken Emery	Source: MCODE for beginners

Analogous to the mainframe AVIEW function, but never stops the program execution (even if user flag 21 is enabled). Use it as alternative to AVIEW.

XTOAL	**[X to Alpha Left]**	Author: Hakan Throrngren	Source: PPCJ V13 N7 p9

Inverse of ATOX, appends to Alpha as leftmost (i.e. last character), the character which code is in the X register.

3.2.2. Display Utilities.

CHRSET	**[Character Set]**	Author: Chris L. Dennis	Source: PPCJ V18 N8 p14

Shows all the characters existing in the complete character set of the calculator. The displaying is sequential, scrolling as new chars are being added to the shown string.

Halfnut machines have substantially more characters that the Fullnut models, notably the lower-case letters of the alphabet –which are used by **aVIEW** as discussed previously.

CTRST	**[Set Display Contrast]**	Author: Michael Katz	Source: HPX V1 N6 p8
CTRST?	**[Find Display Contrast]**	Author: Michael Katz	Source: HPX V1 N6 p8

Use these functions to find out the current display contrast setting of your Halfnut machine, and to modify it accordingly. Possible values are between 0 and 15, and will be returned to X or taken from X depending on which function is being used.

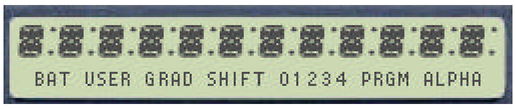
CNT+	**[Increase Contrast]**	Author: Michael Katz	Source: HPX V1 N6 p8
CNT-	**[Decrease Contrast]**	Author: Michael Katz	Source: HPX V1 N6 p8

Used to increment or decrement in one unit the current display contrast settings. No input parameters are required. For instance. CNT+ and CNT- would be equivalent to the following combinations:

```
01 CTRST?             01 CTRST?
02 INCX               02 DECX      (See the Math Functions section below)
03 CTRST              03 CTRST
```

DSTEST	**[Display Test]**	Author: Chris L. Dennis	Source: PPCJ V18 N8 p14

Simultaneously lights up all LCD segments and indicators of the calculator display, preceded by all the comma characters. Use it to check and diagnose whether your display is fully functional. No input parameters are required.



Appendix 2.- Alpha Functions Implementation Comparative Table.

Description	41-CX	SANDBOX	PANAME	CCD	EXT I/O	DEVIL
Alpha ON	AOFF	AOFF	AOFF	AOFF	AOFF	AOFF
Alpha OFF	AON	AON	AON	AON	AON	AON
Append Reg. X	ARCL	ARCL	ARCL	ARCL	ARCL	ARCL
Store Alpha to Register	ASTO	ASTO	ASTO	ASTO	ASTO	ASTO
View Alpha	AVIEW	VIEWA	AVIEW	AVIEW	AVIEW	AVIEW
Clear Alpha	CLA	CLA	CLA	CLA	CLA	CLA
Clear Display	CLD	CLD	CLD	CLD	CLD	CLD
Prompt for Numbers	PROMPT	PROMPT	PROMPT	PROMPT	PROMPT	PROMPT
View Register	VIEW	VIEW	VIEW	VIEW	VIEW	VIEW
Alpha Shift	ASHF	ASHF	ASHF	ASHF	ASHF	ASHF
Alpha Length	ALENG	ALENG	ALENG	-	ALENGIO	ASIZE?
Alpha Rotate	AROT	AROT	AROT	-	-	-
Position within Alpha	POSA	POSA	POSA	-	-	-
Alpha Num	ANUM	ANUM	ANUM	-	-	-
Alpha Num & Delete	-	-	ANUMDEL	-	ANUMDEL	-
X to Alpha Left	-	XTOAL	XTOAL	-	XTOAL	X-AL
X to Alpha Right	XTOA	XTOA	XTOAR	-	XTOAR	X-AX
Right Alpha to X	-	RATOX	ATOXR	-	ATOXR	A-XR
Left Alpha to X	ATOX	ATOX	ATOXL	-	ATOXL	A-XL
Extract Character	-	-	ATOXX	-	ATOXX	A-XX
Substitute Character	-	ASUB	YTOAX	-	YTOAX	Y-AX
Substring	-	MID$	SUB$	-	-	-
Delete Right Character	-	RADEL	-	ABSP	-	-
Delete Left Character	-	LADEL	-	-	-	-
Left Substring	-	LEFT$	-	-	-	-
Right Substring	-	RIGHT$	-	-	-	-
Reverse String	-	AREV	-	-	-	-
Alpha Upper Case	-	UPR$	-	-	-	-
Alpha Lower Case	-	LOW$	-	-	-	-
Lower Case AVIEW	-	aVIEW	-	-	-	-
Remove Leading Zeros	-	REMZER	-	-	-	-
Alpha to Registers	-	A>RG	-	-	-	-
Registers to Alpha	-	RG>A	-	-	-	-
Alpha to Stack	-	A>ST	-	-	-	-
Stack to Alpha	-	ST>A	-	-	-	-
Append Integer	-	AINT	APPX	ARCLI	-	AIPT
Delete from Comma	-	CLAX	-	-	-	-
Delete from Blank	-	-	-	CLA-	-	-
Alpha Prompt	-	-	-	PMTA	-	-
XTOA in Hex	-	-	-	XTOAH	-	-
Append Entry to Alpha	-	-	-	ARCLE	-	-
Input	-	-	-	INPT	-	-

3.3. Mathematical Functions.

-Math Fncts

3.3.1. Hyperbolic Functions.

© RedShift Software and Wilson Holes.

COSH	**[Hyperbolic Sine]**	Author: Nelson F. Crowle	Source: AECROM ROM
SINH	**[Hyperbolic Cosine]**	Author: Nelson F. Crowle	Source: AECROM ROM
TANH	**[Hyperbolic Tangent]**	Author: Nelson F. Crowle	Source: AECROM ROM

Direct hyperbolic functions. Input value placed in X, return value also left in X. The only difference with the original set from the AECROM is that these here will store the original value (the function's argument) into the LASTX register, while those in the AECROM will not.

The formulas used are the well known defining expressions:

sinh x = [exp(x) – exp(-x)]/2
cosh x = [exp(x) + exp(-x)]/2
tanh x = sinh x / cosh x

ACOSH	**[Inverse Hyp. Sine]**	Author: Nelson F. Crowle	Source: AECROM ROM
ASINH	**[Inverse Hyp. Cosine]**	Author: Nelson F. Crowle	Source: AECROM ROM
ATANH	**[Inverse Hyp. Tangent]**	Author: Nelson F. Crowle	Source: AECROM ROM

Inverse hyperbolic functions. Input value placed in X, return value also left in X. The only difference with the original set from the AECROM is that these here will store the original value (the function's argument) into the LASTX register, while those in the AECROM will not.

The formulas used are the well known defining expressions:

asinh x = Ln[x + sqrt(x^2 + 1)]
acosh x = Ln [x + sqrt(x^2 – 1)]
atanh x = Ln [(1+x)/(1-x)] / 2

3.3.2. Base Conversion and special viewer functions.

D>H	**[Decimal to HEX]**	Author: William Graham	Source: PPCJ V12 N6 p19
H>D	**[HEX to Decimal]**	Author: William Graham	Source: PPCJ V12 N6 p19

D>H will convert the decimal number in X into a hexadecimal number placed in Alpha, and it displays its content when done (so there's no need to switch the alpha mode on).
H>D will convert the hexadecimal number in Alpha into its decimal equivalent placed in X.

T>BS	**[Decimal to Base]**	Author: Ken Emery	Source: MCODE for beginners

T>BS will convert decimal numbers (i.e. base TEN) into any other numeric base, lower than 37. If bases greater than 36 are attempted, an error message will be shown.
To use it, place the base number in Y, and the decimal number to convert to it into the X register. The result will be shown in the display, but not stored into Alpha.

DFRAC	**[Decimal to Fraction]**	Author: Frans de Vries	Source: DF V9 N7 p8

Shows in the display the smallest possible fraction that results in the decimal number in X, for the current display precision set. Change the display precision as appropriate to adjust the accuracy of the results. It uses the same algorithm as the PPC ROM "DF".

VMANT	**[View Mantissa]**	Author: Ken Emery	Source: MCODE for beginners

Shows in the display the full mantissa of the number in the X register, 10 digits without exponent.

3.3.3. Sorting and performing Auxiliary Calculations.

REGSORT	**[Registers Sort]**	Author: HaJo David	Source: PPCJ V12 N5 p44

Sorts the contents of the registers specified in the control number in X, defined as: ***bbb,eee***, where "**bbb"** is the begin register number and "**eee"** is the end register number. If the control number is positive the sorting is done in ascending order, if negative it is done in descending order.

STSORT	**[Stack Sort]**	Author: David Phillips	Source: PPCJ V12 N2 p13

Sorts in descending order the contents of the four stack registers, X, Y, Z and T. No input parameters are required.

DECX	**[Decrement X]**	Author: Ross Cooling	Source: PPCJ V12 N12 p21
INCX	**[Increment X]**	Author: Ross Cooling	Source: PPCJ V12 N12 p21

Convenient substitutes for 1+ and 1-, these functions will decrement or increment by one the current content of the X register. Also used instead of ISG X and DSE X, when there's no desire to branch the program execution even if the boundary condition is reached: this saves a NOP line placed right after the conditional instruction.

E3/E+	**[Decimal to Fraction]**	Author: Frans de Vries	Source: DF V9 N7 p8

Divides the value in X by 1,000 and adds one to the result. Very useful to build matrix indices, and to speed up repetitive calculations that appear very frequently.

GEULER	**[Gamma constant]**	Author: Ángel Martin	Source: Sandbox project

Places in X the value of the Euler's gamma constant with 10-digit precision: 5,772156649 E-01
The stack lift is enabled, allowing for normal RPN-style calculations.

3.3.4. Alea jacta est… or the rest of the best.

RAND	**[Random Number]**	Author: Ken Emery	Source: MCODE for beginners

Uses the fractional part of the number in the X register as a seed to generate a random number from it. It uses the same algorithm as the PPC ROM routine RN, thus should yield the same results as that one when using the same seeds.

PRIME?	**[Prime Number Finder]**	Author: Jason DeLooze	Source: PPCJ V11 N7 p30

Determines whether the number in the X register is Prime (i.e. only divisible by itself and one). If not, it returns the smallest divisor found and stores the original number into the LASTX register. YES or NO are shown depending of the result.

When in a program, the execution will skip one step if the result is false (i.e. not a prime number), enabling so the conditional branching options.

Example program:- The following routine shows the prime numbers starting with 3, and using diverse Sandbox Math functions.

```
01  LBL "PRIMES"        05  PRIME?              09  INCX
02  3                   06  VIEW X  <yes>       10  GTO 00
03  LBL 00              07  X#Y?    <no>        11  END
04  RPLX                08  LASTX
```

QREM	**[Quotient Remainder]**	Author: Ken Emery	Source: MCODE for beginners

Calculates the Remainder "R" and the Quotient "Q" of the Euclidean division between the numbers in the Y (dividend) and X (divisor) registers. Q is returned to the Y registers and R is placed in the X register. The general equation is:

Y = Q X + R,

where both Q and R are integers.

QROOT	**[Quadratic Eq. Roots]**	Author: Ángel Martin	Source: Sandbox project

Given the quadratic equation: aX^2 + bx + c = 0, this function calculates its two solutions (or roots). Input the coefficients into the stack registers Z, Y, and X using: a ENTER^ b ENTER^ c

The roots are obtained using the well-known formula: X1,2 = -b/2a +- sqrt[(-b/2a)^2 – c/a]
Upon execution, X1 will be left in Y and X2 will be left in X.

If the argument of the square root is negative, then the roots Z1 and Z2 are complex and conjugated (symmetrical over the X axis), with Real and Imaginary parts defined by:

Re(Z) = -b/2a	Z1 = Re(Z) + i Im(Z)
Im(Z) = sqrt[abs((-b/2a)^2 –c/a)]	Z2 = Re(Z) – i Im(Z)

Upon execution, Im(Z) will be left in Y and Re(Z) will be left in X.

If the roots are real, the function sounds a high-pitch short tone. If the roots are complex, the function sounds a low-pitch short tone and places a zero in the Z register. In a program, the execution will skip one step if the roots are complex, enabling so as well conditional branching.

Example program:- The following routine presents the equation roots in Alpha, according to their Real or Complex nature. It assumes that the coefficients are stored on the stack as specified before.

```
01  LBL "QUAD"           06  ARCL X          11  LBL 00  <Real>    16  ARCL X
02  QROOT                07  "@#"            12  "X1="             17  LBL 01
03  GTO 00 <Real>        08  ARCL Y          13  ARCL Y            18  PROMPT
04  FIX 2    <Complex>   09  "@)"            14  PROMPT            19  END
05  "Z=("                10  GTO 01          15  "X2="
```

Solved for x^2 – 3x + 2 = 0 it returns: X1=2 and X2=1
Solved for x^2 + x + 1 = 0 it returns: Z=(-0,5 # 0,87)

SIGNUM	**[Numeric SIGN]**	Author: Ross Cooling	Source: PPCJ V12 N12 p31

Like the mainframe SIGN function, but returning zero for null arguments. Input value is the argument in the X register, which is also stored in LASTX.

3.3.5. Recall Math and Additional Comparison functions.

RCL+	**[RCL Plus]**	Author: Ross Cooling	Source: PPCJ V14 N4 p16
RCL-	**[RCL Minus]**	Author: Ross Cooling	Source: PPCJ V14 N4 p16
RCL*	**[RCL Times]**	Author: Ross Cooling	Source: PPCJ V14 N4 p16
RCL/	**[RCl Divide]**	Author: Ross Cooling	Source: PPCJ V14 N4 p16

RCL math between the Y register and the register which number is specified in X. It performs the corresponding operation, leaving the result in X and storing the original value in Y into the LASTX register. The value in the register specified by X is not changed.

X>=Y?	**[Is X>=Y?]**	Author: Ken Emery	Source: MCODE for beginners
X>=0?	**[Is X>=0?]**	Author: Ángel Martin	Source: Sandbox project

X>=Y? compares the values of the X and Y registers, skipping one line if false.
X>=0? compares with zero the value in the X register, skipping one line if false.

These functions are arguably "missing" on the mainframe set; a fact partially corrected with the indirect comparison functions of the CX model (X>=NN?), but unfortunately not quite the same.

X=1?	**[Is X=1?]**	Author: Nelson F. Crowle	Source: NFC ROM

A quick and simple way to check whether the value in X equals one. As usual, program execution skips one step if the answer is false.

X=Y?Z?	**[Is X=Y? or Z?]**	Author: Ken Emery	Source: MCODE for beginners

Double comparison, first between the values in the X and Y registers, and if false between the values contained in the X and Z registers. Execution will skip one step if the first condition only isn't met, and two steps if both conditions aren't met.

3.3.6. Complex Arithmetic.

Z+	**[Complex Addition]**	Author: Ángel Martin	Source: Sandbox project
Z-	**[Complex subtraction]**	Author: Ángel Martin	Source: Sandbox project
Z*	**[Complex Multiplication]**	Author: Ángel Martin	Source: Sandbox project
Z/	**[Complex Division]**	Author: Ángel Martin	Source: Sandbox project

These functions will calculate the resulting complex number **Z** of the corresponding operation (+, -, *, /) between two arguments. The input values are the Real and Imaginary parts of the operands Z1 and Z2, stored in the stack as follows:

T: Im(Z1)		T: Im(Z1)
Z: Re(Z1)	*Operation*	Z: Re(Z1)
Y: Im(Z2)	-------------→	Y: Im(**Z**)
X: Re(Z2)		X: Re(**Z**)

Note that one of the arguments (Z1) is kept in its original location of the stack, while the other (Z2) is replaced (and consequently lost) by the result of the operation (**Z**). Note also that the Alpha register is used as scratch registers, and thus its previous contents will be lost.

The formulas used are:

(a + bi) + (c + di) = (a + c) + (b + d) i
(a + bi) - (c + di) = (a - c) + (b - d) i
(a + bi) (c + di) = (ac – bd) + (ad + bc) i
(a + bi)/(c + di) = [(ac + bd)/(c^2 + d^2)] + [(bc – ad)/(c^2 + d^2)] i

1/Z	**[Complex Inversion]**	Author: Ángel Martin	Source: Sandbox project

Calculates the inverse complex number of a given one (Z0), which imaginary and real parts are stored in Y and X respectively. The resulting imaginary and real parts will replace the original argument's as per the scheme below:

$$\begin{array}{lcl} \text{Y: Im(Z0)} & \textit{1/Z} & \text{Y: Im}(\mathbf{Z}) \\ \text{X: Re(Z0)} & \xrightarrow{\qquad} & \text{X: Re}(\mathbf{Z}) \end{array}$$

The formula used is:

1 / (x + yi) = [x / (x^2 + z^2)] – [y / (x^2 + z^2)] i

Example program:- Write a user code routine to calculate the Sine, Cosine and Tangent of a complex number Z=(x + yi), making use of the expressions:

sin z = sin x cosh y + i cos x sinh y
cos z = cos x cosh y - i sin x sinh y

tan z = sin z / cos z

```
01  LBL "ZSIN"
02  STO M
03  COS
04  X<>Y
05  STO N
06  SINH
07  *
08  RCL M
09  SIN
10  RCL N
11  COSH
12  *
13  FS? 02
14  RTN
15  GTO "ZOUT"

16  LBL "ZCOS"
17  STO M
18  SIN
19  X<>Y
20  STO N
21  SINH
22  *
23  CHS
24  RCL M
25  COS
26  RCL N
27 COSH
28  *
29  FS? 02
30  RTN
31  GTO "ZOUT"

32  LBL "ZTAN"
33  SF 02
34  RAD
35  XEQ "ZCOS"
36  X<>Y
37  STO O
38  X<>Y
39  RCL N
40  RCL M
41  XEQ "ZSIN"
42  RCL O
43  R^
44  Z/

45  LBL "ZOUT"
46  FIX 2
47  "Z=("
58  ARCL X
49  "@#"
50 ARCL Y
51  "@)"
52  PROMPT
53  END
```

Where the program assumes RAD mode is enabled, and that the input is done according to the defined process above: Im(Z) in Y and Re(Z) in X. Note that no Data Registers are used.

Solved for Z = (2 + 3i) it returns: SIN Z=(9,15 # -4,17); COS Z = (-4,19 # -9,11)
Solved for Z = (0.25 + 0.25i) it returns: TAN Z = (0,24 # 0,26)

Appendix 3.- Mathematical Functions Implementation Comparison Table.

Function	HP41	Math Pac	Advantage	AECROM	SANDBOX	PPC
SIN	MCODE	MCODE	MCODE	MCODE	MCODE	MCODE
COS	MCODE	MCODE	MCODE	MCODE	MCODE	MCODE
TAN	MCODE	MCODE	MCODE	MCODE	MCODE	MCODE
ASIN	MCODE	MCODE	MCODE	MCODE	MCODE	MCODE
ACOS	MCODE	MCODE	MCODE	MCODE	MCODE	MCODE
ATAN	MCODE	MCODE	MCODE	MCODE	MCODE	MCODE
SINH	-	FOCAL	(FOCAL)	MCODE	MCODE	-
COSH	-	FOCAL	(FOCAL)	MCODE	MCODE	-
TANH	-	FOCAL	-	MCODE	MCODE	-
ASINH	-	FOCAL	-	MCODE	MCODE	-
ACOSH	-	FOCAL	-	MCODE	MCODE	-
ATANH	-	FOCAL	-	MCODE	MCODE	-
CURVFIT	-	-	FOCAL	MCODE	-	FOCAL
DIFEQ	-	FOCAL	FOCAL	-	-	-
FOURIER	-	FOCAL	-	-	-	-
INTEG	-	FOCAL	MCODE	-	-	FOCAL
QROOT	-	(FOCAL)	(FOCAL)	-	MCODE	-
PROOT	-	FOCAL	FOCAL	-	-	-
SOLVE	-	FOCAL	MCODE	-	-	FOCAL
MDET	-	FOCAL	MCODE	-	-	-
MINV	-	FOCAL	MCODE	-	-	-
MTRNP	-	FOCAL	MCODE	-	-	-
MSYS	-	FOCAL	MCODE	-	-	-
ZMOD	MCODE	FOCAL	FOCAL	MCODE	MCODE	FOCAL
Z+	-	FOCAL	FOCAL	-	MCODE	FOCAL
Z-	-	FOCAL	FOCAL	-	MCODE	FOCAL
Z*	-	FOCAL	FOCAL	-	MCODE	FOCAL
Z/	-	FOCAL	FOCAL	-	MCODE	FOCAL
1/Z	-	FOCAL	FOCAL	-	MCODE	-
LNZ	-	FOCAL	FOCAL	-	-	FOCAL
LOGZ	-	FOCAL	FOCAL	-	-	-
e^Z	-	FOCAL	FOCAL	-	-	FOCAL
Z^W	-	FOCAL	FOCAL	-	-	FOCAL
Z^1/W	-	FOCAL	FOCAL	-	-	-
SINZ	-	FOCAL	FOCAL	-	(FOCAL)	FOCAL
COSZ	-	FOCAL	FOCAL	-	(FOCAL)	FOCAL
TANZ	-	FOCAL	FOCAL	-	(FOCAL)	(FOCAL)
ASINZ	-	-	-	-	-	-
ACOSZ	-	-	-	-	-	-
ATANZ	-	-	-	-	-	-
SINHZ	-	-	-	-	-	-
COSHZ	-	-	-	-	-	-
TANHZ	-	-	-	-	-	-
ASINHZ	-	-	-	-	-	-
ACOSHZ	-	-	-	-	-	-
ATANHZ	-	-	-	-	-	-

3.4. MLDL and Special Functions.	

3.4.1. Peeking and Poking.

aNRCL	**[absolute NRCL]**	Author: Ken Emery	Source: MCODe for beginners

Complementary to NNRCL in that it also recalls the contents of the registers without normalization, but more powerful because it uses the absolute address instead of the register number as input in X. Thus it is possible to recall anything from Mail memory, including status registers (from 0 to 17), buffers and Key Assignment areas, and even Extended-Memory registers as well.

RCLB	**[Recall Byte]**	Author: Mark Power	Source: DF V7 N8 p24
STOB	**[Store Byte]**	Author: Mark Power	Source: DF V7 N8 p24
X<>B	**[Exchange Byte]**	Author: Mark Power	Source: DF V7 N8 p24

Byte-level functions to read, write or exchange values into/from Main and Extended Memory. Input parameters are: the byte value in X and the address in M, in ZENROM format. This consists of two alpha characters which corresponding hex codes represent the memory address.

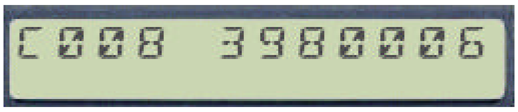
RAMEDIT	**[Ram Editor]**	Author: Hakan Thorgren	Source: PPCJ V13 N4 p26

This function sets the calculator in RAM Editor mode. When invoked from the keyboard, it can take the start absolute address either from the decimal value stored in X, or from a right-justified NNN with the binary address in it. When invoked from a program it takes it from the current position of the program counter.

In either case, the display shows the register and nybble being edited, as well as the contents of the complete register. The cursor can be moved to the left and right with the USER and PRGM keys respectively, and the current digit where it's positioned on will blink on the display.

Direct editing is possible using the redefined hex keyboard. Continuing to scroll in either direction shifts the cursor to the beginning or end of the register (indicated with a short warning tone), but doesn't move up or down to the adjacent registers. Use the "**+**" and "**–**" keys to actually move to the following or previous registers.

The input sequence terminates by pressing R/S or the back arrow key, which exits the RAM editing mode.



3.4.2. Conversions with a twist.

BCD>BIN	**[Binary to BCD]**	Author: Ken Emery	Source: MCODE for beginners
BIN>BCD	**[BCD to Binary]**	Author: Ken Emery	Source: MCODE for beginners

Converts between binary and BCD. Used internally also as subroutines for other functions.

HEX>NNN	**[Decode]**	Author: Ken Emery	Source: MCODE for beginners
NNN>HEX	**[Code]**	Author: Clifford Stern	Source: MCODE for beginners

The Sandbox version of the well-known CODE and DECODE functions. Their usage is probably known to every 41 user, as they've been around for a long time, not the least important included already in the PPC ROM (routines "NH" and "HN").

Simply enter the HEX code in Alpha and execute HEX>NNN to obtain (code) the equivalent binary NNN in X. NNN>HEX will decode the NNN in X into the HEX code in Alpha, and (contrary to other implementations of this function) without leading zeroes (i.e. no left-padding).

HXENTRY	**[Hex Entry]**	Author: Ken Emery	Source: MCODE for beginners
HEXIN	**[Hex Input]**	Author: Hakan Thorgren	Source: PPCJ V13 N4 p13

Direct entry of Non-normalized numbers using its byte's HEX codes. Similar to CODE but interactive. HEXIN allows for a prompt message, if the alpha register contains any string before the function is executed. HXENTRY stores the input code into Alpha as well as returning the NNN into X. Both enable only the keys of the HEX keyboard (0-9 and A-F).

HEX>VSM	**[Hex to VASM Oct]**	Author: Ken Emery	Source: MCODE for beginners
VSM>HEX	**[VASM Oct to HEX]**	Author: Ken Emery	Source: MCODE for beginners

Routines to convert ROM address between HEX and the VASM Octal format used by HP. Input fields are automatically separated by the function, and the keyboard only admits numbers appropriate of the origin base (Hex or Octal).

3.4.3. MLDL Utilities.

GETW	**[Get Word]**	Author: Ángel Martin	Source: Sandbox project

Reads the ROM word which address is in the address field of the binary NNN stored in X, and returns its value into the word field of the same binary NNN in X.

Example program:- This short routine prompts for the absolute address in HEX and returns the HEX value of the word read at such a given address.

```
01  LBL "READW"        07  NNN>HEX
02  16                 08  3
03  WSIZE              09  RIGHT$
04  "aADR=?"           10  VIEWA
05  PMTH               11  END
06  GETW
```

Assuming the SandBox Module is plugged in port 1, execute READW with aADR=8000 to obtain its XROM number value: 008 *(Note: PMTH and WSIZE are from the CCD Module. You can replace lines 02, 03 and 05 with: 4 HPROMPT if you use the HEPAX module, keeping line 04 as is).*

CHKROM	**[Check ROM]**	Author: HP Co.	Source: HP-IL Devel ROM

This function tests the ROM with XROM number in X, to verify whether the value of its checksum word is correct. Input value is the XROM number, and the result is a message with the words "OK" or "BAD". The ROM id# is also shown while performing the calculation. (Sum of all word values, MOD 256).

SUMROM	**[Sum ROM]**	Author: George Ioannou	Source: DF V3 N1 p10

Calculates the ROM Checksum and writes its value into the last word of the page being summed. Prompting function requests the page address (8 to F), to be input on the blinking field.

XQ>XR	**[XEQ to XROM]**	Author: W&W GmbH	Source: RAMBOX ROM

For a user code program which name is in Alpha, this function changes all the global XEQ lines calling other programs in the q-RAM space, converting them into their XROM equivalent.

Use it once the function allocation and FAT is completed, as it will refer to the XROM and function numbers, instead to performing a label search based on the actual name. Execution of the program will be much faster, as the mentioned search will be avoided.

3.4.4. Miscellaneous Hacker Tools.

FDATA	**[Function Data]**	Author: Klaus Huppertz	Source: Prisma, Jan-90

Shows the FAT address and XROM value (the one used for key assignments) of the function input into the function's Alpha prompt. It works equally for mainframe functions, User Code programs in RAM, and MCODE functions in ROM.

Despite being an Alpha prompt function when invoked from the keyboard, FDATA is also programmable: when in a program, the function name will be taken from the Alpha register!

ROMIN	**[ROM In]**	Author: Warren Furlow	Source: www.hp41.com
ROMOUT	**[ROM Out]**	Author: Warren Furlow	Source: www.hp41.com

These two functions will write a ROM file to the HP-IL Mass Storage unit (ROMOUT), and will read it back from it into the page number specified at the execution. If no HP-IL is present an error message will be shown.

MNF	**[Mainframe Function]**	Author: Clifford Stern	Source: PPCJ V12 N3 p37

This function prompts for a three-digit input representative of any mainframe function, as per the codes contained in the HEX Byte tables. Note that some values will invoke strange synthetic routines.

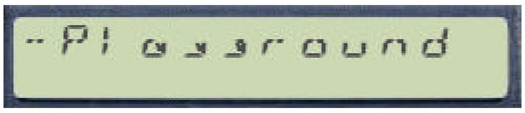
The following table shows some of the functions and their corresponding suffixes. Note how MFN conveniently accesses many of the non-programmable mainframe functions.

MFN Suffix	Mainframe Function
000	CAT_
006	SIZE _ _ _
002	DEL _ _ _
003	CLP _
010	PACK
015	ASN _

MemLost	**[Polling Point]**	Author: Ken Emery	Source: MCODE for beginners

Finally, register allocation will be of 26 data registers after Memory Lost.

3.5. Entertainment & Fun stuff.


-Playground

The few remaining functions deal with alternative sounds and even include one game programmed in MCODE (by far the longest piece of code in the SandBox!). Not to be taken too seriously, it stills provides a playground for those of us who'll never completely grow into adulthood…

BUZZON	**[Buzz On]**	Author: Andreas Meyer	Source: Cursor Magazine
NOBUZZ	**[Buzz Off]**	Author: Ángel Martin	Source: Sandbox project

These functions activate and deactivate the buzzing mode upon CPU activity. Keep it mind that this isn't supposed to be active for long times, or otherwise damage to the calculator beeper could be made.

CLAXON	**[Alternative BEEP]**	Author: Mark Power	Source: DF V7 N7 p12
RASP	**[Alternative BEEP]**	Author: Mark Power	Source: DF V7 N7 p12

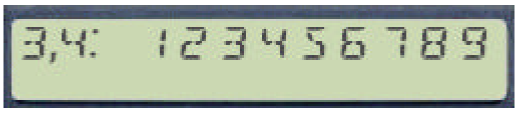
Two other acoustic signals to use instead of BEEP (which admittedly is getting boring after all these years…). Use discretionally as per your programming needs.

ROLLERS	**[High Rollers]**	Author: Ross Cooling	Source: PPCJ V14 N4 p31

MCODE version of the popular High Rollers game.

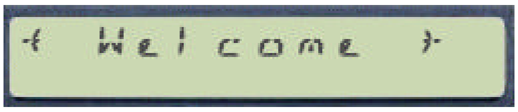
The player starts the game with a string of digits from 1 to 9. Each turn two digits are offered by the calculator, which has kindly rolled the dice for you. These two numbers should be used to remove some of the digits in the original string by using any combination of their sum (which must equal the removed ones).

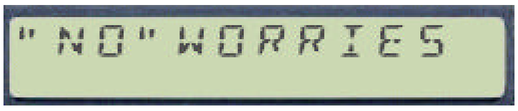
The game ends when all digits have been removed (you win) or when no digit can be removed (you lose). Doubles are saved as extra throws, shown after a colon in the left side of the display. Mind you, it's not that easy to win!


3,4: 123456789

Power ON	**[Polling Point]**	Author: Ángel Martin	Source: Sandbox project
Power OFF	**[Polling Point]**	Author: Ángel Martin	Source: Sandbox project

Finally, the SandBox displays two messages upon power on and power off. The first one is a greetings note, hopefully inviting the use of the system with a friendly touch. The second one is a "No Worries" Aussie-like farewell, inviting to come back at the user's convenience…


-{ Welcome }-


"NO" WORRIES

4. Quick Reference Guide.

Fnc#	Function Name	Description	Input	Output	Author	Source
1	-Sandbox 3d	Header	None	Shows "Welcome" in Alpha	Ángel Martin	SANDBOX Project
2	BIT56	Multi-Function	Sub-Function index	Performs Sub-function	Ángel Martin	SANDBOX Project
2,0	CATALOG	Sub-functions Catalogue	(i=00); none	Lists Function Names	Ángel Martin	SANDBOX Project
2,1	X+Y	Bitwise Addition	(i=01); X and Y	X+Y	Gordon Pegue	PPCJ V2 N5 p38
2,2	Y-X	Bitwise Subtraction	(i=02); X and Y	Y-X	Gordon Pegue	PPCJ V2 N5 p38
2,3	AND	Logical AND	(i=03); X and Y	X AND Y	Gordon Pegue	PPCJ V2 N5 p38
2,4	OR	Logical OR	(i=04); X and Y	X OR Y	Gordon Pegue	PPCJ V2 N5 p38
2,5	NOT	Logical NOT	(i=05); X	Not X	Gordon Pegue	PPCJ V2 N5 p38
2,6	RXR	Rotate x bits Right	(i=06); X: number of bits	Bit Rotation	Gordon Pegue	PPCJ V2 N5 p38
2,7	RXL	Rotate x bits Left	(i=07); X: number of bits	Bit Rotation	Gordon Pegue	PPCJ V2 N5 p38
2,8	BRXL	Rotate Block Right	(i=08); X: number of digits	Block Rotation	Gordon Pegue	PPCJ V2 N5 p38
2,9	RLN	Rotate Left N digits	(i=09);	Digit Rotation	Gordon Pegue	PPCJ V2 N5 p38
2,10	RRN	Rotate Right N digits	(i=10);	Digit Rotation	Gordon Pegue	PPCJ V2 N5 p38
3	BLCAT	Blocs Catalogue	None	Catalogues all Pages	VM Electronics	HEPAX ROM
4	BLNG?	Buffer Length Finder	None	Number of registers used	W&W GmbH	RamBox
5	CALLXM	Call XM Program	File Name in Alpha	Program will execute	Ross Wentworth	PPCJ V12 N3 p48
6	CFX	Clear Flag by X	Flag number in X	Clears Flag	Michael Katz	HPX V1 N6 p7
7	CLEM	Clear EM	None	EM deleted	Hakan Thorngren	PPCJ V13 N2 p14
8	CLRSAF	Clear ST, REG, Alpha, Flags	None	CLX + CLA + CLREG + 0 X<>F	Gordon Pegue	PPCJ V12 N3 p40
9	CRTN?	Curtain Finder	None	Curtain Address	N/A	MMEPROM
10	CSST	Continuous SST	Program Pointer positioned.	Program lines displayed	Phi Trinh	PPCJ V9 N7 p49
11	DREG?	Data Registers Finder	None	Current Size	Ken Emery	MCODE For beginners
12	FC?S	Is flag Clear and Set	Flag number in X	Like FC?C	Ken Emery	PPCJ V11 N6 p11
13	FLNG?	Disk File Length	File Name in Alpha	File Length in X	N/A	MMEPROM
14	FREG?	Free Registers Finder	None	Available Main Memory registers	Ken Emery	MCODE For beginners
15	FS?S	Is Flag Set and Set	Flag number in X	Like FS?C	Ken Emery	PPCJ V11 N6 p11
16	GTEND	Go to .END.	None	Positions pointer on .END.	Ken Emery	MCODE For beginners
17	KACLR	Clear Key Assignments	OK or OKALL in Alpha	Deleted buffer(s)	Hajo David	PPCJ V12 N4 p24
18	KALNG?	KA Length Finder	None	Number of registers used	W&W GmbH	RamBox
19	KAPCK	Pack Key Assignments	None	Packed KA buffer	Hajo David	PPCJ V12 N4 p24
20	LKAOFF	Suspend Local KA	None	Deactivates A-J assignments	Ross Cooling	PPCJ V13 N2 p37
21	LKAON	Activate local KA	None	Reactivates A-J assignments	Ross Cooling	PPCJ V13 N2 p37

Fnc#	Function Name	Description	Input	Output	Author	Source
22	NNRCL	NNN Recall	Register Number in X	Register contents in X	Sid Kelly	PPCJ V12 N10 p6
23	RCLF	Recall Flags Status	None	Flag Status string in X	Hajo David	PPCJ V12 N5 p44
24	REPLX	Stack Replicate	Value in X	Fills the stack with X	J.D. Dodin	Au fond de la HP-41
25	ROMCAT	ROM Catalogue	ROM ID# in X	Catalogues ROM functions	J.D. Dodin	Au fond de la HP-41
26	RSTCHK	Reset Checksum	File Name in Alpha	Resets the Checksum byte	Hakan Thorngren	PPCJ V13 N2 p14
27	SFX	Set Flag by X	Flag number in X	Sets Flag in X	Michael Katz	HPX V1 N6 p7
28	SKIPN	Skip N Steps	Number of lines in X	Skips N lines	Erik Blake	PPCJ V11 N6 p32
29	ST>Sigma	Stack to SREG	Stack full	Stores Stack into Sigma Regs	Zengrange	ZENROM Manual
30	**ST<>ST**	*Multi-Function*	*Sub-Function index*	*Performs Sub-function*	*Ángel Martin*	*SANDBOX Project*
30,0	CATALOG	Sub-functions Catalogue	(i=00); none	Lists Function Names	Ángel Martin	SANDBOX Project
30,1	Y<>Z	Exchange Y and Z	(i=01); Y=y; Z=z	Y=z; Z=y	Ken Emery	MCODE For beginners
30,2	Y<>T	Exchange Y and T	(i=02); Y=y; T=t	Y=t; T=y	Ángel Martin	SANDBOX Project
30,3	Y<>L	Exchange Y and L	(i=03); Y=y; L=l	Y=l; L=y	Ángel Martin	SANDBOX Project
30,4	Z<>T	Exchange Z and T	(i=04); Z=z; T=t	T=z; Z=t	Ángel Martin	SANDBOX Project
30,5	Z<>L	Exchange Z and L	(i=05); Z=z; L=l	Z=l; L=z	Ángel Martin	SANDBOX Project
30,6	T<>L	Exchange T and L	(i=06); T=t; L=l	T=l; L=t	Ángel Martin	SANDBOX Project
30,7	M<>N	Exchange M and N	(i=07); M=m; N=n	M=n; N=m	Ángel Martin	SANDBOX Project
30,8	M<>O	Exchange M and O	(i=08); M=m; O=o	M=o; O=m	Ángel Martin	SANDBOX Project
30,9	N<>O	Exchange N and O	(i=09); N=n, O=o	N=o, O=n	Ángel Martin	SANDBOX Project
30,10	N<>P	Exchange N and P	(i=10); N=n, P=p	N=p, P=n	Ángel Martin	SANDBOX Project
31	STOF	Store Flags Status	Flag Status string in X	Changes flags 0-43	Hajo David	PPCJ V12 N5 p44
32	TOGF	Toggle Flag	Flag number in X	Toggles the flag	Ken Emery	PPCJ V11 N6 p11
33	XMROOM	EM Room	None	Available EM registers	Clifford Stern	PPCJ V12 N3 p38
34	XROM	Input XROM function	Promps for RR:FF	Executes the function	Clifford Stern	PPCJ V12 N3 p37
35	SigmaRG?	Stat Regs Finder	None	Stat Regs Address	N/A	MMEPROM
36	**XTRABOX**	*Multi-Function*	*Sub-Function index*	*Performs Sub-function*	*Ángel Martin*	*SANDBOX Project*
36,0	CATALOG	Function Catalogue	(i=00); none	Lists Function Names	Ángel Martin	SANDBOX Project
36,1	POPADR	POP Return Address	(i=01); none	Destroys First Return Address	Hakan Thorngren	PPCJ V13 N2 p14
36,2	X>ROM	Write to ROM	(i=02); aaaawww as NNN	Writes word in ROM	VM Electronics	HEPAX Manual
36,3	ROM>X	Read from ROM	(i=03); aaaa000 as NNN	Places NNN ROM word in X	VM Electronics	HEPAX Manual
36,4	PGCPY	Page Copy	(i=04); Y:Source, X:Destination	Copies Source into Destination	Ángel Martin	SANDBOX Project
36,5	BUZZON	Buzz Mode On	(i=05); none	Sets buzzer on	Andreas Meyer	Cursor N.2/89 p14
36,6	NOBUZZ	Buzz Mode Off	(i=06); none	Sets buzzer off	Ángel Martin	SANDBOX Project
36,7	LASTRG	Last Register	(i=07); X: Register number	Register Value	Eramco System	MLOS-1A ROM

Fnc#	Function Name	Description	Input	Output	Author	Source
36,8	X>$	Numeric to Alpha Data	(i=08); X: Value	Value as Alpha Data	VM Electronics	HEPAX ROM
36,9	AVOGADR	Avogadro's Number	(i=09); none	6,02214199 E23	Ángel Martin	SANDBOX Project
36,10	eCHARGE	Electron's Charge	(i=10); none	1,6021892 E-19	Simon Bradshow	DF V6 N6 p12
36,11	ABS	Alpha Back Space	(i=11); none	Deletes Rightmost Character	W&W GmbH	CCD ROM
37	**-Window Nut**	**Header**	**None**	**Shows "No Worries" in Alpha**	**Ángel Martin**	**SANDBOX Project**
38	aVIEW	Lower Case AVIEW	Alpha String (<13 Chrs)	Shows ALPHA in Lower Case	Ángel Martin	SANDBOX Project
39	A>RG	Alpha to Memory	4-Register block start in X	Stores Alpha into 4 Registers	Ken Emery	PPCJ V11 N7 p26
40	A>ST	Alpha to Stack	Alpha String	NNN's in Stack	Ángel Martin	SANDBOX Project
41	AINT	Alpha Integer Part	Number in X	Appends Integer part	Frits Ferwerda	-ML ROM
42	ARCLCHR	ARCL Char	# Chars in X, FileName in Alpha	Chars appended to Alpha	Hakan Thorngren	PPCJ V13 N7 p19
43	AREV	Reverse Alpha	Alpha String	Reversed Alpha String	Frans de Vries	DF V10 N8 p8
44	ASUB	Alpha Substitute	Y: position; X:Char	Places char in position	Zengrange	ZENROM Manual
45	CHRSET	Character Set Demo	None	Shows all characters	Chris L. Dennis	PPCJ V18 N8 p14
46	CLAX	CLA from Comma	None	Alpha deleted from Comma	W&W GmbH	CCD ROM
47	CNT+	Increase Contrast	None	Contrast increased by One	Michael Katz	HPX V1 N6 p8
48	CNT-	Decrease Contrast	None	Contrast decreased by One	Michael Katz	HPX V1 N6 p8
49	CTRST	Set Contrast	Value (1 to 15) in X	none	Michael Katz	HPX V1 N6 p8
50	CTRST?	Current Contrast	None	Current contrast setting	Michael Katz	HPX V1 N6 p8
51	DSTEST	Display Test	None	all segments on	Chris L. Dennis	PPCJ V18 N8 p14
52	LADEL	Left Alpha Delete	None	Deletes left char in Alpha	Ross Cooling	PPCJ V12 N2 p16
53	LEFT$	Left String	String length in X	Leaves left chars in alpha	Ross Cooling	PPCJ N13 N2 p8
54	LOW$	Lower Case Alpha	Alpha String	Lower Case Alpha String	Ángel Martin	SANDBOX Project
55	MID$	Sub String	Beginning in Y, length in X	Leaves substring in Alpha	Ross Cooling	PPCJ V12 N2 p29
56	RADEL	Right Alpha Delete	None	Deletes Right char in Alpha	Ross Cooling	PPCJ V12 N12 p10
57	RATOX	Right Alpha to X	None	Right Alpha Char value in X	Ross Cooling	PPCJ V12 N12 p10
58	REMZER	Remove Leading Zeroes	None	Zeroes removed	Ross Cooling	PPCJ V12 N12 p6
59	RG>A	Memory to Alpha	4-Register block start in X	Recalls 4 registers to Alpha	Ken Emery	PPCJ V11 N7 p26
60	RIGHT$	Right String	String length in X	Leaves right chars in alpha	Ross Cooling	PPCJ N13 N2 p8
61	ST>A	Stack to Alpha	NNN's in Stack	Alpha String	Ángel Martin	SANDBOX Project
62	UPR$	Upper Case Alpha	Alpha String	Upper Case Alpha String	Mark Power	DF V8 N1 p10
63	VIEWA	View Alpha	Alpha string	Shows Alpha, not stopping	Ken Emery	MCODE For Beginners
64	XTOAL	X to Alpha Left	Char value in X	Appends char to alpha on left	Hakan Thorngren	PPCJ V13 N7 p9
1	**-Math Fncts**	**Header**	**None**	**Shows "No Worries" in Alpha**	**Ángel Martin**	**SANDBOX Project**
2	ACOSH	Inverse COSH	Number in X	Value in X, x in LASTX	Nelson Crowle	AECROM Module

Fnc#	Function Name	Description	Input	Output	Author	Source
3	ASINH	Inverse SINH	Number in X	Value in X, x in LASTX	Nelson Crowle	AECROM Module
4	ATANH	Inverse TANH	Number in X	Value in X, x in LASTX	Nelson Crowle	AECROM Module
5	COSH	Hyperbolic Cosine	Number in X	Value in X, x in LASTX	Nelson Crowle	AECROM Module
6	D>H	Decimal to Hex	Dec String in Alpha	Hex string in Alpha	William Graham	PPCJ V12 N6 p19
7	DECX	Decrement X	Number in X	Number-1 in X	Ross Cooling	PPCJ V12 N12 p21
8	DFRAC	Decimal to Fraction	Decimal number in X	Fraction in Alpha	Frans de Vries	DF V9 N7 p8
9	E3/E+	Builds Matrix Pointer	N in X	1,00N left in X	Ángel Martin	SANDBOX Project
10	GEULER	Euler's Gamma Constant	None	0,5772156649	Ángel Martin	SANDBOX Project
11	H>D	Hex to Decimal	Hex string in Alpha	Dec String in Alpha	William Graham	PPCJ V12 N6 p19
12	INCX	Increment X	Number in X	Number+1 in X	Ross Cooling	PPCJ V12 N12 p21
13	PRIME?	Is X Prime?	Number in X	Yes: Divisor in X - No: none	Jason Delooze	PPCJ V11 N7 p30
14	QREM	Quotient Remainder	Y:Dividend, X: Divisor	Remainder	Ken Emery	MCODE For Beginners
15	QROOT	Quadratic Roots	X: a, Y:b, Z: c	X: x1, Y: x2; Z: 0 if Complex	Ángel Martin	SANDBOX Project
16	RANDN	Random Number	Seed	Random number	Ken Emery	MCODE For Beginners
17	REGSORT	Register Sort	X: Begin,end	Sorted registers	Hajo David	PPCJ V12 N5 p44
18	RCL+	Recall Sum	Number in Y, Reg# in X	Result in X	Ross Cooling	PPCJ V14 N4 p16
19	RCL-	Recall Subtract	Number in Y, Reg# in X	Result in X	Ross Cooling	PPCJ V14 N4 p16
20	RCL*	Recall Multiply	Number in Y, Reg# in X	Result in X	Ross Cooling	PPCJ V14 N4 p16
21	RCL/	Recall Divide	Number in Y, Reg# in X	Result in X	Ross Cooling	PPCJ V14 N4 p16
22	SIGNUM	Sign Number	Value in X	Sign in X, value to LASTX	Ross Cooling	PPCJ V12 N12 p31
23	SINH	Hyperbolic Sine	Number in X	Value in X, x in LASTX	Nelson Crowle	AECROM Module
24	STSORT	Sort Stack	Stack full	Stack sorted	David Phillips	PPCJ V12 N2 p13
25	T>BASE	Decimal to Base	Y: Base, X: Value	Alpha: new value	Ken Emery	MCODE For Beginners
26	TANH	Hyperbolic Tangent	Number in X	Value in X, x in LASTX	Nelson Crowle	AECROM Module
27	VMANT	View Mantissa	Decimal number	Mantissa	Ken Emery	MCODE For Beginners
28	X>=0?	X>=0?	Like X<-0?	Skips a line if false	Ángel Martin	SANDBOX Project
29	X>=Y?	X>=Y?	Like X<=Y?	Skips a line if false	Ken Emery	MCODE For Beginners
30	X=1?	X=1?	Value in X	Skips a line if false	Nelson Crowle	NFCROM
31	X=Y?Z?	Double Comparison	Values in X, Y, and Z	Skips one or two lines	Ken Emery	MCODE For Beginners
32	Z+	Complex Addition	T:Im1, T: Re1, Y: Im2, X:Re2	Y: ImSum; X: ReSum	Ángel Martin	SANDBOX Project
33	Z-	Complex Subtraction	T:Im1, T: Re1, Y: Im2, X:Re2	Y: ImDiff; X: ReDiff	Ángel Martin	SANDBOX Project
34	Z*	Complex Multiplication	T:Im1, T: Re1, Y: Im2, X:Re2	Y: ImProd, X: ReProd	Ángel Martin	SANDBOX Project
35	Z/	Complex Division	T:Im1, T: Re1, Y: Im2, X:Re2	Y: ImDiv; X:ReDiv	Ángel Martin	SANDBOX Project
36	1/Z	Complex Inversion	Y: Im(Z); X: Re(Z)	Y: ImInv; X: ReInv	Ángel Martin	SANDBOX Project

Fnc#	Function Name	Description	Input	Output	Author	Source
37	**-Hacker Lab**	**Header**	**None**	**None (NOP)**	**Ángel Martin**	**SANDBOX Project**
38	aNNRCL	Absolute addres NNRCL	Absolute address in X	NNN in X	Ken Emery	MCODE For Beginners
39	BCD>BIN	BCD to Binary	BCD in X	NNN in X	Ken Emery	MCODE For Beginners
40	BIN>BCD	Binary to BCD	NNN in X	BCD in X	Ken Emery	MCODE For Beginners
41	CHKROM	Check ROM	XROM Number	Test result message	HP Co.	HP-IL Devel ROM
42	FDATA	Function Data	Prompts for Function Name	Address, Type, KA data in Alpha	Klaus Huppertz	PRISMA, Jan-90
43	GETW	Get Word	Absolute address in X	Word Value in X as NNN	Ángel Martin	SANDBOX Project
44	HEX>NNN	Code	Hex in Alpha	NNN in X	Ken Emery	MCODE For Beginners
45	HEX>VSM	HEX to VASM Oct	Prompts for Hex in Alpha	Oct in Alpha in VASM format	Ken Emery	MCODE For Beginners
46	HEXIN	Enter NNN Directly	Prompts for Hex in Alpha	NNN in X	Hakan Thorngren	PPCJ V
47	HXENTRY	Enter NNN Directly	Prompts for Hex in Alpha	NNN in X	Clifford Stern	MCODE For Beginners
48	MNF	Mainframe Function	Prompts for function code	Executes the function	Clifford Stern	PPCJ V12 N3 p37
49	NNN>HEX	Decode	NNN in X	Hex in Alpha	Clifford Stern	MCODE For Beginners
50	RAMEDIT	RAM Editor	Address in X or PRGM pointer	RAM Editor	Hakan Thorngren	PPCJ V13 N4 p26
51	RCLB	Recall Byte	Address in M	Byte in X	Mark Power	DF V7 N8 p24
52	ROMIN	ROM In	Prompts for Page number	ROM Read in	Warren Furlow	www.hp41.org
53	ROMOUT	ROM Out	Prompts for Page number	ROM written out	Warren Furlow	www.hp41.org
54	STOB	Store Byte	Address in M, Byte in X	Stores Byte	Mark Power	DF V7 N8 p24
55	SUMROM	Checksum Calculation	Prompts for Page number	Checksum updated	George Ioannou	DF V3 N1 p10
56	VSM>HEX	VASM Oct to HEX	Prompt for Oct in Alpha	Hex in Alpha	Ken Emery	MCODE For Beginners
57	X<>B	Exchange Byte	Address in M, Byte in X	Exchanged values	Mark Power	DF V7 N8 p24
58	XQ>XR	XEQ to XROM	Program name in Alpha	XEQ changed to XROM	W&W GmbH	RamBox
59	**-Playground**	**Header**	**None**	**None (NOP)**	**Ángel Martin**	**SANDBOX Project**
60	BUZZON	Buzz Mode On	None	Sets buzzer on	Andreas Meyer	Cursor N.2/89 p14
61	CLAXON	Alternative Beep	None	Distorted Tone	Mark Power	DF V7 N7 p12
62	NOBUZZ	Buzz Mode Off	None	Sets buzzer off	Ángel Martin	SANDBOX Project
63	RASP	Alternative Beep	None	Rasping Tones	Mark Power	DF V7 N7 p12
64	ROLLERS	High Rollers Game	None	Game	Ross Cooling	PPCJ V