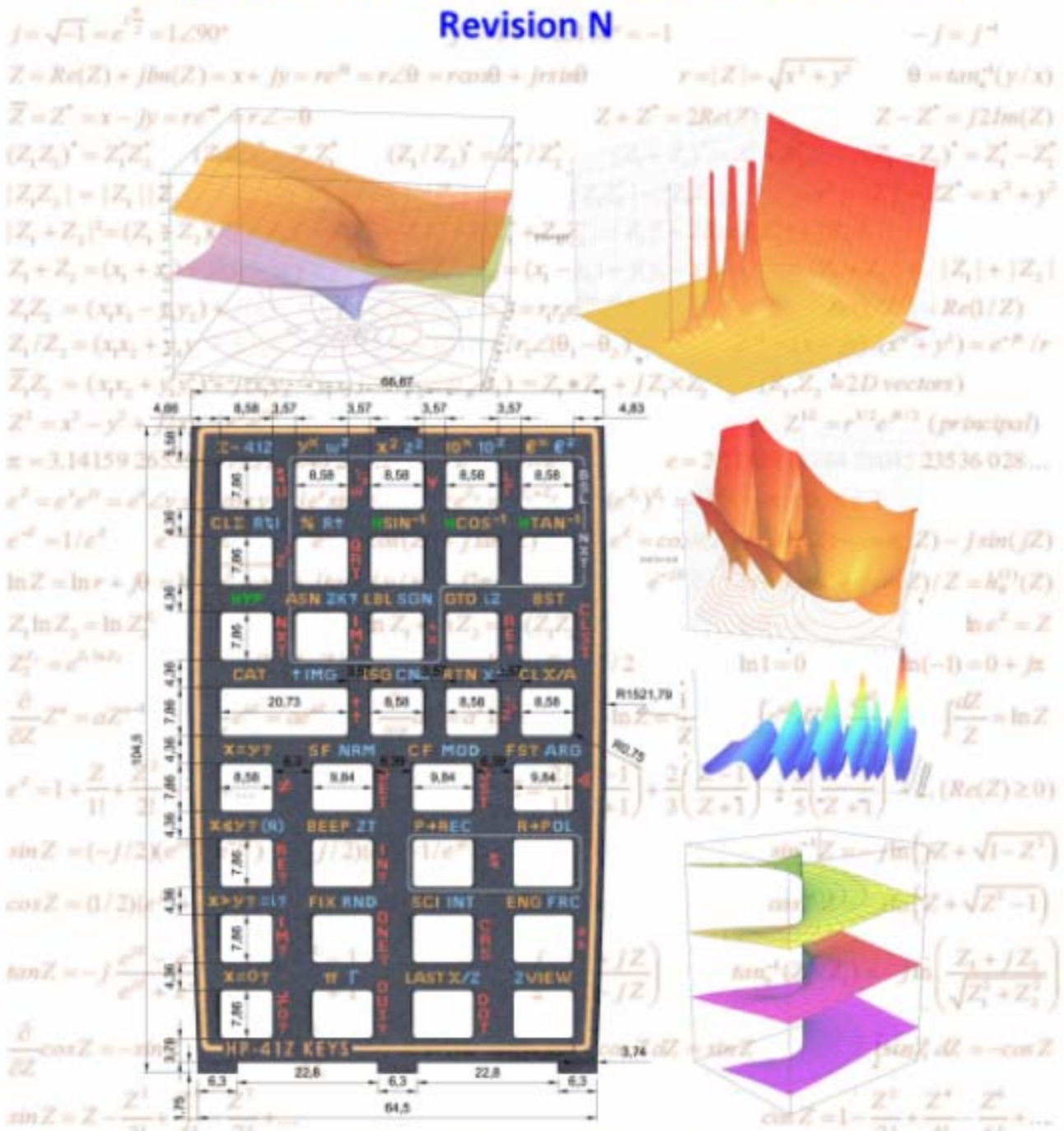


HP-41Z Module

## Revision N

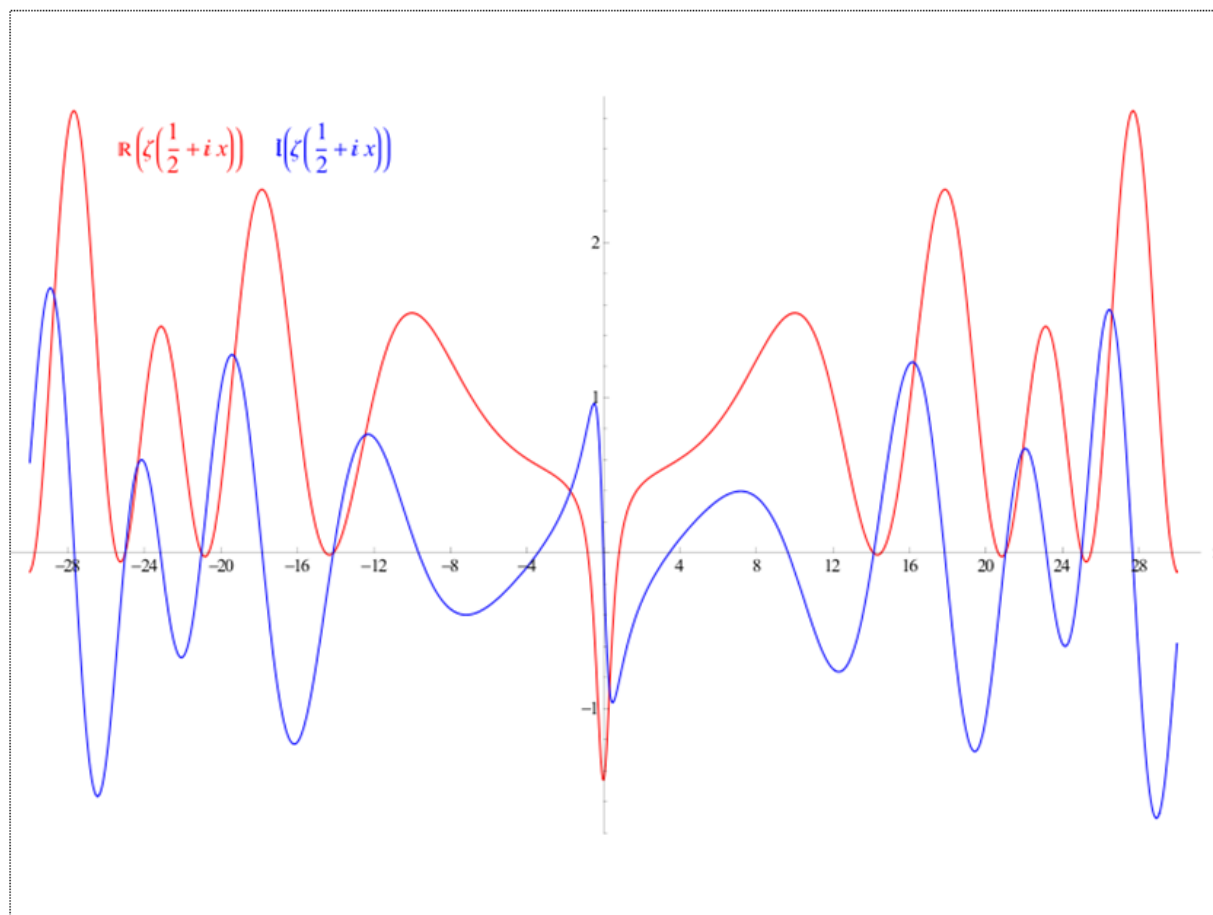


## User's Manual and Quick Reference Guide

Written and developed by: Ángel M. Martín  
June 2013

This compilation, revision A.4.5.

Copyright © 2005-2013 Ángel M. Martín



Published under the GNU software licence agreement.

The author wishes to thank the contributors to this project in various ways, as follows:

W. Doug Wilder, who wrote the code for the non-merged functions in program mode,  
Håkan Thörngren for his assistance and advices on the Memory Buffer implementation,  
Valentín Albillo, who wrote the original "PROOT" FOCAL program,  
M. Luján García, who prepared the 41Z Keys overlay bitmap file,  
Jean-Marc Baillard, a constant reference for all Math routines.

Some graphics taken from <http://www.clarku.edu/~djoyce/complex>, copyright 1999 by David E. Joyce.  
Some graphics taken from <http://www.wikipedia.org>

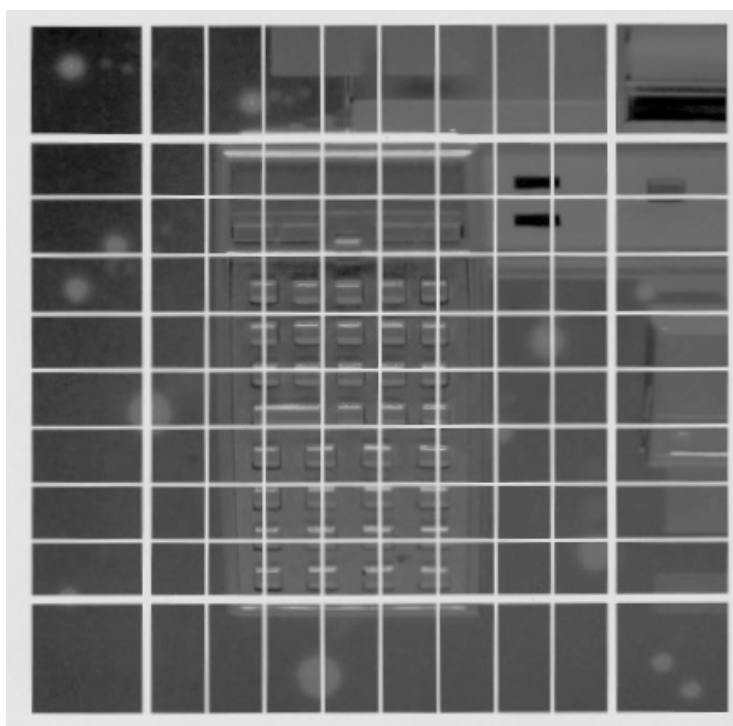
Screen captures taken from V41, Windows-based emulator developed by Warren Furlow. See  
<http://www.hp41.org/>

Original authors retain all copyrights, and should be mentioned in writing by any party utilizing this material. No commercial usage of any kind is allowed.

# Table of Contents.

0. <u>Preamble: a Complex Relapse.</u>	7
1. <u>Introduction.</u>	9
2. <u>Complex Stack, number entering and displaying.</u>	9
2.1 <a href="#">Rectangular vs. Polar modes</a>	10
2.2 <a href="#">Data entry conventions</a>	11
3. <u>User interface enhancements.</u>	12
3.1 <a href="#">Display and Conversion functions</a>	12
3.2 <a href="#">Complex Natural Data entry</a>	13
3.3 <a href="#">The Complex User Assignments</a>	16
3.4 <a href="#">The Complex Keyboard</a>	17
4. <u>Stack and Memory functions.</u>	20
4.1. <a href="#">Stack functions group</a>	20
4.2. <a href="#">ZSTO Math function group</a>	24
5. <u>Complex Math.</u>	25
5.1. <a href="#">Simple Arithmetic</a>	25
5.2. <a href="#">Exponentials and Powers that be</a>	27
5.3. <a href="#">Complex Logarithm</a>	32
6. <u>Complex Geometry</u>	34
6.1 <a href="#">Basic functions</a>	34
6.2 <a href="#">Complex Comparisons</a>	37
7. <u>Complex Trigonometry</u>	40
7.1 <a href="#">Basic Functions</a>	40
8. <u>2D-vectors or complex numbers?</u>	43
8.1 <a href="#">Two parallel worlds</a>	43

9. <u>It's a Gamma world out there</u>	44
9.1. <a href="#">Lanczos approximation</a>	44
9.2. <a href="#">Digamma and LnGamma</a>	46
9.3. <a href="#">Riemann's Zeta function</a>	48
10. <u>Application programs</u>	50
10.0. <a href="#">Delta-Wye Transformation</a>	51
10.1. <a href="#">Solution to quadratic and cubic equations</a>	52
10.2. <a href="#">Lambert W function</a>	55
10.3. <a href="#">Multi-valued functions</a>	56
10.4. <a href="#">Polynomial roots</a>	58
10.5. <a href="#">Solutions to <math>f(z)=0</math></a>	60
10.6. <a href="#">Bessel Functions</a>	64
10.7. <a href="#">Polylogarithm</a>	69
10.8. <a href="#">Lerch Transcendent</a>	70
10.9. <a href="#">Exponential Integrals</a>	71
<u>Appendices.</u>	73
a1.- <a href="#">Complex Buffer functions</a>	73
a2.- <a href="#">Complex Keyboard key-maps</a>	78
a3.- <a href="#">Formula compendium</a>	79
a4.- <a href="#">Quick Reference Guide</a>	80
a5.- <a href="#">Complex functions logic</a>	84







## 41Z Revision 4L – Complex Number Module for the HP-41

### 0. Preamble - A Complex Relapse.

---

The 41Z module was the author's first project to use a combination of both MCODE and math techniques put together in service of a dedicated purpose. The design of the complex stack in particular was the subject of careful implementation and extensive testing – glad to say the effort has paid off and that the design has worked well to date.

This new revision benefits from the usage of Library#4 – a dedicated ROM packed with MCODE routines used frequently and repeatedly by several other modules (SandMath, PowerCL amongst others). Library#4 is located in page 4, and must be present on the system for this version of the 41Z module to work properly. All interaction occurs behind the scenes and transparently to the user.

There is a Library presence check made upon the Calculator ON event, showing an error message if it's not found - but otherwise the library is completely invisible to the user. Refer to the appropriate instructions manual for installation details. For compatibility reasons, make sure you have revision "H" or higher of the Library#4 ROM.

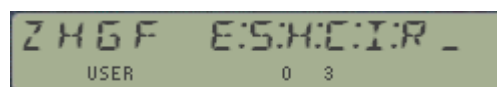
Changing the original code to take advantage of the library took some effort, but the benefits of doing so have been twofold:

0. The revised code is more robust and better structured,
1. A lot of room is recovered and can be used for new functionality.

Putting the reclaimed room to a worthwhile use required dealing with the FAT limitation. The 41Z Module already had 128 functions, meaning that both FAT's were used up – therefore further consolidation and changes to the functions were needed to allow for the new additions.

This is a summary of the most important changes:

1. Added nine new functions to the module, all in the High-Level math section. Seven of them are to calculate the Error function and the Exponential, Sine, Cosine (and their Hyperbolic counterparts) integrals – **ZHGF**, **ZERF**, **ZEI**, **ZCI**, **ZHCI**, **ZSI**, and **ZHSI** - all using the Hypergeometric Function method. The remaining two are **ZLI2** and **ZLIN**, to calculate the Polylogarithm. All of them work with complex arguments.
2. Removed the least relevant functions **ZIMAG**, **ZREAL**, **ZHALF**, and **ZDBL** – easily replaceable by equivalent combinations of standard functions.
3. Usage of section headers, so they can be called in FOCAL programs to perform actual calculations. This is the case for **-ZVECTOR** (which performs **ZGPRD**), **-ZSTACK** (which does **HARMN**) and **-HL ZMATH** (which performs **2^X-1**). These "hidden" functions are only used in dedicated sections of the module, and FOCAL programs.
4. Double-duty usage of the new function **ZHGF** – the Complex Hypergeometric Function (written by Jean-Marc Baillard). In RUN mode it is a new function launcher, grouping the functions that implement this calculation method. In PRGM mode however it "just" performs the actual execution work.





5. Double-duty usage of the Bessel auxiliary function **ZBS#**, now effectively performing (via control flags to decide the case) the same tasks done before by the **ZBS** and **ZBS2** functions. This is used by all the other Bessel functions, for 1<sup>st</sup> and 2<sup>nd</sup>. kinds.
6. Removal of the **ZMTV** entry from the FAT – the function still exists as it was (calculates all the solutions of the multi-value functions), but it has been incorporated to the **ZNEXT** launcher. More on this later on.
7. Adopted the general convention to always use MCODE headers for all functions, even for those which really are FOCAL programs. This improves readability, reduces the code size, and facilitates coding them as extensions to the launchers. The drawback is that the 41 OS interprets the programs to be in PRIVATE mode and therefore you won't be able to see the steps. Use the program listings within this manual instead. Their names are in BLACK font color to differentiate them from the native MCODE ones, which are in BLUE.
8. And last but not least, numerous improvements in the code all throughout the module, rearranged sections and overall improvement in the usability of the functions - notably **NXTNRT** prompts when called from the **ZNEXT** launcher; now allows using the top two key rows for index shortcuts 1-10.

**Warning:** due to all those function removals and additions, this version of the 41Z module has a different function arrangement in the FATs. If you have written your own programs using 41Z functions they will not match the new XROM id#'s and therefore will need to be re-written. At this point in the game this is highly unlikely, but just in case this is to be observed.

**Note also** that (contrary to the SandMath or PowerCL modules) there isn't any "sub-functions" group in the 41Z\_4L module. There simply wasn't enough room available for that, and also the Library#4 was already filled-up – with no room left for extensions to the scheme.

#### Note for Advanced Users:

Even if the 41Z\_4L is an 8k module, it is possible to configure only the first (lower) page as an independent ROM. This may be helpful when you need the upper port to become available for other modules (like mapping the CL's MMU to another module temporarily); or permanently if you don't care about the High Level Math (Special Functions) and 2D-Vectors sections.

Think however that the FAT entries for **ZKBRD**, **^IM/AG** and the other function launchers are in the upper page, so they'll be gone as well if you use the reduced foot-print (4k) version of the 41Z.

Upper Page XROM #01	High-Level Math, Zvectors, Function Launchers
Lower Page XROM #04	41Z main, Z-stack

Note however that it's not possible to do it the other way around; plugging only the upper page of the module will be dysfunctional for the most part and likely to freeze the calculator– rather do not attempt!



## 1. Introduction.

Complex Number handling is perhaps one of the very few areas where the HP-41 didn't have a comprehensive set of native functions, written in machine code and so taking advantage of the speed and programming enhancements derived from it. While both the Math Pack and the Advantage Rom provide FOCAL programs for complex number treatment, neither of them could be properly consider as a full function set to the effect of, for instance, the powerful Matrix handling functions contained in the Advantage Rom (in turn an evolution of those implemented in the CCD Module).

The 41Z module provides a significant number of functions that should address the vast majority of complex number problems, in a user-friendly context, and with full consistency. To that goal this manual should also contribute to get you familiar with their usage and applications, hopefully learning a couple of new things and having some fun during the process.

The implementation provided in this 8k-module is a third-generation code, building on the initial 41Z ROM released by the author in April 2005 – and on the previous version released in 2009. Numerous improvements have been added to the initial function set, notably the addition of a *4-level complex stack*, a *POLAR mode*, and a fully featured *complex mode keyboard*. Memory management is facilitated by prompting functions that deal with complex arguments, like **ZSTO**, **ZSTO Math**, **ZRCL**, **Z<>**, and **ZVIEW** – all of them fully programmable as well.

## 2. Complex Stack, number entering and displaying.

A four-level complex stack is available to the user to perform all complex calculations. The complex stack levels are called **U**, **V**, **W**, and **Z** – from top to bottom. Each level holds two real numbers, the imaginary and real parts of the corresponding complex number. Besides them, a "LastZ" complex register **S** temporarily stores the argument of the last executed function.

41Z Complex Stack		
<b>b11</b>	<b>non-zero</b>	
<b>b10</b>	<b>U</b>	-
<b>b9</b>		-
<b>b8</b>	<b>V</b>	-
<b>b7</b>		-
<b>b6</b>	<b>W</b>	<b>T</b>
<b>b5</b>		<b>Z</b>
<b>b4</b>	<b>Z</b>	<b>Y</b>
<b>b3</b>		<b>X</b>
<b>b2</b>	<b>(S)</b>	-
<b>b1</b>		<b>L</b>
<b>b0</b>	<b>Header</b>	

The complex stack uses a dedicated buffer in main memory. It is created and maintained by the 41Z module and its operation should be transparent to the user. This buffer is independent from the real stack (X, Y, Z, and T registers) but it's important however to understand how they interact with each other. A complex number uses two real stack levels (like X and Y), but a single complex stack level (like **Z** or **W**). The figure on the left shows the relationship between the complex and real stacks, which is automatically maintained upon function execution, as we'll see later on.

The real stack is used to enter the complex number values, real and imaginary parts. The input sequence varies depending on the method used *but all functions will expect the imaginary part in the Y register and the real part in the X register*. More about this later.

The contents of complex and real stack levels are automatically synchronized before and after each complex operation is performed. This may just involve real levels X,Y and complex level **Z** if it's a monadic (or unary) operation requiring a single complex argument, or may also involve real levels Z,T and complex level **W** if it's a dual operation requiring two complex arguments.

**Monadic functions** will assume that the real numbers in X,Y are the most up-to-date values for the real and imaginary parts of the complex argument. They will overwrite the contents of complex level **Z**. This allows quick editing and modification of the complex argument prior to executing the function.

**Dual functions** will assume that the second argument is stored in **W**, that is level 2 of the complex stack, and will thus ignore the values contained in real stack registers Z,T. Note that because the real stack overflows when trying to hold more than four different values, it is not a reliable way to input two complex numbers at once.

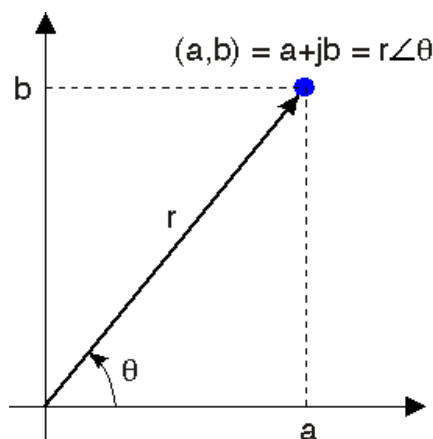
The design objective has been to employ as much as possible the same rules and conventions as for the real number stack, only for complex numbers instead. This has been accomplished in all aspects of data entering, with the exception of automated complex stack lift: with a few exceptions, entering two complex numbers into the complex stack requires pressing **ZENTER^** to separate them.

Once again: entering two complex numbers into the complex stack is accomplished by executing **ZENTER^** to separate the first and second complex number. Exceptions to this rule are the other complex-stack lifting functions, such as **GEUZ**, **ZRCL**, **ZRPL^**, **IMAGINE**, **^ZIMAG**, **^ZREAL**, **^IM/AG**, and the "**Complex Keypad**". Here the left-side symbol "^" (SHIFT-N) represents an input action.

## 2.1 Rectangular vs. Polar forms.

---

The HP-41 sorely lacks a polar vs. Rectangular mode. This limitation is also overcome on the 41Z module, with the functions **POLAR** and **RECT** to switch back and forth between these modes. It uses an internal flag in the complex buffer, not part of the 41 system flags. The operation is simplified in that complex numbers are always stored in their rectangular (or Cartesian) form,  $z=x+yi$ . So while all functions expect the argument(s) in rectangular form, yet the results are shown in the appropriate format as defined by the POLAR or RECT mode. (The notable exception is **ZPOL**, which always returns the value in Polar form). Note also that the POLAR mode is directly affected by the angular mode as well, as it occurs with real argument values.



**Note:** The POLAR display of the complex number requires an additional R-P conversion after the result is calculated in Cartesian form. The Polar form is temporarily stored in the Real stack registers T,Z – which have no active role in the Complex Stack and therefore can always be used as scratch. Once again, no changes are made to either X,Y registers or Complex stack level **Z**.

## 2.2 Data Entry Conventions

And how about complex number entering? Here the world divides in two camps, depending on whether the sequence is: "Re( $z$ ), ENTER<sup>^</sup>, Im( $z$ )" – like on the HP-42S -, or its reverse: "Im( $z$ ), ENTER<sup>^</sup>, Re( $z$ )" – like on the HP-32/33S and other FOCAL programs -. With the 41Z module you can do it either way, but it's important to remember that *regardless of how you introduce the numbers, all functions expect the imaginary part in the Y real-stack register and the real part in the X real-stack register*.

Fast data entry will typically use the sequence Im( $z$ ) , ENTER<sup>^</sup>, Re( $z$ ), followed by the complex function. This is called the "Direct" data entry, as opposed to the "Natural" data entry, which would first input the real part. The 41Z module includes the function "**^IM/AG**" that can be used to input the number using the "Natural" convention (reversed from the Direct one).

Its usage is the same as the "i"-function on the HP-35s, to separate the real and the imaginary parts. The sequence is completed by pressing ENTER<sup>^</sup> or R/S, after which the imaginary part will be left in the Y register and the real part in the X register as explained before.

*(Incidentally, the 42S implementation of the complex stack isn't suitable for a true 4-level, since the COMPLEX function requires two levels prior to making the conversion!)*

Other functions and special functionality in the 41Z module can be used as shortcuts to input purely real or imaginary numbers more efficiently. For instance, to enter the imaginary unit one need only press: 1, **ZIMAG<sup>^</sup>** (which is also equivalent to executing the **IMAGINE** function) – or simply "**ZKBRD, Radix, 1**" using the "complex keypad". And to enter 4 as a complex number, just press: 4, **ZREAL<sup>^</sup>** - or simply "**ZKBRD, 4**" using the "complex keypad".

Incidentally, the 42S implementation fails short from delivering a true 4-level stack, due to the COMPLEX function and the fact that it requires two stack levels to be available to combine the complex number. In this regard the 41Z solution is a better one.



Two (opposite) alternatives to data entry: COMPLEX key on the 42S, and "i" key on the 35S

### 3. User interface enhancements.

Table-3.1: Functions to enhance the user interface.

Index	Function	Group	Description
1	<b>ZK?YN</b>	Usability	Activates and deactivates the Complex Assignments
2	<b>ZKBRD</b>	Usability	Accesses most of the 41Z functions plus special features
3	<b>ZAVIEW</b>	Display	Views complex number in X,Y
4	<b>POLAR</b>	Display	Displays complex numbers in Polar form
5	<b>RECT</b>	Display	Displays complex numbers in Rectangular form
6	<b>^IM/AG</b>	Usability	Inputs Imaginary Part (or Argument) of complex number

These functions facilitate the showing of the complex number on the display, and the conversion between the polar and rectangular forms. They enhance the usability by supplying a system to handle the lack of native complex number treatment capabilities of the calculator.

#### 3.1 Display mode and conversion functions.

<b>ZAVIEW</b>	<b>Complex number AVIEW</b>	Uses ALPHA registers	
---------------	-----------------------------	----------------------	--

Shows the contents of the complex stack level **Z** in the display, using the current complex display mode (POLAR or RECT).:

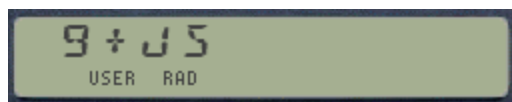
RECT:  $\text{Re}(z) + j \text{Im}(z)$  ; where  $\text{Re}(z)$  is stored in register X and  $\text{Im}(z)$  in register Y.

POLAR:  $\text{Mod}(z) < | \text{Arg}(z)$ ; *shown but not stored in the X,Y stack registers (!)*

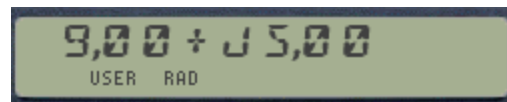
Note that **ZAVIEW** uses the ALPHA register, thus the previous contents of the M, N and O registers will be lost.

The displaying will respect the current DEG, RAD, or GRAD angular mode (in POLAR form), the current FIX, SCI or ENG settings, as well as the number of decimal places selected on the calculator. Note that "j" precedes the imaginary part, as this improves legibility with real-life complex numbers, with decimal imaginary parts.

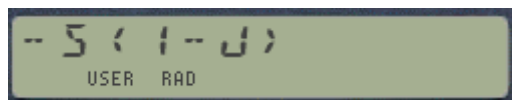
For a simplified visualization, **ZAVIEW** *won't show decimal zeroes if the number is an integer.* This is done automatically regardless of the number of decimal places selected in the calculator; so one can immediately tell whether the real or imaginary parts are true integers as opposed to having some decimal content hidden in the least significant places not shown.



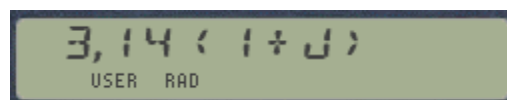
versus:



**ZAVIEW** *will extract common factor if both the real and imaginary parts are equal:*



or also:



Executing the functions **POLAR** and **RECT** will also display the complex number currently stored in X,Y

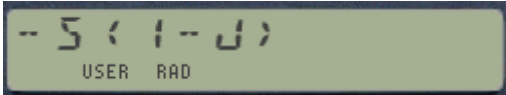
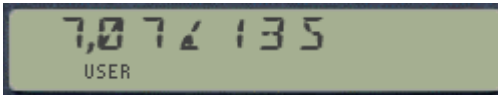
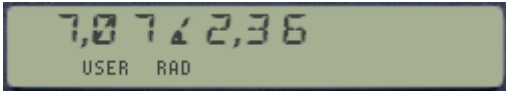
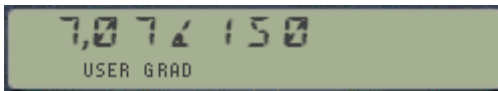
<b>POLAR</b>	<b>Sets POLAR mode on</b>	Displays number	Shows in SET mode
<b>RECT</b>	<b>Sets RECT mode on</b>	Displays number	Shows in SET mode
<b>ZPOL</b>	<b>Convert to Polar</b>	Converts X,Y to POLAR	<i>Always shows in POLAR</i>
<b>ZREC</b>	<b>Convert to Rectangular</b>	Converts X,Y to RECT	Shows in SET mode

**ZPOL** Converts the complex number in the **Z** stack level from rectangular to polar mode. If executed in run mode, the display shows the value of its magnitude (its module) and its argument, as follows:

Mod < Arg ; where:

Mod =  $|z|$  and Arg= $\alpha$   **$[z = |z| * e^{i\alpha}]$**

The argument value will be expressed in the angular settings currently selected: DEG, RAD, or GRAD.

	equals	
	or also	

**ZREC** is the reciprocal function, and will convert the complex number in **Z** (assumed to be in polar form) to rectangular form, showing it on the display (in run mode) in identical manner as **ZAVIEW**.

In fact, if it weren't because of the displaying capabilities, these two functions will be identical to the pair R-P and P-R, standard on the calculator. Recognizing this, they're assigned to the very same position as their real counterparts on the Complex User keyboard.

Notice that contrary to the **POLAR** and **RECT** functions (which only display the values), **ZPOL** and **ZREC** *perform the actual conversion of the values and store them in the stack registers* (complex and real). This is also very useful to enter complex numbers directly in polar form, simply using the sequence: (direct data entry: Angle first, then modulus):

- Arg(z), ENTER^,  $|z|$ , **ZREC** ->  $\text{Re}(z) + j \text{Im}(z)$

### 3.2 Complex Natural Data Entry.

This function belongs to its own category, as an automated way to input a complex number using the "Natural" data entry convention: Real part first, Imaginary part next. Its major advantage (besides allowing the natural data entry sequence) is that *it performs a complex stack lift upon completion of the data entry*, thus there's no need to use **ZENTER^** to input the complex number into the complex stack. That alone justifies its inclusion on the 41Z module.

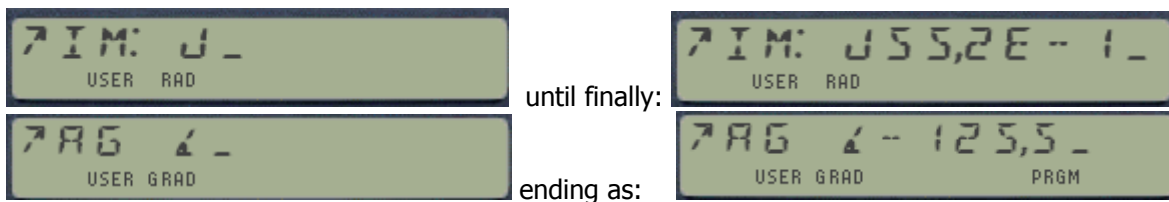
<b>^IM/AG _</b>	<b>Inputs Im(z)/Arg(z) Part</b>	<b>Does Stack Lift</b>	Prompting function
-----------------	---------------------------------	------------------------	--------------------

The function will prompt for the imaginary part (or the argument if in POLAR mode) of the complex number being entered. The design mimics that on the HP-35S calculator, and it's used as a way to separate the two complex parts during the complex number data entering.

*A few important considerations are:*

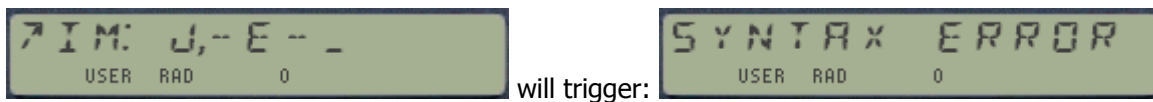
- The real part (or module) must be introduced right *before* calling it, so it's in X during the data entry.
- The keyboard is redefined to allow for numeric digits, RADIX, CHS and EEX as only valid keys.
- The radix symbol used (comma or dot) is controlled by the user flag 28.
- Only one RADIX character will be allowed in the mantissa – and none in the exponent.
- Only nine digits will be used for the mantissa, and two in the exponent. **^IM/AG** will not check for that during the input process, but exceeding entries will simply be ignored.
- Only one EEX can exist in the imaginary part - **^IM/AG** will check for that.
- Only one CHS can be used for the mantissa sign, **^IM/AG** will check for that.
- Multiple CHS can be used for the exponent sign, but **^IM/AG** will apply the arithmetic rules to determine the final sign as follows: odd number is negative, even number is positive.
- Pressing Back Arrow will remove the last entry, be that a number, Radix, EEX or CHS. If the entry is the first one it will cancel the process and will discard the real part as well.
- The sequence must be ended by pressing ENTER^ or R/S.
- The display cue is different depending on the actual complex mode (RECT or POLAR), and it's controlled automatically.
- Upon completion, the complex number is pushed into the **Z** complex stack level, and placed on the X,Y real stack registers as well following the same 41Z convention: real part in X and imaginary part in Y. The complex stack is lifted and the real stack is synchronized accordingly.

The screens below show usage examples in RECT and POLAR modes:



Note: To extract the numeric value from the input string, **^IM/AG** executes the same code as the X-function **ANUM**. All conversion conventions will follow the same **ANUM** logic. Suffice it to say that the implementation of **^IM/AG** is not absolute perfect and you can trip it up if that's what you really want – but it should prevent likely errors that could yield incorrect results. It's a very convenient way to meet this need solving the diverse issues associated with its generic character.

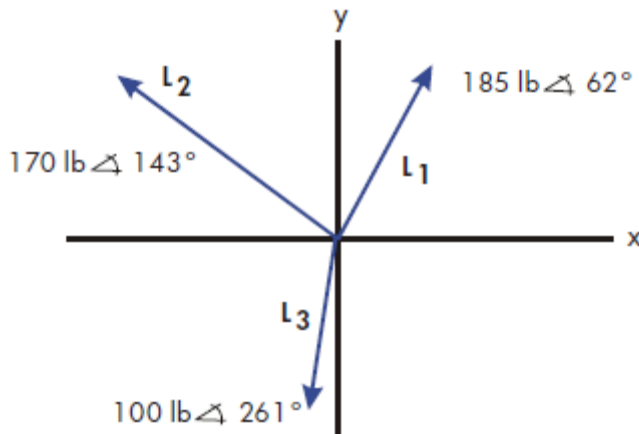
If the input string doesn't yield any sensible numeric result, the message "SYNTAX ERROR" is briefly shown in the display, and the stack is restored to its status prior to executing **^IM/AG**.



Some apparently incorrect syntax constructions will however be properly interpreted by **^IM/AG**, returning a valid imaginary part. This is for instance the case with multiple negative signs in the exponent, or decimal values after negative sign in the mantissa. Such is the flexibility of the ANUM function!

**Example:** Vector Load addition (taken from the 35s User Guide):-

We start by setting POLAR and DEG modes, then using the  $\wedge\text{IM/AG}$  function three times will set the three complex numbers on the complex stack, and finally simply execute the complex addition function  $Z+$  twice:



**POLAR, DEG**

185,  $\wedge\text{IM/AG}$ , 62, ENTER $\wedge$

170,  $\wedge\text{IM/AG}$ , 143, R/S

100,  $\wedge\text{IM/AG}$ , 261, R/S

**Z+, Z+**

**Result: -> 178,9372 <) 111,1489**

Or in Rectangular mode (as it's saved in XY):

**RECT** -> -64,559 + J166,885

Note the following points:

- We used indistinctly ENTER $\wedge$  and R/S to terminate the complex number entry.
- No need to store intermediate results as the complex buffer can hold up to four levels.
- We didn't need to use **ZENTER $\wedge$**  to push the complex numbers into the complex stack because the stack-lift was performed by  $\wedge\text{IM/AG}$ .

With regard to the data entry sequence, one could have used **ZREC** instead of  $\wedge\text{IM/AG}$  – albeit in that case it would have been in “direct mode”, as opposed to the more intuitive natural convention. It also requires pressing **ZENTER $\wedge$**  to push each number into the complex stack.

This is the keystroke sequence and partial results (assuming we're in **POLAR** mode)

62, ENTER $\wedge$ , 185, <b>ZREC</b> , <b>ZENTER<math>\wedge</math></b>	-> 185 <)62
143, ENTER $\wedge$ , 170, <b>ZREC</b> , <b>ZENTER<math>\wedge</math></b>	-> 170 <)143
261, ENTER $\wedge$ , 100, <b>ZREC</b>	-> 100 <)99
Z+, Z+	-> 178,9372 <) 111,1489

**One last remark about data displaying vs. data entry.**- As it was explained before, **ZPOL** will convert the complex number into Polar coordinates, and it will be displayed in **POLAR** form even if **RECT** mode is selected. This is the single one exception all throughout the 41z module, and it will only work immediately after pressing **ZPOL** but not for subsequent executions of **ZAVIEW** – which always expects the number is stored in rectangular form, and therefore will show an incorrect expression.



### 3.3 The Complex User Assignments.

The 41Z module provides a convenient way to do user key assignments *in masse*. Given the parallelisms between the real and complex number functions, the natural choice for many of the functions is “predetermined” to be that of their real counterparts.

A single function is used for the mass-assignment (or de-assignment) action:

<b>ZK?YN</b>	<b>Complex User Assignments</b>		Prompting function
--------------	---------------------------------	--	--------------------

**ZK?YN** automates the assignment and de-assignment of 37 functions. It prompts for a Yes/No answer, as follows:

- Answering “Y” will assign the complex functions to their target keys
- Answering “N” will de-assign them, and
- Pressing “Back Arrow” will cancel the function.
- Any other key input (including ON) will be ignored.

The assignment action will be indicated by the message “Z-KEYS: ON” or “Z-KEYS OFF” in the display during the time it takes to perform, followed by “PACKING” – and possibly “TRY AGAIN” should the enough number of memory registers not exist.

Note that **ZK?YN** is *selective*: any other key assignment not part of the complex functions set will not be modified.

Keycode	Unshifted Keys		Shifted Keys	
11	s+	ZHYP	s-	ZNXT
12	1/X	ZINV	y^x	W^Z
13	SQRT	ZSQRT	x^2	Z^2
14	LOG	ZLOG	10^x	ZALOG
15	LN	ZLN	e^x	ZEXP
21	x<>y	Z<>W	CLs	ZTRP
22	RDN	ZRDN	%	ZRUP
23	SIN	ZSIN	ASIN	ZASIN
24	COS	ZCOS	ACOS	ZACOS
25	TAN	ZTAN	ATAN	ZATAN
33	STO	ZSTO	LBL	n/a
34	RCL	ZRCL	GTO	n/a
41	ENTER^	^IMG	CAT	ZENTER^
42	CHS	n/a	ISG	ZNEG
44	EEX	n/a	CLx	CLZ
51	-	Z-	x=y?	Z=W?
61	+	Z+	x<=y?	Z=WR?
71	*	Z*	x>y?	Z=I?
81	/	Z/	x=0?	Z=0?
83	,	n/a	LASTx	LASTZ
84	R/S	ZAVIEW	VIEW	ZVIEW

Table 3.3. Complex key assignments done by ZK?YN

### 3.4 The Complex Keyboard.

As good as the user assignments are to effectively map out many of the 41Z functions, this method is not free from inconveniences. Perhaps the biggest disadvantage of the Complex Assignments is that it's frequently required to toggle the user mode back and forth, depending on whether it's a complex or a real (native) function to be executed.

Besides that, the Complex Assignments consume a relative large number of memory registers that can be needed for other purposes. Lastly, there are numerous 41Z functions not included on the user assignments map, and no more "logical" keys are available without compromising the usability of the calculator.

To solve these quibbles, the 41Z module provides an alternative method to access the majority of the complex functions, plus some unique additional functionality. It's called the **Complex Keyboard**, accessed by the function **ZKBRD**: a single key assignment unleashes the complete potential of the module, used as a **complex prefix**, or in different combinations with the SHIFT key and with itself.

Figure 3.4. Complex Keyboard overlay (with ZKBRD assigned to Sigma+).  
On the left: the version for V41. On the right, for i41CX



© 2009 M. Luján García.



The 41Z overlay can be downloaded from the HP-41 archive website, at:  
<http://www.hp41.org/LibView.cfm?Command=View&ItemID=893>

To use it with V41 emulator, replace the original file "*large.bmp*" in the V41 directory with the 41Z bitmap file, after renaming it to the same file name.

Here's how to access all the functions using **ZKBRD**:

**a.- Direct functions.** Simply press "**Z**" as a prefix to denote that the next function will operate on a complex argument, and not on a real one. These functions don't have any special marks, as they correspond to the standard functions on the HP-41 keyboard. There are *twenty 41Z functions* directly accessible like these.

**Examples:** Pressing **Z**, LN will execute **ZLN**; pressing **Z**, COS will execute **ZCOS**, etc...  
Pressing **Z**, + will execute **Z+**; pressing **Z**, R/S will execute **ZAVIEW**,

**b.- Shifted functions.** Press "**Z**" followed by the SHIFT key. These functions are either marked in blue when different from the standard SHIFTED ones, or just marked in yellow as part of the standard HP-41 keyboard (like  $x=y?$ , which will execute **Z=W?** if the pressed key sequence is this: **Z**, SHIFT,  $x=y?$ ).

**Examples:** pressing **Z**, SHIFT, LN will execute **ZEXP**; pressing **Z**, SHIFT, SIN will execute **ZASIN**,

Pressing **Z**, SHIFT, R/S will execute **ZVIEW** (a prompting function itself).

There are *thirty-one 41Z functions* accessible using this SHIFTED method.

**c.- Dual (alternate) functions.** Press "**Z**" *twice* as a double prefix to access the dual complex functions and many others. These functions are marked in red, on the right side of each available key.

**Examples.** Pressing **Z**, **Z**, 7 will execute **ZWDET**; pressing **Z**, **Z**, 5 will execute **ZWCROSS**, , and so on with all the "red-labeled" keys.

Pressing **Z**, **Z**, ENTER^ will execute **ZREPL**; pressing **Z**, **Z**, **Z** will execute **Z<>U**

There are *twenty-five 41Z functions* accessible using this Dual method.

**d.- Multi-value functions.** As a particular case of the dual functions case above, the ZNEXT function group is enabled by pressing "**Z**" *twice* and then SHIFT. This group is encircled on the keyboard overlay, and sets the five multi-value functions as follows: **NXTASN**, **NXTACS**, **NXTATN**, **NXTLN**, and **NXTNRT** (this one will also prompt for the root order, as an integer number 0-9).

Notice that pressing SHIFT while in the NEXT section toggles the display to "ZBSL". Use it as a shortcut to access the different Bessel functions of first and second kind provided in the 41, as follows: **ZJBS**, **ZIBS**, **ZKBS**, and **ZYBS**. – as well as **EIZ/IZ**, a particular case of Spherical Hankel  $h_1(0,z)$ .

**e.- Hyperbolic functions.** Press "**Z**" followed by SHIFT *twice* to access the three direct hyperbolics. Pressing SHIFT a third time will add the letter "A" to the function name and will enable the inverse functions. This action toggles with each subsequent pressing of SHIFT. (Watch the 41Z building up the function name in the display as you press the keys...)

**Example:** Pressing **Z**, SHIFT, SHIFT, SHIFT, **SIN** will execute **ZASINH**

**f.- Complex Keypads.** Press "**Z**" followed by a numeric key (0 to 9) to enter the corresponding digit as a complex number in the complex stack. Pressing "**Z**" followed by the Radix key, and then the numeric key will input the digit as an imaginary number as opposed to as a real number into the complex stack. This is a very useful shortcut to quickly input integer real or imaginary values for complex arithmetic or other operations (like multiplying by 2, etc.)

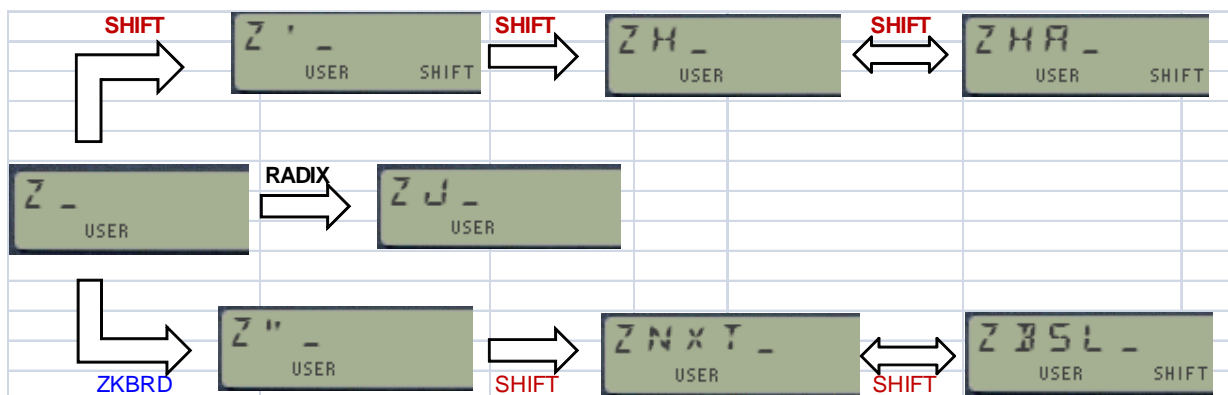
Pressing **Z**, XEQ calls the function **^IM/AG** for the Natural Data entry. This is obviously not shown on the keyboard – which has no changes to the key legends for un-shifted functions. Note that there are three different ways to invoke **^IM/AG**, as follows:

XEQ, ALPHA, SHIFT, N, I, M, /, A, G, ALPHA	-> the standard HP-41 method, or:
<b>Z</b> , SHIFT, ENTER^	-> shown in blue in the overlay, or:
<b>Z</b> , XEQ	-> not shown.

**Other keystrokes.** The 41Z module takes control of the calculator keyboard when **ZKBRD** is executed. Available keys are determined by the partial key sequence entered, as defined on the 41Z Keys overlay and as explained above. Pressing **USER** or **ALPHA** will have no effect, and pressing **ON** at any time will shut the calculator off. The *back arrow* key plays its usual important role during data entering, and also undoes the last key pressed during a multi-shifted key sequence. Try it by yourself and you'll see it's actually easier than giving examples on how it works here.

In summary: a complete new keyboard that is accessed by the "**Z**" blue prefix key. This being the only requisite, it's a near-perfect compromise once you get used to it – but if you don't like it you can use the User Assignments , the choice is yours.

The figure below shows the main different modes of the **ZKBRD** function, the real cornerstone of the 41Z module:



Press the Back-arrow key to bring the command chain back to the starting point (ZKBRD). Pressing it twice shows "NULL" and cancels out the sequence.

Pressing non-relevant keys (i.e. those not supposed to be included in the corresponding mode) causes the display to blink, and maintain the same prompt (no action taken).

## 4. Stack and Memory functions.

Let **Z** and **W** be the lower two levels of the complex stack, and *z* and *w* two complex numbers stored in **Z** and **W** respectively.  $\mathbf{Z} = \text{Re}(z) + j \text{Im}(z)$ ;  $\mathbf{W} = \text{Re}(w) + j \text{Im}(w)$

Note the use of "j" to express the imaginary unit, instead of "i". This isn't done to favor those EE's in the audience (you know who *we* are), but rather due to the displaying limitations of the 41 display: no lower-case letters for either i or j, and better-looking for the last one in caps.

Note also that despite their being used interchangeably, the complex stack register "**Z**" – in bold font – and the real stack register "Z" – in regular font – are not the same at all.

Table-4.1: Stack and memory function group.

Index	Function	Name	Description
1	<b>ZTRP</b>	Re(z)<>Im(z)	Exchanges (transposes) Re and Im for number in level <b>Z</b> .
2	<b>ZENTER^</b>	Complex ENTER^	Enters X,Y into complex level <b>Z</b> , <b>lifts complex stack</b> .
3	<b>ZREPL</b>	Complex Stack Fill	Fills complex stack with value(s) in X,Y
4	<b>ZRDN</b>	Complex Roll Down	Rolls complex stack down
5	<b>ZRUP</b>	Complex Roll Up	Rolls complex stack up
6	<b>ZREAL^</b>	Inputs real Z	Enters value in X as real-part only complex number
7	<b>ZIMAG^</b>	Inputs imaginary Z	Enters value in X as imaginary complex number
8	<b>Z&lt;&gt;W</b>	Complex Z<>W	Swaps complex levels <b>Z</b> and <b>W</b>
9 (*)	<b>Z&lt;&gt;ST __</b>	Complex Z<> level	Swaps complex levels <b>Z</b> and any stack level (0-4)
10 (*)	<b>ZRCL __</b>	Complex Recall	Recalls complex number from memory to level <b>Z</b>
11 (*)	<b>ZSTO __</b>	Complex Storage	Stores complex number in <b>Z</b> into memory
12 (*)	<b>Z&lt;&gt; __</b>	Complex Exchange	Exchanges number in level <b>Z</b> and memory
13 (*)	<b>ZVIEW __</b>	Complex Display	Shows Complex number stored in memory register
14	<b>CLZ</b>	Clears Level Z	Deletes complex level <b>Z</b>
15	<b>CLZST</b>	Clears Complex Stack	Clears all complex levels <b>U</b> , <b>V</b> , <b>W</b> , and <b>Z</b>
16	<b>ZREAL</b>	Extracts real part	<i>Removed</i> . Replace with: X<>Y, CLX, X<>Y
17	<b>ZIMAG</b>	Extracts Imag part	<i>Removed</i> . Replace with: CLX
18	<b>LASTZ</b>	Last number used	Recovers the last complex number used

(\*) Note: These functions are **fully programmable**. When used in a program their argument is taken from the next program line, see below for details.

### 4.1 Stack and memory functions group.

Let's start with the individual description of these functions in more detail, beginning with the simplest.

<b>ZTRP</b>	<b>Z Transpose</b>	Does Re <>Im	
-------------	--------------------	--------------	--

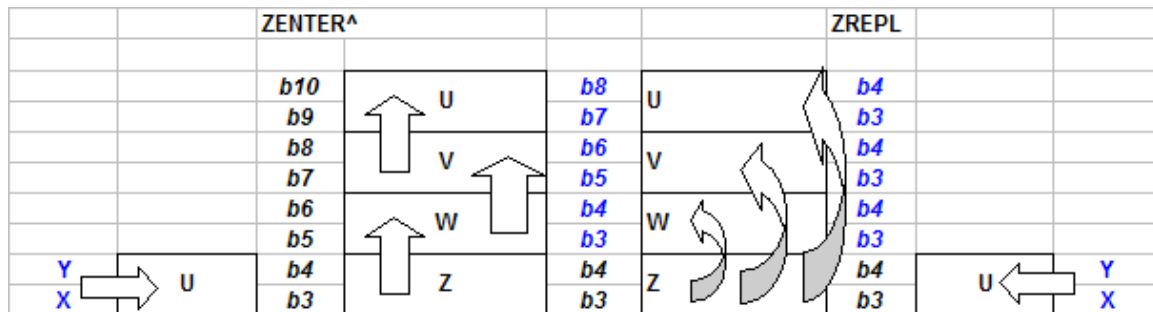
This function's very modest goal is to exchange the real and imaginary parts of the complex number stored in the **Z** level of the complex stack.

Hardly a worthwhile scope, you'd say, considering that the standard function X<>Y does the same thing? Indeed it is quite similar (and as such it's logically assigned to the shifted X<>Y key). But it's not quite the same, as in run mode **ZTRP** also shows on the display the complex number after transposing their real and imaginary parts. Besides, as it was mentioned in the introduction, this function may play an important role during data entry: it is the one to use when entering the real part first, as per the following sequence: Re(*z*), ENTER^, Im(*z*), **ZTRP**

Thus its use is analogous to the "COMPLEX" function on the HP-42S, also required to enter the complex number in the stack, from its two real components. Note that the other, alternative data entering sequence doesn't require using ZTRP, although the order of the real and imaginary parts is reversed (and arguably less intuitive):  $\text{Im}(z)$ , ENTER<sup>^</sup>,  $\text{Re}(z)$ . Either one of these two is entirely adequate once you become familiar with it and get used to using it - it's your choice.

<b>ZENTER<sup>^</sup></b>	<b>Enters X,Y into levels Z, W</b>	<b>Does Stack lift</b>	
<b>ZRPL<sup>^</sup></b>	<b>Fills complex stack</b>		

**ZENTER<sup>^</sup>** enters the values in X,Y as a complex number in the **Z** stack level, and performs stack lift (thus duplicates **Z** into **W** as well – and **U** is lost due to the complex stack spill-over). As said in the introduction, *always* use **ZENTER<sup>^</sup>** to perform stack lift when entering two (or more) complex numbers into the complex stack. This is required for the correct operation of dual complex functions, like **Z+**, or when doing chain calculations using the complex stack (which, unlike the real XYZT real stack, it does NOT have an automated stack lift triggered by the introduction of a new real number).



**ZRPL<sup>^</sup>** simply fills the complex stack with the values in the real registers X,Y. This is convenient in chained calculations (like the Horner method for polynomial evaluation). If executed in run mode it also displays the number in **Z**. This is in fact a common characteristic of all the functions in the 41Z module, built so to provide visual feedback on the action performed.

<b>ZREAL<sup>^</sup></b>	<b>Enters X in Z as (x+j0)</b>	<b>Does Stack Lift</b>	
<b>ZIMAG<sup>^</sup></b>	<b>Enters X in Z as (0+jX)</b>	<b>Does Stack Lift</b>	

These functions enter the value in X either as a purely real or purely imaginary number in complex form in the **Z** stack level, and perform stack lift. If executed in run mode it also displays the number in **Z** upon completion.

<b>CLZ</b>	<b>Clears complex stack level Z</b>		
<b>CLZST</b>	<b>Clears complete complex stack</b>		
<b>ZREAL</b>	Extracts Real part from Z	Removed.	X<>Y, CLX, X<>Y
<b>ZIMAG</b>	Extracts Imaginary part from Z	Removed.	CLX

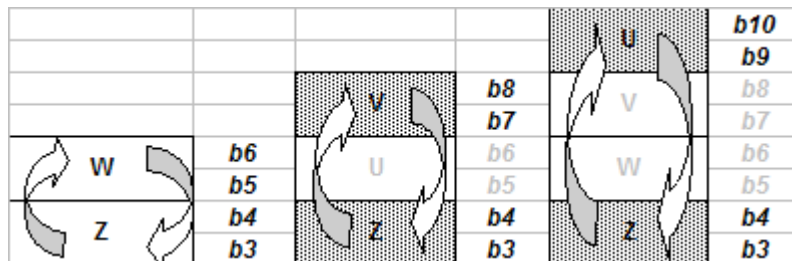
Use these four functions to partially or completely clear (delete) the contents of the complex stack **Z** level, or the complete complex stack. No frills, no caveats. The real stack will also be cleared appropriately.



<b>Z&lt;&gt;ST (*)</b>	<b>Exchanges Z and Stack</b>	Level# = 0,1,2,3,4	Prompting function
<b>Z&lt;&gt;V</b>	<b>Exchanges Z and V</b>		
<b>Z&lt;&gt;W</b>	<b>Exchanges Z and W</b>		

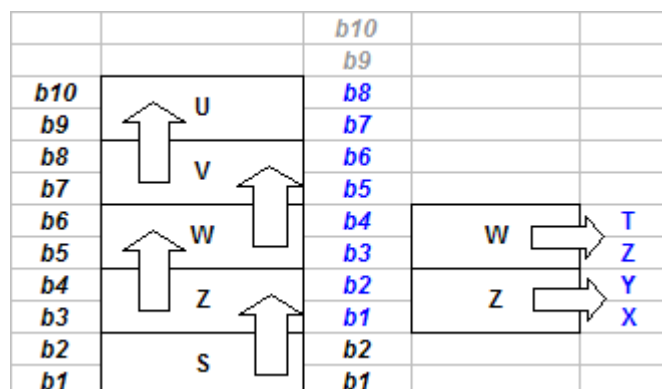
(\*) Fully programmable, see note in previous page.

Use these functions to swap the contents of the **Z** and **U/V/W** levels of the complex stack respectively. As always, the execution ends with **ZAVIEW** in run mode, displaying the new contents of the **Z** register.(which is also copied into the XY registers).



<b>LASTZ</b>	<b>Recalls last number used to Z</b>	<b>Does Stack Lift</b>	
--------------	--------------------------------------	------------------------	--

Similar to the LASTX function, **LASTZ** recalls the number used in the immediate preceding operation back to the **Z** level of the complex stack. A complex stack lift is performed, pushing the contents of **Z** up to the level **W**, and losing the previous content of **U**.



The majority of functions on the 41Z module perform an automated storage of their argument into the LastZ register, enabling the subsequent using of **LASTZ**. This will be notated in this manual when appropriate under each function description.

**Example:** to calculate  $[(z^2 + z)/2]$  simply press: **ZSQRT, LASTZ, Z+, ZHALF**

**Example:** Calculate the following expression without using any data registers:

$$F(z) = \text{Ln} [ z + \text{SQR}(z^2 + 1) ], \text{ for } z = 20+20i$$

Solution:

20, ENTER^, **ZRPL**  
**Z^2, 1, ZREAL^, Z+**  
**ZSQRT, Z+, ZLN**

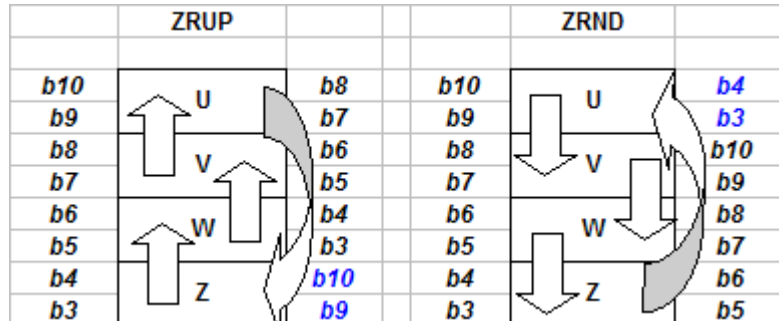
-> puts 20+20i in all 4 levels of the complex stack  
 -> could have used "1, +" as a more direct method  
 -> **4,035+J0,785**

Congratulations! You just calculated the hyperbolic arcsine of (20+20i).



<b>ZRDN</b>	<b>Rolls complex stack down</b>		
<b>ZRUP</b>	<b>Rolls complex stack up</b>		

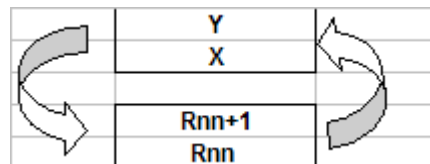
Like their real stack counterparts, these functions will roll the complex stack down or up respectively. If executed in run mode it also displays the number in **Z**. Real stack registers will be synchronized accordingly.



Be aware that although **ZRDN** and **ZRUP** do not perform stack lift, they update the Z complex register with the values present in X,Y upon the function execution. This behavior is common across all 41Z functions.

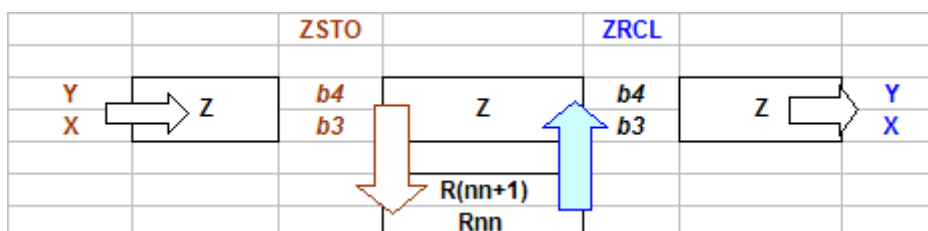
<b>ZVIEW __</b>	<b>Displays Complex Register value</b>		Prompting function
<b>Z&lt;&gt; __</b>	<b>Exchanges Z and complex register</b>		Prompting function

Like its real counterparts, these functions view or exchange the content of the complex stack level **Z** with that of the complex storage register given as its argument. Two standard storage registers are used, as per the above description.



<b>ZRCL __</b>	<b>Recall from Complex Register</b>	<b>Does Stack lift</b>	Prompting function
<b>ZSTO __</b>	<b>Store in Complex Register</b>		Prompting function

Like their real counterparts, these functions are used to Recall or store the complex number in Z from or into the complex register which number is specified as the function's argument. In fact two (real) storage registers are used, one for the imaginary part and another for the real part. This means that CRnn corresponds to the real storage registers Rnn and R(nn+1).



**ZRCL** will perform complex stack lift upon recalling the contents of the memory registers to the Z stack level. Also note that, following the 41Z convention, **ZSTO** will overwrite the Z level with the contents of X,Y if these were not the same. This allows walk-up complex data entering.

These functions are **fully programmable**. When in program mode (either running or SST execution), the index input is ignored, and their argument is taken from the following program line after the function. For this reason they are sometimes called *non-merged* functions. In fact, the number denoting the argument can have any combination of leading zeroes (like 001, 01, 1 all resulting in the same). Moreover, when the argument is zero then such index following line can be omitted if any non-numeric line follows the function. This saves bytes.

This implementation was written by W. Doug Wilder, and it is even more convenient than the one used by the HEPAX module for its own multi-function groups.

Similar to the real counterparts, keys on the first two rows can be used as **shortcut for indexes 1-9**. Note that **indirect addressing is also supported** (say **ZRCL IND \_ \_**) pressing the SHIFT key – *in RUN mode only* (i.e. not programmatically). In program mode you can make use of the fact that the **indirect addressing is nothing more than adding 128 to the address**, thus it can be handled by simply adding such factor to the index in the second program line.

Also note that despite being possible to invoke, their logic doesn't support the use of the stack registers. (**ZRCL ST \_ \_**); and certainly neither the combination of both, indirect and stack addressing (**ZRCL IND ST \_ \_**). If you use these, unpredicted (and wrong) results will occur. The same can be said if you press the arithmetic keys (+, -, \*, /): **simply don't**.

Lastly, a NONEXISTENT message will be shown if the storage register is not available in main memory. Registers can be made available using the SIZE function of the calculator.

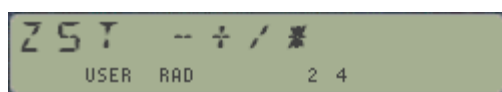
## 4.2. ZSTO Math function group.

<b>ZST+ _ _</b>	<b>Recall from Complex Register</b>		Prompting function
<b>ZST- _ _</b>	<b>Store in Complex Register</b>		Prompting function
<b>ZST* _ _</b>	<b>Recall from Complex Register</b>		Prompting function
<b>ZST/ _ _</b>	<b>Store in Complex Register</b>		Prompting function

The newest addition to the 41Z function set.- allow storage math in a concise format, saving bytes and programming steps in FOCAL programs. Their equivalence with standard functions would have to be done using four steps, and disturbing the Complex Stack as follows:

- 1.- ZENTER^,
- 2.- Z<>(nn)
- 3.- MATH (Z+, Z-, Z\*, Z/)
- 4.- Z<>(nn)

Functions are fully programmable using the non-merged technique. These functions can also be accessed using the Z-keyboard from its own dedicated launcher, pressing **[Z]**, **[A]** and then **[STO]**.



## 5. Complex Math.

Complex numbers are much more than a simple extension of the real numbers into two dimensions. The Complex Plane is a mathematical domain with well-defined, own properties and singularities, and it isn't in the scope of this manual to treat all its fundamental properties. On occasions there will be a short discussion for a few functions (notably the logarithms!), and some analogies will be made to their geometric equivalences, but it is assumed throughout this manual that the user has a good understanding of complex numbers and their properties.

### 5.1. Arithmetic and Simple Math.

Table-5.1:- Arithmetic functions.

Index	Function	Formula	Description
1	<b>Z+</b>	$Z=w+z$	Complex addition
2	<b>Z-</b>	$Z=w-z$	Complex subtraction
3	<b>Z*</b>	$Z=w*z$	Complex multiplication
4	<b>Z/</b>	$Z=w/z$	Complex division
5a	<b>ZINV</b>	$Z=1/z$	Complex inversion, direct formula
5b	<b>1/Z</b>	$Z=1/r e^{(-iArg)}$	Complex inversion, uses TOPOL
6	<b>ZDBL</b>	$z=2*z$	Removed. Replace with: 2, ST* Z, *
7	<b>ZHALF</b>	$z= z/2$	Removed. Replace with: 2, ST/ Z, /
8	<b>ZRND</b>	$Z=rounded(z)$	Rounds Z to display settings precision
9	<b>ZINT</b>	$Z=Int(z)$	Takes integer part for Re(z) and Im(z)
10	<b>ZFRC</b>	$Z=Frc(z)$	Takes fractional part for Re(z) and Im(z)
11	<b>ZPI*</b>	$Z=z\pi$	Simple multiplication by pi

Here's a description of the individual functions within this group.

<b>Z+</b>	<b>Complex addition</b>	$Z=w+z$	Does LastZ
<b>Z-</b>	<b>Complex subtraction</b>	$Z=w-z$	Does LastZ
<b>Z*</b>	<b>Complex multiplication</b>	$Z=w*z$	Does LastZ
<b>Z/</b>	<b>Complex division</b>	$Z=w/z$	Does LastZ

Complex arithmetic using the RPN scheme, with the first number stored in the **W** stack level and the second in the **Z** stack level. The result is stored in the **Z** level, the complex stack drops (duplicating **U** into **V**), and the previous contents of **Z** is saved in the LastZ register.

<b>ZINV</b>	<b>Direct Complex inversion</b>	$Z=1/z$	Does LastZ
<b>1/Z</b>	<b>Uses POLAR conversion</b>	$Z=1/r e^{(-iArg)}$	Does LastZ

Calculates the reciprocal of the complex number stored in **Z**. The result is saved in **Z** and the original argument saved in the LastZ register. Of these two the direct method is faster and of comparable accuracy – thus it's the preferred one, as well as the one used as subroutine for other functions.

This function would be equivalent to a particular case of **Z/**, where  $w=1+0j$ , and not using the stack level **W**. Note however that **Z/** implementation is not based on the **ZINV** algorithm [that is, making use of the fact that :  $w/z = w * (1/z)$ ], but based directly on the real and imaginary parts of both arguments.

**Example.** Calculate  $z/z$  using **ZINV** for  $z=i$

We'll use the direct data entry, starting w/ the imaginary part:

1, ENTER^, 0, <b>ZINV</b>	-> 0-j1
<b>LASTZ</b>	-> 0+j1
<b>Z*</b>	-> 1+j0

Note that *integer numbers are displayed without decimal zeroes*, simplifying the visual display of the complex numbers.

<b>ZDBL</b>	<b>Doubles Z</b>	$Z=2*z$	Does LastZ
<b>ZHALF</b>	<b>Halves Z</b>	$Z=z/2$	Does LastZ

These two functions are provided to save stack level usage and programming efficiency. The same result can also be accomplished using their generic forms (like **Z\*** and **Z/**, with  $w=2+0j$ ), but the shortcuts are faster and simpler to use.

**Example.** Taken from the HP-41 Advantage manual, page 97.

Calculate:  $z_1/(z_2+z_3)$ ; for:  $z_1=(23+13i)$ ;  $z_2=(-2+i)$ , and  $z_3=(4-3i)$

If the complex stack were limited to 2 levels deep, we would need to calculate the inverse of the denominator and multiply it by the numerator, but using the 4-level deep complex stack there's no need to resort to that workaround. We can do as follows:

13, ENTER, 23, <b>ZENTER^</b>	-> 23+j13
1, ENTER^, 2, CHS, <b>ZENTER^</b>	-> -2+j1
3, CHS, ENTER^, 4, <b>Z+</b>	-> 2(1-j)
<b>Z/</b>	-> 2,500+j9

Note that 41Z *automatically takes common factor when appropriate*, and that integer numbers are displayed without decimal zeroes to simplify the visual display of the complex numbers. Non-integers are displayed using the current decimal settings, but of course full precision (that is 9 decimal places) is always used for the calculations (except in the rounding functions).

<b>ZRND</b>	<b>Rounds Complex number</b>	$Z=\text{Rounded}(z)$	Does LastZ
<b>ZINT</b>	<b>Takes integer parts</b>	$Z=\text{Int}[\text{Re}(z)]+j\text{Int}[\text{Im}(z)]$	Does LastZ
<b>ZFRC</b>	<b>Takes Fractional parts</b>	$Z=\text{Frc}[\text{Re}(z)]+j\text{Frc}[\text{Im}(z)]$	Does LastZ

These functions will round, take integer part or fractional part both the real and imaginary parts of the complex number in **Z**. The rounding is done according to the current decimal places specified by the display settings.

<b>ZPI*</b>	<b>Multiplies by pi</b>	$Z=\pi*z$	Does LastZ
-------------	-------------------------	-----------	------------

Simple multiplication by pi, used as a shortcut in the Bessel FOCAL programs. Has better accuracy than the FOCAL method, as it used internal 13-digit math.

## 5.2. Exponential and powers that be.

Table-5.2: Exponential group.

Index	Function	Formula	Description
1a	<b>ZEXP</b>	$Z = \text{REC}(e^x, y)$	Complex exponential (method one)
1b	<b>e^Z</b>	See below	Complex Exponential (method two)
2	<b>Z^2</b>	$Z = \text{REC}(r^2, 2\alpha)$	Complex square
3a	<b>ZSQRT</b>	Algebraic Formula	Principal value of complex square root
3b	<b>SQRTZ</b>	$Z = \text{REC}(r^{1/2}, \alpha/2)$	Principal value of complex square root
4	<b>W^Z</b>	$Z = e^z * \text{Ln}(w)$	Complex to complex Power
5	<b>W^1/Z</b>	$Z = e^{1/z} * \text{Ln}(w)$	Complex to reciprocal complex Power
6	<b>X^Z</b>	$Z = e^z * \text{Ln}(x)$	Real to complex power
7	<b>X^1/Z</b>	$Z = e^{1/z} * \text{Ln}(x)$	Real to reciprocal complex power
8	<b>Z^X</b>	$Z = e^x * \text{Ln}(z)$	Complex to real Power
9	<b>Z^1/X</b>	$Z = e^{1/x} * \text{Ln}(z)$	Complex to reciprocal real Power
10	<b>ZALOG</b>	$Z = e^z * \text{Ln}(10)$	Complex decimal power
11	<b>NXTRTN</b>	$Z = z * e^{j 2\pi/N}$	Next value of complex nth. Root

Looking at the above formula table it's easy to realize the importance of the exponential and logarithmic functions, as they are used to derive many of the other functions in the 41Z module. It is therefore important to define them properly and implement them in an efficient way.

The 41Z module includes two different ways to calculate the complex exponential function. The first one is based on the trigonometric expressions, and the second one uses the built-in polar to rectangular routines, which have enough precision in the majority of practical cases. The first method is slightly more precise but takes longer computation time.

<b>ZEXP</b>	<b>Complex Exponential</b>	$Z = \text{REC}(e^x, y)$	Does LastZ
<b>e^Z</b>	<b>Complex Exponential</b>	Trigonometric	Does LastZ

One could have used the rectangular expressions to calculate the result, as follows:

$$e^z = e^x * (\cos y + i \sin y), \text{ thus: } \text{Re}(z) = e^x * \cos y ; \text{ and: } \text{Im}(z) = e^x * \sin y$$

and this is how the function **e^Z** has been programmed. It is however more efficient (albeit slightly less precise) to work in polar form, as follows:

$$\text{since } z = x + iy, \text{ then } e^z = e^{(x+iy)} = e^x * e^{iy},$$

and to calculate the final result we only need to convert the above number to rectangular form.

**Example.-** Calculate  $\exp(z^2)$ , for  $z = (1+i)$

1, ENTER^, **ZENTER^**                      -> 1(1+j)  
 2, CHS, **Z^X**                                -> 0-j0,500  
**ZEXP**     -> 0,878-j0,479

Another method using **W^Z** and the complex keypad function (**ZREAL^**):

1, ENTER^, **ZENTER^**                      -> 1(1+j)  
 2, CHS, **ZREAL^**                           -> -2+j0  
**W^Z, ZEXP**                                -> 0,878-j0,479

or alternatively, this shorter and more efficient way: (leaves **W** undisturbed)

1, ENTER<sup>^</sup>, **Z<sup>2</sup>**, **ZINV**, **ZEXP** -> 0,878-j0,479

Note how this last method doesn't require using **ZENTER<sup>^</sup>** to terminate the data input sequence, as the execution of monadic functions will automatically synchronize the complex stack level Z with the contents of the real X,Y registers.

<b>Z<sup>2</sup></b>	<b>Complex square</b>	$Z = \text{REC}(r^2, 2\alpha)$	Does LastZ
<b>ZSQRT</b>	<b>Complex square root</b>	Algebraic Formula	Does LastZ
<b>SQRTZ</b>	<b>Complex square root</b>	$Z = \text{REC}(r^{1/2}, \alpha/2)$	Does LastZ

Two particular cases also where working in polar form yields more effective handling. Consider that:

$$Z^2 = |z|^2 * e^{2i\alpha}, \text{ and:}$$

$$\text{Sqrt}(z) = z^{1/2} = \text{Sqrt}(|z|) * e^{i\alpha}, \text{ where } \alpha = \text{Arg}(z),$$

It is then simpler first converting the complex number to its polar form, and then apply the individual operations upon its constituents, followed by a final conversion back to the rectangular form.

Note that this implementation of **ZSQRT** only offers one of the two existing values for the square root of a given complex number. The other value is easily obtained as its opposite, thus the sum of both square roots is always zero.

Such isn't exclusive to complex arguments, for the same occurs in the real domain – where there are always 2 values,  $x_1$  and  $-x_1$ , that satisfy the equation  $\text{SQRT}[(x_1)^2]$ .

As with other multi-valued functions, the returned value is called the *principal value* of the function. See section 6 ahead for a more extensive treatment of this problem.

<b>W<sup>Z</sup></b>	<b>Complex to complex Power</b>	$Z = e^{[z * \text{Ln}(w)]}$	Does LastZ
<b>W<sup>1/Z</sup></b>	<b>Complex to reciprocal Power</b>	$Z = e^{[\text{Ln}(w)/z]}$	Does LastZ

The most generic form of all power functions, calculated using the expressions:

$$w^z = \exp[z * \text{Ln}(w)], \text{ and}$$

$$w^{1/z} = \exp[\text{Ln}(w) / z]$$

The second function is a more convenient way to handle the reciprocal power, but it's obviously identical to the combination **ZINV**, **W<sup>Z</sup>**.

**Example:** calculate the inverse of the complex number 1+2i using **W<sup>Z</sup>**:- Then obtain its reciprocal using **ZINV** to verify the calculations.

2, ENTER <sup>^</sup> , 1, <b>ZENTER<sup>^</sup></b>	number stored in level <b>W</b> (also as: 1, ENTER <sup>^</sup> , 2, <b>ZTRP</b> )
0, ENTER <sup>^</sup> , -1	exponent -1 stored in level <b>Z</b> (also as: -1, ENTER <sup>^</sup> , 0, <b>ZTRP</b> )
<b>W<sup>Z</sup></b>	result: 0,200-j0,400
<b>ZINV</b>	result: 1,000+j2

Note that the final result isn't exact – as the decimal zeroes in the real part indicate there's a loss of precision in the calculations.

<b>Z^X</b>	<b>Complex to real power</b>	$Z=e^{[x*\ln(z)]}$	Does LastZ
<b>Z^1/X</b>	<b>Complex to reciprocal real</b>	$Z=e^{[\ln(z)/x]}$	Does LastZ
<b>X^Z</b>	<b>Real to complex power</b>	$Z=e^{[z*\ln(x)]}$	Does LastZ
<b>X^1/Z</b>	<b>Real to reciprocal complex</b>	$Z=e^{[1/z*\ln(x)]}$	Does LastZ
<b>ZALOG</b>	<b>10 to complex power</b>	$Z=e^{[z*\ln(10)]}$	Does LastZ

These five functions are calculated as particular examples of the generic case  $W^Z$ . Their advantage is a faster data entry (not requiring inputting the zero value) and a better accuracy in the results

**Z^1/X** is identical to:  $1/X$ , **Z^X**

**X^1/Z** is identical to: **RDN**, **ZINV**,  $R^{\cdot}$ , **X^Z**

Data entry is different for hybrid functions, with mixed complex and real arguments. As a rule, the second argument is stored into its corresponding stack register, as follows:

- $x$  into the real stack register **X** for **Z^X** and **Z^1/X**
- $z$  into the complex stack register **Z** for **X^Z** and **X^1/Z**

The first argument needs to be input first, since this is an RPN implementation.

Because **ZALOG** is a monadic function, it expects  $z$  in the stack level **Z**, and thus it doesn't disturb the complex stack.

**Example:** Calculate  $(1+2i)^3$  and  $3^{(1+2i)}$

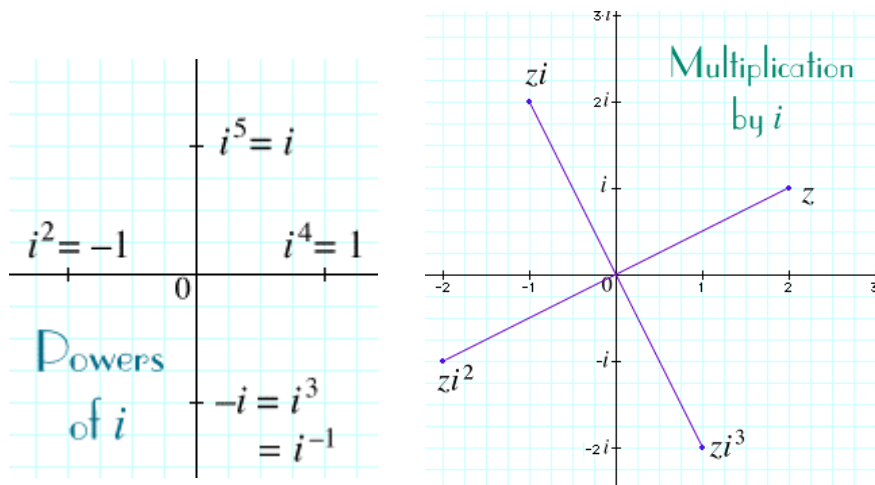
2, ENTER^, 1, **ZENTER^**, 3, **Z^X**

results:  $(1+2i)^3 = -11-2j$

2, ENTER^, 1, **ZENTER^**, 3, **X^Z**

results:  $3^{(1+2i)} = 1+0j$

**Example:** Verify the powers of the imaginary unit, as per the picture below.- You can use either **Z^X**, with  $z=(0+i)$  and  $x=1,2,3,4,5$ ; or alternatively **W^Z**, with  $w=(0+i)$  and  $z=(1+0i), (2+0i), (3+0i)$ , etc.



This keystroke sequence will quickly address the even powers:

0, ENTER^, 1, **ZTRP**

->  $0 + j1$       $i$

**Z^2**

->  $-1 + j0$       $i^2 = -1$

**Z^2**

->  $1 + j0$       $i^4 = 1$



Whilst this will take care of the rest (and also in general):

0, ENTER^, 1, <b>ZTRP</b>	-> 0 + j1	i
3, <b>Z^X</b>	-> 0 - j1	i <sup>3</sup> = -i
<b>LASTZ</b>	-> 0 + j1	
5, <b>Z^X</b>	-> 0 + j1	i <sup>5</sup> = i

Note in this example that for enhanced usability **Z^X** stores the original argument in the LastZ register, even though it wasn't strictly located in the **Z** level of the complex stack. The same behavior is implemented in **X^Z**.

Alternatively, using **W^Z** and **ZREPL**:

1, ENTER^, 0, <b>ZREPL</b>	-> 0 + j1	i
0, ENTER^, 2, <b>W^Z</b>	-> -1 + j0	i <sup>2</sup> = -1
<b>ZRDN</b>	-> 0 + j1	i
0, ENTER^, 3, <b>W^Z</b>	-> 0 - j1	i <sup>3</sup> = -i
<b>ZRDN</b>	-> 0 + j1	i
0, ENTER^, 4, <b>W^Z</b>	-> 1 + j0	i <sup>4</sup> = 1
<b>ZRDN</b>	-> 0 + j1	i
0, ENTER^, 5, <b>W^Z</b>	-> 0 + j1	i <sup>5</sup> = i

**Examples-** Calculate the value of:  $z = 2^{1/(1+i)}$ ; and  $z=(1+i)^{1/2}$

These two have a very similar key sequence, but they have different meaning:

Solution: 1, ENTER^, ENTER^, 2, <b>X^1/Z</b>	-> 1,330 – j0,480
Solution: 1, ENTER^, ENTER^, 2, <b>Z^1/X</b>	-> 1,099 + j0,455

<b>NXTNRT</b>	<b>Next value of Nth. Root</b>	$Z=z0 \cdot e^{j 2\pi/N}$	$z0$ is the principal value
---------------	--------------------------------	---------------------------	-----------------------------

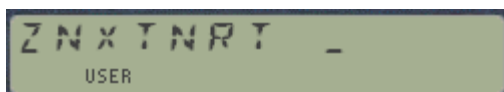
In its general form, the solution to the Nth. Root in the complex plane admits multiple solutions. This is because of its logarithmic nature, since the logarithm is a multi-valued function (see discussion in next section).

$$Z^{1/N} = e^{[\ln(z)/N]} = e^{[\ln(|z|)+i(\alpha+2\pi)]/N} = e^{[\ln(|z|)+i\alpha]/N} * e^{j 2\pi/N}$$

From this we derive the general expression: **Next(z<sup>1/N</sup>) = z<sup>1/N</sup> \* e<sup>(j 2 π /N)</sup>**

thus there are N different Nth. Roots, all separated by (2π over N). See the geometric interpretation on section 7 ahead for further discussion on this.

When executed in a program or RUN mode, data entry for this function expects N in the X register, and **z** in the **Z** complex stack level. However when the Complex Keyboard shortcut is used, the index N is prompted as part of the entry sequence – a much more convenient way.



Shortcut: Z, Z, SHIFT, SQRT

**Example:** - Calculate the two square roots of 1.

0, ENTER^, 1, **ZENTER^**, 2, **Z^1/X**                   -> 1+j0  
2, **NXTNRT** (plus **ZRND**)                               -> -1+j0

Note that the previous root is temporarily stored in the LastZ register:  
**LASTZ**   -> 1+j0 (previous root)

See section 9 for a general application program to calculate the n different Nth. Roots of a complex number

**Example:** - Calculate the three cubic roots of 8.

Using "direct" data entering: [Im(z), ENTER^, Re(z)]

0, ENTER^, 8, **ZENTER^**, 3, **Z^1/X**                   -> 2+j0  
3, **NXTNRT**   -> -1,000+j1,732  
3, **NXTNRT**   -> -1,000-j1,732

Note: for this example use the *Complex Keyboard* **ZKBRD** to execute **NXTNRT**, as follows:

**Z**, **Z**, SHIFT, SQRT, and then input 3 at the last prompt.

**Example:** Calculate both quadratic roots of 1+2i.

2, ENTER^, 1, **ZSQRT**                   gives the first root:     $z = 1,272+0,786 j$   
2, **NXTNRT**                               gives the second root:  $z = -1,272-0,786 j$   
2, **NXTNRT**                               reverts to the first, principal value, of the root.

This verifies that both roots are in fact on the same straight line, separated 180 degrees from each other and with the same module.

**Example:** Calculate the three cubic roots of 1+2i.

2, ENTER^, 1, **ZENTER^**               inputs z in the complex stack level **Z**  
3, 1/X, **Z^X**                           gives the main root:     $z = 1,220+0,472 j$   
3, **NXTNRT**                               gives the second root:  $z = -1,018+0,82 j$   
3, **NXTNRT**                               give the third and last:  $z = -0,201-1,292 j$

In the next section we'll discuss the logarithm in the complex plane, a very insightful and indeed interesting case study of the multi-valued functions.

### 5.3. Complex Logarithm.

Table-x: Logarithm group.

Index	Function	Formula	Description
1	<b>ZLN</b>	$Z = \text{Ln} z  + i\alpha$	Principal value of natural logarithm
2	<b>ZLOG</b>	$Z = \text{Ln}(z) / \text{Ln}10$	Principal value of decimal logarithm
3	<b>ZWLOG</b>	$Z = \text{Ln}(z) / \text{Ln}(w)$	Base-w logarithm of z
4	<b>NXTLN</b>	$Z = z + 2\pi j$	Next value of natural logarithm

The first thing to say is that a rigorous definition of the logarithm in the complex plane requires that its domain be restricted, for if we defined it valid in all the plane, such function wouldn't be continuous, and thus neither *holomorphic* (or expressible as series of power functions).

This can be seen intuitively if we consider that:

Since:  $z = |z| \cdot e^{i\rho}$ , then:

$$\text{Ln } z = \text{Ln } |z| + \text{Ln } (e^{i\rho}) = \text{Ln}(|z|) + i\rho$$

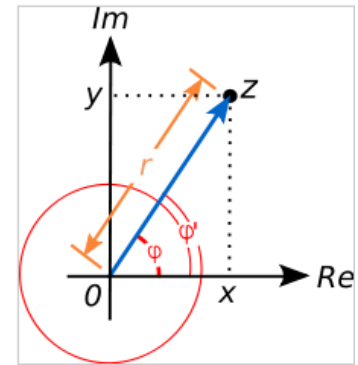
But also

$$z = |z| \cdot e^{i(\rho+2\pi)} = |z| \cdot e^{i(\rho+4\pi)} = \dots = |z| \cdot e^{i(\rho+2\pi n)}$$

Then we'd equally have multiple values of its logarithm, as follows:

$$\text{Ln}(z) = \text{Ln}(|z|) + i\rho = \text{Ln}(|z|) + i(\rho+2\pi) = \dots \quad \text{Or generally:}$$

$$\text{Ln } z = \text{Ln}|z| + i(\rho+2\pi n); \quad \text{where } n \text{ is a natural number.}$$

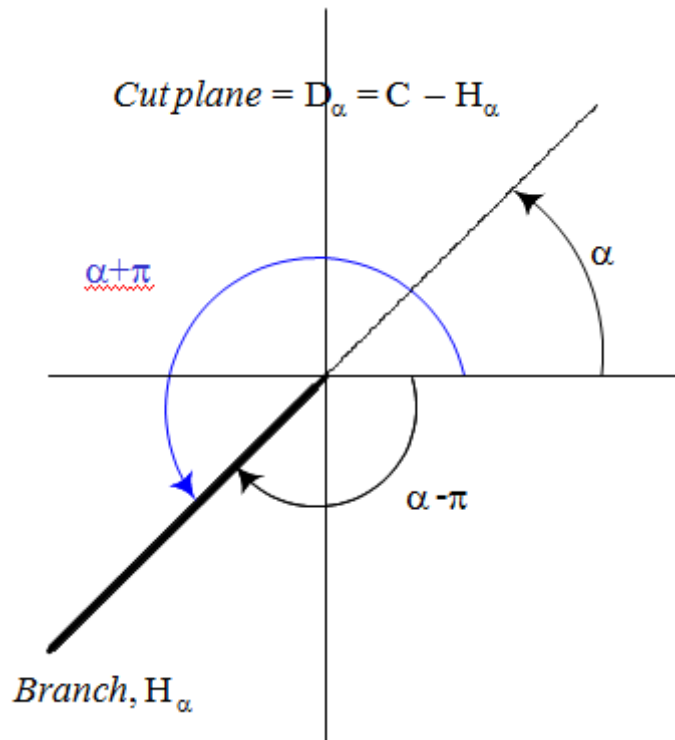


To deal with this multi-valued nature of the function, mathematicians define the different **branches of the complex logarithm**, -  $\log_\alpha$  - as the single one and only logarithm which argument is comprised between  $(\alpha - \pi)$  and  $(\alpha + \pi)$ , thus within the open interval  $] \alpha - \pi, \alpha + \pi [$

Its domain isn't the whole complex plane, but it excludes a semi-straight line, centered at the origin, that forms an angle  $\alpha$  with the real axis, as shown in the picture. Such set is called the "**torn**" or **cut complex plane at angle  $\alpha$** ". Thus the principal value of the logarithm really should be called  $\text{Log}_0$ , as it tears (or cuts) the complex plane by the real negative semi-axis, or otherwise  $\alpha = 0$ . This means it is *NOT defined* for any negative numbers, and when those need to be subject of its application, a different cut should be chosen.

Therefore all arguments should be comprised between 180 and -180 degrees, as it would correspond to this definition of " $\text{Log}_0$ ".

In practicality, the values calculated by **ZLN** always lie within this interval, since they use the internal routines of the calculator, [TOPOL] and [TOREC].



The angle  $\alpha$  should not be confused with the base of the logarithm, which is always the number  $e$  – that is, there are natural logarithms.

(See [http://en.wikipedia.org/wiki/Branch\\_point](http://en.wikipedia.org/wiki/Branch_point) for a more rigorous description of this subject).

After this theoretical discussion, let's see the functions from the 41Z module:-

<b>ZLN</b>	<b>Natural logarithm</b>	$Z = \text{Ln} z  + i\alpha$	Does LastZ
------------	--------------------------	------------------------------	------------

Calculates the principal value of the natural logarithm, using the expression:

$$\text{Ln } z = \text{Ln}|z| + i\gamma, \quad \text{where } \gamma = \text{Arg}(z) \text{ belongs to } ]-\pi, \pi]$$

**Example:** check that:  $z = \text{Ln}(e^z)$ , for  $z = (1+i)$  and  $z = (2+4i)$

1, ENTER^, **ZEXP, ZLN**      -> 1,000+j1,000  
 4, ENTER^, 2, **ZEXP, ZLN**      -> 2-j2,283

How do you explain the last result? Is it correct? Try executing **NXTLN** (see below) on it...

**NXTLN**      -> 2+j4,000    - that's more like it!

<b>ZLOG</b>	<b>Decimal logarithm</b>	$Z = \text{Ln}(z) / \text{Ln}10$	Does LastZ
-------------	--------------------------	----------------------------------	------------

Calculates the principal value of the decimal logarithm using the expression:

$$\text{Log } z = \text{Ln } z / \text{Ln}(10)$$

**Example:** check that:  $z = \text{Log}(10^z)$ , for  $z = (1+i)$  and  $z = (2+4i)$

1, ENTER^, **ZALOG, ZLOG**      -> 1(1+j)  
 4, ENTER^, 2, **ZALOG, ZLOG**      -> 2+j1,271

How do you explain the last result? Is it correct? Have you found a bug on the 41Z?

<b>ZWLOG</b>	<b>Base-W Logarithm</b>	$Z = \text{Ln}(z) / \text{Ln}(w)$	Does LastZ
--------------	-------------------------	-----------------------------------	------------

General case of ZLOG, which has  $w=10$ . This is a dual function,

$$\text{Log } z = \text{Ln } z / \text{Ln } w$$

<b>NXTLN</b>	<b>Next Natural logarithm</b>	$Z = z_0 + 2\pi j$	$z_0$ is the principal value
--------------	-------------------------------	--------------------	------------------------------

Calculates the next value of the natural logarithm, using the expression:

$$\text{Next}(\text{Ln } z) = \text{Ln}(z) + 2\pi j$$

So the different logarithms are "separated"  $2\pi$  in their imaginary parts. This works both "going up" as well as "going down", thus each time **NXTLN** is executed two values are calculated and placed in complex levels Z and W. You can use **Z<>W** to see them both.

## 6. Complex geometry.

The next set of functions admits a geometrical interpretation for their results. Perhaps one of the earliest ways to approach the complex numbers was with the analogy where the real and imaginary parts are equivalent to the two coordinates in a geometric plane.

Table-6.1: Complex geometric group.

Index	Function	Formula	Description
1	<b>ZMOD</b>	$ z  = \text{SQR}(x^2 + y^2)$	Module or magnitude of a complex number
2	<b>ZARG</b>	$\alpha = \text{ATAN}(y/x)$	Phase or angle of a complex number
3 <sup>a</sup>	<b>ZNEG</b>	$Z = -z$	Opposite of a complex number
3b	<b>ZCHSX</b>	$Z = (-1)^x * z$	Opposite (by X) of a complex number
4	<b>ZCONJ</b>	$Z = x - yj$	Conjugated of a complex number
5	<b>ZSIGN</b>	$Z = z/ z $	Sign of a complex number
6	<b>ZNORM</b>	$Z =  z ^2$	Norm of a complex number
7	<b>Z*I</b>	$Z = z * i$	Rotates z 90 degrees counter clockwise
8	<b>Z/I</b>	$Z = z/i$	Rotates z 90 degrees clockwise

In fact, various complex operations admit a geometrical interpretation. An excellent reference source for this can be found at the following URL: <http://www.clarku.edu/~djoyce/complex>.

Let's see the functions in detail.

<b>ZMOD</b>	<b>Module of z</b>	$ z  = \text{SQR}(x^2 + y^2)$	Does LastZ
<b>ZARG</b>	<b>Argument of z</b>	$\alpha = \text{ATAN}(y/x)$	Does LastZ

This pair of functions calculates the module (or magnitude) and the argument (or angle) of a complex number, given by the well-known expressions:

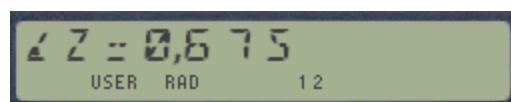
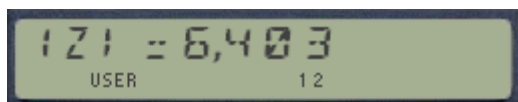
$$|z| = \text{SQR}(x^2 + y^2)$$

$$\alpha = \text{ATAN}(y/x)$$

Since they use the internal [TOPOL] routine (like R-P does), the argument will always be given between 180 and -180 degrees (or equivalent in the selected angular mode).

The result is saved in the complex **Z** register, and the real X,Y stack levels – as a complex number with zero imaginary part. The original complex number is stored in the Last**Z** register. The other complex stack levels **W**, **V**, **U** aren't disturbed.

These functions display a meaningful description when used in run mode, as can be seen in the pictures below, for  $z = 5 + 4j$  and RAD mode.

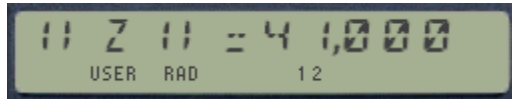


<b>ZNORM</b>	<b>Norm of z</b>	$  Z   =  z ^2$	Does LastZ
--------------	------------------	-----------------	------------

This function calculates the norm of a complex number, also known as the square of its module"

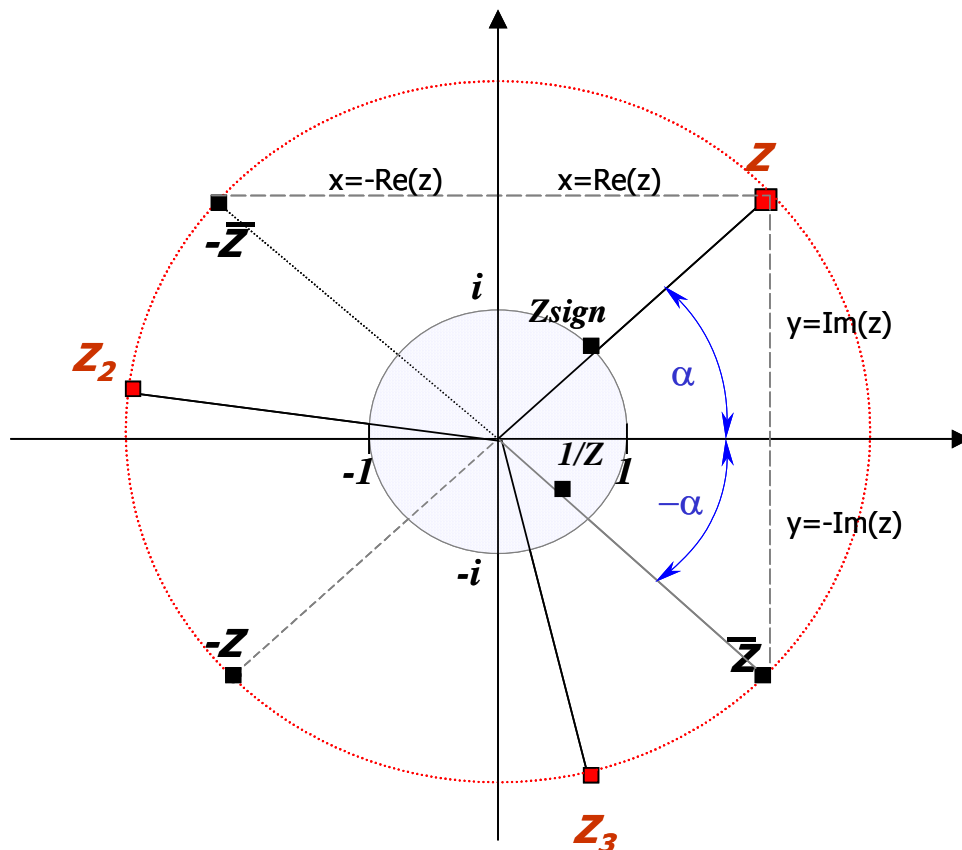
$$||z|| = |z|^2; \text{ thus: } Z_{\text{norm}} = x^2 + y^2$$

When executed in run mode, the display shows a meaningful representation for it, like in the example below, also for  $z = 4 + 5j$  :



<b>ZSIGN</b>	<b>Module of z</b>	$Z = z/ z $	Does LastZ
--------------	--------------------	-------------	------------

This function calculates the sign of a complex number. As an extension to the SIGN function for the real domain, it is a complex number with magnitude of one (i.e. located on the unit circle), that indicates the direction of the given original number. Thus obviously:  $Z_{\text{sign}} = z / |z|$



The figure above shows the unit circle and the relative position in the complex plane for the opposite ( $-z$ ), conjugate ( $z^*$ ), and opposite conjugate ( $-z^*$ ) of a given number  $z$ . Note that the inverse of  $z$  ( $1/z$ ) will be located inside of the unit circle, and over the direction defined by the negative of its argument  $[-\text{Arg}(z)]$

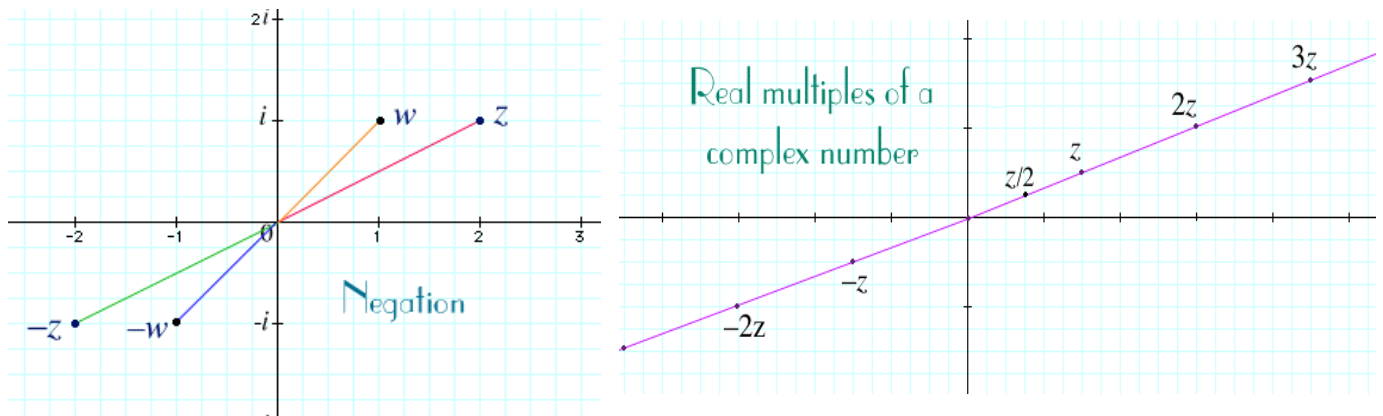
Note that if  $z$  happens to be a cubic root of another number (i.e.  $z^3$ ), then the other two roots ( $z_2$  and  $z_3$ ) will have the same module and be located at 120 degrees from each other, on the red circle line.

<b>ZNEG</b>	<b>Opposite of z</b>	$Z = -z$	Does LastZ
<b>ZCHSX</b>	<b>Opposite of z by X</b>	$Z = (-1)^x * z$	Does LastZ
<b>ZCONJ</b>	<b>Conjugate of z</b>	$Z = x - yj$	Does LastZ

This pair of functions calculate the opposite- or the multiple-opposite by  $(-1)^x$  – and the conjugate of a complex number  $z = x + yi$ , as follows:

$$-z = -x - yi, \text{ and } z^* = x - yi$$

See the figure below for the geometric interpretation of **ZNEG** and multiplication by real numbers:



<b>Z*I</b>	<b>Multiply by i</b>	$Z = z * i$	Rotates z 90 deg ccw
<b>Z/I</b>	<b>Divide by i</b>	$Z = z / i$	Rotates z 90 deg cw

The main role of these two functions is as subroutines for the trigonometric set, and they are also provided for completion sake. Their geometric interpretation is a 90 degrees rotation of the complex number either clockwise or counter-clockwise respectively.

These functions are used as subroutines for several others, like the direct and inverse trigonometric. The dependencies between hyperbolic and trigonometric ultimately involves multiplication by  $i$ , which is really a matter of swapping the real and imaginary parts, with the appropriate sign change in each case.



## 6.2 Complex Comparisons.

The 41Z module includes a comprehensive set of comparison checks, based on the complex numbers themselves and their modules (for relative position in the complex plane). Checks for purely real or imaginary cases are also provided. The main utilization for these functions is in program mode, as conditional decisions under program control based on the different values.

Table 6.2. Complex comparisons function group.

Index	Function	Formula	Description
1	<b>Z=0?</b>	Is $z=0$ ?	Checks if $z$ is zero
2	<b>Z#0?</b>	Is $z \neq 0$ ?	Checks if $z$ is not zero
3	<b>Z=I?</b>	Is $z=i$ ?	Checks if $z$ is the imaginary unit
4	<b>Z=W?</b>	Is $z=w$ ?	Checks if $z$ and $w$ are the same
5	<b>Z=WR?</b>	Is $z=w$ rounded?	Checks if rounded $z$ and rounded $w$ are the same
6	<b>Z#W?</b>	Is $z \neq w$ ?	Checks if $z$ and $w$ are different
7	<b>ZUNIT?</b>	Is $ z =1$ ?	Checks if $z$ is on the unit circle
8	<b>ZIN?</b>	Is $ z <1$ ?	Checks whether $z$ is inside the unit circle
9	<b>ZOUT?</b>	Is $ z >1$ ?	Checks whether $z$ is outside the unit circle
10	<b>ZREAL?</b>	Is $z$ a real number?	Checks whether $\text{Im}(z)=0$
11	<b>ZIMAG?</b>	Is $z$ true imaginary?	Checks whether $\text{Re}(z)=0$
12	<b>ZINT?</b>	Is $z$ true integer?	Checks whether $\text{Im}(z)=0$ and $\text{FRC}[\text{Re}(z)]=0$

It's well known that, contrary to real numbers, the complex plane isn't an ordered domain. Thus we can't establish ordered relationships between two complex numbers like they are done with real ones (like  $x>y$ ,  $x<y$ ?, etc.).

There are however a few important cases that can also be used with complex numbers, as defined by the following functions.- As it is standard, they respond to the "do if true" logic, skipping the next program line when false.

<b>Z=W?</b>	<b>Compares z with w</b>	Are they equal?	
<b>Z#W?</b>	<b>Compares z with w</b>	Are they different?	
<b>Z=WR?</b>	<b>Compares z with w rounded</b>	Are they equal?	
<b>Z=0?</b>	<b>Compares z with zero</b>	Are they equal?	
<b>Z#0?</b>	<b>Compares z with zero</b>	Are they different?	
<b>Z=I?</b>	<b>Compares z with i</b>	Are they equal?	

The first two functions compare the contents of the **Z** and **W** stack levels, checking for equal values of both the real and imaginary parts.

$$Z=w \text{ iff } \text{Re}(z)=\text{Re}(w) \text{ and } \text{Im}(z)=\text{Im}(w)$$

The third function, **Z=WR?** Will establish the comparison *on the rounded values of the four real numbers*, according to the current display settings on the calculator (i.e. number of decimal places shown). This is useful when programming iterative calculations involving conditional decisions.

$$\text{Rnd}(z) = \text{Rnd}(w) \text{ iff } \text{rnd}[\text{Re}(z)]=\text{rnd}[\text{Re}(w)] \text{ and } \text{rnd}[\text{Im}(z)] = \text{rnd}[\text{Im}(w)]$$

The remaining three functions on the table are particular applications of the general cases, checking whether the **Z** complex stack level contains zero or the imaginary unit:

$z=0$  iff  $\text{Re}(z)=0$  and  $\text{Im}(z)=0$

$z=i$  iff  $\text{Re}(z)=0$  and  $\text{Im}(z)=1$

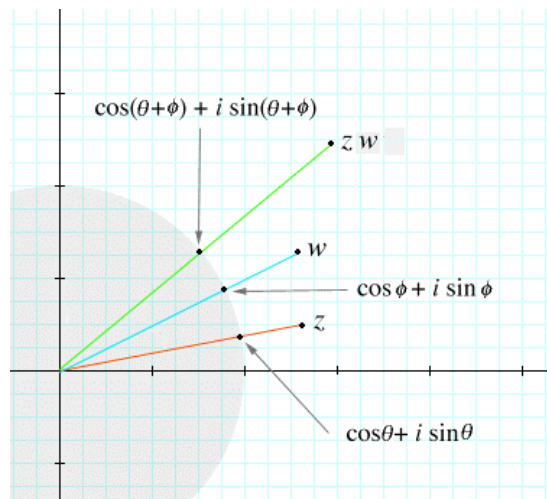
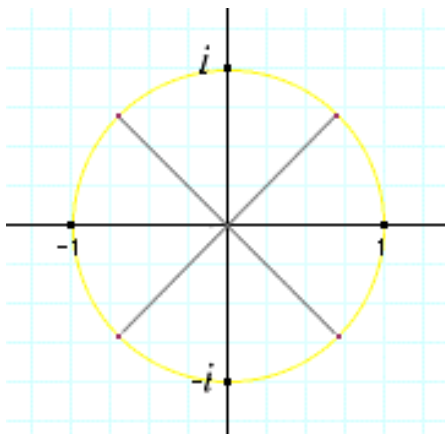
Some of the inverse comparisons can be made by using standard functions, as follows:

- use **X#0?** To check for  $Z \neq 0$ ? Condition
- Use **X#0?** To check for  $Z \neq i$ ? Condition

<b>ZUNIT?</b>	Checks if $z$ is on the unit circle		
<b>ZIN?</b>	Checks if $ z  < 1$		
<b>ZOUT?</b>	Checks if $ z  > 1$		

These three functions base the comparison on the actual location of the complex number referred to the unit circle: inside of it, on it, or outside of it. The comparison is done using the number's modulus,

Unit Circle

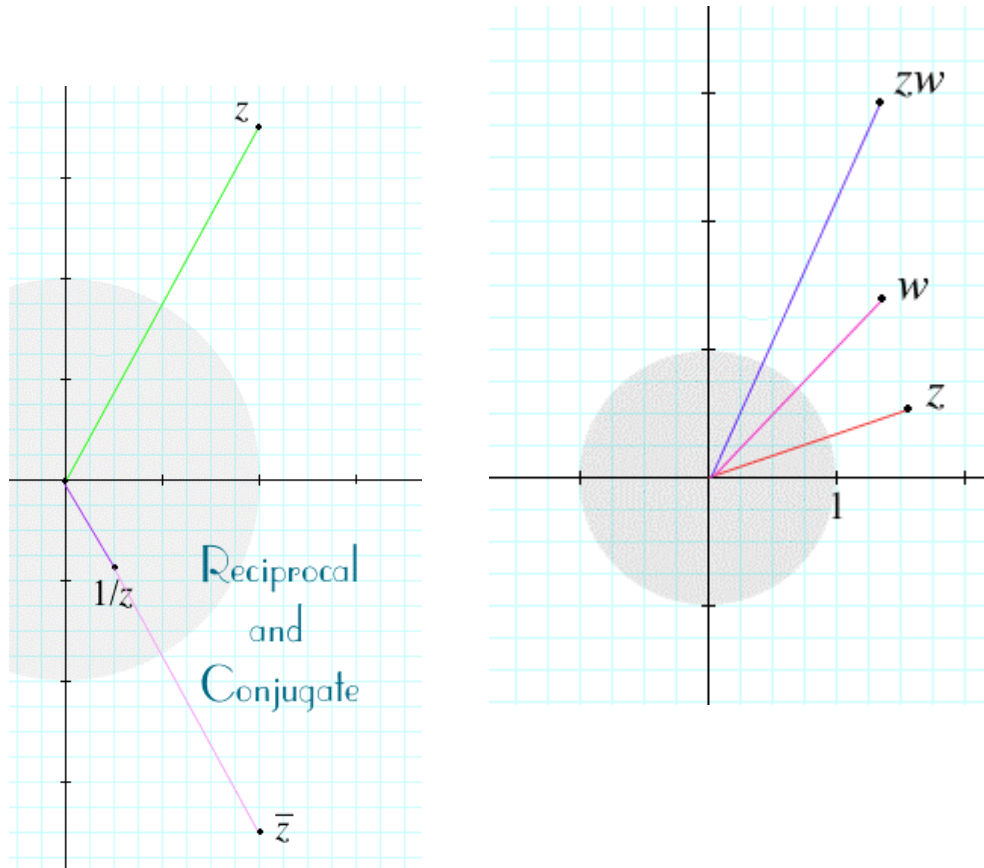


as a measure of the distance between the number and the origin.

**Example:** For  $z=4+5j$ , calculate its sign and verify that it's located on the unit circle:

5, ENTER^, 4, **ZSIGN**, → result:  $Z_{\text{sign}} = 0,62+0,78j$   
**ZUNIT?** → result: "YES"  
 DEG, **POLAR** → result:  $1,00 < 51,34$  (in degrees)

In program mode the behavior is ruled by the "do if true" rule, skipping the next line if false.



<b>ZREAL?</b>	Checks if $z$ is purely real		
<b>ZIMAG?</b>	Checks if $z$ is purely imaginary		
<b>ZINT?</b>	Checks if $z$ is an integer		

The first two functions check whether the complex number is purely a real or imaginary number.

Do not mistake these comparison functions with the other pair, **{ZREAL and ZIMAG}**, which cause the number to change to become either real or imaginary – nor with **{ZREAL^ and ZIMAG^}**, which are used to input complex numbers of the selected type based on the value stored in the real stack level X.

The third one extends the scope of ZREAL?, adding the condition of being a true integer number:

- **ZINT?** True means **ZREAL?** True, and  $\text{FRC}(\text{Re}(z))=0$

Do not mistake it with **ZINT**, which causes the complex number to have no decimal figures in BOTH its real and imaginary parts – *therefore it's result not a Real number!*

**ZINT?** Is used in the FOCAL programs to calculate Bessel Function, as a quick and effective way to determine if the order is integer – which triggers different expressions for the formulas.

Like it occurs with any built-in comparison function, there's no action taken on the original number, which will remain unchanged.

## 7. Complex Trigonometry.

Table 7.1. Complex trigonometry function group.

Index	Function	Formula	Description
1	<b>ZSIN</b>	$\sin z = -i * \sinh(iz)$	Complex Sine
2	<b>ZCOS</b>	$\cos z = \cosh(iz)$	Complex Cosine
3	<b>ZTAN</b>	$\tan z = -i * \tanh(iz)$	Complex Tangent
4	<b>ZHSIN</b>	$\sinh z = 1/2 * [e^z - e^{-z}]$	Complex Hyperbolic Sine
5	<b>ZHCOS</b>	$\cosh z = 1/2 * [e^z + e^{-z}]$	Complex Hyperbolic Cosine
6	<b>ZHTAN</b>	$\tanh z = (e^z - e^{-z}) / (e^z + e^{-z})$	Complex Hyperbolic Tangent

And their inverses:

7	<b>ZASIN</b>	$\operatorname{asin} z = -i * \operatorname{asinh}(iz)$	Complex Inverse Sine
8	<b>ZACOS</b>	$\operatorname{acos} z = \pi/2 - \operatorname{asin} z$	Complex inverse Cosine
9	<b>ZATAN</b>	$\operatorname{atan} z = -i * \operatorname{atanh}(iz)$	Complex Inverse Tangent
10	<b>ZHASIN</b>	$\operatorname{asinh} z = \operatorname{Ln}[z + \operatorname{SQ}(z^2 + 1)]$	Complex Inverse Hyperbolic Sine
11	<b>ZHACOS</b>	$\operatorname{acosh} z = \operatorname{Ln}[z + \operatorname{SQ}(z^2 - 1)]$	Complex Inverse Hyperbolic Cosine
12	<b>ZHATAN</b>	$\operatorname{atanh} z = 1/2 * \operatorname{Ln}[(1+z)/(1-z)]$	Complex Inverse Hyperbolic Tangent

This section covers all the trigonometric and hyperbolic functions, providing the 41Z with a complete function set. In fact, their formulas would suggest that despite their distinct grouping, they are nothing more than particular examples of logarithm and exponential functions (kind of "*logarithms in disguise*").

Their usage is simple: the argument is taken from the complex-**Z** level and *always* saved on the LastZ register. The result is placed on the complex-**Z** level. Levels **W**, **V**, **U** are preserved in all cases, including the more involved calculations with **ZTAN** and **ZATAN** (those with the devilish names), for which extensive use of scratch and temporary internal registers is made.

The formulas used in the 41Z are:

$\sin z = -i * \sinh(iz)$	$\sinh z = 1/2 * [e^z - e^{-z}]$
$\cos z = \cosh(iz)$	$\cosh z = 1/2 * [e^z + e^{-z}]$
$\tan z = -i * \tanh(iz)$	$\tanh z = (e^z - e^{-z}) / (e^z + e^{-z})$
$\operatorname{asin} z = -i * \operatorname{asinh}(iz)$	$\operatorname{asinh} z = \operatorname{Ln}[z + \operatorname{SQ}(z^2 + 1)]$
$\operatorname{acos} z = \pi/2 - \operatorname{asin} z$	$\operatorname{acosh} z = \operatorname{Ln}[z + \operatorname{SQ}(z^2 - 1)]$
$\operatorname{atan} z = -i * \operatorname{atanh}(iz)$	$\operatorname{atanh} z = 1/2 * \operatorname{Ln}[(1+z)/(1-z)]$

So we see that interestingly enough, the hyperbolic functions are used as the primary ones, also when the standard trigonometric functions are required. This could have also been done the other way around, with no particular reason why the actual implementation was chosen.

**Example.** Because of their logarithmic nature, also the inverse trigonometric and hyperbolic functions will be multi-valued. Write a routine to calculate all the multiple values of **ASIN** z.

01 <b>LBL "ZASIN"</b>	08 <b>ZRCL</b>	15 <b>ZAVIEW</b>
02 <b>ZASIN</b>	09 <b>ZNEG</b>	16 <b>PSE</b>
03 <b>ZSTO</b>	10 <b>ZSTO</b>	17 <b>E</b>
04 <b>ZAVIEW</b>	11 <b>RCL 02</b>	18 <b>ST+ 02</b>
05 <b>E</b>	12 <b>PI</b>	19 <b>GTO 00</b>
06 <b>STO 02</b>	13 <b>*</b>	20 <b>END</b>
07 <b>LBL 00</b>	14 <b>+</b>	



- the multiple values for  $\text{ACOS}(z)$  –in yellow circles– are placed on the same two straight lines, and are separated at intervals of  $2\pi$  length on each line.
- the multiple values for  $\text{ATAN}(z)$  –in brown triangles– are placed on the upper of those straight lines, separated at intervals of  $\pi$  length on it.
- the multiple values for  $\text{Ln}(z)$  –in blue squares– are placed on the vertical straight line  $x=\text{Re}[\text{LN}(z)]$ , and separated at intervals of  $2\pi$  length on it.
- the three different values for  $z^{1/3}$  are placed in the circle  $r=|z|^{1/3}$ , and are separated at 120 degrees from each other (angular interval).

<b>NXTASN</b>	<b>Next Complex ASIN</b>		Does LastZ
<b>NXTACS</b>	<b>Next Complex ACOS</b>		Does LastZ
<b>NXTATN</b>	<b>Next Complex ATAN</b>		Does LastZ

Let  $z_0$  be the principal value of the corresponding inverse trigonometric function. Each of these three functions returns two values,  $z_1$  and  $z_1'$  placed in complex stack levels **Z** and **W**.  $z_1$  will be shown if the function is executed in RUN mode. You can use **Z<>W** to see the value stored in **W** (that is,  $z_1'$ )

The NEXT values  $z$  and  $z_1'$  are and given by the following recursion formulas:

Next ZASIN:

$$Z_1 = Z_0 + 2\pi i$$

$$Z_1' = -Z_0 + \pi i$$

Next ZACOS:

$$Z_1 = Z_0 + 2\pi i$$

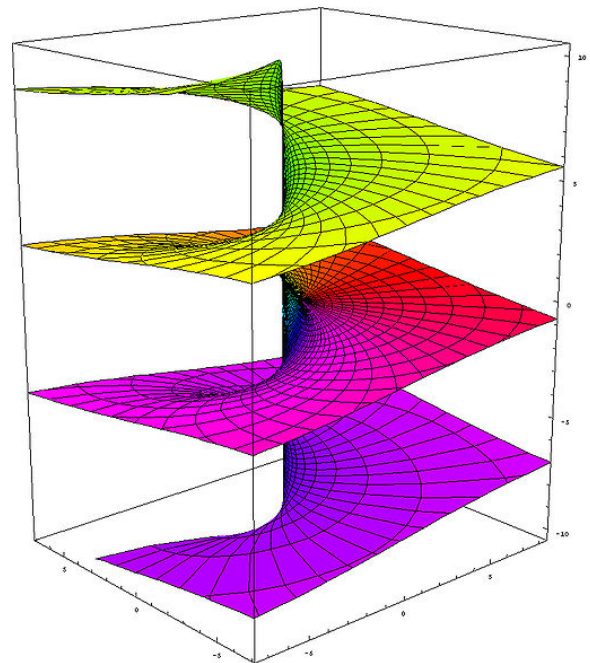
$$Z_1' = -Z_0 + 2\pi i$$

Next ZATAN:

$$Z_1 = Z_0 + \pi i$$

$$Z_1' = Z_0 - \pi i$$

The figure on the right plots the multi-valued imaginary part of the complex logarithm function, which shows the branches. As a complex number  $z$  goes around the origin, the imaginary part of the logarithm goes up or down:



For further information on multi-valued complex functions see the following excellent reference:

[http://en.wikipedia.org/wiki/Branch\\_point](http://en.wikipedia.org/wiki/Branch_point)

Note: See section 9 ahead for further details on multi-valued functions, with the FOCAL driver program **ZMTV** (ZmulTiValue) that calculates all the consecutive results of the eight multi-value functions.

## 8. 2D-vectors or complex numbers?

One of the common applications for complex numbers is their treatment as 2D vectors. This section covers the functions in 41Z that deal with vector operations between 2 complex numbers.

Table 8.1. 2D vectors function group.

Index	Function	Formula	Description
1	<b>ZWANG</b>	$\text{Arg}(ZW) = \text{Arg}(Z) - \text{Arg}(W)$	Angle between 2 vectors
2	<b>ZWDIST</b>	$ W-Z  = \text{SQR}[(Wx-Zx)^2 - (Wy-Zy)^2]$	Distance between 2 points
3	<b>ZWDOT</b>	$Z*W = Zx*Wx + Zy*Wy$	2D vector Dot product
4	<b>ZWCROSS</b>	$Z \times W =  z  *  w  * \text{Sin}(\text{Angle})$	2D vector Cross product
5	<b>ZWDET</b>	$ ZW  = Wx*Zy - Wy*Zx$	2D determinant
6	<b>ZWLINE</b>	$a=(Y1-Y2) / (X1-X2)$ $b=Y2 - a*X2$	Equation of line through two points

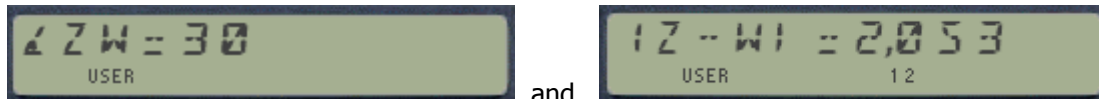
These functions use **W** and **Z** levels of the complex stack, leaving the result in level **Z** after performing complex stack drop. The original contents of **Z** is saved in the LastZ register.

The following screen captures from V41 show the different displays for these functions:

Let  $z = 4 < 45$  degrees, and  $w = 3 < 75$  degrees .

45, ENTER^, 4, **ZREC**                      -> 2,828(1+j)  
**ZREPL**    [don't forget or Z will be overwritten]  
 75, ENTER^, 3, **ZREC**                      -> 0,776 + 2,898j

- 2) **ZWANG**, - angle defined between both vectors (in degrees in this case)
- 2) **ZRDN** , **LASTZ**, **ZWDIST** – distance between both complex numbers



The angle will be expressed in the selected angular unit.

- 2) **ZRDN** , **LASTZ**, **ZWDOT** - dot product of both vectors
4. **ZRDN**, **LASTZ**, **ZWCROSS** - magnitude of the cross product of both vectors



5. **ZRDN**, **LASTZ**, **ZWDET** - magnitude of the determinant of both vectors
6. **ZRDN**, **LASTZ**, **ZWLINE** - equation of the straight line linking both points



(\*) Note that despite having a simpler formula, **ZWDET** shows less precision than **ZWCROSS**.



## 9. It's a Gamma/Zeta world out there.

This section describes the different functions and programs included on the 41Z that deal with the calculation of the Gamma and Zeta functions in the complex plane. A group of five functions in total, two completely written in machine code and three as FOCAL programs, plus a couple of example application programs to complement it.

Table 9.1. Gamma function group.

<b>ZGAMMA</b>	<b>Complex Gamma function</b>	for $z \neq -k$ , $k=\text{integer}$	Does LastZ
<b>-HL ZMTH</b>	<b>Auxiliary Product</b>	$\text{PROD}[(z+n); n=1,..6]$	Does LastZ
<b>ZPSI</b>	<b>Complex Digamma (Psi)</b>	see below	FOCAL program
<b>ZLNG</b>	<b>Gamma Logarithm</b>	see below	FOCAL program
<b>ZZETA</b>	<b>Complex Riemann Zeta</b>	For $z \neq 1$	FOCAL program

**ZGAMMA** uses the Lanczos approximation to compute the value of Gamma. An excellent reference source is found under <http://www.rskey.org/gamma.htm>, written by Viktor T. Toth. Remark that **ZGAMMA** is implemented completely in machine code, even for  $\text{Re}(z) < 0$  using the reflection formula for analytical continuation.

For complex numbers on the positive semi-plane [ $\text{Re}(z) > 0$ ], the formula used is as follows

$$\Gamma(z) = \frac{\sum_{n=0}^N q_n z^n}{\prod_{n=0}^N (z+n)} (z+5.5)^{z+0.5} e^{-(z+5.5)}$$

$q_0 =$	75122.6331530
$q_1 =$	80916.6278952
$q_2 =$	36308.2951477
$q_3 =$	8687.24529705
$q_4 =$	1168.92649479
$q_5 =$	83.8676043424
$q_6 =$	2.5066282

$$\Gamma(1-z) \Gamma(z) = \frac{\pi}{\sin(\pi z)}$$

And the following identity (reflection formula) is used for numbers in the negative semi-plane: [ $\text{Re}(z) < 0$ ]: which can be re-written as:  $\Gamma(z) * \Gamma(-z) = -\pi / [z * \text{Sin}(\pi z)]$

For cases when the real part of the argument is negative [ $\text{Re}(z) < 0$ ], **ZGAMMA** uses the analytical continuation to compute the reflection formula – all internal in the MCODE and transparent to the user.

Example 1.- Calculate  $\Gamma(1+i)$

1, ENTER^, **ZGAMMA** -> "RUNNING...", followed by -> 0,498-j0,155

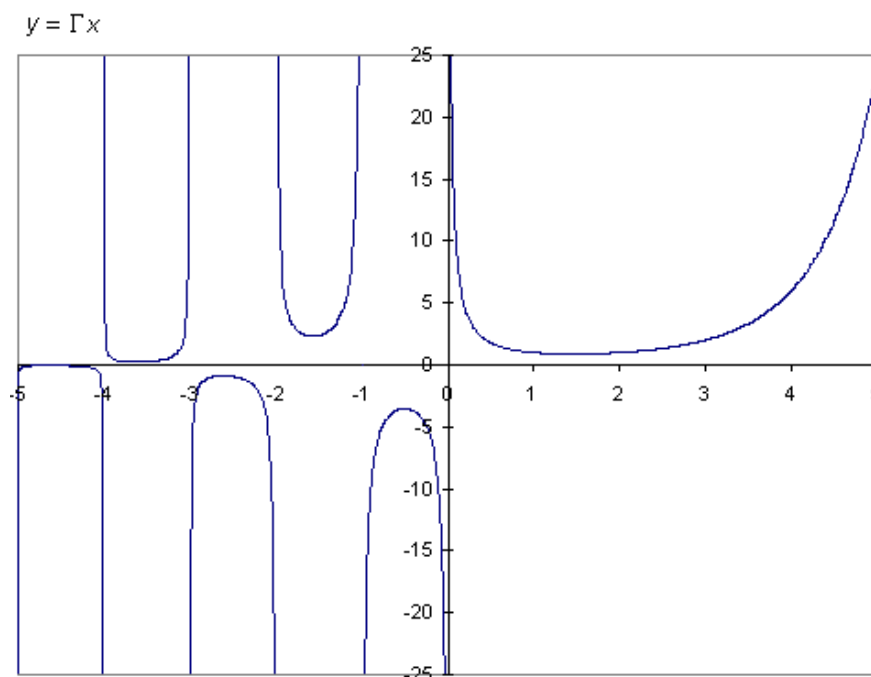
Example 2.- Verify that  $\Gamma(1/2) = \text{SQR}(\pi)$

0, ENTER^, 0.5, **ZGAMMA** -> 1,772 + j0  
 PI, SQRT, **ZREAL^**, **Z-** -> -2,00E-9 + j0

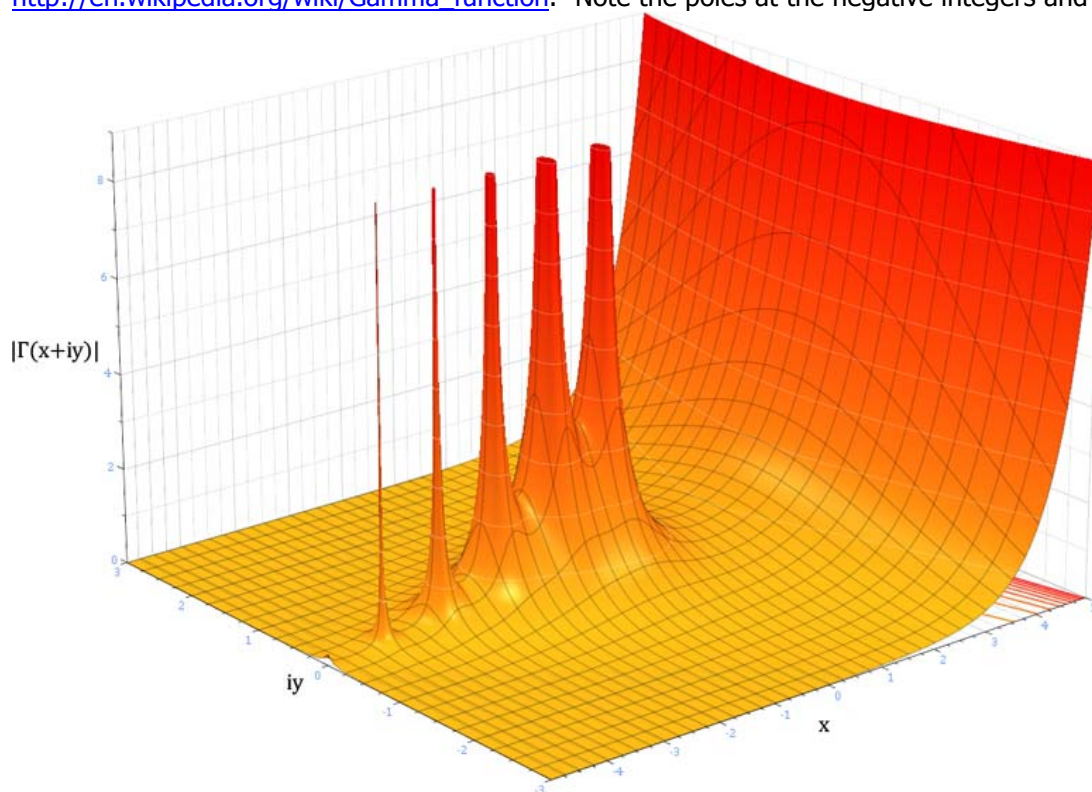
Example 3.- Calculate  $\Gamma(-1.5+i)$

1, ENTER^, 1.5, CHS, **ZGAMMA** -> 0,191 + j0,174

The graphic below (also from the same web site) shows Gamma for real arguments. Notice the poles at  $x=0$  and negative integers.



The following graphic showing the ***module of the Complex Gamma*** function is taken from [http://en.wikipedia.org/wiki/Gamma\\_function](http://en.wikipedia.org/wiki/Gamma_function).- Note the poles at the negative integers and zero.



**Example:** Use **ZLNG** to calculate  $G(1+i)$  and compare it with the value obtained by **ZGAMMA**

1, ENTER^, **ZGAMMA**, **LASTZ**, **ZLNG**, **ZEXP**, **Z-**      -> 2,400E-9+j3,000E-10

**Program listings.-**

The two FOCAL programs listed below calculate the Digamma and the Gamma functions for complex arguments. The first one is an example using the asymptotic approximation as described below, whilst the second one is an extension of the MCODE function **ZGAMMA**, using the reflection formula for arguments with  $\text{Re}(z) < 1$  (programmed in turn as another MCODE function, **ZGNZG**).

01	LBL "ZPSI"	26	Z/	01	LBL "ZG"
02	ZREPL	27	21	02	ZENTER^
03	7 E-3	28	1/X	03	X<>Y
04	STO O	29	ZREAL^	04	X#0?
05	CLZ	30	Z-	05	GTO 00
06	LBL 00	31	Z*	06	X<>Y
07	Z<>W	32	0,1	07	X>0?
08	RCL O	33	+	08	GTO 00
09	INT	34	Z*	09	INT
10	+	35	1	10	LASTX
11	ZINV	36	-	11	X#Y?
12	Z+	37	Z*	12	GTO 00
13	ISG O	38	12	13	0
14	GTO 00	39	ZREAL^	14	1/X
15	ZSTO	40	Z/	15	LBL 00
16	1	41	ZRCL (00)	16	ZRDN
17	Z<>W	42	ZLN	17	CF 00
18	8	43	LASTZ	18	X<0?
19	+	44	ZHALF	19	SF 00
20	ZINV	45	Z+	20	FS? 00
21	ZSTO (00)	46	Z-	21	ZNEG
22	Z^2	47	ZRCL	22	ZGAMMA
23	ZREPL	48	1	23	FC? 00
24	20	49	Z-	24	GTO 01
25	ZREAL^	50	ZAVIEW	25	LASTZ
		51	END	26	ZGNZG
				27	Z<>W
				28	Z/
				29	LBL 01
				30	ZAVIEW
				31	END

for x>8

$\Psi(x) = \ln x - 1/(2x) - 1/(12x^2) + 1/(120x^4) - 1/(252x^6) + 1/(240x^8)$   
together with the relationship:  $\Psi(x+1) = \Psi(x) + 1/x$

Approximation for Digamma when  $x > 8$

$$\Psi(x) = \log(x) - \frac{1}{2x} - \frac{1}{12x^2} + \frac{1}{120x^4} - \frac{1}{252x^6} + O\left(\frac{1}{x^8}\right)$$

programmed as:  $u^2\{[(u^2/20-1/21)u^2 + 1/10]u^2 - 1\}/12 - [\ln u + u/2]$ ,

where  $u=1/x$ ; and using the following precision correction factor when  $x < 8$

$$\Psi(x+1) = \Psi(x) + \frac{1}{x}.$$

The next expression shows Stirling's approximation for Gamma:

$$\Gamma(z) \approx \sqrt{\frac{2\pi}{z}} \left( \frac{z}{e} \sqrt{z \sinh \frac{1}{z} + \frac{1}{810z^6}} \right)^z,$$

The following two programs calculate the Logarithm of the Gamma function for complex arguments. The first one uses the Stirling approximation, with a *correction factor* to increase the precision of the calculation. This takes advantage of the **ZGPRD** function, also used in the Lanczos approximation.

$$2 \ln \Gamma(z) \approx \ln(2\pi) - \ln z + z \left( 2 \ln z + \ln \left( z \sinh \frac{1}{z} + \frac{1}{810z^6} \right) - 2 \right),$$

correction factor: **LnG(z) = LnG(z+7) – Ln[PROD(z+k)|k=1,2..6]**

The second one applies the direct definition by calculating the summation until there's no additional contribution to the partial result when adding more terms. In addition to being much slower than the Stirling method, this is also dependent of the display precision settings and thus not the recommended approach. It is not included on the 41Z but nevertheless is an interesting example of the utilization of some of its functions, like **Z=WR?** And the memory storage registers, **ZSTO** and **ZRCL**.

$$\ln \Gamma(z) = -\gamma z - \ln z + \sum_{k=1}^{\infty} \left[ \frac{z}{k} - \ln \left( 1 + \frac{z}{k} \right) \right]$$

01	LBL "ZLNG"		01	LBL "ZLNG2"	
02	7		02	1	
03	+		03	STO 02	
04	ZSTO (00)		04	RDN	
05	Text-0	NOP	05	ZSTO (00)	
06	6		06	XEQ 05	
07	CHS		07	LBL 00	
08	Z^X		08	ZENTER^	
09	810		09	XEQ 05	
10	ST/ Z		10	Z+	
11	/		11	Z=WR?	
12	ZRCL (00)		12	GTO 02	
13	ZINV		13	GTO 00	
14	ZSINH		14	LBL 02	
15	ZRCL (00)		15	ZRCL (00)	
16	Z*		16	ZLN	
17	Z+		17	Z-	
18	ZLN		18	ZRCL (00)	
19	ZRCL (00)		19	0,5772156649	
20	ZLN		20	ST* Z	
21	ZDBL		21	*	
22	Z+		22	Z-	
23	2		23	ZAVIEW	
24	-		24	RTN	
25	ZRCL (00)		25	LBL 05	
26	Z*		26	ZRCL (00)	
27	ZRCL (00)		27	RCL 02	
28	ZLN		28	ST/ Z	
29	Z-		29	/	
30	PI		30	ZENTER^	
31	ST+ X		31	1	
32	LN		32	+	
33	+		33	ZLN	
34	ZHALF		34	Z-	
35	ZRCL (00)		35	1	
36	Text-0	NOP	36	ST+ 02	
37	7		37	RDN	
38	-		38	END	
39	ZGPRD				
40	ZLN				
41	Z-				
42	ZAVIEW				
43	END				

Max ZREG#	Size
n/a	1
0	2
	3
1	4
	5
2	6
	7
3	8
	9
4	10
	11
5	12
	13
6	14
	15
7	16
	17
8	18
	19
9	20
	21
10	22
	23
11	24
	25
...	...

The table on the right shows the correspondence between the complex register number( CRnn) and the required SIZE in the calculator. Note that a minimum of SIZE 003 is required for CR00 to exist.

## Riemann's Zeta function.

Included in the 41Z is an implementation of the Borwein algorithm to calculate the Zeta function. Considering the task at hand this does an excellent job, providing accurate results in acceptable execution times. Obviously won't win the speed contest, nor will it help you find non-trivial zeroes outside of the critical strip ☺

*Example:* calculate  $\zeta(2)$

2, **ZREAL**^, **ZZETA**                    -> 1,645+J0  
FIX 9                                        -> 1,644934066

The program is a modified version of JM Baillard's ZETAZ, written for complex arguments – only adapted to use the 41Z complex stack and related functions. See the program listing in next page if interested. The algorithm is summarized as follows:

1. For the case  $\text{Re}(z) < 0.5$  , 2 formulas may be used

$$\zeta(z) = \zeta(1-z) 2^z \pi^{z-1} \sin(\pi(z/2)) \Gamma(1-z)$$

$$\zeta(z) = \zeta(1-z) \pi^{z-1/2} \Gamma((1-z)/2) / \Gamma(z/2)$$

2. If  $\text{Re}(z) \geq 0.5$

$$\zeta(z) = \chi(z) / (1-2^{1-z})$$

where:

$$\chi(z) = \sum \{ (-1)^k / k^z \}, k=0,1,2,\dots$$

is calculated by:

$$\chi(z) = (-1)^{dn} \sum \{ (-1)^k (dk-dn) / (k+1)^z \}, k=0 \text{ to } n-1$$

where:

$$dk = n \sum \{ (n+j-1)! 4^j / ((n-j)!(2j)!) \}, j=0 \text{ to } k$$

with an error:

$$|e| \leq (3/(3+\sqrt{8}))^n [1+2 \text{Im}(z)] \exp [p \text{Im}(z) / 2]$$

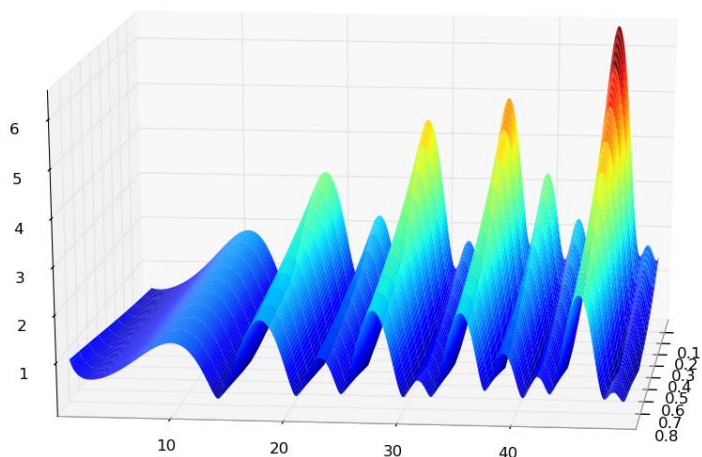
Note that dk is calculated using the following approach:

$$dk = e(0)+e(1)+\dots+e(k)$$

where :

$$e(0)=1 \text{ and}$$

$$e(j+1) = \frac{2(n^2 - j^2) e(j)}{[(1+j)(2j+1)]}$$



**FOCAL program for ZZETA:-** Uses R00 to R11. No Flags used.

01	LBL "ZZETA"		51	ST+ X		101	X^2
02	X=0?		52	LN1+X		102	RCL 10
03	GTO 00		53	+		103	DSE X
04	.5		54	3 E10		104	NOP
05	CHS		55	LN		105	X^2
06	ZAVIEW		56	+		106	-
07	RTN		57	8		107	ST+ X
08	LBL 00		58	SQRT		108	/
09	CF 00		59	3		109	STO 03
10	ZSTO 03	R06 - Re(z)	60	+		110	ST+ 05
11	ZSTO 00	R07 - Im(z)	61	LN		111	DSE 10
12	.5		62	/		112	GTO 01
13	X<=Y?		63	INT		113	RCL 05
14	GTO 00		64	E		114	ST/ 08
15	SF 00		65	+		115	ST/ 09
16	SIGN		66	STO 10		116	RCL 07
17	-		67	STO 02		117	CHS
18	ZNEG	R06: - Im(z)	68	LASTX		118	STO 11
19	ZSTO 03	R07: 0,5-Re(z)	69	STO 11		119	RCL 06
20	XEQ 00		70	STO 03		120	CHS
21	ZRCL 00		71	STO 05		121	2
22	ZNEG		72	CHS		122	LN
23	E		73	X<>Y		123	*
24	+		74	Y^X		124	E
25	2		75	CHS		125	RAD
26	ST/ Z		76	STO 04		126	P-R
27	/		77	CLX		127	ENTER^
28	ZGAMMA		78	STO 08		128	DEG
29	Z*		79	STO 09		129	E
30	ZRCL 00		80	LBL 01		130	ST+ 11
31	.5		81	ZRCL 03		131	-
32	-		82	ZNEG		132	RCL 11
33	PI		83	RCL 10		133	2^X-1
34	X^Z		84	X^Z		134	ST* Z
35	Z*		85	RCL 05		135	X<> T
36	ZRCL 00		86	RCL 04		136	ST* T
37	2		87	CHS		137	ST+ T
38	ST/ Z		88	STO 04		138	RDN
39	/		89	*		139	+
40	ZGAMMA		90	ST* Z		140	ZST/ 04
41	Z/		91	*		141	ZRCL 04
42	ZAVIEW		92	ZST+ 04		142	FC? 00
43	RTN		93	RCL 10		143	ZAVIEW
44	LBL 00		94	ENTER^		144	END
45	PI		95	ST+ Y			
46	2		96	ST* Y			
47	/		97	-			
48	RCL 06	Re(z)	98	RCL 03			
49	ABS		99	*			
50	ST* Y		100	RCL 02			

## 10. Application programs.

Most of the following functions in the 41Z are in reality FOCAL programs (the exceptions being **ZAWL** and **ZHGF**), included as application examples because of their applicability and as a way to illustrate actual programming of the complex number functions of the module.

Index	Function	Description	Author
0	<b>ZWYE</b>	Delay-Wye Conversions	AM
1	<b>ZQRT</b>	Roots of Quadratic equation	AM
2	<b>ZCRT</b>	Roots of Cubic equation	AM
3	<b>ZMTV</b>	Multi-valued functions	AM
4	<b>ZPROOT</b>	Roots of a polynomial of any degree	Valentín Albillo
5	<b>ZSOLVE</b>	Solves $f(z)=0$ by secant method	AM
6	<b>ZWL</b>	Lambert-W function	JM Baillard & AM
7	<b>ZAWL</b>	Inverse of Lambert-W	AM
8	<b>ZLIN</b>	Polylogarithm	AM
8b	<b>ZLI2</b>	Dilogarithm	AM
9	<b>ZLRCH</b>	Lerch Transcendent Function	AM
10	<b>ZHGF</b>	Hypergeometric Function	JM Baillard

*Note 0.*- Program **ZWYE** is not included in the module, all the others are.

*Note 1.*- All of these functions appear on CAT'2 as M-Code entries, instead of as FOCAL programs. This is achieved by using a clever technique shown by W. Doug Wilder (author of the BLDROM), which allows cleaner and convenient program listings (no ugly "XROM" description before the program title).

These programs cannot be copied into main memory using COPY. Another drawback is that they are interpreted as PRIVATE by the 41 OS, nor could they be "looked-up" using GTO + global LBL, since there's no global LBL for them.

*Note 2.*- In version 9L the FAT entry for **ZMTV** was removed – the same functionality exists accessed via the launcher menus. Refer to the following sections for details.

*Note 3.*- The Hypergeometric Function is the preferred method used for the calculation of the Exponential Integrals and the Error function – which have been programmed as simple FOCAL examples of the former. See the descriptions in the SandMath module users' Manual for additional reference.

*Note 4.*- The programs supplied for the Polylogarithm and Lerch functions are simplified and necessarily non-rigorous, not using contour integrals or residues. See the references below for a formal treatment of the problem, clearly exceeding the scope of this manual.-

<http://rspa.royalsocietypublishing.org/content/459/2039/2807.full.pdf>  
<http://rspa.royalsocietypublishing.org/content/463/2080/897.full.pdf>



## 10.0.- EE's Time: Delta-Wye Transformation.

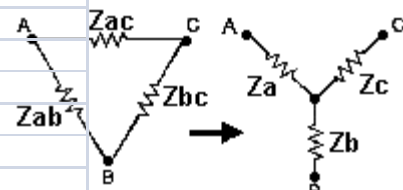
To open this section of the manual here's a token of appreciation for the EE audiences – using the 41Z to tackle a classic: Delta-Wye impedance transformation for 3-phase systems. The simple program below is all there is to it – behold the power of the 41Z complex stack in action :-)

Delta <-> Why conversions		
LBL "D-Y"	LBL "DYD"	
SF 00	ZRCL 00	$Z_a / Z_{ab}$
GTO 00	ZRCL 01	$Z_b / Z_{bc}$
LBL "Y-D"	Z+	$Z_a+Z_b / Z_{ab}+Z_{bc}$
CF 00	FC? 00	
LBL 00	GTO 01	
"Za"	ZRCL 02	$Z_{ab}$
FS? 00	Z+	$Z_{ab}+Z_{bc}+Z_{ca}$
" -b"	ZINV	$1/(Z_{ab}+Z_{bc}+Z_{ca})$
" -=?"	ZRPL^	
PROMPT	ZRCL 00	$Z_{ab}$
ZSTO 00	ZRCL 02	$Z_{ca}$
"Zb"	Z*	$Z_{ab}Z_{ca}$
FS? 00	Z*	$Z_a = Z_{ab}Z_{ca}$
" -c"	Z<>W	$1/(Z_{ab}+Z_{bc}+Z_{ca})$
" -=?"	ZRCL 01	$Z_{bc}$
PROMPT	ZRCL 00	$Z_{ab}$
ZSTO 01	Z*	$Z_{ab}Z_{bc}$
"Zc"	Z*	$Z_b = Z_{ab}Z_{bc}/(Z_{ab}+Z_{bc}+Z_{ca})$
FS? 00	ZRUP	$1/(Z_{ab}+Z_{bc}+Z_{ca})$
" -a"	ZRCL 02	$Z_{ca}$
" -=?"	ZRCL 01	$Z_{bc}$
PROMPT	Z*	$Z_{bc}Z_{ca}$
ZSTO 02	Z*	$Z_c = Z_{bc}Z_{ca}/(Z_{ab}+Z_{bc}+Z_{ca})$
XEQ "DYD"	RTN	
ZSTO 02	LBL 01	
ZRDN	LASTZ	$Z_b$
ZSTO 01	ZRCL 00	$Z_a$
ZRDN	Z*	$Z_aZ_b$
ZSTO 00	ZRCL 02	$Z_c$
ZRDN	Z/	$Z_aZ_b/Z_c$
ZRDN	Z+	$Z_{ab} = Z_a+Z_b+Z_aZ_b/Z_c$
ZVIEW 00	ZRCL 01	$Z_b$
ZVIEW 01	ZRCL 00	$Z_a$
ZVIEW 02	Z/	$Z_b/Z_a$
RTN	ZRCL 02	$Z_c$
	Z*	$Z_bZ_c/Z_a$
	LASTZ	$Z_c$
	Z+	$Z_c+Z_bZ_c/Z_a$
	ZRCL 01	$Z_b$
	Z+	$Z_b+Z_c+Z_bZ_c/Z_a$
	ZRCL 00	$Z_a$
	ZRCL 01	$Z_b$
	Z/	$Z_a/Z_b$
	ZRCL 02	$Z_c$
	Z*	$Z_aZ_c/Z_b$
	LASTZ	$Z_c$
	Z+	$Z_c+Z_aZ_c/Z_b$
	ZRCL 00	$Z_a$
	Z+	$Z_a+Z_c+Z_aZ_c/Z_b$
	RTN	

$$Z_1 = \frac{Z_b Z_c}{Z_a + Z_b + Z_c}$$

$$Z_2 = \frac{Z_c Z_a}{Z_a + Z_b + Z_c}$$

$$Z_3 = \frac{Z_a Z_b}{Z_a + Z_b + Z_c}$$



## 10.1 Solution of quadratic and cubic equations.

<b>ZCRT</b>	<b>Roots of cubic equation</b>	Main routine	
<b>ZQRT</b>	<b>Roots of quadratic equation</b>	Main routine	
<b>ZQUAD</b>	Driver for ZCRT	Application Program	Not included

**ZQRT** Solves the roots of a quadratic equation with complex coefficients, as follows:

$$C_1 * z^2 + C_2 * z + C_3 = 0; \text{ where } C_1, C_2, C_3, \text{ and } z \text{ are complex numbers}$$

By applying the general formula:

$$z_{1,2} = [-C_2 \pm \sqrt{C_2^2 - 4C_1C_3}] / 2C_1$$

Example 1. - find out the roots of  $(1+i)z^2 + (-1-i)z + (1-i) = 0$

**XEQ "ZQUAD"**                      *"aZ^2+bZ+c=0"*, followed by:  
   "IM^RE a=?"  
 1, ENTER^, R/S                    "IM^RE b=?"  
 1, CHS, ENTER^, R/S            "IM^RE c=?"  
 1, CHS, ENTER^, 1, R/S        *"RUNNING..."* followed by:  
   " 1,300+j0,625"  
 R/S                                    "-0,300-j0,625"

We can see that both roots are NOT conjugate of each other, as it occurs with real coefficients.

Note that **ZQUAD** is just a driver for **ZQRT**, which expects the three complex coefficients stored in levels **V**, **W**, and **Z** of the complex stack. Note also that *no memory registers are used*, and all calculations are performed using exclusively the complex stack. The core of the program is from lines 16 to 37, or just 21 programming steps to resolve both roots.

### Program listing.-

1	LBL "ZQUAD"	12	PROMPT	23	ZHALF	34	Z+
2	"aZ^2+bZ+c=0"	13	"RUNNING..."	24	ZNEG	35	ZRDN
3	AVIEW	14	AVIEW	25	ZENTER^	36	Z+
4	PSE	15	LBL "ZQRT"	26	ZENTER^	37	ZRUP
5	"IM^RE a=?"	16	ZENTER^	27	Z^2	38	SF 21
6	PROMPT	17	ZRUP	28	ZRUP	39	ZAVIEW
7	ZENTER^	18	Z/	29	Z-	40	Z<>W
8	"IM^RE b=?"	19	LASTZ	30	ZSQRT	41	CF 21
9	PROMPT	20	ZRUP	31	ZENTER^	42	ZAVIEW
10	ZENTER^	21	Z<>W	32	ZNEG	43	END
11	"IM^RE c=?"	22	Z/	33	ZRUP		

Example 2. Obtain the three roots of  $(1+2i)z^3 - (2-i)z - 3i = 0$

We type: 2, ENTER^, 1, **[Z]**, 0, **ZENTER^**, 1, ENTER, 2, CHS, **[Z]**, **[,]**, 3, **ZNEG**  
 to obtain the three solutions in the complex stack, as follows:

**XEQ "ZCRT"**    → z1 = -0,117-J0,910  
**ZRDN**            → z2 = -0,922+J1,047  
**ZRDN**            → z3 = 1,039-J0,136

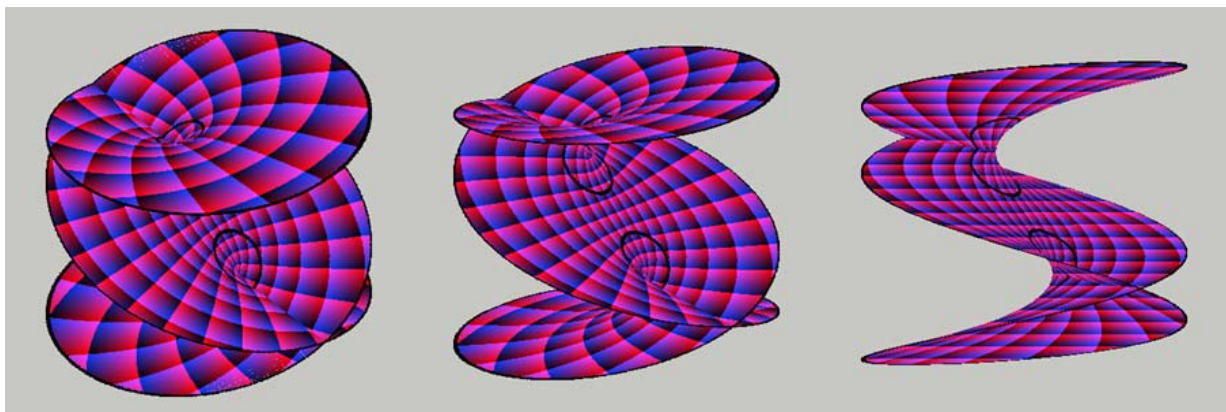
## Two ways to skin the third-degree Equation Cat.

The programs below show two alternative solutions for the third degree equation roots. Note the existing symmetry between them, in fact identical until step 31. The version on the left is the implemented in the 41Z module. Both use a variation of the Cardano-Vieta formulas involving some trigonometry tricks that notably reduce the number of steps.

1	LBL "ZCRT"	Main version	LBL "ZCRT2"	Alternative Version
2	ZRUP	$a3$	ZRUP	$a3$
3	Z/	$a0/a3$	Z/	$a0/a3$
4	ZSTO (00)	$a0'$	ZSTO (00)	$a0'$
5	Z<>W	$a1$	Z<>W	$a1$
6	LASTZ	$a3$	LASTZ	$a3$
7	Z/	$a1/a3$	Z/	$a1/a3$
8	ZSTO 01	$a1'$	ZSTO 01	$a1'$
9	ZRUP	$a'2$	ZRUP	$a'2$
10	LASTZ	$a3$	LASTZ	$a3$
11	Z/	$a2/a3$	Z/	$a2/a3$
16	3		3	
17	ST/ Z		ST/ Z	
18	/		/	
19	ZSTO 02	$a2' / 3$	ZSTO 02	$a2' / 3$
12	Z^2	$a2^2 / 9$	Z^2	$a2^2 / 9$
13	3		3	
14	ST* Z		ST* Z	
15	*	$a2^2 / 3$	*	$a2^2 / 3$
20	Z-	$a1-a2^2 / 3$	Z-	$a1-a2^2 / 3$
21	ZRCL 02	$a2 / 3$	ZRCL 02	$a2 / 3$
22	Z^3	$a2^3 / 27$	Z^3	$a2^3 / 27$
23	ZDBL	$2 a2^3 / 27$	ZDBL	$2 a2^3 / 27$
24	ZRCL 01	$a1$	ZRCL 01	$a1$
25	ZRCL 02	$a2 / 3$	ZRCL 02	$a2 / 3$
26	Z*	$a1*a2 / 3$	Z*	$a1*a2 / 3$
27	Z-	$(a2^3 / 27) - (a1*a2 / 3)$	Z-	$(a2^3 / 27) - (a1*a2 / 3)$
28	ZRCL (00)	$a0$	ZRCL (00)	$a0$
29	Z+	$q = a0 + (a2^3 / 27) - (a1*a2 / 3)$	Z+	$q = a0 + (a2^3 / 27) - (a1*a2 / 3)$
30	ZHALF	$q/2$	ZHALF	$q/2$
31	Z<>W	$p$	Z<>W	$p$
32	3		-3	
33	ST/ Z		ST/ Z	
34	/	$p/3$	/	$-p/3$
35	Z/	$3q/2p$	Z/	$-3q/2p$
36	LASTZ	$p/3$	LASTZ	$-p/3$
37	ZSQRT	$\text{sqr}(p/3)$	ZSQRT	$\text{sqr}(-p/3)$
38	ZSTO (00)		ZSTO (00)	
39	Z/	$3q/2p / \text{sqr}(p/3)$	Z/	$-3q/2p / \text{sqr}(-p/3)$
40	ZHASIN		ZASIN	
41	3		3	
42	ST/ Z		ST/ Z	
43	/	$1/3 \text{ asin}[ ]$	/	
44	ZRPL^	Fill complex stack	ZRPL^	Fill complex stack
45	,002		,002	
46	STO 02		STO 02	
47	RDN		RDN	
48	LBL 02	Data output loop	LBL 02	Data output loop
49	RCL 02		RCL 02	

50	INT		INT	
51	120	$2k\pi/3$	120	$2k\pi/3$
52	D-R		D-R	
53	*		*	
54	ST+ Z	add to imaginary part	+	add to real part
55	RDN	tidy up stack	ZSIN	
56	ZHSIN		ZRCL (00)	
57	ZRCL (00)		Z*	
58	Z*		ZDBL	
59	ZDBL		ZRCL 02	$a2/3$
60	ZNEG		Z-	
61	ZRCL 02	$a2/3$	ZAVIEW	
62	Z-		ZRUP	save in Z-stack
63	ZAVIEW	Show progress...	ISG 02	Increase counter
64	ZRUP	save in Z-stack	GTO 02	Go for next
65	ISG 02	Increase counter	END	done
66	GTO 02	Go for next		
67	END	done		

As you can see the density of 41Z functions is remarkable. The 41Z complex function set and complex stack enables the programmer to treat complex calculations as though they used real numbers, not worrying about the real or imaginary parts but working on the complex number as single entity. In fact, exercising some care (notably to ensure complex stack lift), you could almost translate one-to-one many FOCAL programs by replacing the standard functions with the equivalent complex ones. That's why it's important that the function set be as complete as possible, and that the complex stack implementation follows the same RPN conventions.



## 10.2 Lambert W function.

<b>ZWL</b>	<b>Lambert W(z)</b>		FOCAL program
<b>ZAWL</b>	<b>Inverse of Lambert-W</b>	$z * e^z$	Does LastZ

These two functions provide a dedicated way to compute the Lambert-W function and its inverse. The FOCAL program uses an iterative method to compute  $W(z)$ , using  $z_0=1+\ln(z)$  as initial guess for  $\text{Re}(z)>0$ , and simply  $z_0=(1+i)$  elsewhere.

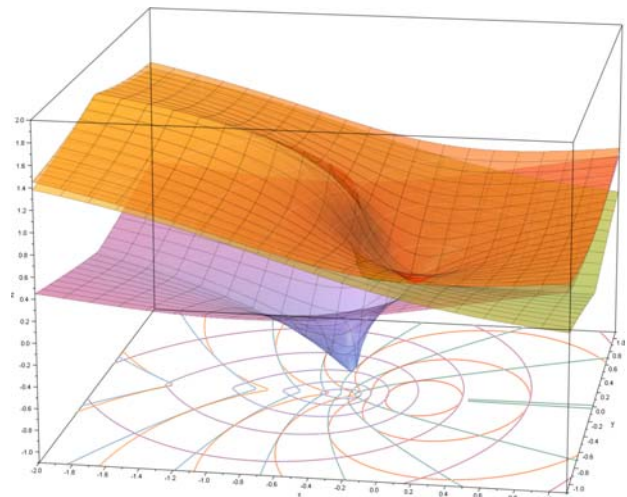
This program is based on a real-mode version written by JM Baillard, just applying the seamless transposition method provided by the 41Z module. In the vast majority of cases convergence is provided for all complex arguments, with 8-decimal digits accuracy. It uses the **Z=WR?** Function on FIX 8 mode to determine that two consecutive iterations are equal.

The inverse function is a simple product:  $W^{-1}(z) = z * e^z$ .

Not worth the FAT entry, you say? For one thing, doing it in MCODE allows for 13-digit accuracy in the results. Besides how often will you forget the exact formula? Better safe than sorry...

1	<b>LBL "ZWL"</b>	
2	<b>Z=0?</b>	
3	<b>GTO 00</b>	
4	<b>ZSTO (00)</b>	
5	<b>E</b>	
6	<b>+</b>	
7	<b>Z=0?</b>	
8	<b>ISG Y(2)</b>	
9	<b>ZLN</b>	
10	<b>FIX 8</b>	
11	<b>LBL 01</b>	
12	<b>ZREPL</b>	
13	<b>ZNEG</b>	
14	<b>ZEXP</b>	
15	<b>ZRCL (00)</b>	
16	<b>Z*</b>	
17	<b>Z-</b>	
18	<b>Z&lt;&gt;W</b>	
19	<b>E</b>	
20	<b>+</b>	
21	<b>Z/</b>	
22	<b>Z-</b>	
23	<b>Z=WR?</b>	
24	<b>GTO 00</b>	
25	<b>GTO 01</b>	
26	<b>LBL 00</b>	
27	<b>FIX 3</b>	
28	<b>ZAVIEW</b>	
29	<b>END</b>	

Another version using SOLVE is listed in section 10.5.1, with slightly more accurate results , but significantly slower execution and a few trouble spots (near  $1/e$  and  $-1/e$ ).



### 10.3 Multi-valued Functions.

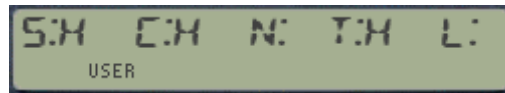
<b>ZMTV</b>	<b>Multi-valued functions</b>		
-------------	-------------------------------	--	--

This program calculates all possible values for the multi-valued functions, including the  $n$  different  $N^{\text{th}}$  roots of a complex number, all the inverse trigonometric and hyperbolic, plus the logarithm itself (source of all the multi-valued scenarios).

Due to the 64-function limit of the 41 ROM FAT structure. these routines are all part of a common entry into the module catalog. To access it you use the **ZNEXT** prompt, followed by the **XEQ** key – i.e: **[Z]**, **[“A”]**, **[SHIFT]**, **[“K”]**

When invoked, the program prompts a menu of choices as follows:

A – ASIN      B – ACOS      C: Nth. Root      D: ATAN      E: Ln  
a – HSIN      b.- HACOS      d.- HATAN



Or more succinctly:

For each case the program will calculate the principal value followed by all the other values with each subsequent pressing of [R/S]. Remember that the top keys need to be free from user assignments for this scheme to work, as per the 41 OS conventions.

All trigonometric functions expect  $z$  into the **Z** level of the complex stack. Data entry is the same for all of them except for the  $n$ -th root, which expects  $N$  in the real-stack register  $X$ , and  $z$  in **Z**. Only the first  $N$  values will be different, running into cyclical repetition if continued.

This is a simple program, mostly written to document an example for the 41Z functions. Use it to get familiar with these concepts, and to understand fully the NXT function set as well.

**Example:** Obtain all values of ASIN  $[\sin(1+j)]$

```
1, ENTER^, ZSIN      -> 1,298+j0,635
ZMTV                -> "S:H C:H N: T:H L:"
[A]                  -> 1,000+j1
R/S                    -> 2,142-j1
R/S                    -> 7,283+j1
R/S                    -> 8,425-j1
etc...
```

**Alternatively,** using the **NXTASN** function:

Note that here we start with the first value of the function, i.e.  $1+j$

```
1, ENTER^, NXTASN    -> 7,238+j1
Z<>W                -> 2,142-j1
NXTASN               -> 8,425-j1
NXTASN               -> 14,708-j1
```

**Program listing.-** Alternative version, superseded in revision 4L.

Note the use of flag 22 for numeric entry: the catalog of functions will display continuously until one choice is made, (expected between 1 and 8), and all initial prompting will be skipped.

1	LBL "ZMTV"	48	LBL 93	95	LBL 92
2	CF 22	49	ZASIN	96	ZHACOS
3	LBL 20	50	ZSTO	97	GTO 07
4	"FCN#.=? 1-8"	51	ZAVIEW	98	LBL 96
5	AVIEW	52	E	99	ZHATAN
6	PSE	53	STO 02	100	LBL 06
7	PSE	54	LBL 03	101	ZAVIEW
8	FC? 22	55	ZRCL	102	PSE
9	GTO 90	56	ZNEG	103	PI
10	INT	57	ZSTO	104	+
11	ABS	58	RCL 02	105	GTO 06
12	90	59	PI	106	LBL 97
13	+	60	*	107	ZLN
14	RDN	61	+	108	LBL 07
15	SF 25	62	ZAVIEW	109	ZAVIEW
16	GTO IND T	63	PSE	110	PSE
17	GTO 20	64	E	111	NXTLN
18	LBL 90	65	ST+ 02	112	GTO 07
19	CF 21	66	GTO 03	113	LBL 98
20	"1:- ZACOS"	67	LBL 91	114	CF 00
21	AVIEW	68	ZACOS	115	"N=?"
22	PSE	69	ZSTO	116	PROMPT
23	"2:- ZACOSH"	70	ZAVIEW	117	ABS
24	AVIEW	71	E	118	INT
25	PSE	72	STO 02	119	X=0?
26	"3:- ZASIN"	73	LBL 01	120	RTN
27	AVIEW	74	ZRCL	121	STO 00
28	PSE	75	RCL 02	122	E
29	"4:- ZASINH"	76	ST+X	123	-
30	AVIEW	77	PI	124	STO 01
31	PSE	78	*	125	X=0?
32	"5:- ZATAN"	79	STO 03	126	SF 00
33	AVIEW	80	+	127	E
34	PSE	81	ZAVIEW	128	+
35	"6:- ZATANH"	82	PSE	129	1/X
36	AVIEW	83	ZRCL	130	Z^X
37	PSE	84	ZNEG	131	SF 21
38	"7:- ZLN"	85	RCL 03	132	ZAVIEW
39	AVIEW	86	+	133	FS?C 00
40	PSE	87	ZAVIEW	134	GTO 08
41	"8:- Z^1/N"	88	PSE	135	LBL 05
42	AVIEW	89	E	136	RCL 00
43	PSE	90	ST+ 02	137	NXTNRT
44	GTO 20	91	GTO 01	138	ZAVIEW
45	LBL 95	92	LBL 94	139	DSE 01
46	ZATAN	93	ZHASIN	140	GTO 05
47	GTO 06	94	GTO 07	141	LBL 08
				142	CF 21
				143	END



## 10.4 Roots of Complex Polynomials.

<b>ZPROOT</b>	<b>Roots of Polynomials</b>	<b>By Valentín Albillo</b>
---------------	-----------------------------	----------------------------

This program calculates all the roots of a polynomial of degree  $n$ , and with complex coefficients. It is therefore *the most general case of polynomial root finders* that can possibly be used, as it also will work when the coefficients are real.

This program is a wonderful example of FOCAL capabilities, and very well showcases the versatility of the HP-41C (even without the 41Z module). It was first published on PPC Technical Notes, PPCTN – the journal of the Australian chapter of the PPC.

1	LBL "ZPROOT"		44	CF 00		87	E-3		130	GTO 02
2	SIZE?		45	CHS		88	ST+ 01		131	RCL 08
3	"DEGREE=?"		46	STO 04		89	RCL 03		132	ST* Z
4	PROMPT		47	FIX 2		90	STO IND 05		133	*
5	STO Z		48	RND		91	RCL 04		134	DSE 08
6	ST+X	2N	49	FIX 6		92	STO IND 06		135	GTO 02
7	11		50	X#0?		93	DSE 00		136	RTN
8	+	2N+11	51	GTO 01		94	GTO 06		137	LBL 00
9	X>Y?		52	SIGN		95	TONE 5		138	ZENTER^
10	PSIZE		53	STO 04		96	RCL 01		139	RCL 04
11	RCL Z		54	LBL 01		97	INT		140	RCL 03
12	STO 00	N	55	RCL 00		98	E1		141	Z*
13	STO 03	N	56	STO 08		99	-		142	RCL IND 05
14	9,008		57	SF 01		100	E3		143	FS? 01
15	+		58	XEQ 11		101	/		144	RCL 08
16	STO 01	N+9,008	59	R-P		102	ST- 05		145	FS? 01
17	STO 05	N+9,008	60	1/X		103	FIX 3		146	*
18	X<>Y	2N+11	61	STO 07		104	SF 21		147	+
19	E		62	X<>Y		105	LBL 10		148	FS? 00
20	-	2N+10	63	CHS		106	ISG 00		149	STO IND 05
21	STO 02	2N+10	64	STO 08		107	NOP		150	X<>Y
22	STO 06		65	CF 01		108	RCL IND 06		151	RCL IND 06
23	FIX 0		66	XEQ 11		109	RCL IND 05		152	FS? 01
24	CF 29		67	ZENTER^		110	ZAVIEW		153	RCL 08
25	LBL 05		68	RCL 08		111	DSE 06		154	FS? 01
26	"IM^RE("	N	69	RCL 07		112	DSE 05		155	*
27	ARCL 03		70	P-R		113	GTO 10		156	+
28	"@)=?"		71	Z*		114	CF 21		157	FS? 00
29	PROMPT		72	ST- 03		115	SF 29		158	STO IND 06
30	STO IND 05		73	X<>Y		116	RTN		159	X<>Y
31	X<>Y		74	ST- 04		117	LBL 11		160	FS? 01
32	STO IND 06	N-1	75	ZRND		118	RCL 01		161	DSE 08
33	DSE 03		76	Z#0?		119	STO 05		162	LBL 02
34	X<>Y		77	GTO 01		120	RCL 02		163	DSE 06
35	DSE 06		78	FIX 0		121	STO 06		164	DSE 05
36	DSE 05		79	"FOUND ROOT#"		122	FC? 01		165	GTO 00
37	GTO 05		80	ARCL 00		123	GTO 13		166	END
38	RCL 03		81	AVIEW		124	E-3			
39	LBL 06		82	SF 00		125	ST+ 05			
40	"SOLVING..."		83	XEQ 11		126	LBL 13			
41	AVIEW		84	E		127	RCL IND 06			
42	SF 25		85	ST+ 05		128	RCL IND 05			
43	SF 99		86	ST+ 06		129	FC? 01			

Example 1.- Calculate the three roots of:  $x^3 + x^2 + x + 1$

XEQ "ZPROOT"	-> "DEGREE=?"
3, R/S	-> "IM^RE (3)=?"
0, ENTER^, 1, R/S	-> "IM^RE (2)=?"
0, ENTER^, 1, R/S	-> "IM^RE (1)=?"
0, ENTER^, 1, R/S	-> "IM^RE (0)=?"
0, ENTER^, 1, R/S	-> "SOLVING..."
	-> "FOUND ROOT#3", and "SOLVING..."
	-> "FOUND ROOT#2", and "SOLVING..."
	-> "FOUND ROOT#1"
	→ -5,850E-14-j1 (that is, -i)
	→ 5,850E-14+j1 (that is, i)
	→ -1+j1,170E-13 (that is, -1)

Example 2.- Calculate the four roots of:  $(1+2i)*z^4 + (-1-2i)*z^3 + (3-3i)*z^2 + z - 1$

XEQ "ZPROOT"	-> "DEGREE=?"
4, R/S	-> "IM^RE (4)=?"
2, ENTER^, 1, R/S	-> "IM^RE (3)=?"
2, CHS, ENTER^, 1, CHS, R/S	-> "IM^RE (2)=?"
3, CHS, ENTER^, CHS, R/S	-> "IM^RE (1)=?"
0, ENTER^, 1, R/S	-> "IM^RE (0)=?"
0, ENTER^, 1, CHS, R/S	-> "SOLVING..."
	-> "FOUND ROOT#4", and "SOLVING..."
	-> "FOUND ROOT#3", and "SOLVING..."
	-> "FOUND ROOT#2", and "SOLVING..."
	-> "FOUND ROOT#1"
	→ 1,698+J0,802 R/S
	→ -0,400-J0,859 R/S
	→ 0,358+J0,130 R/S
	→ -0,656-J0,073

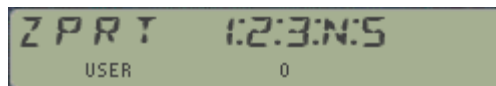
The four solutions are:

$z_1 = 1,698 + 0,802 j$	or: 1,878 <) 25,27
$z_2 = -0,400 - 0,859 j$	or: 0,948 <)-114,976
$z_3 = 0,358 + 0,130 j$	or: 0,381 <) 9,941
$z_4 = -0,656 - 0,073 j$	or: 0,660 <)-173,676

### Root Finders Launcher - all together now.

A convenient grouping of the previous applications provides access to the root finders for the first, second, third and n-th. degree polynomials, as well as the general-purpose **ZSOLVE**. To access it just press:

[Z], [A], [H], [SHIFT]



The first-degree option is for function **ZWLINE** - not strictly a root finder but being such a simple case it's convenient to have it also in the group.

For **ZQRT** and **ZCRT** the coefficients are expected to be in the complex stack prior to the execution – whilst **ZPROOT** and **ZSOLVE** will prompt for the required entries.

Note that the [SHIFT] key toggles between this launcher and the Exponential Integrals one, see section 10.7 later in the manual.

## 10.5 Solution to $f(z)=0$ .

The next application uses the Secant Method to obtain roots of a complex equation, given two estimations of the solution. A general discussion on root-finding algorithms is beyond the scope of this manual – this example is intended to show the capabilities of the 41Z module, in particular how programming with complex numbers becomes as simple as doing it for real numbers using the native function set.

See the following link for further reference on this subject (albeit just for real variable):  
[http://en.wikipedia.org/wiki/Secant\\_method](http://en.wikipedia.org/wiki/Secant_method)

The secant method is defined by the recurrence relation:

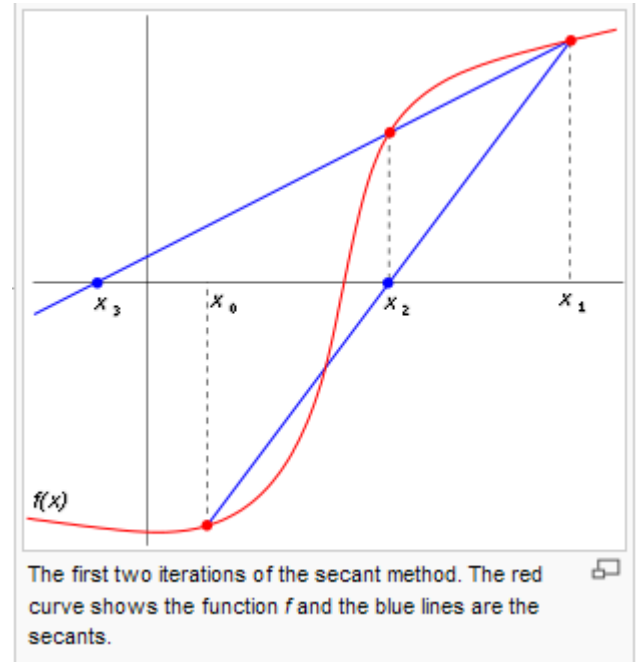
$$x_{n+1} = x_n - \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} f(x_n).$$

which will be calculated until there's no significant contribution to the new value – as determined by the function **Z=WR?**.

### Program listing:-

As it's the case with this type of programs, the accuracy of the solution depends of the display settings, and the convergence (i.e. likelihood to find a root) will depend on the initial estimations.

The program works internally with 8-digit precision, therefore will largely benefit from the turbo-mode settings on V41 to dramatically reduce the execution time.



1	LBL "ZSOLVE"	20	Z<>W	39	Z-
2	FS? 06	21	ZSTO (00)	40	Z*
3	GTO 06	22	XEQ IND 06	41	ZNEG
4	AON	23	ZSTO	42	ZRCL
5	"F. NAME=?"	24	2	43	1
6	PROMPT	25	LBL 01	44	Z+
7	AOFF	26	ZRCL	45	ZENTER^
8	ASTO 06	27	1	46	Z<>
9	PREC=?	28	XEQ IND 06	47	1
10	PROMPT	29	ZREPL	48	Z=WR?
11	FIX IND X	30	Z<>	49	GTO 02
12	"Z1=? (Y^X)"	31	2	50	GTO 01
13	PROMPT	32	Z-	51	LBL 02
14	ZENTER^	33	Z#0?	52	FC? 06
15	"Z2=? (Y^X)"	34	Z/	53	FIX 3
16	PROMPT	35	ZRCL	54	FC? 06
17	LBL 06	36	1	55	ZAVIEW
18	ZSTO	37	ZENTER^	56	END
19	1	38	Z<> (00)		

User flag 06 is for subroutine usage: when set, the data input will be skipped. In that case the relevant data is expected to be in the appropriate registers, as follows:

CR03= Initial estimation z1,  
CR04 = initial estimation z2  
R12 = Function's name,  
FIX set manually to required precision.

Example 1.- Calculate one root of the equation: **Sinh(z) + z<sup>2</sup> + pi = 0**

Which we easily program using 41Z functions as follows:

LBL "ZT", **ZHSIN**, **LASTZ**, **Z<sup>2</sup>**, **Z+**, PI, +, END.

Using the initial estimations as z0=0, and z1=1+i, we obtain:

Root = -0,27818986 + j 1,81288037

Example 2.- Calculate two roots of the equation: **e<sup>(z)</sup> = z**  
programmed as follows: LBL "ZE", **ZEXP**, **LASTZ**, **Z-**, END

using the estimations: {z0=-1-j & z1=1+j} - note that both roots are conjugated!

Root1 = 0,3181315 + j 1,3372357

Root2 = 0,3181315 - j 1,3372357

Example 3.- Calculate the roots of the polynomials from section 10.1 and 10.3:

**P2 = (1+i)\*z<sup>2</sup> + (-1-i)\*z + (1-i)** - re-written as:  $z[(-1-i)-z(1+i)] + (1-i)$

**P3 = z<sup>3</sup> + z<sup>2</sup> + z + 1** - re-written as:  $z[1+z(1+z)] + 1$

**P4 = (1+2i)\*z<sup>4</sup> + (-1-2i)\*z<sup>3</sup> + (3-3i)\*z<sup>2</sup> + z - 1**  
- re-written as:  $z\{1+z[(3-3i)-z[(1+2i)-z(1+2i)]]\}-1$

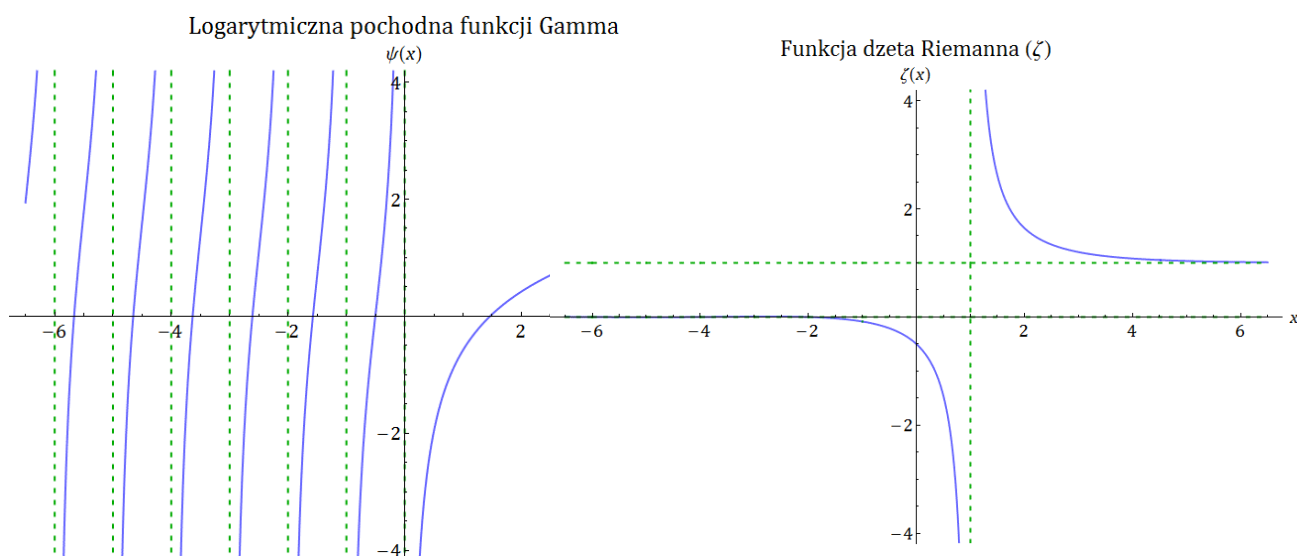
Use the following estimations for the P4 example:-

{z0=-1-j ; z1=1+j}	for root #1,	{z0=1+j ; z1=2+2j}	for root #2,
{z0=-2j ; z1= 2j}	for root #3,	{z0= 4j ; z1= 5j}	for root #4

### **ZSOLVE Register Usage.**

Notice that to avoid register incompatibilities **ZSOLVE** uses registers R06 – R12. This allows its direct application to calculate zeroes of functions using the lower register range (which is the typical case), like the Exponential integral and associates, which in turn all use registers R00 – R05. So there's no need to use cumbersome REGMOVE program steps with its memory-hungry control words.

Lastly, a few other excellent programs written by Jean-Marc Baillard address the general solution to the equation  $f(z)=0$ . They don't use functions from the 41Z module, but are mentioned here for their obviously close related content. The programs can be found at the following link: <http://www.hpmuseum.org/software/41/41cmpxf.htm>



### 10.5.1. Application example;- Using ZSOLVE to calculate the Lambert W function.

In this example we see a few techniques applied together, combining the capabilities of the 41Z in a convenient way. The solution is a direct application of the definition, requiring very simple extra programming – albeit with the logical slow performance.

The Lambert  $W$  function is given by the following functional equation:

$$z = W(z) e^{W(z)}, \text{ for every complex number } z.$$

Which cannot be expressed in terms of elementary functions, but can be properly written with the following short program:

1	LBL "ZWL"
2	ZSTO
3	4
4	ZLN
5	ZENTER^
6	E
7	+
8	SF 06
9	"*W"
10	ASTO 06
11	ZSOLVE
12	ZAVIEW
13	RTN
14	LBL "*W"
15	ZEXP
16	LASTZ
17	Z*
18	ZRCL
19	4
20	Z-
21	END

The complex value is expected to be in the **Z** complex stack level, and X,Y registers upon initialization. Set the FIX manually for the required precision.

Because **ZSOLVE** uses all the complex stack levels and registers 0 to 6 (Note: *this was changed in revision 4L – see pg. 59*), the argument is saved in the complex register 4 – corresponding to real registers 8 and 9, thus a SIZE 10 or higher is required (see register correspondence map below).

We solve for  $W(z)=z$ , using as the function initial estimations the logarithm of the same argument and the same point plus one, perhaps not a refined choice but sufficient to ensure convergence in the majority of cases. Some calculated values are:

$$W(0) = 0$$

$$W(1) = \Omega \approx 0.56714329\dots$$

$$W(e) = 1$$

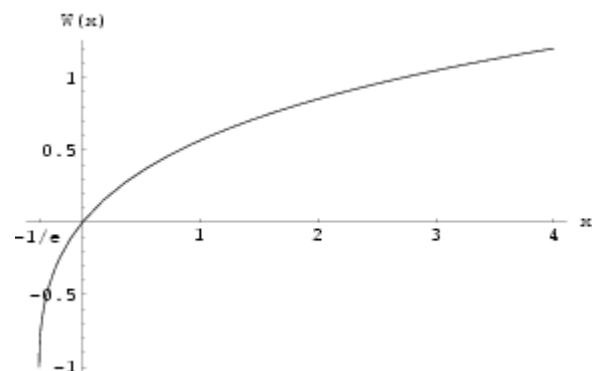
$$W(-1) \approx -0.31813 - 1.33723i$$

This example is not meant to compete with a dedicated program using an iterative algorithm, yet it showcases the versatility of the approach. The obvious speed shortcomings are diminished when ran on the 41CL or modern emulators like V41.

The Taylor series of  $W_0$  around 0 is given by:

$$W_0(x) = \sum_{n=1}^{\infty} \frac{(-n)^{n-1}}{n!} x^n$$

Another technique (somehow a brute-force approach) would employ this definition to calculate successive terms of the summation until their contribution to the sum is negligible. This method would only be applicable within the convergence region.



See the following links for further references on the Lambert W function:

[http://en.wikipedia.org/wiki/Lambert\\_W\\_function](http://en.wikipedia.org/wiki/Lambert_W_function)

<http://mathworld.wolfram.com/LambertW-Function.html>

## 10.6 Bessel functions.

This section represents an interesting "*tour de force*" within the 41Z module – taking the humble 41 system to the realm of true high-level math. Use it or leave it, it's all a matter of choice – but programming techniques and valid algorithms are always interesting, despite its obvious speed shortcomings.

Index	Function	Description	
1	<b>ZJBS</b>	Complex Bessel J function	First kind
2	<b>ZIBS</b>	Complex Bessel I function	First kind
3	<b>ZBS</b>	Subroutine for all J, I and K,Y	First & Second Kind
4	<b>ZKBS</b>	Complex Bessel K function	Second kind
5	<b>ZYBS</b>	Complex Bessel Y function	Second kind
6	<b>ZLIN</b>	Complex Polylogarithm	
7	<b>EIZ/IZ</b>	Spherical Hankel first kind order zero	
8	<b>ZSHK1</b>	Spherical Hankel first kind	
7	<b>ZSHK2</b>	Spherical Hankel second kind	

See the paper "*Bessel functions on the 41 with the SandMath Module*" by the author, for an extensive description of the (real-number) Bessel Functions on the 41 system. In fact, following the "*do it as it's done with real numbers*" standard philosophy of the 41Z module, the complex versions of these programs are very similar to those real-number counterparts described in said paper.

The formulae used are as follows:

$$J(n,z) = \sum \{U_k \mid k=1,2,\dots\} * (z/2)^n / \Gamma(n+1)$$

$$U(k) = -U(k-1) * (z/2)^2 / k(k+n)$$

$$U(0) = 1$$

$$Y_n(x) = [ J_n(x) \cos(n\pi) - J_{-n}(x) ] / \sin(n\pi)$$

$$K_n(x) = (\pi/2) [ I_n(x) - I_{-n}(x) ] / \sin(n\pi)$$

$$n \# \dots -3 ; -2 ; -1 ; 0 ; 1 ; 2 ; 3 ..$$

Like for the real arguments case, there is one auxiliary functions **ZBS#**, used to perform intermediate calculations needed by the main programs: **ZJBS**, **ZIBS** (first kind), and **ZYBS**, **ZKBS** (second kind). Other auxiliary functions are:

- **ZGEU** Euler's gamma constant as a complex number, and
- **HARMN** to obtain the harmonic number of a given integer: (uses "**-ZSTACK**")

$$H(n) = \sum [1/k] \mid k=1,2\dots n \quad (*)$$

The expressions used to calculate the results are different for integer orders (remember the singularities of Gamma), requiring special branches of the main routines. For that reason two other functions have been added to the 41Z as follows:

- **ZINT?** to determine integer condition, and
- **ZCHSX** to simplify calculation of  $z*(-1)^k$

Both the function order and the argument are complex numbers, which are expected to be on complex stack levels **W** (order) and **Z** (argument) prior to the execution of the function. The result is placed on the Z-level complex stack.

Below are the program listings for each particular case.-



a) Bessel Functions of the first kind. Uses R00 – R08. Uses Flags 0-1

1	LBL ZJBS		48	Z*	n
2	CF 00		49	ZRCL 00	n
3	GTO 00		50	RCL M	k
4	LBL ZIBS		51	+	n+k
5	SF 00		52	LASTX	k
8	LBL 00		53	ST* Z	k(n+k)
8	CF 01		54	*	
8	Z<>W		55	Z/	
9	ZINT?	is n integer?	56	ZSTO 02	U(k)
10	XEQ 05		57	ZRCL 03	SUM(k-1)
11	Z<>W		58	Z+	SUM(k)
12	ZHALF	z/2	59	ZENTER^	
13	XROM "ZBS"		60	Z<> 03	SUM(k-1)
14	FS? 01	n integer	61	Z=W?	
15	RCL 01		62	GTO 01	
16	FS? 01		63	E	
17	ZCHSX	$J(-n, z) = (-1)^n J(n, z)$	64	ST+ M	k=k+1
18	LBL 04		65	GTO 02	
19	ZAVIEW		66	LBL 01	
20	RTN		67	ZRCL 00	n
21	LBL 05		68	INCX	(n+1)
22	X<0?	n<0?	69	CF 02	
23	SF 01		70	X<0?	
24	ABS		71	SF 02	
25	RTN		72	X<0?	
26	LBL "ZBS"		73	ZNEG	-z
27	Z#0?		74	ZGAMMA	
28	GTO 00		75	FC? 02	
29	Z=W?		76	GTO 00	
30	E		77	LASTZ	-z
31	GTO 04		78	ZGNGZ	
32	LBL 00		79	Z<>W	
33	-ZSTACK	running...	80	Z/	
34	ZSTO 01	(z/2)	81	LBL 00	
35	Z<>W	n	82	Z/	
36	ZSTO 00	n	83	ZRCL 01	(z/2)
37	E	1	84	ZRCL 00	n
38	ZREAL	1+J0	85	W^Z	(z/2)^n
39	ZSTO 02	1+J0	86	Z*	
40	ZSTO 03	1+J0	87	END	
41	STO M	k=1			
42	LBL 02				
43	ZRCL 01			CR00 - n	
44	Z^2	(z/2)^2		CR01 - Z/2	
45	ZRCL 02	Uk-1		CR02 - Uk	
46	FC? 00			CR03 - SUM	
47	ZNEG			CR04 - result	

**Examples:-** Calculate JBS(1+i, -1-i) and IBS(-0.5+i; 1-0.5i)

1, ENTER^, ZENTER^, ZNEG, ZJBS

--> -8,889 + j 2,295

1, ENTER^, 0.5, CHS, ZENTER^, ENTER^, 1, ZIBS

--> 3,421 + j 1,178

b) *Bessel functions of the second kind.* Uses R00 – R08. Uses flags 0-2

1	<b>LBL "ZB1"</b>	<b>SUM{f(n,x)}</b>	1	<b>LBL "ZB2"</b>	<b>SUM{g(n,x)}</b>
2	CLZ		2	CLZ	
3	ZSTO 02	$J_n / I_n$	3	ZSTO 03	reset partial SUM
4	ZSTO 04	SUM	4	RCL 00	ABS(n)
5	STO 01	$k=0$	5	X=0?	$n=0?$
6	<b>LBL 02</b>		6	RTN	skip it
7	XEQ 10	summing term	7	DECX	
8	Z=0?	$x=0?$	8	E3	
9	<b>GTO 01</b>	ignore term	9	/	$0,00(n-1)$
10	ZRCL 04	$S(k-1)$	10	STO 08	
11	Z+	$S(k)$	11	<b>LBL 05</b>	
12	ZENTER^		12	ZRCL 01	$x/2$
13	Z<> 04		13	RCL 08	$k,00(n-1)$
14	Z=W?	are they equal?	14	INT	
15	RTN	Final result(s)	15	STO 01	$k$
16	<b>LBL 01</b>		16	ST+ X	$2k$
17	E	increase index	17	RCL 00	$n$
18	ST+ 01	$k=k+1$	18	-	$2k-n$
19	<b>GTO 02</b>		19	Z^X	$(x/2)^{(2k-n)}$
20	<b>LBL 10</b>	<b>Function to Sum</b>	20	RCL 00	$n$
21	ZRCL 01	$x/2$	21	RCL 01	$k$
22	RCL 01	$k$	22	-	$n-k$
23	ST+ X	$2k$	23	DECX	$n-k-1$
24	RCL 00	$n$	24	FACT	$(n-k-1)!$
25	+	$2k+n$	25	RCL 01	$k$
26	Z^X	$(x/2)^{(2k+n)}$	26	FACT	$k!$
27	ZENTER^		27	/	$(n-k-1)! / K!$
28	RCL 01	$k$	28	ST* Z	
29	FACT	$k!$	29	*	[**]
30	LASTX	$k$	30	FC? 00	is it Yn?
31	RCL 00	$n$	31	<b>GTO 00</b>	
32	+	$k+n$	32	RCL 01	$k$
33	FACT	$(k+n)!$	33	ZCHSX	$(-1)^k * \text{term}$
34	*	$k! * (k+n)!$	34	<b>LBL 00</b>	
35	ZREAL		35	ZRCL 03	
36	Z/	$k\text{-th. Term}$	36	Z+	
37	FS? 00	is it Kn?	37	ZSTO 03	
38	<b>GTO 00</b>		38	ISG 08	
39	RCL 01	$k$	39	<b>GTO 05</b>	$(k+1),00(n-1)$
40	ZCHSX	$[\text{term}] * (-1)^k$	40	ZRCL 03	
41	<b>LBL 00</b>		41	FC? 00	is it Yn?
42	Z<> 02	<b>ZST+ 02</b>	42	RTN	
43	ZRCL 02		43	RCL 00	$n$
44	Z+	$f(k) + \text{SUM}(k-1)$	44	ZCHSX	$\text{SUM} * (-1)^n$
45	Z<> 02	$J_n / I_n$	45	END	
46	ZENTER^				
47	RCL 01	$k$			
48	HARMN	$H(k)$			
49	LASTX	$k$			
50	RCL 00	$n$			
51	+	$k+n$			
52	HARMN	$H(k+n)$			
53	+	$H(k)+H(k+n)$			
54	ZREAL				
55	Z*				
56	END				

Note: functions DECX and INCX can be replaced by standard FOCAL sequences:

DECX = 1, -  
INCX = 1, +

1	<b>LBL "ZYBS"</b>	Integer Index	47	<b>LBL 05</b>	integer orders
2	CF 00		48	CF 01	
3	GTO 00		49	X<0?	negative
4	<b>LBL "ZKBS"</b>		50	SF 01	
5	SF 00		51	ABS	
6	<b>LBL 00</b> ←		52	STO 00	
7	ZHALF		53	XROM "ZB2"	
8	ZSTO 01	(z/2)	54	ZNEG	-[SUM*(-1)^n]
9	Z<>W		55	ZSTO 03	
10	ZINT?		56	XROM "ZB1"	to obtain both!
11	<b>GTO 05</b>		57	ZRCL 03	
12	Z<>W		58	Z<>W	
13	XROM "ZBS"		59	Z-	
14	FS? 00		60	ZRCL 01	x/2
15	GTO 00		61	ZLN	Ln(x/2)
16	ZRCL 00		62	<b>GEU</b>	g
17	PI		63	+	g+Ln(x/2)
18	ST* Z		64	ZRCL 02	J(n,x) or I(n,x)
19	*		65	Z*	[ ]*J/I(n,x)
20	ZCOS		66	ZDBL	
21	Z*		67	Z+	K(n,x)/Y(n,x)
22	<b>LBL 00</b> ←		68	FC? 00	is it Yn?
23	ZSTO 04		69	<b>GTO 04</b> →	FINAL STEPS
24	ZRCL 00	n	70	RCL 00	n
25	ZNEG	-n	71	<b>INCX</b>	(n+1)
26	ZRCL 01	(z/2)	72	ZCHSX	K(n,x)*(-1)^(n+1)
27	XROM "ZBS"		73	ZHALF	
28	ZRCL 04		74	<b>GTO 03</b> →	Exit
29	Z<>W		75	<b>LBL 04</b> ←	Yn
30	Z-		76	PI	
31	ZRCL 00	-n	77	ST/Z	
32	ZNEG	n	78	/	
33	PI		79	FC? 01	negative index?
34	ST* Z		80	<b>GTO 03</b> →	Exit
35	*		81	RCL 00	n
36	ZSIN		82	ZCHSX	
37	Z/		83	<b>LBL 03</b> ←	
38	FC? 00		84	ZSTO 03	
39	<b>GTO 03</b> →	Exit	85	ZAVIEW	
40	PI		86	END	
41	2				
42	/				
43	CHS				
44	ST* Z				
45	*				
46	<b>GTO 03</b> ←	Exit			

The formulae used for integer orders are as follows:

$$\pi Y_n(x) = 2[\gamma + \ln x/2] J_n(x) - \sum (-1)^k f_k(n,x) - \sum g_k(n,x)$$

$$(-1)^{n+1} 2 K_n(x) = 2 [\gamma + \ln x/2] I_n(x) - \sum f_k(n,x) - (-1)^n \sum (-1)^k g_k(n,x)$$

$$g_k(n,x) = (x/2)^{2k-n} [(n-k-1)! / k!]; \quad k=0,2,\dots,(n-1)$$

$$f_k(n,x) = (x/2)^{2k+n} [H(k) + H(n+k)] / [k! (n+k)!]; \quad k=0,1,2,\dots$$

**Example:-** Calculate **KBS**(-0.5+i; 1-0,5i)

1, ENTER<sup>^</sup>, 0,5, CHS, **ZENTER<sup>^</sup>**,  
ENTER<sup>^</sup>, 1, **ZKBS** → 0,348 + j 0,104

**Example:-** Calculate **YBS**(-1,-1)

0, ENTER<sup>^</sup>, 1, CHS, **ZENTER<sup>^</sup>**,  
**ZYBS** → - 0,781 + j 0,880

This last example shows how even real arguments can yield complex results.

**Example.-** Calculate **JBS** and **IBS** for (1+2i, -1-3i)

2, ENTER<sup>^</sup>, 1, **ZENTER<sup>^</sup>**  
3, CHS, ENTER<sup>^</sup>, 1, CHS, **ZIBS** → 35,813 - j 191,737

2, ENTER<sup>^</sup>, 1, **ZENTER<sup>^</sup>**  
3, ENTER<sup>^</sup>, 1, **ZNEG, ZJBS** → - 257,355 - j 12,633

Note: Using the Complex Keyboard shortcuts the Bessel function group can be accessed pressing SHIFT when the NEXT indicator is shown, as per the following sequence:

**Z, Z**, [SHIFT], [SHIFT] -> then [**I**], [**J**], for **ZJBS** and **ZJBS** or [**K**], [**L**] for **ZKBS** and **ZYBS**.

The same group can be used to access **ZWL** & **ZAWL** (Complex Lambert and its inverse) and **EIZ/IZ**, the Spherical Hankel function of first kind and order zero  $h^{(1)}(0,z)$



The key maps below summarizes all the special assignments in the [**BSSL**] (left) and [**NEXT**] (right) groups. Notice that the mnemonics  $h^{(1)}n$  and  $h^{(2)}n$  correspond to the **ZSH1** and **ZHS2** functions. Note as well the inclusion of the "alternative" versions **SQRTZ**, **e<sup>^</sup>Z** and **1/Z** in the [NEXT] group – so you can quickly compare them with the main functions for accuracy and speed.



## 10.7 Polylogarithm.

The Polylogarithm (also known as Jonquière's function) is a special function  $\text{Li}_s(z)$  that is defined by the infinite sum, or power series

$$\text{Li}_s(z) = \sum_{k=1}^{\infty} \frac{z^k}{k^s} = z + \frac{z^2}{2^s} + \frac{z^3}{3^s} + \dots$$

Only for special values of the order  $s$  does the Polylogarithm reduce to an elementary function such as the logarithm function. The above definition is valid for all complex orders  $s$  and for all complex arguments  $z$  with  $|z| < 1$ ; it can be extended to  $|z| \geq 1$  by the process of analytic continuation. See the reference: <http://people.reed.edu/~crandall/papers/Polylog.pdf>

The implementation of the Polylogarithm is a very rudimentary one, more as an example of direct porting of the real variable routine than anything else. It's based on Jean-Marc's version, that can be found at: <http://hp41programs.yolasite.com/dilogarithm.php>

Both parameters can be complex numbers, although the series representation used forces the condition that  $z$  must be inside the unit circle, that is  $|z| < 1$ . The program will stop with an error message if  $|z| > 1$ . Note also that this method is not valid either for points on the unit circle,  $|z| = 1$ . You can use function **ZLI2** for the dilogarithm, which also works in this case.

In terms of its usage,  $s$  is expected to be in level-2 of the complex stack (W), and  $z$  in level-1 (Z). Let's see a couple of examples.

Example 1. Calculate  $\text{Li}(2; 0.3+0.4i)$

0, ENTER^, 2, **ZENTER^** → 2+J0  
(the Z-keypad version: **[Z]**, 2 does the same easier)

.4, ENTER^, .3, XEQ "ZLIN" → 0,166+J0,560,

or with FIX 9 settings:

Re = 0,326456748  
Im = 0,565254656

Example 2. Calculate  $\text{Li}(1+i, 0.3+0.4i)$

1, ENTER^, .ZENTER^ → 1(1+J)  
.4, ENTER^, .3, XEQ "ZLIN" → 0,326+J0,565

or with FIX 9 settings:

Re = 0,326456748  
Im = 0,565254656

As you can see the program listing doesn't get any easier – so despite its limitations (long execution time, no analytic continuation) it's worthwhile including in the module.

Note that **ZLIN** is a FOCAL program, and therefore the argument  $z$  won't be saved in the LastZ complex register.

01	LBL "ZLIN"
02	" Z >1"
03	ZOUT?
04	PROMPT
05	ZSTO 01
06	Z<W
07	ZSTO 02
08	CLX
09	STO 06
10	E
11	ZSTO 00
12	CLZ
13	LBL 01
14	ZENTER^
15	ZRCL 00
16	ZRCL 01
17	Z*
18	ZSTO 00
19	ISG 06
20	NOP
21	ZRCL 02
22	RCL 06
23	X^Z
24	Z/
25	Z+
26	Z#W?
27	GTO 01
28	ZAVIEW
29	END

## 10.8 Lerch Transcendent Function.

The Lerch Transcendent function can be seen as an extension of the Polylogarithm, and therefore it's easy to modify the previous program to the more general case – adding a third argument " $\alpha$ " as follows:

$$\Phi(z, s, \alpha) = \sum_{n=0}^{\infty} \frac{z^n}{(n + \alpha)^s}.$$

note that contrary to the Polylogarithm case, the summation starts at  $n=0$ ; not at  $n=1$ . This would represent an issue if the power function returned a **DATA ERROR** condition for zero exponent (the zero-th. term being  $z^0 / 0^s$ ). However the 41Z implementation returns zero for this case, and therefore we can use the same program to calculate both the Polylogarithm and Lerch function – taking  $\alpha=0$  for the additional argument in Lis:

$$\text{Li}(s, z) \sim \text{Lerch}(z, s, 0)$$

To be sure the above expression is just a programming trick, but it's not mathematically correct. The proper relationship between both functions is given by:

$$\text{Li}_s(z) = z\Phi(z, s, 1).$$

Example 1. Calculate

$$\Phi(0.3+0.4i; 3+4i; 1+2i)$$

4, ENTER^, 3, **ZENTER^** → 3+J4  
 2, ENTER^, 1, **ZENTER^** → 1+J2  
 .4, ENTER^, .3, XEQ "**ZLRCH**" → 7,658-J1,515,

or with FIX 9 settings:

Re = 7,658159105  
 Im = -1,515114367

Notice the input order convention for the arguments, with  $z$  always entered last, in the Z-level of the complex stack.

Other useful relationships also involving the Lerch Transcendent functions are shown below:

Riemann Zeta: (\*)  
 $\zeta(s) = \Phi(1, s, 1).$

Legendre Chi:  
 $\chi_n(z) = 2^{-n} z \Phi(z^2, n, 1/2).$

(\*) The convergence is very slow, thus using the dedicated **ZZETA** program is a much more convenient approach.

01	<b>LBL "ZLRCH"</b>	
02	<b>" Z &gt;1"</b>	
03	<b>ZOUT?</b>	
04	PROMPT	
05	<b>ZSTO 01</b>	x
06	<b>CLZ</b>	
07	<b>SIGN</b>	
08	<b>ZSTO 00</b>	$x^0 = 1$
09	<b>ZRDN</b>	
10	<b>ZSTO 02</b>	a
11	<b>Z&lt;&gt;W</b>	
12	<b>ZSTO 03</b>	s
13	<b>ZNEG</b>	-s
14	<b>W^Z</b>	$1/a^s$
15	<b>LBL 01</b>	$\Sigma(k-1)$
16	<b>ZRCL 01</b>	x
17	<b>ZRCL 00</b>	$x^{(k-1)}$
18	<b>Z*</b>	$x^k$
19	<b>ZSTO 00</b>	
20	<b>ZRCL 02</b>	$a+k-1$
21	<b>E</b>	
22	<b>ST+ 05</b>	
23	<b>+</b>	$a+k$
24	<b>ZRCL 03</b>	s
25	<b>W^Z</b>	$(a+k)^s$
26	<b>Z/</b>	$x^k / (a+k)^s$
27	<b>Z+</b>	$\Sigma k$
28	<b>Z#WR?</b>	
29	<b>GTO 01</b>	
30	<b>ZAVIEW</b>	
31	<b>END</b>	

## 10.9 Exponential Integrals.

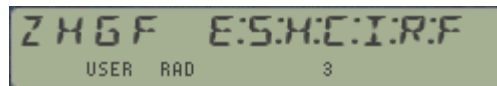
New in revision 4L, this section groups the Exponential Integral and related functions – all calculated using the Hypergeometric function representation.

Index	Function	Description	
1	<b>ZHGF</b>	Complex Hypergeometric function	<i>Author: Jean-Marc Baillard</i>
2	<b>ZEI</b>	Complex Exponential Integral	
3	<b>ZCI</b>	Complex Cosine Integral	
4	<b>ZHCI</b>	Complex Hyperbolic Cosine Integral	
5	<b>ZSI</b>	Complex Sine integral	
6	<b>ZHSI</b>	Complex Hyperbolic Sine Integral	
7	<b>ZERF</b>	Complex Error function	

The key enabler for this group is of course the MCODE implementation of the Complex Hypergeometric function **ZHGF** – written by Jean-Marc Baillard. See the excellent web-site at:  
<http://hp41programs.yolasite.com/complexhypergeo.php>

The rest of the functions are easily obtained as simple and short FOCAL programs, using the well-know equivalence expressions. Their argument is a complex number, taken from the Z-level of the complex stack (XY registers). In terms of usability they are grouped in their own launcher, invoked by pressing **[H]** at the **Z"** prompt; that is:

**[Z]**, **[A]**, **[H]** →



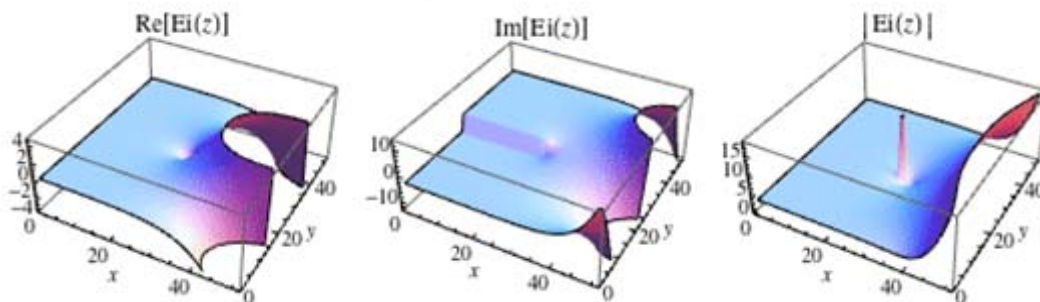
### Examples.-

Calculate  $\text{erf}(1+i)$  and  $\text{Ei}(1+i)$

1, ENTER<sup>^</sup>, **[Z]**, **[A]**, **[H]**, **["R"]** → 1,316+J0,190  
 1, ENTER<sup>^</sup>, **[Z]**, **[A]**, **[H]**, **["E"]** → 1,765+J2,388

Calculate Ei, Ci, Si and their hyperbolic counterparts for the same argument  $z=(1+i)$

1, ENTER<sup>^</sup>, **[Z]**, **[A]**, **[H]**, **["S"]** → 1,104+J0,882  
 1, ENTER<sup>^</sup>, **[Z]**, **[A]**, **[H]**, **["H"]** → 0,882+J1,104  
 1, ENTER<sup>^</sup>, **[Z]**, **[A]**, **[H]**, **["C"]** → 0,882+J0,287  
 1, ENTER<sup>^</sup>, **[Z]**, **[A]**, **[H]**, **["I"]** → 0,882+J1,284



See the program listing in next page, showing the economy of programming when using a power horse like **ZHGF** to do all the heavy lifting for you.



FOCAL Listing: Exponential integrals. Uses R00 – R05

```

01  LBL "ZERF"
02  ZENTER^
03  Z^2
04  E
05  STO 01
06  1.5
07  STO 02
08  CLX
09  E
10  R^
11  R^
12  ZHGF
13  LASTZ
14  ZNEG
15  ZEXP
16  Z*
17  Z*
18  PI
19  SQRT
20  1/X
21  ST+ X
22  ST* Z
23  *
24  ZAVIEW
25  END

```

```

01  LBL "ZSI"
02  SF 00
03  GTO 00
04  LBL "ZHSl"
05  CF 00
06  LBL 00
07  ZENTER^
08  ZHALF
09  Z^2
10  FS? 00
11  ZNEG
12  .5
13  STO 01
14  3
15  *
16  STO 02
17  STO 03
18  CLX
19  E
20  ENTER^
21  2
22  R^
23  R^
24  ZHGF
25  Z*
26  ZAVIEW
27  END

```

```

01  LBL "ZEI"
02  E
03  STO 01
04  STO 02
05  E
06  +
07  STO 03
08  STO 04
09  ENTER^
10  R^
11  R^
12  ZHGF
13  LASTZ
14  Z*
15  LASTZ
16  GTO 01
17  LBL "ZCI"
18  SF 00
19  GTO 00
20  LBL "ZHCl"
21  CF 00
22  LBL 00
23  ZENTER^
24  ZHALF
25  Z^2
26  FS? 00
27  ZNEG
28  ZENTER^
29  E
30  STO 01
31  STO 02
32  CLX
33  2
34  STO 03
35  STO 04
36  1.5
37  STO 05
38  ST+ X
39  R^
40  R^
41  ZHGF
42  Z*
43  Z<>W
44  LBL 01
45  ZLN
46  Z+
47  GEUZ
48  Z+
49  ZAVIEW
50  END

```

## Appendix 1.- Complex Buffer functions.

This appendix lists the buffer handling functions included in the 41Z, and thus are not related to the Complex Number treatment per se. This set is only useful to diagnose problems or to bypass the normal execution of the module's "standard" functions, therefore its usage is not recommended to the casual user (i.e. do it at your own risk!).

Function	Description	Input	Output
<b>-HP 41Z</b>	Initializes Z Buffer	None	Buffer created
<b>CLZB</b>	Clears Z buffer	None	Buffer cleared
<b>L1=XY?</b>	Is L1 equal to XY?	None	Y/N, skip if false
<b>L1&lt;&gt;L _</b>	<b>Swap L1 &amp; Level</b>	<b>Level# as suffix</b>	levels exchanged
<b>L1&lt;&gt;LX</b>	Swap L1 & Level	level in X	levels exchanged
<b>L2=ZT?</b>	Is L2 equal to ZT?	None	Y/N, skip if false
<b>L2&gt;ZT</b>	Copies L2 into ZT	None	L2 copied to ZT
<b>LVIEW _</b>	<b>View Level</b>	<b>Level# as suffix</b>	<i>Transposed value!</i>
<b>LVIEWX</b>	View level by X	level in X	<i>Transposed value!</i>
<b>PREMON</b>	Copies XY into L0 and finds Zbuffer	Re(z) in X; Im(z) in Y	none
<b>PSTMON</b>	Copies XY into L1 and synch's up	Complex stack Z	Re(z) in X; Im(z) in Y
<b>RG&gt;ZB _ _</b>	<b>Copies registers to Z buffer</b>	<b>Reg# as suffix</b>	data copied from registers
<b>ST&gt;ZB</b>	Copies real stack to L1 & L2	None	stack copied to buffer
<b>XY&gt;L _</b>	<b>Copies XY into Level</b>	<b>Level# as suffix</b>	XY copied to LEVEL
<b>XY&gt;L0</b>	Copies XY into L0	Re(z) in X; Im(z) in Y	XY copied to L0
<b>XY&gt;L1</b>	Copies XY into L1	Re(z) in X; Im(z) in Y	XY copied to L1
<b>ZB&gt;RG _ _</b>	<b>copies buffer to registers</b>	<b>Reg# as suffix</b>	data copied to registers
<b>ZB&gt;ST</b>	Copies L1 & L2 into real stack	None	buffer copied to Stack
<b>ZBDROP</b>	Drops Z buffer one level	None	levels dropped
<b>ZBHEAD</b>	Z buffer Header info	None	header register in ALPHA
<b>ZBLIFT</b>	Lifts Z buffer one level	None	buffer lifted
<b>ZBSHOW</b>	Shows Z Buffer	None	shows header & all levels

(\*) Items highlighted in yellow indicate prompting functions.

**Buffer layout.** The complex buffer has 5 levels, labelled L0 to L4; that's 10 memory registers plus the header and footer registers – for a total of 12 registers. The function names in this group use the Level number (L0 to L4) to identify each level, as opposed to the **U**, **V**, **W**, and **Z** notation employed in previous sections of the manual.

	Buffer Layout		Buffer Details
<b>b11</b>	<b>non-zero</b>	-	The buffer has 12 memory registers
<b>b10</b>	<b>L4 (U)</b>	-	Buffer registers are labeled b0 to b11
<b>b9</b>		-	Header is located at the bottom
<b>b8</b>	<b>L3 (V)</b>	-	A non-zero register is at the top
<b>b7</b>		-	Each Level uses two buffer registers
<b>b6</b>	<b>L2 (W)</b>	T	Levels are labeled L0 to L4
<b>b5</b>		Z	
<b>b4</b>	<b>L1 (Z)</b>	Y	
<b>b3</b>		X	
<b>b2</b>	<b>L0 (S)</b>	L	
<b>b1</b>		-	
<b>b0</b>	<b>Header</b>	-	

The buffer header (b0 register) is placed at the lowest memory address. It contains the buffer id#, its size, and its initial address (when it was first created – no updates if it's re-allocated later on).

**Buffer creation** is done automatically by the 41Z module upon power on (when the 41 awakes from deep sleep), using the corresponding poll point in the module. The contents of the real stack registers XYZT is copied into the buffer levels L1 & L2 upon initialization.

The buffer is maintained by the 41 OS, which handles it when modifying the layout of main memory – either changing the SIZE settings, or modifying the user key assignments. The buffer id# is 8, and thus should be compatible with any other memory buffer that uses a different id# (an example of which are the TIMER alarms, with id#=10).

Should for any reason the buffer get damaged or erased (like when using the function **CLZB**), the message “**NO Z-STACK**” would appear when trying to execute any of the 41Z module functions. *To manually re-create the complex buffer* simply execute the first function in the module, “**-HP 41Z**” - either by using XEQ or the Complex Keyboard sequence “**Z**, SHIFT, **Z**”. This requires at least 12 memory registers to be available or the error message “NO ROOM” will be shown.

Because the buffer can be dynamically re-allocated by the 41 OS upon certain circumstances, it's not possible to store its address to be reused by the functions. *Every 41Z function would first seek out the buffer address prior to proceeding with its calculation.* Fortunately this takes very little overhead time.

**Buffer synchronization** with the appropriate real-stack levels is also performed automatically by the 41Z functions, as follows:

- In the input phase (pre-execution), monadic functions will copy the XY contents into level L1 prior to executing their code. Dual functions will do the same for the second argument **Z**, and will use the current contents of the L2 level as first argument **W**.
- In the output phase (post-execution) the results will be placed in the complex buffer levels and then copied to the real stack registers as appropriate: XY for monadic functions, and XZYT for dual functions.

That's the reason why the real stack should just be considered as a *scratch pad* to prepare the data (like doing math on the real values), as only levels X,Y will be used. You must use **ZENTER^** to push the **W** argument into the complex level L2. In other words: real stack registers T,Z will be ignored!

The same consideration applies when performing chain calculations: because there's no automated complex stack lift, *the result of a monadic function would be overwritten by the subsequent input unless it is first pushed into the complex stack*, using **ZENTER^** or another 41Z function that does stack lift.

**Example:** Calculate  $\text{Ln}(1+i) + (2-i)$

The following sequence use the direct data entry, entering Im(z) first.

1, ENTER^, **ZLN**, **ZENTER^**, 1, CHS, ENTER^, 2, **Z+**                      -> 2,347-j0,215

*Some functions perform stack lift by default, and thus **ZENTER^** is not required before them.*

They are as follows:

- **LASTZ**
- **ZRCL** \_ \_
- **ZREAL^** (also when using the complex real keypad, Z plus digit key)
- **ZIMAG^** (also when using the complex imaginary keypad, Z, radix, plus digit key)
- **^IM/AG** - probably the most intricate function in the module

The following sequence uses natural data entry - entering Re(z) first - as an alternative method for the previous example. Note that because **^IMG** does stack lift, it's not necessary to use **ZENTER^**

1, **^IMG**, 1, R/S, **ZLN**, 2, **^IMG**, 1, CHS, R/S, **Z+** -> 2,347-j0,215

Buffer synchronization with the real stack registers can be tested and forced using the following functions in this group:

<b>L1=XY?</b>	- Tests for the first buffer level and XY registers
<b>XY&gt;L1</b>	- Copies X,Y into level L1
<b>L2=ZT?</b>	- Tests for second buffer level and Z,T registers
<b>L2&gt;ZT</b>	- Copies L2 into registers Z,T
<b>ST&gt;ZB</b>	- Copies real stack XYZT to buffer levels L1 & L2
<b>ZB&gt;ST</b>	- Copies L1 & L2 to the real stack XYZT

To dump the complete contents of the complex buffer into memory registers and back you can use these two complementary functions:

<b>ZB&gt;RG _ _</b>	- Copies complex buffer to memory registers
<b>RG&gt;ZB _ _</b>	- Copies memory registers to complex buffer

Note that **RG>ZB** won't check for valid header data, thus it expects the contents to be correct – like with a previously execution of **ZB>RG**. Remember that the header register is a non-normalized number (NNN), thus do not recall it using RCL or X<>.

Other functions to manipulate the contents of the buffer levels are:

<b>L1&lt;&gt;L _</b>	- swaps buffer level L1 and level given by prompt
<b>L1&lt;&gt;LX</b>	- swaps buffer level L1 and level input in X
<b>XY&gt;L0</b>	- copies registers X,Y into buffer level L0 (used to save arguments into LastZ)
<b>XY&gt;L _</b>	- copies registers X,Y into buffer level given by prompt
<b>ZBDROP</b>	- drops contents of complex buffer one level (used during <b>ZRDN</b> )
<b>ZBLIFT</b>	- lifts contents of complex buffer one level (used by <b>ZRUP</b> , <b>ZENTER^</b> and others)

All these functions act on the complex buffer, but will not display the “resulting” complex number (i.e. will not trigger **ZAVIEW** upon completion). To see (view) the contents of the buffer levels without altering their position you can use the following functions:

<b>LVIEW _</b>	- prompts for level number (0 – 4)
<b>LVIEWX</b>	- expects level number in X
<b>ZBSHOW</b>	- lists the contents of all buffer levels
<b>ZBHEAD</b>	- shows in Alpha the decoded buffer header

Note that with these functions all complex level contents will be shown transposed, that is: Im(z) + j Re(z).

finally, the other two functions are auxiliary and mainly used to perform action between the two lower and upper 4k-pages within the 41Z module: (\*)

**PREMON** - Finds Z Buffer address, Copies XY into L0 and checks X,Y for ALPHA DATA

**PSTMON** - Copies the Z complex level into X.Y

(\*) *Note: FAT entries for these two functions were removed in newer versions of the module.*

Because of its relevance and importance within the 41Z module, the following section lists the buffer creation and interrogation routines – pretty much the heart of the implementation. Consider that they are called at least twice every time a function is executed and you'll appreciate their crucial role in the whole scheme!



Remember that the buffer is refreshed (or created) each time the calculator is turned on, and that it gets reallocated when key assignments or other buffers (like timer alarms) are made – yet it's theoretically possible that it gets "unsynchronized" or even lost altogether, and therefore the assignment to the **-HP 41Z** function as well.

1	BUFFER	SYNCH2	A996	39C	PT= 0	Synch2: write XYZT into [b3-b6]
2	BUFFER	(from upper)	A997	130	LDI S&X	
3	BUFFER		A998	003	CON: 3	X reg address
4	BUFFER		A999	146	A=A+C S&X	pointer to b3
5	BUFFER		A99A	270	RAM SELECT	select stack reg
6	BUFFER		A99B	266	C=C-1 S&X	decrease stack pointer
			A99C	0EE	C<>B ALL	store stack pointer in B
b6	L2	T	A99D	038	READ DATA	stack value
b5		Z	A99E	0AE	A<>C ALL	swap stack value and bk address
b4	L1	Y	A99F	270	RAM SELECT	select bk
b3		X	A9A0	0AE	A<>C ALL	swap bk address and stack value
			A9A1	12F0	WRIT DATA	write stack value into bk
13	BUFFER		A9A2	166	A=A+1 S&X	b(k+1) address
14	BUFFER		A9A3	0EE	C<>B ALL	restore stack address
15	BUFFER		A9A4	3DC	PT=PT+1	
16	BUFFER		A9A5	054	?PT=4	loop 4 times
17	BUFFER		A9A6	3A3	JNC -12d	
18	BUFFER		A9A7	3E0	RTN	
1	INITIALIZE	Header	A9A8	0AB	"+"	
2	INITIALIZE	Header	A9A9	01A	"Z"	
3	INITIALIZE	Header	A9AA	031	"1"	
4	INITIALIZE	Header	A9AB	034	"4"	
5	INITIALIZE	Header	A9AC	020	" "	
6	INITIALIZE	Header	A9AD	010	"P"	Check for Library#4 first, then
7	INITIALIZE	Header	A9AE	008	"H"	Create IO buffer 880C000...
8	INITIALIZE	Header	A9AF	02D	"_"	- unless it's already there
9	INITIALIZE	-HP 41Z	A9B0	04E	C=0 ALL	Programmable!
10	INITIALIZE		A9B1	15C	PT= 6	
11	INITIALIZE		A9B2	110	LD@PT- 4	put "4000" in ADR field
12			A9B3	330	FETCH S&X	
13	first we must check for Lib#4		A9B4	106	A=C S&X	put byte in A[S&X]
14	because [CHKBUF] resides		A9B5	130	LDI S&X	
15	in there (!)		A9B6	023	CON:	signature value
16			A9B7	366	?A#C S&X	are they different?
17	INITIALIZE		A9B8	0BB	JNC +23d	no, next thing
18	INITIALIZE		A9B9	320	DSPTOG	display ON
19	INITIALIZE		A9BA	3C1	?PNC XQ	Enable & Clear Disp
20	INITIALIZE		A9BB	0B0	->2CF0	[CLLCDE]
21	INITIALIZE		A9BC	3BD	?PNC XQ	Message Line
22	INITIALIZE		A9BD	01C	->07EF	[MESSL]
23	INITIALIZE		A9BE	00E	"N"	
24	INITIALIZE		A9BF	00F	"O"	
25	INITIALIZE		A9C0	020	" "	
26	INITIALIZE		A9C1	00C	"L"	
27	INITIALIZE		A9C2	009	"I"	"NO LIBRARY"
28	INITIALIZE		A9C3	002	"B"	
29	INITIALIZE		A9C4	012	"R"	
30	INITIALIZE		A9C5	001	"A"	
31	INITIALIZE		A9C6	012	"R"	
32	INITIALIZE		A9C7	219	"Y"	
33	INITIALIZE		A9C8	3DD	?PNC XQ	Left Justified format
34			A9C9	0AC	->2BF7	[LEFTJ]
35	halting here means the rest		A9CA	108	SETF 8	
36	of polling points won't be done!		A9CB	201	?PNC XQ	
37			A9CC	070	->1C80	[MSG105]
38	INITIALIZE		A9CD	3ED	?PNC GO	HALT execution
39	INITIALIZE		A9CE	08A	->22FB	[ERR110]



40	INITIALIZE	NEXT	A9CF	0D9 ?NC XQ	←	Check for CX OS
41	INITIALIZE		A9D0	110 ->4436		[NOCX4]
42	INITIALIZE	ZSTACK	A9D1	130 LDI S&X		
43	INITIALIZE		A9D2	008 CON: 8		Buffer id# in C(0)
			A9D3	2A9 ?NC XQ		Check for Buffer
b11	non-zero	-	A9D4	10C ->43AA		[CHKBF4]
b10	L4	-	A9D5	0A3 JNC +20d		Not Found - Create it !!
b9		-	A9D6	038 READ DATA		reload id# in MS field
b8	L3	-	A9D7	2DC PT= 13		
b7		T	A9D8	210 LD@PT- 8		Buffer id#
b6	L2	Z	A9D9	2F0 WRIT DATA		
b5		Y	A9DA	375 PORT DEP: ←		0.- write XYZT into [b3-b6]
b4	L1	X	A9DB	03C XQ		to initialize L1 & L2
b3		-	A9DC	196 ->A996		[SYNCH2]
b2	L0	L	A9DD	130 LDI S&X		A[S&X] holds b7 address
b1			A9DE	004 CON: 4		adds 4 to it
b0	Header		A9DF	206 C=C+A S&X		b11 addr
			A9E0	270 RAM SLCT		non-zero the last buffer reg
57	INITIALIZE		A9E1	2F0 WRIT DATA		this should do it
58	INITIALIZE		A9E2	04E C=0 ALL		
59	INITIALIZE		A9E3	270 RAM SLCT		Select Chip0
60	INITIALIZE		A9E4	2CC ?FSET 13		Exit if PRG Running
61	INITIALIZE		A9E5	360 ?C RTN		
62	INITIALIZE		A9E6	3AD PORT DEP:		Show X,Y
63	INITIALIZE		A9E7	08C GO		(Respects ZMODE)
64	INITIALIZE		A9E8	02A ->AC2A		[ZAVIEW]
65	INITIALIZE	CREATE	A9E9	066 A<>B S&X ←		First free reg. address (from .END.)
66	INITIALIZE		A9EA	04E C=0 ALL		
67	INITIALIZE		A9EB	270 RAM SLCT		Select Chip0
68	INITIALIZE		A9EC	285 ?NC XQ		
69	INITIALIZE		A9ED	014 ->05A1		[MEMLFT]
70	INITIALIZE		A9EE	106 A=C S&X		number of "free regs"
71	INITIALIZE		A9EF	130 LDI S&X		Must be at least 12 free regs.
72	INITIALIZE		A9F0	00C CON: 12		(header + 5 complex stack levels)
73	INITIALIZE		A9F1	306 ?A<C S&X		Enough Memory?
74	INITIALIZE		A9F2	33D PC GO		Show "No Room" msg
75	INITIALIZE		A9F3	0C3 ->30CF		[NORMER]
76	INITIALIZE		A9F4	0E6 B<>C S&X		First free reg. address (from .END.)
77	INITIALIZE		A9F5	270 RAM SLCT		select buffer header
78	INITIALIZE		A9F6	106 A=C S&X		buffer address in A S&X
79	INITIALIZE		A9F7	2DC PT= 13		
80	INITIALIZE		A9F8	210 LD@PT- 8		Buffer id#
81	INITIALIZE		A9F9	210 LD@PT- 8		Buffer id#
82	INITIALIZE		A9FA	010 LD@PT- 0		Buffer size
83	INITIALIZE		A9FB	310 LD@PT- C		Buffer size
84	INITIALIZE		A9FC	2F0 WRIT DATA		Store Header Reg
85	INITIALIZE		A9FD	2EB JNC -35d		A9CA
1	BUFFER	NOBUFER	A9FE	215 ?NC XQ		Build Msg - all cases
2	BUFFER		A9FF	0FC ->3F85		[APRMSG2]
3	BUFFER		AA00	00E "N"		
4	BUFFER		AA01	00F "O"		
5	BUFFER		AA02	020 " "		
6	BUFFER		AA03	01A "Z"		
7	BUFFER		AA04	02D " "		MSG: "NO Z-STACK"
8	BUFFER		AA05	013 "S"		
9	BUFFER		AA06	014 "T"		
10	BUFFER		AA07	001 "A"		
11	BUFFER		AA08	003 "C"		
12	BUFFER		AA09	20B "K"		
13	BUFFER		AA0A	1F1 ?NC GO		Left!, Show and Halt
14	BUFFER		AA0B	0FE ->3F7C		[APEREX]

Notice how we finish with **ZAVIEW** to show the current complex number in the stack upon buffer creation. [CHKBUF] does not create the buffer, but reads its address into register A and the content of the header into register C.



## Appendix 2. Complex Keyboard key maps.

The following table shows the detailed key map supported by the **ZKBRD** complex keyboard function launcher.

Level					Function	Level					Function
I	II	III	IV	V	Name	I	II	III	IV	V	Name
Z	1/X				ZINV	Z		$\sqrt{x}$			-HP 41Z
Z	SQRT				ZSQRT	Z		$Y^X$			$W^Z$
Z	LOG				ZLOG	Z		$X^2$			$Z^2$
Z	LN				ZLN	Z		$10^X$			ZALOG
Z	$X \leftrightarrow Y$				$Z \leftrightarrow W$	Z		$e^X$			ZEXP
Z	RDN				ZRDN	Z		$X \leftrightarrow Y$			ZTRP
Z	SIN				ZSIN	Z		RDN			ZRUP
Z	COS				ZCOS	Z		ASIN			ZASIN
Z	TAN				ZTAN	Z		ACOS			ZACOS
Z	XEQ				$\wedge$ IMG _	Z		ATAN			ZATAN
Z	STO				ZSTO _ _	Z		ASN			ZK?YN
Z	RCL				ZRCL _ _	Z		LBL			ZSIGN
Z	SST				$Z \leftrightarrow$ _ _	Z		GTO			Z*I
Z	ENT^				ZENTER^	Z		CAT			$\wedge$ IMG _
Z	CHS				ZNEG	Z		ISG			ZCONJ
Z	EEX				$Z^X$	Z		RTN			$X^Z$
Z	-				Z-	Z		CLX			CLZ
Z	+				Z+	Z		$X=Y?$			$Z=W?$
Z	*				Z*	Z		SF			ZNORM
Z	/				Z/	Z		CF			ZMOD
Z	0-9				Z0-Z9	Z		FS?			ZARG
Z	R/S				ZAVIEW	Z		$X \leq Y?$			$Z=WR?$
Z	,	0-9			ZJ0-ZJ9	Z		BEEP			ZTONE
Z	Z	1/X			$W^1/Z$	Z		P-R			ZREC
Z	Z	SQRT			ZPSI	Z		R-P			ZPOL
Z	Z	LOG			ZLNG	Z		$X>Y?$			$Z=I?$
Z	Z	LN			$e^Z$	Z		FIX			ZRND
Z	Z	$X \leftrightarrow Y$			$Z \leftrightarrow V$	Z		SCI			ZINT
Z	Z	RDN			ZQRT	Z		ENG			ZFRC
Z	Z	XEQ			ZIMAG^	Z		$X=0?$			$Z=0?$
Z	Z	STO			ZREAL^	Z		PI			ZGAMMA
Z	Z	RCL			Z/	Z		LASTX			LASTZ
Z	Z	SST			CLSTZ	Z		VIEW			ZVIEW _ _
Z	Z	ENT^			ZRPL	Z			SIN		ZSINH
Z	Z	EEX			$Z^1/X$	Z			COS		ZCOSH
Z	Z	-			$Z\#W?$	Z			TAN		ZTANH
Z	Z	7			ZWDET	Z				SIN	ZASINH
Z	Z	8			ZWDIST	Z				COS	ZACOSH
Z	Z	9			ZWANG	Z				TAN	ZATANH
Z	Z	+			ZREAL?	Z	Z		SQRT		ZNXTNRT _
Z	Z	4			ZIN?	Z	Z		LN		ZNXTLN
Z	Z	5			ZWCROSS	Z	Z		SIN		ZNXTASN
Z	Z	*			ZIMAG?	Z	Z		COS		ZNXTACS
Z	Z	1			ZUNIT?	Z	Z		TAN		ZNXTATN
Z	Z	2			ZWLINE	Z	Z			LOG	ZKBS
Z	Z	/			$Z\#0?$	Z	Z			LN	ZYBS
Z	Z	0			ZOUT?	Z	Z			COS	ZIBS
Z	Z	,			ZWDOT	Z	Z			TAN	ZJBS
Z	Z	Z			$Z \leftrightarrow U$	Z	Z			SIN	ZWL
						Z	Z			SQRT	EIZ/IZ

## **Appendix 3.- Formula Compendium.**

Elementary complex numbers and functions – By W. Doug Wilder.

$$\begin{aligned}
j &= \sqrt{-1} = e^{j\frac{\pi}{2}} = 1 \angle 90^\circ & j^2 &= e^{j\pi} = 1 \angle 180^\circ = -1 & -j &= j^{-1} \\
Z &= \operatorname{Re}(Z) + j\operatorname{Im}(Z) = x + jy = re^{j\theta} = r \angle \theta = r \cos \theta + j r \sin \theta & r &= |Z| = \sqrt{x^2 + y^2} & \theta &= \tan^{-1}(y/x) \\
\bar{Z} &= Z^* = x - jy = re^{-j\theta} = r \angle -\theta & Z + Z^* &= 2\operatorname{Re}(Z) & Z - Z^* &= j2\operatorname{Im}(Z) \\
(Z_1 Z_2)^* &= Z_1^* Z_2^* & (Z_1 Z_2)^* &= Z_2^* Z_1^* & (Z_1 / Z_2)^* &= Z_1^* / Z_2^* & (Z_1 + Z_2)^* &= Z_1^* + Z_2^* & (Z_1 - Z_2)^* &= Z_1^* - Z_2^* \\
|Z_1 Z_2| &= |Z_1| |Z_2| & |Z_1 / Z_2| &= |Z_1| / |Z_2| & |Z_1 Z_2^*| &= |Z_1 Z_2| & r^2 &= |Z|^2 = ZZ^* = x^2 + y^2 \\
|Z_1 + Z_2|^2 &= (Z_1 + Z_2)(Z_1 + Z_2)^* = Z_1 Z_1^* + Z_1 Z_2^* + Z_2 Z_1^* + Z_2 Z_2^* = |Z_1|^2 + 2\operatorname{Re}(Z_1 Z_2^*) + |Z_2|^2 \\
Z_1 + Z_2 &= (x_1 + x_2) + j(y_1 + y_2) & Z_1 - Z_2 &= (x_1 - x_2) + j(y_1 - y_2) & |Z_1 + Z_2| &\leq |Z_1| + |Z_2| \\
Z_1 Z_2 &= (x_1 x_2 - y_1 y_2) + j(x_1 y_2 + y_1 x_2) = r_1 r_2 \angle (\theta_1 + \theta_2) = r_1 r_2 e^{j(\theta_1 + \theta_2)} & \operatorname{Re}(1/Z^*) &= \operatorname{Re}(1/Z) \\
Z_1 / Z_2 &= (x_1 x_2 + y_1 y_2 + j(y_1 x_2 - x_1 y_2)) / (x_2^2 + y_2^2) = r_1 / r_2 \angle (\theta_1 - \theta_2) & Z^{-1} &= (x - jy) / (x^2 + y^2) = e^{-j\theta} / r \\
\bar{Z}_1 Z_2 &= (x_1 x_2 + y_1 y_2) + j(x_1 y_2 - y_1 x_2) = r_1 r_2 \angle (\theta_2 - \theta_1) = Z_1 \bullet Z_2 + j Z_1 \times Z_2 & (Z_1, Z_2 &= 2D \text{ vectors}) \\
Z^2 &= x^2 - y^2 + j2xy = r^2 e^{j2\theta} & Z^{1/2} &= r^{1/2} e^{j\theta/2} \text{ (principal)} \\
\pi &= 3.14159\ 26535\ 89793\ 23846\ 264... & e^{Z+j2\pi n} &= e^Z & e &= 2.7182818284\ 59045\ 23536\ 028... \\
e^Z &= e^x e^{jy} = e^x \angle y = e^x \cos y + j e^x \sin y & e^{Z_1} e^{Z_2} &= e^{Z_1 + Z_2} & (e^{Z_1})^{Z_2} &= e^{Z_1 Z_2} \quad (-\pi < \theta_1 \leq \pi) \\
e^{-Z} &= 1/e^Z & e^{hZ} &= Z & e^Z &= \cosh(Z) + \sinh(Z) = \cos(jZ) - j \sin(jZ) \\
\ln Z &= \ln r + j\theta = \ln \sqrt{x^2 + y^2} + j \tan^{-1}(y/x) + j2\pi n & e^{-jZ} / (-jZ) &= (\sin Z + j \cos Z) / Z = h_0^{(2)}(Z) \\
Z_1 \ln Z_2 &= \ln Z_2^{Z_1} & \ln Z_1 + \ln Z_2 &= \ln(Z_1 Z_2) & \ln 0 &= \infty & \ln e^Z &= Z \\
Z_2^{Z_1} &= e^{Z_1 \ln Z_2} & \operatorname{Log}_a Z &= \ln Z / \ln a & \ln j &= 0 + j\pi/2 & \ln 1 &= 0 & \ln(-1) &= 0 + j\pi \\
\frac{\partial}{\partial Z} Z^a &= a Z^{a-1} & \frac{\partial}{\partial Z} e^{aZ} &= a e^{aZ} & \frac{\partial}{\partial Z} a^Z &= a^Z \ln a & \frac{\partial}{\partial Z} \ln Z &= \frac{1}{Z} & \int e^{aZ} dZ &= \frac{e^{aZ}}{a} & \int \frac{dZ}{Z} &= \ln Z \\
e^Z &= 1 + \frac{Z}{1!} + \frac{Z^2}{2!} + \frac{Z^3}{3!} + ... & \ln Z &= \frac{2}{1} \left( \frac{Z-1}{Z+1} \right) + \frac{2}{3} \left( \frac{Z-1}{Z+1} \right)^3 + \frac{2}{5} \left( \frac{Z-1}{Z+1} \right)^5 + ... \quad (\operatorname{Re}(Z) \geq 0) \\
\sin Z &= (-j/2)(e^{jZ} - e^{-jZ}) = (-j/2)(e^{jZ} - 1/e^{jZ}) & \sin^{-1} Z &= -j \ln(jZ + \sqrt{1-Z^2}) \\
\cos Z &= (1/2)(e^{jZ} + e^{-jZ}) = (1/2)(e^{jZ} + 1/e^{jZ}) & \cos^{-1} Z &= -j \ln(Z + \sqrt{Z^2-1}) \\
\tan Z &= -j \frac{e^{jZ} - e^{-jZ}}{e^{jZ} + e^{-jZ}} = -j \frac{e^{j2Z} - 1}{e^{j2Z} + 1} & \tan^{-1} Z &= -\frac{j}{2} \ln \left( \frac{1+jZ}{1-jZ} \right) & \tan^{-1}(Z_2/Z_1) &= -j \ln \left( \frac{Z_1 + jZ_2}{\sqrt{Z_1^2 + Z_2^2}} \right) \\
\frac{\partial}{\partial Z} \cos Z &= -\sin Z & \frac{\partial}{\partial Z} \sin Z &= \cos Z & \int \cos Z dZ &= \sin Z & \int \sin Z dZ &= -\cos Z \\
\sin Z &= Z - \frac{Z^3}{3!} + \frac{Z^5}{5!} - \frac{Z^7}{7!} + ... & \cos Z &= 1 - \frac{Z^2}{2!} + \frac{Z^4}{4!} - \frac{Z^6}{6!} + ... \\
\sinh Z &= (1/2)(e^Z - e^{-Z}) = -j \sin(jZ) & \sinh^{-1} Z &= \ln(Z + \sqrt{Z^2+1}) \\
\cosh Z &= (1/2)(e^Z + e^{-Z}) = \cos(jZ) & \cosh^{-1} Z &= \ln(Z + \sqrt{Z^2-1}) \\
\tanh Z &= \frac{e^Z - e^{-Z}}{e^Z + e^{-Z}} = \frac{e^{2Z} - 1}{e^{2Z} + 1} = -j \tan(jZ) & \tanh^{-1} Z &= \frac{1}{2} \ln \left( \frac{1+Z}{1-Z} \right) \\
\cos Z &= \cos x \cosh y - j \sin x \sinh y & \sin Z &= \sin x \cosh y + j \cos x \sinh y
\end{aligned}$$

## Appendix 4.- Quick Reference Guide.

The tables in the following four pages list all 41Z functions in alphabetical order.

#	Function	Description	Formula	Input	Output	Comments
1	<b>-HP 41Z</b>	<b>Initializes Complex Stack</b>	<b>Z=XY; W=ZT</b>	<b>none</b>	<b>Initializes Z buffer &amp; ZAVIEW</b>	<b>runs on CALC ON</b>
2	<b>W^1/Z</b>	Complex $Y^{1/X}$	$w^{1/z} = \exp(\ln w / z)$	w in <b>W</b> ; z in <b>Z</b> (XY)	$w^{1/z}$ in <b>Z</b> (XY)	<i>Drops Buffer</i>
3	<b>W^Z</b>	Complex $Y^X$	$w^z = \exp(z \cdot \ln w)$	w in <b>W</b> ; z in <b>Z</b> (XY)	$w^z$ in <b>Z</b> (XY)	<i>Drops Buffer</i>
4	<b>X^1/Z</b>	Hybrid $Y^X$	$a^{1/z} = \exp(1/z \cdot \ln a)$	x in X reg; z in Y,Z regs	$x^{1/z}$ in <b>Z</b> (XY)	<i>does LastZ</i>
5	<b>X^Z</b>	Hybrid $Y^X$	$a^z = \exp(z \cdot \ln a)$	x in X reg; z in Y,Z regs	$x^z$ in <b>Z</b> (XY)	<i>does LastZ</i>
6	<b>Z+</b>	Complex addition	$(x1+x2) + i(y1+y2)$	w in <b>W</b> ; z in <b>Z</b> (XY)	w+z in <b>Z</b> (XY)	<i>Drops Buffer, LastZ</i>
7	<b>Z-</b>	Complex subtraction	$w-z = w + (-z)$	w in <b>W</b> ; z in <b>Z</b> (XY)	w-z in <b>Z</b> (XY)	<i>Drops Buffer, LastZ</i>
8	<b>Z*</b>	Complex multiplication	$(x1*x2 - y1*y2) + i(x1*y2 + y1*x2)$	w in <b>W</b> ; z in <b>Z</b> (XY)	w*z in <b>Z</b> (XY)	<i>Drops Buffer, LastZ</i>
9	<b>Z/</b>	Complex division	$w/z = w * (1/z)$	w in <b>W</b> ; z in <b>Z</b> (XY)	w/z in <b>Z</b> (XY)	<i>Drops Buffer, LastZ</i>
10	<b>Z^1/X</b>	Hybrid $Y^X$	$z^{1/n} = r^{1/n} * \exp(i*Arg/n)$	x in X reg; z in Y,Z regs	$z^{1/x}$ in <b>Z</b> (XY)	<i>does LastZ</i>
11	<b>Z^2</b>	Complex $X^2$	$z^2 = r^2 * \exp(2i*Arg)$	z in <b>Z</b> (XY)	$z^2$ in <b>Z</b> , (XY)	<i>does LastZ</i>
12	<b>Z^3</b>	Cubic power	$z = z^3$	z in <b>Z</b> (Im in Y, Re in X)	result in <b>Z</b> (XY)	<i>more accurate than Z^X</i>
13	<b>Z^X</b>	Hybrid $Y^X$	$z^n = r^n * \exp(i*n*Arg)$	x in X reg; z in Y,Z regs	$z^x$ in <b>Z</b> , (XY)	<i>does LastZ</i>
14	<b>Z=0?</b>	Is z=0?	is z=0?	z in <b>Z</b> (XY)	YES/NO (skips if false)	
15	<b>Z=i?</b>	Is z=i?	is z=i?	z in <b>Z</b> (XY)	YES/NO (skips if false)	
16	<b>Z=W?</b>	Is z=w?	is z=w?	w in <b>W</b> ; z in <b>Z</b> (XY)	YES/NO (skips if false)	
17	<b>Z=WR?</b>	are z & w equal if rounded?	is Rnd(z)=Rnd(w)?	w in <b>W</b> ; z in <b>Z</b> (XY)	YES/NO (skips if false)	
18	<b>Z#0?</b>	is z equal to zero?	is z#0?	z in <b>Z</b> (XY)	YES/NO (skips if false)	
19	<b>Z#W?</b>	Is z equal to w?	is z=w?	w in <b>W</b> ; z in <b>Z</b> (XY)	YES/NO (skips if false)	
20	<b>ZACOS</b>	Complex ACOS	$\text{acos } z = \pi/2 - \text{asin } z$	z in <b>Z</b> (XY)	$\text{acos}(z)$ in <b>Z</b> (XY)	<i>does LastZ</i>
21	<b>ZALOG</b>	Complex $10^X$	$e^{[z \cdot \ln(10)]}$	z in <b>Z</b> (XY)	$10^z$ in <b>Z</b> (X,Y) and ALPHA	<i>does LastZ</i>
22	<b>ZASIN</b>	Complex ASIN	$\text{asin } z = -i * \text{asinh}(iz)$	z in <b>Z</b> (XY)	$\text{asin}(z)$ in <b>Z</b> , (XY)	<i>does LastZ</i>
23	<b>ZATAN</b>	Complex ATAN	$\text{atan } z = -i * \text{atanh}(iz)$	z in <b>Z</b> (XY)	$\text{atan}(z)$ in <b>Z</b> (XY)	<i>does LastZ</i>
24	<b>ZCOS</b>	Complex COS	$\cos z = \cosh(iz)$	z in <b>Z</b> (XY)	$\cos(z)$ in <b>Z</b> (XY)	<i>does LastZ</i>
25	<b>ZEXP</b>	Complex $e^X$	$e^x * e^{iy}$	z in <b>Z</b> (XY)	$e^z$ in <b>Z</b> (XY) and ALPHA	<i>does LastZ</i>
26	<b>ZHACOS</b>	Complex Hyp. ACOS	$\text{acosh } z = \ln[z + \text{SQ}(z^2 - 1)]$	z in <b>Z</b> (XY)	$\text{acosh}(z)$ in <b>Z</b> (XY)	<i>does LastZ</i>
27	<b>ZHASIN</b>	Complex Hyp. ASIN	$\text{asinh } z = \ln[z + \text{SQ}(z^2 + 1)]$	z in <b>Z</b> (XY)	$\text{asinh}(z)$ in <b>Z</b> (XY)	<i>does LastZ</i>
28	<b>ZHATAN</b>	Complex Hyp. ATAN	$\text{atanh } z = 1/2 * \ln[(1+z)/(1-z)]$	z in <b>Z</b> (XY)	$\text{atanh}(z)$ in <b>Z</b> (XY)	<i>does LastZ</i>
29	<b>ZHCOS</b>	Complex Hyp. COS	$\cosh z = 1/2 * [e^z + e^{-z}]$	z in <b>Z</b> (XY)	$\cosh(z)$ in <b>Z</b> (XY)	<i>does LastZ</i>
30	<b>ZHSIN</b>	Complex Hyp. SIN	$\sinh z = 1/2 * [e^z - e^{-z}]$	z in <b>Z</b> (XY)	$\sinh(z)$ in <b>Z</b> (XY)	<i>does LastZ</i>
31	<b>ZHTAN</b>	Complex Hyp. TAN	$\tanh z = (e^z - e^{-z}) / (e^z + e^{-z})$	z in <b>Z</b> (XY)	$\tanh(z)$ in <b>Z</b> (XY)	<i>does LastZ</i>
32	<b>ZIMAG?</b>	is Im(z)=0?	is Im(z)=0?	z in <b>Z</b> (XY)	YES/NO (skips if false)	
33	<b>ZIN?</b>	Is z inside the unit circle?	is  z <1?	z in <b>Z</b> (XY)	YES/NO (skips if false)	
34	<b>ZINT?</b>	Checks if Z is an integer number	are Im(z)=0 and FRC[Re(z)]=0?	z in <b>Z</b> (Im in Y, Re in X)	YES/NO (skips if false)	<i>used in Bessel fncs</i>

#	Function	Description	Formula	Input	Output	Comments
35	ZINV	Complex Inversion	$x/(x^2 + y^2) - i y/(x^2 + y^2)$	z in Z (XY)	1/z in Z (XY) and ALPHA	does LastZ
36	ZLN	Complex LN	$\ln(z) = \ln(r) + i \cdot \text{Arg}$	z in Z (XY)	$\ln(z)$ in Z (XY)	does LastZ
37	ZLOG	Complex LOG	$\log(z) = \ln(z)/\ln(10)$	z in Z (XY)	$\log(z)$ in Z (X,Y)	does LastZ
38	ZNEG	Complex CHS	$-z = -x - iy$	z in Z (XY)	-z in Z (XY)	does LastZ
39	ZOUT?	Is z outside the unit circle?	is $ z  > 1$ ?	z in Z (XY)	YES/NO (skips if false)	
40	ZPI*	Product by pi	$z \cdot \pi$	z in Z (XY)	result in Z (XY)	more accurate than FOCAL
41	ZREAL?	Is Re(z)=0?	Is $\text{Re}(z)=0$ ?	z in Z (XY)	YES/NO (skips if false)	
42	ZRND	Rounds Z to display settings	rounded values to display	z in Z (XY)	Rounded Re & Im in Z (XY)	does LastZ
43	ZSIN	Complex SIN	$\sin z = -i \cdot \sinh(iz)$	z in Z (XY)	$\sin(z)$ in Z (XY)	does LastZ
44	ZSQRT	Complex SQRT (Direct)	$\text{sqr}(z) = \text{sqr}(r) \cdot e^{i \cdot \text{Arg}/2}$	z in Z (XY)	main value of $z^{1/2}$ in Z (XY)	does LastZ
45	ZTAN	Complex TAN	$\tan z = -i \cdot \tanh(iz)$	z in Z (XY)	$\tan(z)$ in Z (XY)	does LastZ
46	ZUNIT?	Is z on the unit circle?	is $ z =1$ ?	z in Z (XY)	YES/NO (skips if false)	
47	-ZSTACK	Section Header	n/a	none	Shows "Running..." msg	
48	CLZ	Clears Z	$\text{Re}(z)=0=\text{Im}(z)$	none	Z level (XY) cleared	
49	CLZST	Clears Z-Stack	n/a	none	Z-Stack Cleared	
50	LASTZ	Complex LASTX	n/a	none	Last z in X,Y regs;	Lifts Buffer
51	ZAVIEW	Shows Complex Z	n/a	z in Z (XY)	Shows z in ALPHA	
52	ZENTER^	Copies Z into the W register	n/a	z in Z (XY)	Pushes z one level Up	Lifts Buffer
53	Z<>__	Complex Exchange	n/a	Reg# as suffix	Exchanges Z with regs contents	Prompting
54	Z<>ST__	Exchanges Z and L#	n/a	z in XY, level# in prompt	z in L#; L# in L1 & X,Y	Prompting
55	ZTRP	Exchanges Re(Z) and Im(Z)	$z_{\text{Trp}} = y + ix$	z in Z (XY)	$\text{Im}(z)$ in X, $\text{Re}(z)$ in Y	does LastZ
56	Z<>W	Exchange Z and W (L2)	n/a	w in W, z in Z (XY)	z in L2 & Z,T w in L1 & X,Y	
57	ZIMAG^	Enter imaginary number	n/a	$\text{Im}(z)$ in X	zero in X; $\text{Im}(z)$ in Y	Lifts Buffer
58	ZRCL__	Complex RCL	n/a	Reg# as suffix	z in X,Y - lifts stack	Lifts Buffer, Prompting
59	ZRDN	Z-Stack Roll Down	n/a	Stack Levels	Rolls Down stack	Drops Buffer
60	ZREAL^	Enter Real number in Z	n/a	$\text{Re}(z)$ in X	$\text{Re}(z)$ in X; Zero in Y	Lifts Buffer
61	ZRPL^	Replicates z in all levels	$L4=L3=L2=L1$	z in Z (XY)	z in all 4 levels	Lifts Buffer
62	ZRUP	Z-Stack Roll Up	n/a	Stack Levels	Rolls Up stack	Lifts Buffer
63	ZSTO__	Complex STO	n/a	Reg# as suffix	Stores z in consecutive regs	Prompting
64	ZVIEW__	Complex View	n/a	Reg# as suffix	Shows z in ALPHA	Prompting
1	^IM/AG_	Natural Data Entry	Re ^ IM or r ^ arg	Re(z) in X, Im(Z) as suffix	z in Z (XY), stack lifted	Prompting, Lifts Buffer
2	1/Z	alternative ZINV (Uses TOPOL)	$1/r \cdot \exp(-i \arg)$	z in Z (XY)	1/z in X,Y registers and ALPHA	does LastZ
3	e^Z	alternative ZEXP	$e^z = e^x \cdot (\cos y + i \sin y)$	z in Z (XY)	$\exp(z)$ in Z (XY)	does LastZ
4	EIZ/IZ	spherical hankel h1(0,z)	$h^{(1)}(0,z) = \exp(i \cdot z) / i \cdot z$	z in Z (XY)	result in Z (XY)	does LastZ
5	NXTACS	Next ACOS Value	$z_{1,2} = +/- z_0 + 2p$	z0 in Z (XY)	z1 in W, z2 in Z (XY)	does LastZ
6	NXTASN	Next ASIN Value	$z_{1,2} = +/- z_0 + 2p/2$	z0 in Z (XY)	z1 in W, z2 in Z (XY)	does LastZ

#	Function	Description	Formula	Input	Output	Comments
7	NXTATN	Next ATAN value	$z1,2 = z0 \pm p$	$z0$ in <b>Z</b> (XY)	$z1$ in <b>W</b> , $z2$ in <b>Z</b> (XY)	does LastZ
8	NXTLN	Next Ln(z)	$\text{next}(k) = \text{Ln}(z) + 2kpJ$	LN(z) in <b>Z</b> (XY) regs	$z1$ in <b>W</b> , $z2$ in <b>Z</b> (XY)	does LastZ
9	NXTRTN	Next Complex Root	$\text{next}(k) = z^{1/n} * e^{(2kp/n)J}$	$n$ in X reg.; $z^{1/n}$ in Z,Y regs	$z^{1/n} * e^{(2p/n)J}$ in <b>Z</b> (XY)	does LastZ
10	SQRTZ	Alternative SQRT (Uses TOPOL)	$\text{sqr}(z) = \text{sqr}(r) * e^{(i*Arg/2)}$	$z$ in <b>Z</b> (XY)	main value of $z^{1/2}$ in <b>Z</b> (XY)	does LastZ
11	Z*I	Multiplies by I (90 deg. Rotation)	$iz = -\text{Im}(z) + i \text{Re}(z)$	$z$ in <b>Z</b> (XY)	$z*i$ in L1 & XY	does LastZ
12	ZCHSX	Sign Change by X	$(-1)^n * z$	$x$ in X reg; $z$ in Y,Z regs	$\{(-1)^x * z\}$ in <b>Z</b> (XY)	does LastZ
13	ZGEU	Euler's gamma constant	$\gamma = 0.577215665$	none	$\gamma$ constant as complex	Lifts Buffer
14	ZK?YN	Block Key Assignments	$n/a$	prompt-driven	Makes / Removes assignments	may do PACKING
15	ZKBRD _	Complex keyboard launcher	$n/a$	Prompt-driven	Launches function	prompting, launcher
16	ZST+ _ _	STO Addition	$cR + z$	$z$ in <b>Z</b> (XY)	Adds $z$ to complex register#	prompting
17	ZST- _ _	STO Subtraction	$cR - z$	$z$ in <b>Z</b> (XY)	Subtract $z$ from complex register#	prompting
18	ZST* _ _	STO Multiply	$cR * z$	$z$ in <b>Z</b> (XY)	Multiplies $z$ to complex register#	prompting
19	ZST/ _ _	STO Divide	$cR / z$	$z$ in <b>Z</b> (XY)	Divides complex register by $z$	prompting
20	ZWLOG	Base-w Logarithm	base $w$ in W, arg. In Z	$w$ in <b>W</b> , $z$ in <b>Z</b> (XY)		Drops Buffer, LastZ
21	ZVECTOR	Section Header	$n/a$	none	Displays Revision Number	
22	POLAR	Sets POLAR mode on	sets the Polar flag in Buffer	none	shows $\text{Re}(z) + j \text{Im}(z)$	
23	RECT	Sets RECT mode on	clears the Polar flag in Buffer	none	shows $r \angle \arg$	
24	ZARG	Argument of Z	$\text{atan}(y/x)$	$z$ in <b>Z</b> (XY)	$\text{Arg}(z)$ in X, (Y reg void)	zeroes Y, LastZ
25	ZCONJ	Complex Conjugate	$\text{conj} = x - iy$	$z$ in <b>Z</b> (XY)	Inverts sign of $\text{Im}(z)$	does LastZ
26	ZMOD	Module of Z	$ z  = \text{sqr}(x^2 + y^2)$	$z$ in <b>Z</b> (XY)	$\text{Mod}(z)$ in X, (Y reg void)	zeroes Y, LastZ
27	ZNORM	Norm of Z (i.e. square of Module)	$ z ^2 =  z  ^2$	$z$ in <b>Z</b> (XY)	$(\text{mod}(z)^2)$ in X,Y	zeroes Y, LastZ
28	ZPOL	Converts to Polar notation	R-P	$z$ in <b>Z</b> (XY)	$\text{Mod}(z)$ in X; $\text{Arg}(z)$ in Y	does LastZ
29	ZREC	Convers to Rectangular notation	P-R	$\text{Mod}(z)$ in X; $\text{Arg}(z)$ in Y	$\text{Re}(z)$ in X; $\text{Im}(z)$ in Y	does LastZ
30	ZSIGN	Complex SIGN	$\text{sign} = z/ z $	$z$ in <b>Z</b> (XY)	$z/\text{Mod}(z)$ in X,Y	does LastZ
31	ZWANG	Angle between Z and W	$\text{arg}(zw) = \text{Arg}(z) - \text{Arg}(w)$	$z$ in <b>Z</b> (XY)	$\text{ang}(z,w)$ in X (Y void)	Drops Buffer LastZ
32	ZWCROSS	Cross product of Z and W	$z \times w =  z  *  w  * \sin(\text{Angle})$	$w$ in <b>W</b> , $z$ in <b>Z</b> (XY)	$z \times w$ in X (Y void)	Drops Buffer LastZ
33	ZWDET	Determinant of Z and W	$ zw  = x2*y1 - y2*x1$	$w$ in <b>W</b> , $z$ in <b>Z</b> (XY)	$\det(z,w)$ in X (Y void)	Drops Buffer LastZ
34	ZWDIST	Distance between Z and W	$ w-z  = \text{SQR}[(x2-x1)^2 - (y2-y1)^2]$	$w$ in <b>W</b> , $z$ in <b>Z</b> (XY)	$\text{dist}(z,x)$ in X (Y void)	Drops Buffer LastZ
35	ZWDOT	Dot product of Z and W	$z*w = x1*x2 + y1*y2$	$w$ in <b>W</b> , $z$ in <b>Z</b> (XY)	$\text{dot}(z,w)$ in X, (Y void)	Drops Buffer LastZ
36	ZWLINE	Line equation defined by Z and W	$a = (y1-y2) / (x1-x2)$	$w$ in <b>W</b> , $z$ in <b>Z</b> (XY)	$y = ax + b$ in ALPHA; $b$ in Y, $a$ in X	Drops Buffer LastZ
37	-HL ZMATH	Section Header	Calculates $2^x - 1$	$x$ in X	Result in X	used in ZZETA
38	ZAWL	Inverse of Lambert W	$z * e^z$	$z$ in <b>Z</b> (XY)	result in <b>Z</b> (XY)	does LastZ
39	ZBS#	Bessel subroutine 1st./2nd. Kind	see manual, Flag 6 controls case	$w$ in W, $z/2$ in Z	$w$ in cR00, $z/2$ in cR01	FOCAL
40	ZCI	Cosine Integral	$\text{Ci}(z) = -(z^{2/4}) F_{23}(1, 1; 2, 2; 3/2, -z^{2/4})$	$z$ in <b>Z</b> (XY)	result in <b>Z</b> (XY)	FOCAL
41	ZCRT	Complex Cubic Eq. Roots	Cubic equation roots	A,B,C,D in Z-Stack	roots in <b>V</b> , <b>W</b> , and <b>Z</b> (XY) levels	FOCAL
42	ZEI	Exponential Integral	$\text{Ei} = \gamma + \ln z  + z * F_{22}(1,1; 2,2; z)$	$z$ in <b>Z</b> (XY)	result in <b>Z</b> (XY)	FOCAL

#	Function	Description	Formula	Input	Output	Comments
43	ZERF	Error Function	$\text{erf}(z) = 2z/\sqrt{\pi} e^{-(z^2)} F_{11}(1, 3/2; z^2)$	z in <b>Z</b> (XY)	result in <b>Z</b> (XY)	FOCAL
44	ZGAMMA	Complex G(z) for z#0, -1, -2...	Lanczos approximation	z in <b>Z</b> (XY)	G(z) in <b>Z</b> (XY)	uses reflection for $\text{Re}(z)<0$
45	ZHCI	Hyperbolic Cosine Integral	$\text{Chi}(z) = (z^2/4) F_{23}(1, 1; 2, 2; 3/2, z^2/4)$	z in <b>Z</b> (XY)	result in <b>Z</b> (XY)	FOCAL
46	ZHGF	Hypergeometric Function	See manual	see manual	result in <b>Z</b> (XY)	by Jean-Marc Baillard
47	ZHSI	Hyperbolic Sine Integral	$\text{Shi}(z) = z * F_{12}(1/2, 3/2, 3/2, z^2/4)$	z in <b>Z</b> (XY)	result in <b>Z</b> (XY)	FOCAL
48	ZIBS	Bessel I function	see manual	w in <b>W</b> , z in <b>Z</b> (XY)	I(w,z) in <b>Z</b> (XY)	FOCAL
49	ZJBS	Bessel J function	see manual	w in <b>W</b> , z in <b>Z</b> (XY)	J(w,z) in <b>Z</b> (XY)	FOCAL
50	ZKBS	Bessel K function	see manual	w in <b>W</b> , z in <b>Z</b> (XY)	K(w,z) in <b>Z</b> (XY)	FOCAL
51	ZLI2	Dilogarithm	$\text{Li}(2,z) = \sum (z^k/k^2); k=1,2...$	z in <b>Z</b> (XY)	result in <b>Z</b> (XY)	by Jean-Marc Baillard
52	ZLIN	Polylogarithm	$\text{Li}(s,z) = \sum (z^k/k^s); k=1,2...$	order w in <b>W</b> ; arg. z in <b>Z</b>	result in <b>Z</b> (XY)	FOCAL
53	ZLNG	Gamma Logarithm function	Stirling method w/ correction	z in <b>Z</b> (XY)	result in <b>Z</b> (XY)	FOCAL
54	ZLRCH	Lerch Transcendent	$\text{Fi}(z,s,a) = \sum [z^k/(k+a)^s]; k=0,1...$	s,a, z in <b>U</b> , <b>W</b> , and <b>Z</b> (XY)	result in <b>Z</b> (XY)	FOCAL
55	ZPROOT	Roots of complex polynomials	Iterative	Prompt-driven	roots in <b>W</b> and <b>Z</b> (XY) levels	by Valentin Albillo
56	ZPSI	Complex Digamma	Approximation	z in <b>Z</b> (XY)	Psi(z) in X,Y regs. And ALPHA	FOCAL
57	ZQRT	Complex Quadratic Eq. Roots	Quadratic ecuation roots	A,B,C in Zstack	Calculates roots of equation	FOCAL
58	ZSHK1	Spherical Hankel h1	$h^{(1)}(w,z)$	order w in <b>W</b> ; arg. z in <b>Z</b>	result in <b>Z</b> (XY)	FOCAL
59	ZSHK2	Spherical Hankel h2	$h^{(2)}(w,z)$	order w in <b>W</b> ; arg. z in <b>Z</b>	result in <b>Z</b> (XY)	FOCAL
60	ZSI	Sine Integral	$\text{Si}(z) = z * F_{12}(1/2, 3/2, 3/2, -z^2/4)$	z in <b>Z</b> (XY)	result in <b>Z</b> (XY)	FOCAL
61	ZSOLVE	Solves for F(z)=0	Newton's method	Fnc. name in R06	Calculates one root for f(z)	FOCAL
62	ZWL	Lambert W function	see manual	z in <b>Z</b> (XY)	W(z) in <b>Z</b> (XY)	FOCAL
63	ZYBS	Bessel Y function	see manual	w in <b>W</b> , z in <b>Z</b> (XY)	Y(w,z) in <b>Z</b> (XY)	FOCAL
64	ZZETA	Riemann Zeta function	Borwein Algorithm	z in <b>Z</b> (XY)	result in <b>Z</b> (XY)	by Jean-Marc Baillard



1	<b>-ZBUFFER</b>	<b>Section Header</b>	n/a	None	None	
2	<b>CLZB</b>	Clears Z buffer	n/a	None	buffer cleared	
3	<b>L1=XY?</b>	is L1 equal to XY?	n/a	None	Y/N, skip if false	
4	<b>L1&lt;&gt;L _ _</b>	Swap L1 & Level	n/a	Level# as suffix	levels exchanged	Prompting
5	<b>L1&lt;&gt;L2</b>	Swap L1 & L2	n/a	None	levels exchanged	
6	<b>L1&lt;&gt;L3</b>	Swap L1 & L3	n/a	None	levels exchanged	
7	<b>L1&lt;&gt;L4</b>	Swap L1 & L4	n/a	None	levels exchanged	
8	<b>L1&lt;&gt;LX</b>	Swap L1 & Level	n/a	level in X	levels exchanged	
9	<b>L2=ZT?</b>	is L2 equal to ZT?	n/a	None	Y/N, skip if false	
10	<b>L2&gt;ZT</b>	Copies L2 into ZT	n/a	None	L2 copied to ZT	
11	<b>LVIEW _</b>	View Level	n/a	Level# as suffix	Transposed value!	Prompting
12	<b>LVIEWX</b>	View level by X	n/a	level in X	Transposed value!	
13	<b>PREMON</b>	Copies XY into L0 and finds Zbuffer	n/a	Re(z) in X; Im(z) in Y	none	
14	<b>PSTMON</b>	Copies XY into L1 and synch's up	n/a	Re(z) in X; Im(z) in Y	None	
15	<b>RG&gt;ZB _ _</b>	Copies registers to Z buffer	n/a	Reg# as suffix	data copied from registers	Prompting
16	<b>ST&gt;ZB</b>	Copies real stack to L1 & L2	n/a	None	stack copied to buffer	
17	<b>XY&gt;L _</b>	Copies XY into Level	n/a	Level# as suffix	XY copied to LEVEL	Prompting
18	<b>XY&gt;L0</b>	Copies XY into L0	n/a	Re(z) in X; Im(z) in Y	XY copied to L0	
19	<b>XY&gt;L1</b>	Copies XY into L1	n/a	Re(z) in X; Im(z) in Y	XY copied to L1	
20	<b>ZB&gt;RG _ _</b>	copies buffer to registers	n/a	Reg# as suffix	data copied to registers	Prompting
21	<b>ZB&gt;ST</b>	Copies L1 & L2 into real stack	n/a	None	buffer copied to Stack	
22	<b>ZBDROP</b>	Drops Z buffer one level	n/a	None	levels dropped	Drops Buffer
23	<b>ZBHEAD</b>	Zbuffer Header info	n/a	None	header register in ALPHA	
24	<b>ZBLIFT</b>	Lifts Z buffer one level	n/a	None	buffer lifted	Lifts Buffer
25	<b>ZBVIEW</b>	Shows Z Buffer	n/a	None	shows header & all levels	FOCAL
26	<b>-B UTILS</b>	<b>Section Header</b>	n/a	None	None	
27	<b>B?</b>	Does buffer exist?	n/a	buffer id# in X	YES/NO (skips if false)	CCD Module
28	<b>BLIST</b>	lists all buffers existing	n/a	none	list in Alpha	D. Yerka
29	<b>BLNG?</b>	Buffer length	n/a	buffer id# in X	buffer size in X	CCD Module
30	<b>BX&gt;RG</b>	copies buffer to registers	n/a	buffer id# in X	data copied into R00 to end	David Assm
31	<b>CLB</b>	Clear buffer	n/a	buffer id# in X	Clears buffer from memory	CCD Module
32	<b>FINDBX</b>	finds buffer address	n/a	buffer id# in X	buffer address in X	D. Yerka
33	<b>MAKEBX</b>	makes buffer in RAM	n/a	(id#,size) in X	buffer created	D. Yerka
34	<b>RG&gt;BX</b>	copies registers to buffer	n/a	Data in R00 to Rnn	Copied to Buffer	David Assm

(\*) Buffer functions have been moved to the BUFFERLAND Module, under a dedicated section for the 41Z case.

## Appendix 5.- Buffer logic function table.

		Pre-Exec					Post-Exec					
		Alpha in XY	XY to L0	XY to L1	Buffer LIFT	L2 -> ZT		Buffer DROP	XY into L1	L1,2 -> XYZT	ZAVIEW	
1	<a href="#">- HP-41 Z</a>	Initialize Buffer	yes	no	yes	no		no	no	no	yes	
2	<a href="#">W^Z</a>	Power	yes	yes	no	no	PREDUAL	yes	yes	yes	yes	POSTDUAL
3	<a href="#">Z+</a>	Addition	yes	yes	no	no	PREDUAL	yes	yes	yes	yes	POSTDUAL
4	<a href="#">Z-</a>	Substraction	yes	yes	no	no	PREDUAL	yes	yes	yes	yes	POSTDUAL
5	<a href="#">Z*</a>	Multiply	yes	yes	no	no	PREDUAL	yes	yes	yes	yes	POSTDUAL
6	<a href="#">Z/</a>	Divide	yes	yes	no	no	PREDUAL	yes	yes	yes	yes	POSTDUAL
7	<a href="#">ZWANG</a>	Angle between	yes	yes	no	no	PREDUAL	yes	yes	yes	no	PSTDUAL-2
8	<a href="#">ZWCROSS</a>	Cross Product	yes	yes	no	no	PREDUAL	yes	yes	yes	no	PSTDUAL-2
9	<a href="#">ZWDET</a>	Determinat	yes	yes	no	no	PREDUAL	yes	yes	yes	no	PSTDUAL-2
10	<a href="#">ZWDIST</a>	Distance	yes	yes	no	no	PREDUAL	yes	yes	yes	no	PSTDUAL-2
11	<a href="#">ZWDOT</a>	Dot Product	yes	yes	no	no	PREDUAL	yes	yes	yes	no	PSTDUAL-2
12	<a href="#">ZWLINE</a>	Line Equation	yes	yes	no	no	PREDUAL	yes	yes	yes	no	PSTDUAL-2
13	<a href="#">Z=W?</a>	is Z=W?	yes	no	yes	no	PREDUL-2	no	no	no	no	
14	<a href="#">Z=WR?</a>	is Z=W round?	yes	no	yes	no	PREDUL-2	no	no	no	no	
15	<a href="#">Z#W?</a>	is Z not W?	yes	no	yes	no	PREDUL-2	no	no	no	no	
16	<a href="#">Z=0?</a>	is Z Zero?	yes	no	yes	no	PREMON-2	no	no	no	no	
17	<a href="#">Z#0?</a>	is Z not zero?	yes	no	yes	no	PREMON-2	no	no	no	no	
18	<a href="#">Z=i?</a>	is Z = i?	yes	no	yes	no	PREMON-2	no	no	no	no	
19	<a href="#">ZREAL?</a>	Is Z real?	yes	no	yes	no	PREMON-2	no	no	no	no	
20	<a href="#">ZIMAG?</a>	Is Z imag?	yes	no	yes	no	PREMON-2	no	no	no	no	
21	<a href="#">ZIN?</a>	Z <1?	yes	no	yes	no	PREMON-2	no	no	no	no	
22	<a href="#">ZOUT?</a>	Z >1?	yes	no	yes	no	PREMON-2	no	no	no	no	
23	<a href="#">ZUNIT?</a>	Z =1?	yes	no	yes	no	PREMON-2	no	no	no	no	
24	<a href="#">X^Z</a>	Hybrid Power	yes	yes	no	no	PREMON	no	yes	yes	yes	POSTMON
25	<a href="#">Z^2</a>	Z^2	yes	yes	no	no	PREMON	no	yes	yes	yes	POSTMON
26	<a href="#">Z^X</a>	Z^X	yes	yes	no	no	PREMON	no	yes	yes	yes	POSTMON
27	<a href="#">ZACOS</a>	ACOS	yes	yes	no	no	PREMON	no	yes	yes	yes	POSTMON
28	<a href="#">ZACOSH</a>	ACOSH	yes	yes	no	no	PREMON	no	yes	yes	yes	POSTMON
29	<a href="#">ZALOG</a>	10^Z	yes	yes	no	no	PREMON	no	yes	yes	yes	POSTMON
30	<a href="#">ZASIN</a>	ASIN	yes	yes	no	no	PREMON	no	yes	yes	yes	POSTMON

31	<a href="#">ZASINH</a>	ASINH	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
32	<a href="#">ZATAN</a>	ATAN	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
33	<a href="#">ZATANH</a>	ATANH	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
34	<a href="#">ZCONJ</a>	X-Yj	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
35	<a href="#">ZCOS</a>	COS	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
36	<a href="#">ZCOSH</a>	COSH	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
37	<a href="#">ZDBL</a>	2*Z	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
38	<a href="#">ZEXP</a>	E^Z	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
39	<a href="#">ZHALF</a>	Z/2	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
40	<a href="#">ZINV</a>	1/Z	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
41	<a href="#">ZLN</a>	Ln(Z)	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
42	<a href="#">ZINT</a>		yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
43	<a href="#">ZFRC</a>		yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
44	<a href="#">ZLOG</a>	Log(Z)	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
45	<a href="#">ZNEG</a>	-Z	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
46	<a href="#">ZRND</a>	rounded Z	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
47	<a href="#">ZSIGN</a>	Sign(Z)	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
48	<a href="#">ZSIN</a>	SIN	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
49	<a href="#">ZSINH</a>	SINH	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
50	<a href="#">ZSQRT</a>	Square Root	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
51	<a href="#">ZTAN</a>	TAN	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
52	<a href="#">ZTANH</a>	TANH	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
53	<a href="#">ZTRP</a>	Re<->Im	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
54	<a href="#">ZARG</a>	Zarg	yes	yes	no	no	no	PREMON	no	yes	yes	no	PSTMON-2
55	<a href="#">ZMOD</a>	Z	yes	yes	no	no	no	PREMON	no	yes	yes	no	PSTMON-2
56	<a href="#">ZNORM</a>	Z ^2	yes	yes	no	no	no	PREMON	no	yes	yes	no	PSTMON-2
57	<a href="#">ZREC</a>	Rectangular	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
58	<a href="#">ZPOL</a>	Polar Notation	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
59	<a href="#">e^Z</a>	alternate ZEXP	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
60	<a href="#">EIZ/IZ</a>	function	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
61	<a href="#">Z^1/X</a>	hybrid power	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
62	<a href="#">Z*I</a>	rotation	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
63	<a href="#">Z/I</a>	rotation	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
64	<a href="#">NXTASN</a>	Next ASIN	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
65	<a href="#">NXTACS</a>	Next ACOS	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
66	<a href="#">NXTATN</a>	Next ATAN	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON

67	<a href="#">NXTLOG</a>	Next LN	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
68	<a href="#">NXTNRT</a>	Next Nth. Root	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
69	<a href="#">ZAVIEW</a>	Output Z	yes	no	no	no	no		no	no	no	yes	
70	<a href="#">CLZ</a>	Clear Z	no	no	no	no	no		no	yes	yes	yes	POSTMON
71	<a href="#">ZIMAG</a>	Clear Re(z)	no	yes	no	no	no		no	yes	yes	yes	POSTMON
72	<a href="#">ZREAL</a>	Clear Im(z)	no	yes	no	no	no		no	yes	yes	yes	POSTMON
73	<a href="#">CLZST</a>	Clear Zstack	no	no	no	no	no		no	no	yes	yes	POSTMON-3
74	<a href="#">Z↔</a>	Exchange	yes	no	no	no	no	PREMON	no	yes	yes	yes	POSTMON
75	<a href="#">Z↔W</a>	Exchange Stack	yes	no	yes	no	no	PREMON-2	no	no	yes	yes	POSTMON-3
76	<a href="#">Z↔R</a>	Exchange Stack	yes	no	yes	no	no	PREMON-2	no	no	yes	yes	POSTMON-3
77	<a href="#">Z↔S</a>	Exchange Stack	yes	no	yes	no	no	PREMON-2	no	no	yes	yes	POSTMON-3
78	<a href="#">LASTZ</a>	last argument	yes	no	yes	yes	no	PREMON-2	no	no	yes	yes	POSTMON-3
79	<a href="#">ZR^</a>	Roll Up Zstack	yes	no	yes	yes	no	PREMON-2	no	no	yes	yes	POSTMON-3
80	<a href="#">ZRCL</a>	Recall to Z	yes	no	yes	yes	no	PREMON-2	no	yes	yes	yes	POSTMON
81	<a href="#">IMAGINE</a>	inputs Im(z)	yes	no	yes	yes	no	PREMON-2	no	yes	yes	yes	POSTMON
82	<a href="#">ZENTER^</a>	Enter level	yes	no	yes	yes	no	PREMON-2	no	no	yes	yes	POSTMON-3
83	<a href="#">ZREAL^</a>	Input number	yes	no	no	yes	no	PREMON	no	yes	yes	yes	POSTMON
84	<a href="#">ZIMAG^</a>	Input number	yes	no	no	yes	no	PREMON	no	yes	yes	yes	POSTMON
85	<a href="#">ZRDN</a>	Roll Down ZSTK	yes	no	yes	no	no	PREMON-2	yes	no	yes	yes	POSTMON-3
86	<a href="#">ZREPL</a>	Replicates Z	yes	no	yes	no	no	PREMON-2	no	no	yes	yes	POSTMON-3
87	<a href="#">ZSTO</a>	Stores Z	yes	no	yes	no	no	PREMON-2	no	no	yes	yes	POSTMON-3