Complex Number Module

Written and developed by Ángel Martin

May 2021
Published under the GNU software licence agreement.

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

Jean-Marc Baillard, a constant reference for all math routines. He also contributed programs for the Riemann’s Zeta, Poly-Gamma and Complete Elliptic Integrals.

Greg McClure, who contributed the Complex Derivative engine and the Continued Fractions.

Håkan Thörngren for his assistance and advice on the Memory Buffer implementation.

W. Doug Wilder, who wrote the initial code for the non-merged functions in program mode.

Valentín Albillo, who wrote the original "PROOT" FOCAL program,

M. Luján García, who prepared the 41Z Keys overlay bitmap file.


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. Preamble: a Complex Relapse. - Reloaded. ............................................ 7

1. Introduction.
   1.1. Function Launchers. ................................................................. 8
   1.2. Last Function functionality .................................................... 9

2. Complex Stack, number entering and displaying.
   2.1 Rectangular vs. Polar modes .................................................... 11
   2.2 Data entry conventions .......................................................... 12

   3.1 Display and Conversion functions .............................................. 13
   3.2 Complex Natural Data entry ..................................................... 14
   3.3 The Complex User Assignments ................................................. 17
   3.4 The Complex Keyboard ............................................................ 18

4. Stack and Memory functions.
   4.1 Stack functions group .............................................................. 22
   4.2 ZSTO/ZRCL Math function groups ............................................ 26

5. Complex Math.
   5.1 Simple Arithmetic ....................................................................... 27
   5.2 Exponentials and Powers that be ............................................... 29
   5.3 Complex Logarithm .................................................................... 34

6. Complex Geometry
   6.1 Basic functions ........................................................................... 36
   6.2 Complex Comparisons .................................................................. 39

7. Complex Trigonometry
   7.1 Basic Functions ........................................................................... 42
   7.2 Complex Fibonacci numbers ....................................................... 45

8. 2D-vectors or complex numbers?
   8.1 Two parallel worlds ..................................................................... 46
   8.2 Alternate Displaying: Quads and Tones ....................................... 47
9. Polynomial Roots and Evaluations

9.1. Polynomial Evaluation 48
9.2. Polynomial Primitive and Derivatives 49
9.3. Solution to quadratic and cubic equations 50
9.4. General Polynomial Root Finder 53

10. It’s a Gamma-Zeta world out there

10.1. Lanczos approximation 56
10.2. Digamma and LnGamma 58
10.3 Poly-Gamma function 60
10.4 Inverse Gamma and Catalan Numbers 61
10.5. Riemann’s Zeta function 62
10.6. Lambert W function 64

11. Complex Means, Elliptic Integrals and DFT.

11.1. Arithmetic, Harmonic and Geometric Means 65
11.2. Dual-means: AGM and GHM 65
11.3. Complete Elliptic Integrals K(m), E(m) 66
11.4. Incomplete Elliptic Integrals, F(a; m) and E(a; m) 67
11.5. Discrete Fourier Transform (DFT) 70

12. General Methods and Special Functions

12.0. Real variable Functions 72
12.1. Multi-valued functions Driver 73
12.2. Solutions to f(z)=0 75
12.3. Successive Approximation Method 81
12.4. Complex Function Derivatives 83
12.5. Complex Continued Fractions 87
12.6. Bessel Functions 91
12.7. Spherical Hankel functions 96
12.8. Weber & Anger functions 97
12.9. Dilogarithm and Polylogarithm 100
12.10. Lerch Transcendent 101
12.11. Exponential Integrals 102

Appendices.

a0.- Appendix: Delta-Wye Transformation 104
a0.- Saving & Restoring the Z-Stack in X-Memory 105
a1.- Complex Buffer functions 107
a2.- Complex Keyboard key-maps 113
a3.- Formula compendium 114
a4.- Quick Reference Guide 115
a5.- Complex functions logic 119
Appendix 0. – Sketch of the 41Z Launchers Map

The figure below shows the hierarchy and dependencies between all launchers. Note that only those choices prompting to other levels are shown, not all prompting functions (like ZSTO, ZRCL, Z<>), ZVIEW, ^IM/AG, etc.

Note. Within reason, this module adopts 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.
41Z Deluxe – Complex Number Module for the HP-41

0. Preamble - A Complex Relapse - Reloaded

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 bank-switching and 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 “Q” 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: The revised code is more robust and better structured, plus a lot of room was recovered and used for new functionality.

The following summarizes the most important changes in the Deluxe version:

1. Extended the memory access functions functionality to fully support the stack registers. Both directly and with indirect arguments, with dedicated prompts and interrelationships. Furthermore, the function arguments are now entered as non-merged program lines directly by the function itself. This implementation applies to ZRCL, ZSTO, Z<> , and ZVIEW and is a direct port from the Total_Rekal module applied to complex registers. Also added RCL Math functions to the set.

2. Implemented an auxiliary FAT to allocate many other additions – mostly in the High-Level Math but not exclusively. Also re-instated the less relevant functions (such as ZIMAG, ZREAL, ZHALF, ZDBL, etc.) as sub-functions in the auxiliary FAT. The auxiliary FAT is also the home for all the second-tier sub-launchers underneath the main ΣZL function, such as ZMTV, ZHYP, ZNXT, ZBSL, etc.

3. Added sub-function launchers – ZF$ by name and ZF# by index#. This implementation is analogous to other modules and adheres to the U/I guidelines developed for sub-function design and usage. ALPHA prompts will be made directly in the by-name launchers. The sub-function index is added as a non-merged program line in PRGM mode.

4. Convenient implementation of the “Last Function” functionality – for direct re-execution of the last used function without retyping its name or navigating the launchers and menu structures. All functions called from any of the dedicated launchers will be captured, included main-FAT entries or functions from other modules as well.

5. Addition of MCODE implementations of the Continued Fractions evaluation (ZCF2V) and 10-point Complex Derivative Engine (ZDERV) – both written by Greg McClure. Use it to calculate the first and second derivatives of a user-defined function programmed in memory as a FOCAL routine using the 41Z functions.
6. Added set of MCODE functions for Complex Means (Arithmetic, Geometric, Harmonic and their dual forms), as well as Polynomial Evaluation, including the first and second derivatives and its primitive. The complex ZAGM will also be used for the Elliptic Integrals routines.

7. Added set of functions to calculate the Complete and incomplete Elliptic integrals of first and second kinds - with complex amplitudes or modulus. Some routines require the SandMath module to work.

8. Added multiple functions 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, the Complex Hypergeometric Function (written by Jean-Marc Baillard). ZERF, ZEI, ZCI, ZHCI, ZSI, and ZHSI - all using the Hypergeometric Function method. The remaining three are ZLERCH for the Lerch transcendental, plus ZLI2 and ZLIN, to calculate the Polylogarithm. All of them work with complex arguments.

9. 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/or FOCAL programs. This includes double-duty usage of the new function ZHGF – In RUN mode it is a new function launcher, grouping the functions that implement this calculation method. However in a running program it performs the actual execution work.

10. Added two MCODE functions for the Discrete Fourier Transform calculation on a set of complex data points, direct and inverse. ZDFT and ZIDFT will work on a set of complex data registers defined by its control word bbb.eee in X – returning the transformed set to a contiguous set of registers following that sample.

11. And last but not least, numerous changes in the code all throughout the module, rearranged sections and overall improvement in the consistency and usability of the functions - notably NXTNRT prompts when called from the ZNEXT launcher; now allows using the top two key rows (A – J) for index shortcuts 1-10.

**Warning:** due to all those function removals and additions, this version of the 41Z module has slightly different function arrangement in the FATs. If you have written your own programs using 41Z functions they may 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 for Advanced Users:**

The 41Z Deluxe is a bank-switched module. The bank switching will happen on both pages simultaneously therefore the module should not be plugged on “straddled” port configurations. Note also that you cannot configure only one page of the 41Z Deluxe module – therefore the footprint will always take a complete external port in the ROM bus.

<table>
<thead>
<tr>
<th>Page</th>
<th>Bank-1</th>
<th>Bank-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Upper Page</td>
<td>Main FAT w/ High-Level Math, Zvectors,</td>
<td>Function Tables and Launcher M-Code</td>
</tr>
<tr>
<td>XROM #04</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lower Page</td>
<td>Main and Aux-FATs, Z-stack</td>
<td>Lower-level Math Routines. MCODE only.</td>
</tr>
<tr>
<td>XROM #01</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1. Introduction.

Complex Number handling is perhaps the most notable area 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 16k-module is a fourth-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, ZRCL, (both with full math support), Z<>, and ZVIEW – all of them fully programmable as well.

1.1. Launchers and Last Function functionality.

The 41Z Deluxe includes full support for the "LASTF" functionality. This is a handy choice for repeat executions of the same function (i.e. to execute again the last-executed function), without having to type its name or navigate the different launchers to access it.

The implementation is not universal – it only covers functions invoked using the dedicated launchers, but not those called using the mainframe XEQ function. The following table summarizes the launchers that include this feature:

<table>
<thead>
<tr>
<th>Module</th>
<th>Launchers</th>
<th>LASTF Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>41Z &quot;Deluxe&quot;</td>
<td>ΣZL, ZHGF, ZPRT, ZNEXT, ZBSL, ZHYP</td>
<td>Captures (sub)fnc id#</td>
</tr>
<tr>
<td></td>
<td>ZF$</td>
<td>Captures fnc NAME</td>
</tr>
<tr>
<td></td>
<td>ZF#</td>
<td>Captures (sub)fnc id#</td>
</tr>
<tr>
<td></td>
<td>CAT+ (XEQ)</td>
<td>Captures (sub)fnc id#</td>
</tr>
</tbody>
</table>

LASTF Operating Instructions

The Last Function feature is triggered by pressing the radix key (decimal point - the same key used by LastX) twice at the “Z:” prompt. When this feature is invoked, it first shows “LASFT” briefly in the display, quickly followed by the last-function name. Keeping the key depressed for a while shows “NULL” and cancels the action. In RUN mode the function is executed, and in PRGM mode it's added as a program step if programmable, or directly executed if not programmable.

If no last-function record yet exists, the error message “NO LASTF” is shown. If the buffer #9 (used to store the last function id# code) is not present, the error message is “NO BUF” instead.
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.

<table>
<thead>
<tr>
<th>41Z Complex Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>b11</td>
</tr>
<tr>
<td>b10</td>
</tr>
<tr>
<td>b09</td>
</tr>
<tr>
<td>b08</td>
</tr>
<tr>
<td>b07</td>
</tr>
<tr>
<td>b06</td>
</tr>
<tr>
<td>b05</td>
</tr>
<tr>
<td>b04</td>
</tr>
<tr>
<td>b03</td>
</tr>
<tr>
<td>b02</td>
</tr>
<tr>
<td>b01</td>
</tr>
<tr>
<td>b00</td>
</tr>
</tbody>
</table>

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 **ZGEU, LASTZ, ZPI, ZRCL, ZRPL^, 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 is stored as values in Polar form). However, direct manual entry of complex in polar form is also possible using the **^IM/AG** function described in the following pages.

Note also that the POLAR mode is directly affected by the angular mode as well, as it occurs with real argument values.

\[
\begin{align*}
(a,b) &= a+jb = r \angle \theta \\
\end{align*}
\]

**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 typically 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^, Im(z)" – like on the HP-42S –, or its reverse: "Im(z), ENTER^, 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^, 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^ 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^ (which is also equivalent to executing the IMAGINE function) – or simply “ΣZL, Radix, 1” using the “complex keypad”. And to enter 4 as a complex number, just press: 4, ZREAL^ - or simply “ΣZL, 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.

<table>
<thead>
<tr>
<th>Index</th>
<th>Function</th>
<th>Group</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ZK?YN</td>
<td>Usability</td>
<td>Activates and deactivates the Complex Assignments</td>
</tr>
<tr>
<td>2</td>
<td>ΣZL_</td>
<td>Usability</td>
<td>Accesses most of the 41Z functions plus special features</td>
</tr>
<tr>
<td>3</td>
<td>ZAVIEW</td>
<td>Display</td>
<td>Views complex number in X,Y (prompts for # decimal places)</td>
</tr>
<tr>
<td>4</td>
<td>POLAR</td>
<td>Display</td>
<td>Displays complex numbers in Polar form</td>
</tr>
<tr>
<td>5</td>
<td>RECT</td>
<td>Display</td>
<td>Displays complex numbers in Rectangular form</td>
</tr>
<tr>
<td>6</td>
<td>^IM/AG_</td>
<td>Usability</td>
<td>Inputs Imaginary Part (or Argument) of complex number</td>
</tr>
</tbody>
</table>

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.

**ZAVIEW _ _ _**  
Complex number AVIEW  
Uses ALPHA registers  
Prompts for # decimal places

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

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. In RUN mode you have the choice to input the number of decimal places in the function's prompt – whilst in Program it'll use the selected settings 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 enhanced visualization, **ZAVIEW won't show decimal zeros 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.

![Examples](image)

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

![Examples](image)

Executing the functions **POLAR** and **RECT** will also display the complex number currently stored in X,Y...
### POLAR
Sets POLAR mode on
Displays number
Shows in SET mode

### RECT
Sets RECT mode on
Displays number
Shows in SET mode

### ZPOL
Convert to Polar
Converts X,Y to POLAR
Always shows in POLAR

### ZREC
Convert to Rectangular
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:

\[
\text{Mod} < \text{Arg} \quad \text{; where:} \\
\text{Mod} = |z| \quad \text{and} \quad \text{Arg}=\alpha \\
\Rightarrow z = |z| * e^{i\alpha}
\]

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

\[\begin{array}{c|c}
-5 (1 - j) & \text{equals} \\
7.07 \pm 2.36 & \text{7.07} \pm 135 \text{ USER} \\
7.07 \pm 150 & \text{USER RAD}
\end{array}\]

**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):

\[- \text{Arg}(z), \text{ENTER}^\wedge, |z|, \text{ZREC} \rightarrow \text{Re}(z) + J \text{Im}(z)\]

### 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^\wedge to input the complex number into the complex stack. That alone would justify its inclusion on the 41Z module.

\[\begin{array}{c|c|c}
\wedge \text{IM/AG } & \text{Inputs Im}(z)/\text{Arg}(z) \text{ Part} & \text{Does Stack Lift} \\
\text{Prompting function}
\end{array}\]

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. \(^\text{IM/AG}\) 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 - \(^\text{IM/AG}\) will check for that.
• Only one CHS can be used for the mantissa sign, \(^\text{IM/AG}\) will check for that.
• Multiple CHS can be used for the exponent sign, but \(^\text{IM/AG}\) 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\(^{\uparrow}\) 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:

```
\[\text{^IM/AG} \text{ J -} \]
\[\text{USER} \]
\[\text{\text{^IM/AG} \text{ J -} \text{ USER}}\]
```

```
\[\text{\text{^IM/AG} \text{ J S5.2E - 1 -}}\]
\[\text{USER} \]
```

```
\[\text{\text{^IM/AG} \text{ J - 125.5 -}}\]
\[\text{USER} \]
```

Note: To extract the numeric value from the input string, \(^\text{IM/AG}\) executes the same code as the X-function \texttt{ANUM}. All conversion conventions will follow the same \texttt{ANUM} logic. Suffice it to say that the implementation of \(^\text{IM/AG}\) 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 \(^\text{IM/AG}\).

```
\[\text{\text{^IM/AG} \text{ J, - E -}}\]
\[\text{USER \ RAD \ 0} \]
```

will trigger:

```
\[\text{SYNTAX ERROR} \]
\[\text{USER \ RAD \ 0} \]
```

Some apparently incorrect syntax constructions will however be properly interpreted by \(^\text{IM/AG}\), 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 \texttt{ANUM} function!

\textit{Example: Vector Load addition} (taken from the 35s User Guide):-
We start by setting POLAR and DEG modes, then using the \(^{\text{IM/AG}}\) function three times will set the three complex numbers on the complex stack, and finally simply execute the complex addition function \(Z+\) twice:

\[
\begin{align*}
\text{POLAR, DEG} \\
185, \text{ } ^{\text{IM/AG}}, 62, \text{ Enter}^\uparrow \\
170, \text{ } ^{\text{IM/AG}}, 143, \text{ R/S} \\
100, \text{ } ^{\text{IM/AG}}, 261, \text{ R/S} \\
\text{Z+}, \text{ Z+} \\
\text{Result: } & - > 178,9372 < \) 111,1489 \\
\end{align*}
\]

Or in Rectangular mode (as it’s saved in XY):
\[
\text{RECT} \quad - > -64,559 + J166,885
\]

Note the following points:

- We used indistinctly \(\text{Enter}^\uparrow\) 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 \(\text{ZEnter}^\uparrow\) to push the complex numbers into the complex stack because the stack-lift was performed by \(^{\text{IM/AG}}\).

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

This is the keystroke sequence and partial results (assuming we’re in \text{POLAR} mode)

\[
\begin{align*}
62, \text{ Enter}^\uparrow, & 185, \text{ ZREC, ZEnter}^\uparrow \quad - > 185 <)62 \\
143, \text{ Enter}^\uparrow, & 170, \text{ ZREC, ZEnter}^\uparrow \quad - > 170 <)143 \\
261, \text{ Enter}^\uparrow, & 100, \text{ ZREC} \quad - > 100 <)-99 \\
\text{Z+}, \text{ Z+} \quad & - > 178,9372 <) 111,1489
\end{align*}
\]

\textbf{One last remark about data displaying vs. data entry}:- As it was explained before, \(\text{ZPOL}\) will convert the complex number into Polar coordinates, and it will be displayed in POLAR form even if \text{RECT} mode is selected. This is the single one exception all throughout the 41z module, and it will only work immediately after pressing \(\text{ZPOL}\) but not for subsequent executions of \(\text{ZAVIEW}\) – 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:

<table>
<thead>
<tr>
<th>ZK?YN _</th>
<th>Complex User Assignments</th>
<th>Prompting function</th>
</tr>
</thead>
</table>

**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 – and display the Z-level content.
- 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.

<table>
<thead>
<tr>
<th>Keycode</th>
<th>Unshifted Keys</th>
<th>Shifted Keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>s+</td>
<td>s-</td>
</tr>
<tr>
<td>12</td>
<td>1/X</td>
<td>y^x</td>
</tr>
<tr>
<td>13</td>
<td>SQRT</td>
<td>z^2</td>
</tr>
<tr>
<td>14</td>
<td>LOG</td>
<td>10^x</td>
</tr>
<tr>
<td>15</td>
<td>LN</td>
<td>e^x</td>
</tr>
<tr>
<td>21</td>
<td>x&lt;&gt;y</td>
<td>CLS</td>
</tr>
<tr>
<td>22</td>
<td>RDN</td>
<td>%</td>
</tr>
<tr>
<td>23</td>
<td>SIN</td>
<td>ASIN</td>
</tr>
<tr>
<td>24</td>
<td>COS</td>
<td>ACS</td>
</tr>
<tr>
<td>25</td>
<td>TAN</td>
<td>ATAN</td>
</tr>
<tr>
<td>33</td>
<td>STO</td>
<td>LBL</td>
</tr>
<tr>
<td>34</td>
<td>RCL</td>
<td>n/a</td>
</tr>
<tr>
<td>41</td>
<td>ENTER^</td>
<td>CAT</td>
</tr>
<tr>
<td>42</td>
<td>CHS</td>
<td>ISG</td>
</tr>
<tr>
<td>44</td>
<td>EEX</td>
<td>CLX</td>
</tr>
<tr>
<td>51</td>
<td>-</td>
<td>x=y?</td>
</tr>
<tr>
<td>61</td>
<td>+</td>
<td>x&lt;&gt;y?</td>
</tr>
<tr>
<td>71</td>
<td>^</td>
<td>x&gt;y?</td>
</tr>
<tr>
<td>81</td>
<td>i</td>
<td>x=0?</td>
</tr>
<tr>
<td>82</td>
<td>n/a</td>
<td>LASTx</td>
</tr>
<tr>
<td>84</td>
<td>R/S</td>
<td>ZAVIEW</td>
</tr>
</tbody>
</table>

*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 ΣZL: 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 ΣZL assigned to Sigma+).
On the left: the version for V41. On the right, for i41CX

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 \( \textcolor{red}{ZL} \):

- **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 \( Z\text{LN} \); pressing \( Z \), COS will execute \( Z\text{COS} \), etc...
  Pressing \( Z \), + will execute \( Z+ \); pressing \( Z \), R/S will execute \( Z\text{VIEW} \).

- **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 \( Z\text{EXP} \); pressing \( Z \), SHIFT, SIN will execute \( Z\text{ASIN} \),
  Pressing \( Z \), SHIFT, R/S will execute \( Z\text{VIEW} \) (a prompting function itself).

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

- **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 \( Z\text{WDET} \); pressing \( Z \), \( Z \), 5 will execute \( Z\text{WCROSS} \), , and so on with all the "red-labeled" keys.

  Pressing \( Z \), \( Z \), ENTER^ will execute \( Z\text{REPL} \); pressing \( Z \), \( Z \), \( Z \) will execute \( Z<>U \)

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

- **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: \( \text{NXTASN}, \text{NXTACS}, \text{NXTATN}, \text{NXTLN} \), and \( \text{NXTNRT} \) (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: \( \text{ZJBS}, \text{ZIBS}, \text{ZKBS}, \text{ZYBS} \) – as well as \( \text{EIZ/IZ} \), a particular case of Spherical Hankel \( h_1(0,z) \).

- **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, \( \text{SIN} \) will execute \( Z\text{ASINH} \)

- **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 $^\text{IM/AG}$ 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 $^\text{IM/AG}$, as follows:

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

**Other keystrokes**. The 41Z module takes control of the calculator keyboard when $\Sigma ZL$ 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.

**Quick Recap**:

The figure below shows the main different modes of the $\Sigma ZL$ function, the real cornerstone of the 41Z module:

- **SHIFT**
- **Z**, USER
- **SHIFT**, **Z**
- **H**, USER
- **SHIFT**, **Z**
- **H**, USER
- **S**
- **T**, USER
- **Z**
- **STO**, USER

Press the Back-arrow key to bring the command chain back to the starting point ($\Sigma ZL$). 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. Z = Re(z) + j Im(z); W = Re(w) + j 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 latter 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.

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

(*) 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.

ZTRP

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): Im(\(z\)), ENTER^, 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.

| ZENTER^ | Enters X,Y into levels Z, W | Does Stack Lift |
| ZRPL^ | Fills complex stack |

ZENTER^ 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^ 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^ 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.

| ZREAL^ | Enters X in Z as (x+j0) | Does Stack Lift |
| ZIMAG^ | Enters X in Z as (0+jX) | Does Stack Lift |

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.

| CLZ | Clears complex stack level Z |
| CLZST | Clears complete complex stack |
| ZREAL | Extracts Real part from Z | Sub-function X<>Y, CLX, X<>Y |
| ZIMAG | Extracts Imaginary part from Z | Sub-function 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. Note that contrary to the real CLX function, the execution of CLZ will save the complex number in the complex LastX level "S".
Exchanges Z and Stack

Level# = 0,1,2,3,4

Prompting function (*)

(*) Fully programmable, see note in following pages.

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).

Z<>W
Exchanges Z and W

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).

LastZ
Recalls last number used to Z

Does Stack Lift

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: Z^2, LastZ, Z+, ZHALF

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

F(z) = Ln [ z + SQR(z^2 + 1)], for z= 20+20i

Solution:

Z^2, ENTER^, ZRPL -> puts 20+20i in all 4 levels of the complex stack

Z^2, 1, ZREAL^, Z+ -> could have used "1, +" as a more direct method

ZSQRT, Z+, ZLN -> 4,035+J0,785

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

<table>
<thead>
<tr>
<th>ZRCL</th>
<th>Recall from Complex Register</th>
<th>Does Stack lift</th>
<th>Prompting function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSTO</td>
<td>Store in Complex Register</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 ZRnn corresponds to the real storage registers R2nn and R(2nn+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 input index 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 line can be omitted if any non-numeric line follows the function. This saves bytes and makes programs more legible.

The original implementation was written by W. Doug Wilder, and it was almost as powerful and convenient as the one used by the HEPAX module for its own multi-function groups. I enhanced it further with an automated parameter input feature: when entered into a program, the index input will be added automatically in a second program line by the function.

Similar to the real counterparts, keys on the first two rows can be used as shortcut for indexes 1-10.

Note that indirect addressing is also supported (say ZRCL IND _ _) pressing the SHIFT key. Also note that in the Deluxe edition of the 41Z, their logic fully supports the use of the complex stack registers (i.e. ZRCL ZL _ _ followed by a Z-stack level: {U, V, W, Z, and S}) pressing the RADIX key; as well as the combination of both indirect and stack addressing (i.e. ZRCL IND ST _ _ followed by a REAL stack / data register number) sequentially pressing the SHIFT and RADIX keys. This extends the model of the native calculator functions to the complex data registers, where obviously an indirect pointer is always a real number by definition.

For example:

```
ZSTO ZL 01
```

Where the left prompt will only allow for one of the five complex Z-Stack levels letters, and the right prompt will allow for any of the 16 choices available as real stack (including the synthetic registers as well - be careful with those!).

Note that as of revision “O” of the Library#4 module, in program mode the argument entered by the function will be automatically entered in the second program step for the IND, ZL, and IND ST cases. In fact the indirect addressing is nothing more than adding 128 to the address, (or 0x80 Hex) thus it is handled by simply adding such factor to the index in the prompt line. Similarly, by adding 112 (or 0x70 Hex) for complex Z-stack levels, or the addition of both 0x80- and 0x70 (i.e. 240) for the IND ST combination.

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

Note for advanced users: Pressing the EEX key will also activate the prompt-lengthener adding a third field to the prompt. This is of limited usability since for Complex registers it would require setting a real SIZE above 200 in the calculator.
4.2. ZSTO/ZRCL Math function groups.

<table>
<thead>
<tr>
<th>Function</th>
<th>Name</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZST+</td>
<td>Store Adding to ZReg</td>
<td>Prompting function</td>
</tr>
<tr>
<td>ZST-</td>
<td>Store subtracting from ZReg</td>
<td>Prompting function</td>
</tr>
<tr>
<td>ZST*</td>
<td>Store multiplying to ZReg</td>
<td>Prompting function</td>
</tr>
<tr>
<td>ZST/</td>
<td>Store Dividing by ZReg</td>
<td>Prompting function</td>
</tr>
<tr>
<td>ZRC+</td>
<td>Add ZReg to Z</td>
<td>Prompting function</td>
</tr>
<tr>
<td>ZRC-</td>
<td>Subtract ZReg from Z</td>
<td>Prompting function</td>
</tr>
<tr>
<td>ZRC*</td>
<td>Multiply Z by ZReg</td>
<td>Prompting function</td>
</tr>
<tr>
<td>ZRC/</td>
<td>Divide Z by ZReg</td>
<td>Prompting function</td>
</tr>
</tbody>
</table>

One of the newest additions to the 41Z function set.- allow storage and recall 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)

With the support of Z-stack registers and INDirection it is possible to use the same shortcuts and conveniences as there are available for the real case in the standard calculator. For example to multiply a number by two you use ZST+, RADIX, “Z”:

Which expects a letter representing the complex stack. i.e. {Z, W, U, V}, and “S” for LastZ.

Complex Stack manipulation is now simply a matter of using Z<> _ _ with the corresponding Z-stack level letter in the RADIX prompts. Not to be confused with the stack level# input required by the function Z<>ST _ _, which only allows decimal values between 1-5 as valid entries. Similar but not the same – in particular when it comes to INDirection.

*These functions are fully programmable* using the same non-merged technique described in the previous page for the standard cases. Like them, the argument is automatically entered by the function as a second program line. The same considerations apply for Z-Stack and indirect registers, which are automatically entered in the non-merged program step.

The RCL/STO Math functions can be accessed directly from the ZRCL and ZSTO prompts by pressing the corresponding arithmetic key. In fact, you can “navigate” your way about all the choices between the three memory access functions ZRCL, ZSTO and Z<> as well as their arithmetic extensions (excluding Z<>) simply by pressing any of the appropriate keys during the prompts of any of them.
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>
<thead>
<tr>
<th>Index</th>
<th>Function</th>
<th>Formula</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$Z+$</td>
<td>$Z = w + z$</td>
<td>Complex addition</td>
</tr>
<tr>
<td>2</td>
<td>$Z-$</td>
<td>$Z = w - z$</td>
<td>Complex subtraction</td>
</tr>
<tr>
<td>3</td>
<td>$Z^*$</td>
<td>$Z = w^* z$</td>
<td>Complex multiplication</td>
</tr>
<tr>
<td>4</td>
<td>$Z/\pi$</td>
<td>$Z = w / \pi$</td>
<td>Complex division</td>
</tr>
<tr>
<td>5a</td>
<td>ZINV</td>
<td>$Z = 1/z$</td>
<td>Complex inversion, direct formula</td>
</tr>
<tr>
<td>5b</td>
<td>$1/Z$</td>
<td>$Z = 1/r \ e^{(-iArg)}$</td>
<td>Complex inversion, uses TOPOL</td>
</tr>
<tr>
<td>6</td>
<td>ZDBL</td>
<td>$z = z^* z$</td>
<td>Sub-function. Can be replaced with: $2, ST^* Z, *$</td>
</tr>
<tr>
<td>7</td>
<td>ZHALF</td>
<td>$z = z/2$</td>
<td>Sub-function. Can be replaced with: $2, ST / Z, /$</td>
</tr>
<tr>
<td>8</td>
<td>ZRNND</td>
<td>$z = rounded(z)$</td>
<td>Rounds $z$ to display settings precision</td>
</tr>
<tr>
<td>9</td>
<td>ZINT</td>
<td>$z = \text{Int}(z)$</td>
<td>Takes integer part for Re($z$) and Im($z$)</td>
</tr>
<tr>
<td>10</td>
<td>ZFRC</td>
<td>$z = \text{Fr}(z)$</td>
<td>Takes fractional part for Re($z$) and Im($z$)</td>
</tr>
<tr>
<td>11</td>
<td>ZPI*</td>
<td>$z = z\pi$</td>
<td>Simple multiplication by $\pi$</td>
</tr>
</tbody>
</table>

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

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.

- **ZINV** Direct Complex inversion $Z = 1/z$ Does LastZ
- **1/Z** Uses POLAR conversion $Z = 1/r \ e^{(-iArg)}$ 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 $Z/\pi$, where $w = 1+0j$, and not using the stack level $W$. Note however that $Z/\pi$ 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:

\[
\begin{align*}
1, \text{ ENTER}^\wedge, 0, & \text{ ZINV} \quad \rightarrow 0-j1 \\
\text{LASTZ} & \rightarrow 0+j1 \\
Z^* & \rightarrow 1+j0
\end{align*}
\]

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

<table>
<thead>
<tr>
<th>ZDBL</th>
<th>Doubles Z</th>
<th>(Z=2*Z)</th>
<th>Does LastZ</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZHALF</td>
<td>Halves Z</td>
<td>(Z=Z/2)</td>
<td>Does LastZ</td>
</tr>
</tbody>
</table>

These two sub-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: \(z_1=(23+13i); \ qquad z_2=(-2+i), \qquad \text{and} \ qquad z_3=(4-3i)\)

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:

\[
\begin{align*}
13, \text{ ENTER}, 23, & \quad \text{ZENTER}^\wedge \quad \rightarrow 23+j13 \\
1, \text{ ENTER}^\wedge, 2, \text{ CHS}, & \quad \text{ZENTER}^\wedge \quad \rightarrow -2+j1 \\
3, \text{ CHS}, \text{ ENTER}^\wedge, 4, & \text{ Z+} \quad \rightarrow 2(1-j) \\
Z/ & \rightarrow 2,500+j9
\end{align*}
\]

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).

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

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.

<table>
<thead>
<tr>
<th>ZPI*</th>
<th>Multiplies by pi</th>
<th>(Z=\pi*z)</th>
<th>Does LastZ</th>
</tr>
</thead>
</table>

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.

<table>
<thead>
<tr>
<th>Index</th>
<th>Function</th>
<th>Formula</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a</td>
<td>ZEXP</td>
<td>Z=REC(e^x, y)</td>
<td>Complex exponential (method one)</td>
</tr>
<tr>
<td>1b</td>
<td>E^Z</td>
<td>See below</td>
<td>Complex Exponential (method two)</td>
</tr>
<tr>
<td>2</td>
<td>Z^2</td>
<td>Z=REC(r^2, 2)</td>
<td>Complex square</td>
</tr>
<tr>
<td>3a</td>
<td>ZSQRT</td>
<td>Algebraic Formula</td>
<td>Principal value of complex square root</td>
</tr>
<tr>
<td>3b</td>
<td>SQRTZ</td>
<td>Z=REC(r^1/2, 0/2)</td>
<td>Principal value of complex square root</td>
</tr>
<tr>
<td>4</td>
<td>W^Z</td>
<td>Z=e^z*Ln(w)</td>
<td>Complex to complex Power</td>
</tr>
<tr>
<td>5</td>
<td>W^1/Z</td>
<td>Z=e^1/z*Ln(w)</td>
<td>Complex to reciprocal complex Power</td>
</tr>
<tr>
<td>6</td>
<td>X^1/Z</td>
<td>Z=e^x<em>z</em>Ln(x)</td>
<td>Real to complex power</td>
</tr>
<tr>
<td>7</td>
<td>X^1/X</td>
<td>Z=e^x<em>z</em>Ln(x)</td>
<td>Real to reciprocal complex power</td>
</tr>
<tr>
<td>8</td>
<td>Z^X</td>
<td>Z=e^x<em>z</em>Ln(z)</td>
<td>Complex to real Power</td>
</tr>
<tr>
<td>9</td>
<td>Z^1/X</td>
<td>Z=e^1/z*Ln(z)</td>
<td>Complex to reciprocal real Power</td>
</tr>
<tr>
<td>10</td>
<td>ZALOG</td>
<td>Z=e^z*Ln(10)</td>
<td>Complex decimal power</td>
</tr>
<tr>
<td>11</td>
<td>NXTRTN</td>
<td>Z=z*e^z^2/N</td>
<td>Next value of complex nth. Root</td>
</tr>
</tbody>
</table>

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.

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

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

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

since \( z = x + iy \), then \( e^z = e^{x+i}y \) = \( e^x \times e^i\),
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
   ZEXP -> 0 – j 0,500

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

1. ENTER^, ZENTER^ -> 1(1+j)
2. CHS, ZREAL^ ZEXP -> -2 + j 0

3. ZEXP -> 0,878 – j 0,479
or alternatively, this shorter and more efficient way: (leaves \( W \) undisturbed)

1, ENTER\(^\wedge\), \( Z^2, \text{ZINV, ZEXP} \) -> 0,878 – j 0,479

Note how this last method doesn’t require using \( \text{ZENTER}^\wedge \) 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.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Formula/Algorithm</th>
<th>Does LastZ</th>
</tr>
</thead>
<tbody>
<tr>
<td>( Z^2 )</td>
<td>Complex square</td>
<td>( Z = \text{REC}(r^2, \theta) )</td>
<td></td>
</tr>
<tr>
<td>( 	ext{ZSQRT} )</td>
<td>Complex square root</td>
<td>Algebraic Formula</td>
<td></td>
</tr>
<tr>
<td>( 	ext{SQRTZ} )</td>
<td>Complex square root</td>
<td>( Z = \text{REC}(r^{1/2}, \theta/2) )</td>
<td></td>
</tr>
</tbody>
</table>

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

\[
Z^2 = |z|^2 \cdot e^{2i\alpha}, \quad \text{and:} \quad \text{Sqrt}(z) = z^{1/2} = \text{Sqrt}(|z|) \cdot e^{i\alpha}, \quad \text{where } \alpha = \text{Arg}(z),
\]

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 \( \text{ZSQRT} \) 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.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Formula/Algorithm</th>
<th>Does LastZ</th>
</tr>
</thead>
<tbody>
<tr>
<td>( W^Z )</td>
<td>Complex to complex Power</td>
<td>( Z = e^{z \cdot \text{Ln}(w)} )</td>
<td></td>
</tr>
<tr>
<td>( W^{1/Z} )</td>
<td>Complex to reciprocal Power</td>
<td>( Z = e^{\text{Ln}(w) / z} )</td>
<td></td>
</tr>
</tbody>
</table>

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

\[
w^z = \exp[z \cdot \text{Ln}(w)], \quad \text{and:} \quad w^{1/z} = \exp[\text{Ln}(w) / z]
\]

The second function is a more convenient way to handle the reciprocal power, but it’s obviously identical to the combination \( \text{ZINV, W}^Z \).

**Example:** calculate the inverse of the complex number \( 1+2i \) using \( W^Z \):- Then obtain its reciprocal using \( \text{ZINV} \) to verify the calculations.

2, ENTER\(^\wedge\), 1, \( \text{ZENTER}^\wedge \) exponent \(-1\) stored in level \( W \) (also as: 1, ENTER\(^\wedge\), 2, \( \text{ZTRP} \))

\[
\begin{align*}
\text{number stored in level } W \\
\text{exponent } -1 \text{ stored in level } Z \\
\text{result: } 0,200 \text{ – j } 0,400 \\
\text{result: } 1,000 \text{ + j } 2
\end{align*}
\]

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.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Formula</th>
<th>Does LastZ</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Z^X)</td>
<td>Complex to real power</td>
<td>(Z = e^{x \cdot \ln(z)})</td>
<td></td>
</tr>
<tr>
<td>(Z^{1/X})</td>
<td>Complex to reciprocal real</td>
<td>(Z = e^{\ln(z)/x})</td>
<td></td>
</tr>
<tr>
<td>(X^Z)</td>
<td>Real to complex power</td>
<td>(Z = e^{z \cdot \ln(x)})</td>
<td></td>
</tr>
<tr>
<td>(X^{1/Z})</td>
<td>Real to reciprocal complex</td>
<td>(Z = e^{1/z \cdot \ln(x)})</td>
<td></td>
</tr>
<tr>
<td>(ZALOG)</td>
<td>10 to complex power</td>
<td>(Z = e^{z \cdot \ln(10)})</td>
<td></td>
</tr>
</tbody>
</table>

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^, \(X^Z\)

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 \(3^{(1+2i)}\)

- \(2, \text{ENTER}^\wedge, 1, \text{ZENTER}^\wedge, 3, Z^X\) results: \((1+2i)^3 = -11 - j2\)
- \(2, \text{ENTER}^\wedge, 1, \text{ZENTER}^\wedge, 3, X^Z\) results: \(3^{(1+2i)} = -1,759 + j2,430\)

**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 \(z=(1+i), (2+i), (3+i), \text{etc.}\)

This keystroke sequence will quickly address the even powers:

- \(0, \text{ENTER}^\wedge, 1, \text{ZTRP}\)  -> \(0 + j1\)
- \(Z^2\)  -> \(-1 + j0\) \(i^2 = -1\)
- \(Z^2\)  -> \(1 + j0\) \(i^4 = 1\)
Whilst this will take care of the rest (and also in general):

\[
\begin{align*}
0, \text{ ENTER}^\wedge, 1, \text{ ZTRP} & \quad \rightarrow \quad 0 + j1 \quad i \\
3, \text{ Z}^\wedge X & \quad \rightarrow \quad 0 - j1 \quad i^2 = -i \\
\text{LASTZ} & \quad \rightarrow \quad 0 + j1 \\
5, \text{ Z}^\wedge X & \quad \rightarrow \quad 0 + j1 \quad \text{i}^5 = i
\end{align*}
\]

Note in this example that for enhanced usability \(Z^\wedge X\) 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 \(X^\wedge Z\).

Alternatively, using \(W^\wedge Z\) and \(Z\text{REPL}\):

\[
\begin{align*}
1, \text{ ENTER}^\wedge, 0, \text{ ZREPL} & \quad \rightarrow \quad 0 + j1 \quad i \\
0, \text{ ENTER}^\wedge, 2, \text{ W}^\wedge Z & \quad \rightarrow \quad -1 + j0 \quad i^2 = -1 \\
\text{ZRDN} & \quad \rightarrow \quad 0 + j1 \quad i \\
0, \text{ ENTER}^\wedge, 3, \text{ W}^\wedge Z & \quad \rightarrow \quad 0 - j1 \quad i^3 = -i \\
\text{ZRDN} & \quad \rightarrow \quad 0 + j1 \quad i \\
0, \text{ ENTER}^\wedge, 4, \text{ W}^\wedge Z & \quad \rightarrow \quad 1 + j0 \quad i^4 = 1 \\
\text{ZRDN} & \quad \rightarrow \quad 0 + j1 \quad i \\
0, \text{ ENTER}^\wedge, 5, \text{ W}^\wedge Z & \quad \rightarrow \quad 0 + j1 \quad i^5 = i
\end{align*}
\]

**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, \text{ ENTER}^\wedge, \text{ ENTER}^\wedge, 2, \text{ X}^\wedge 1/Z \quad \rightarrow \quad 1,330 – j0,480

**Solution:** 1, \text{ ENTER}^\wedge, \text{ ENTER}^\wedge, 2, \text{ Z}^\wedge 1/X \quad \rightarrow \quad 1,099 + j0,455

<table>
<thead>
<tr>
<th>NXTNRT</th>
<th>Next value of Nth. Root</th>
<th>(z^\wedge N = e^{\frac{\text{Ln}(z)}{N}})</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZO</td>
<td>(z_0^\wedge N = \text{e}^{\frac{\text{Ln}(z)}{N}})</td>
<td></td>
</tr>
</tbody>
</table>

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).

\[
Z^\wedge 1/N = e^{\frac{\text{Ln}(z)}{N}} = e^{\frac{\text{Ln}(|z|)+i(\alpha+2\pi)}{N}} = e^{\frac{\text{Ln}(|z|)+i\alpha}{N}} * e^{j \frac{2\pi}{N}}
\]

From this we derive the general expression: \(\text{Next}(z^\wedge 1/N) = z^\wedge 1/N * e^{j \frac{2\pi}{N}}\)

thus there are \(N\) different Nth. Roots, all separated by \((2\pi\text{ over }N)\). 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.
**Example:** Calculate the *two* square roots of 1.

\[
\begin{align*}
0, \text{ ENTER}^\wedge, & \ 1, \ \text{ZENTER}^\wedge, \ 2, \ \text{Z}^1/X \quad \rightarrow \ 1 + j \ 0 \\
2, \ \text{NXTNRT (plus ZRND)} \quad & \rightarrow \ -1 + j \ 0 \\
\end{align*}
\]

Note that the previous root is temporarily stored in the LastZ register:

\[
\text{LASTZ} \quad \rightarrow \ 1 + j \ 0 \quad \text{(previous root)}
\]

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: \([\text{Im}(z), \ \text{ENTER}^\wedge, \ \text{Re}(z)]\)

\[
\begin{align*}
0, \ \text{ENTER}^\wedge, & \ 8, \ \text{ZENTER}^\wedge, \ 3, \ \text{Z}^1/X \quad \rightarrow \ 2 + j \ 0 \\
\text{NXTNRT}_\ 3 & \rightarrow \ -1,000 + j \ 1,732 \\
\text{NXTNRT}_\ 3 & \rightarrow \ -1,000 - j \ 1,732 \\
\end{align*}
\]

Note: for this example use the *Complex Keyboard* \(\mathbb{ZL}\) to execute \(\text{NXTNRT}\), as follows:

\(\mathbb{Z}, \ \mathbb{Z}, \ \text{SHIFT}, \ \text{SQRT}, \ \text{and then input} \ 3 \ \text{at the last prompt.}\)

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

\[
\begin{align*}
2, \ \text{ENTER}^\wedge, & \ 1, \ \text{ZSQRT} \quad \text{gives the first root:} \quad z= 1,272 + j \ 0,786 \\
\text{NXTNRT}_\ 2 & \quad \text{gives the second root:} \quad z=-1,272 - j \ 0,786 \\
\text{NXTNRT}_\ 2 & \quad \text{reverts to the first, principal value, of the root.} \\
\end{align*}
\]

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 \(1 + 2i\).

\[
\begin{align*}
2, \ \text{ENTER}^\wedge, & \ 1, \ \text{ZENTER}^\wedge \quad \text{inputs} \ z \ \text{in the complex stack level} \ \mathbb{Z} \\
3, \ 1/X, \ \text{Z}^X & \quad \text{gives the main root:} \quad z= 1,220 + j \ 0,472 \\
\text{NXTNRT}_\ 3 & \quad \text{gives the second root:} \quad z=-1,018 + j \ 0,82 \\
\text{NXTNRT}_\ 3 & \quad \text{give the third and last:} \quad z=-0,201 - j \ 1,292 \\
\end{align*}
\]

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.

<table>
<thead>
<tr>
<th>Index</th>
<th>Function</th>
<th>Formula</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ZLN</td>
<td>$Z = \ln</td>
<td>z</td>
</tr>
<tr>
<td>2</td>
<td>ZLOG</td>
<td>$Z = \frac{\ln(z)}{\ln(10)}$</td>
<td>Principal value of decimal logarithm</td>
</tr>
<tr>
<td>3</td>
<td>ZWLOG</td>
<td>$Z = \frac{\ln(z)}{\ln(w)}$</td>
<td>Base-$w$ logarithm of $z$</td>
</tr>
<tr>
<td>4</td>
<td>NXTLN</td>
<td>$Z = z + 2n\pi i$</td>
<td>Next value of natural logarithm</td>
</tr>
</tbody>
</table>

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: $z = |z|e^{i\theta}$, then:
\[
\ln z = \ln |z| + \ln (e^{i\theta}) = \ln(|z|) + i\theta
\]

But also
\[
z = |z|e^{i(\rho + 2\pi n)} = |z|e^{i(\rho + 4\pi n)} = \ldots = |z|e^{i(\rho + 2\pi n)}
\]

Then we’d equally have multiple values of its logarithm, as follows:
\[
\ln(z) = \ln(|z|) + i\theta = \ln(|z|) + i(\rho + 2\pi n) = \ldots
\]

Or generally:
\[
\ln z = \ln |z| + i(\rho + 2\pi n); \quad \text{where } n \text{ is a natural number.}
\]

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 for a more rigorous description of this subject).

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Expression</th>
<th>Does LastZ</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZLN</td>
<td>Natural logarithm</td>
<td>$Z=\ln</td>
<td>z</td>
</tr>
</tbody>
</table>

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

$$\ln z = \ln|z| + i\gamma, \quad \text{where} \quad \gamma = \text{Arg}(z) \in \left]-\pi, \pi\right]$$

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

| 1, ENTER\(^\wedge\), ZEXP, ZLN | -> $1,000 + j 1,000$ |
| 4, ENTER\(^\wedge\), 2, ZEXP, ZLN | -> $2 – j 2,283$ |

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

**NXTLN**

-> $2 + j 4,000$  - that’s more like it!

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Expression</th>
<th>Does LastZ</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZLOG</td>
<td>Decimal logarithm</td>
<td>$Z=\ln(z)/\ln(10)$</td>
<td>Yes</td>
</tr>
</tbody>
</table>

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

$$\log z = \frac{\ln z}{\ln(10)}$$

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

| 1, ENTER\(^\wedge\), ZALOG, ZLOG | -> $1(1+j)$ |
| 4, ENTER\(^\wedge\), 2, ZALOG, ZLOG | -> $2 + j 1,271$ |

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Expression</th>
<th>Does LastZ</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZWLOG</td>
<td>Base-W Logarithm</td>
<td>$Z=\ln(z)/\ln(w)$</td>
<td>Yes</td>
</tr>
</tbody>
</table>

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

$$\log z = \frac{\ln z}{\ln w}$$

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Expression</th>
<th>Does LastZ</th>
</tr>
</thead>
<tbody>
<tr>
<td>NXTLN</td>
<td>Next Natural logarithm</td>
<td>$Z=z_0+2\pi j$</td>
<td>No</td>
</tr>
</tbody>
</table>

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

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

So the different logarithms are “separated” a distance of value $2\pi$ 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.

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.**

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

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](http://www.clarku.edu/~djoyce/complex).

Let’s see the functions in detail.

<table>
<thead>
<tr>
<th>Function</th>
<th>Formula</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZMOD</td>
<td>$</td>
<td>z</td>
</tr>
<tr>
<td>ZARG</td>
<td>$\alpha = \text{ATAN}(y/x)$</td>
<td>Does LastZ</td>
</tr>
</tbody>
</table>

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

\[
|z| = \sqrt{x^2 + y^2} \\
\alpha = \text{ATAN}(y/x)
\]

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 LastZ 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.
ZNORM  Norm of z  \[ ||Z|| = |z|^2 \] Does LastZ

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

\[ ||z|| = |z|^2 \]

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 \( z = 4 + 5 \text{j} \):

```
> Z
```

ZSIGN  Module of z  \[ Z = z/|z| \] 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: \( Z\text{sign} = z / |z| \)

![Complex number diagram](image)

The figure above shows the unit circle and the relative position in the complex plane for the opposite (-z), conjugate (zc), and opposite conjugate (-zc) 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.
This pair of functions calculate the opposite- or the multiple-opposite by \((-1)^x\) – and the conjugate of a complex number \(z=x+y\ \text{i}\), as follows:

\[-z = -x -y\ \text{i}, \quad \text{and} \quad z^* = x - y\ \text{i}\]

See the figure below for the geometric interpretation of \textbf{ZNEG} and multiplication by real numbers:

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.

<table>
<thead>
<tr>
<th>Index</th>
<th>Function</th>
<th>Formula</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Z=0?</td>
<td>Is z=0?</td>
<td>Checks if z is zero</td>
</tr>
<tr>
<td>2</td>
<td>Z#0?</td>
<td>Is z≠0?</td>
<td>Checks if z is not zero</td>
</tr>
<tr>
<td>3</td>
<td>Z=i?</td>
<td>Is z=i?</td>
<td>Checks if z is the imaginary unit</td>
</tr>
<tr>
<td>4</td>
<td>Z=W?</td>
<td>Is z=w?</td>
<td>Checks if z and w are the same</td>
</tr>
<tr>
<td>5</td>
<td>Z=WR?</td>
<td>Is z=w rounded?</td>
<td>Checks if rounded z and rounded w are the same</td>
</tr>
<tr>
<td>6</td>
<td>Z#W?</td>
<td>Is z≠w?</td>
<td>Checks if z and w are different</td>
</tr>
<tr>
<td>7</td>
<td>ZUNIT?</td>
<td>Is</td>
<td>z</td>
</tr>
<tr>
<td>8</td>
<td>ZIN?</td>
<td>Is</td>
<td>z</td>
</tr>
<tr>
<td>9</td>
<td>ZOUT?</td>
<td>Is</td>
<td>z</td>
</tr>
<tr>
<td>10</td>
<td>ZREAL?</td>
<td>Is z a real number?</td>
<td>Checks whether Im(z)=0</td>
</tr>
<tr>
<td>11</td>
<td>ZIMAG?</td>
<td>Is z true imaginary?</td>
<td>Checks whether Re(z)=0</td>
</tr>
<tr>
<td>12</td>
<td>ZINT?</td>
<td>Is z true integer?</td>
<td>Checks whether Im(z)=0 and RRC[Re(z)]=0</td>
</tr>
<tr>
<td>13</td>
<td>ZGSS?</td>
<td>Is z Gaussian?</td>
<td>Checks whether Re(z) and Im(z) are both integers</td>
</tr>
<tr>
<td>14</td>
<td>ZQUAD</td>
<td>Shows Quad# msg.</td>
<td>Sets corresponding User Flag, clears others.</td>
</tr>
</tbody>
</table>

It’s well know 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.

\[
\begin{align*}
Z=W? & \quad \text{Compares z with w} & \text{Are they equal?} \\
Z\neq W? & \quad \text{Compares z with w} & \text{Are they different?} \\
Z=WR? & \quad \text{Compares z with w rounded} & \text{Are they equal?} \\
Z=0? & \quad \text{Compares z with zero} & \text{Are they equal?} \\
Z\neq 0? & \quad \text{Compares z with zero} & \text{Are they different?} \\
Z=i? & \quad \text{Compares z with i} & \text{Are they equal?}
\end{align*}
\]

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.

\[
z=w \iff \text{Re}(z)=\text{Re}(w) \quad \text{and} \quad \text{Im}(z)=\text{Im}(w)
\]

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.

\[
\text{Rnd}(z) = \text{Rnd}(w) \iff |\text{Re}(z)|=|\text{Re}(w)| \quad \text{and} \quad |\text{Im}(z)|=|\text{Im}(w)|
\]
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:

\[
\begin{align*}
    z = 0 & \text{ iff } \Re(z) = 0 \text{ and } \Im(z) = 0 \\
    z = i & \text{ iff } \Re(z) = 0 \text{ and } \Im(z) = 1
\end{align*}
\]

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

- use \( X#0? \) To check for \( Z#0? \) Condition
- Use \( X#0? \) To check for \( Z#I? \) Condition

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZUNIT?</td>
<td>Checks if ( z ) is on the unit circle</td>
<td>(</td>
</tr>
<tr>
<td>ZIN?</td>
<td>Checks if (</td>
<td>z</td>
</tr>
<tr>
<td>ZOUT?</td>
<td>Checks if (</td>
<td>z</td>
</tr>
</tbody>
</table>

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, 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:

\[
\begin{align*}
5, \text{ ENTER}^\wedge, 4, \text{ ZSIGN} & \rightarrow \text{ result: Zsign} = 0.625 + j 0.781 \\
\text{ZUNIT?} & \rightarrow \text{ result: “YES”} \\
\text{DEG, POLAR} & \rightarrow \text{ result: } 1.00 < 51.34 \text{ (in degrees)}
\end{align*}
\]

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

<table>
<thead>
<tr>
<th>ZQUAD</th>
<th>Returns Quadrant# for ( z )</th>
<th>Sets flag 1-2-3-4</th>
<th>Shows message</th>
</tr>
</thead>
</table>

A new function to display the quadrant number (1 to 4) and set the user flag matching its value.

In the display:

```
USER: RAD 3
```

the message goes away in a few instants.
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 FRC(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 an 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.

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

And their inverses:

<table>
<thead>
<tr>
<th>Index</th>
<th>Function</th>
<th>Formula</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>ZASIN</td>
<td>( \text{asinz} = -i \text{asinh}(iz) )</td>
<td>Complex Inverse Sine</td>
</tr>
<tr>
<td>8</td>
<td>ZACOS</td>
<td>( \text{acosz} = \pi/2 - \text{asinz} )</td>
<td>Complex Inverse Cosine</td>
</tr>
<tr>
<td>9</td>
<td>ZATAN</td>
<td>( \text{atanz} = -i \text{atanhz}(iz) )</td>
<td>Complex Inverse Tangent</td>
</tr>
<tr>
<td>10</td>
<td>ZHASIN</td>
<td>( \text{asinhz} = \text{Ln}[z + \sqrt{z^2 + 1}] )</td>
<td>Complex Inverse Hyperbolic Sine</td>
</tr>
<tr>
<td>11</td>
<td>ZHACOS</td>
<td>( \text{acoshz} = \text{Ln}[z + \sqrt{z^2 - 1}] )</td>
<td>Complex Inverse Hyperbolic Cosine</td>
</tr>
<tr>
<td>12</td>
<td>ZHATAN</td>
<td>( \text{atanhz} = 1/2 \text{Ln}[(1+z)/(1-z)] )</td>
<td>Complex Inverse Hyperbolic Tangent</td>
</tr>
</tbody>
</table>

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:

\[
\begin{align*}
\sin z & = -i \sinh (iz) \\
\cos z & = \cosh (iz) \\
\tan z & = -i \tanh (iz) \\
\text{asinz} & = -i \text{asinh}(iz) \\
\text{acosz} & = \pi/2 - \text{asinz} \\
\text{atanz} & = -i \text{atanhz}(iz) \\
\text{sinh} z & = 1/2 [e^z - e^{-z}] \\
\text{cosh} z & = 1/2 [e^z + e^{-z}] \\
\text{tanh} z & = (e^z - e^{-z})/(e^z + e^{-z}) \\
\text{asinh} z & = \text{Ln}[z + \sqrt{z^2 + 1}] \\
\text{acosh} z & = \text{Ln}[z + \sqrt{z^2 - 1}] \\
\text{atanhz} & = 1/2 \text{Ln}[(1+z)/(1-z)]
\end{align*}
\]

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 LBL “ZASIN”
02 ZASIN
03 ZSTO 00
04 ZAVIEW
05 E
06 STO 02
07 LBL 00
08 ZRCL 00
09 ZNEG
10 ZSTO 00
11 RCL 02
12 PI
13 * 
14 +
```

```
The 41Z module includes functions to calculate next values for complex ASIN, ACOS and ATAN, as follows: **NXTASN, NXTACS, and NXTATN**. Using the first one the program above changes to this very simplified way:

```
01 LBL "ZASIN2"  04 ZAVIEW  07 END
02 ZASIN
03 LBL 00
04 NXTASN
05 GTO 00
```

The key map is shown in the figure on the right, and can be accessed using:

- **Z**, [SHIFT] for the direct ones, and
- **Z**, [SHIFT], [SHIFT] for the inverses.

Using the general expressions we can obtain the multiple values of a given function from its principal value “Z” of a given function, as follows:

- the multiple values for ASIN(z) -in green squares- are placed on the two straight lines parallel to the x axis, y=Im[ASIN(z)] and y=−Im[ASIN(z)], and are separated at intervals of 2π length on each line.
- the multiple values for 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 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 Ln(z) –in blue squares– are placed on the vertical straight line $x=\text{Re}[\ln(z)]$, 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).

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>LastZ Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>NXTASN</td>
<td>Next Complex ASIN</td>
<td>Does LastZ</td>
</tr>
<tr>
<td>NXTACS</td>
<td>Next Complex ACOS</td>
<td>Does LastZ</td>
</tr>
<tr>
<td>NXTATN</td>
<td>Next Complex ATAN</td>
<td>Does LastZ</td>
</tr>
</tbody>
</table>

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:

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

Next ZACOS:

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

Next ZATAN:

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

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

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.
7.2 Complex Fibonacci Numbers. \{ZFIB\}

This short routine uses Binet’s formula applied to the complex domain to calculate the Fibonacci number of a given complex “index”. The result is another complex number that for integer cases coincides with the well-known Fibonacci series of course.

Binet’s formula interpolation to noninteger real indexes (below left) provides an easy expression for the determination that guarantees real values also for the interpolated Fibonacci numbers

\[
\begin{align*}
 f_x^* &= \frac{\varphi^x - \cos(\pi x)\varphi^{-x}}{\sqrt{5}} \\
 F_n &= \frac{\varphi^n - (-\varphi)^{-n}}{\sqrt{5}}
\end{align*}
\]

But it is in the complex domain (above right) when things become quite interesting, as can be seen in the graphical representations below, showing the locus of output results when the input values are negative real numbers (figure 1) and positive real value (figure 2). Note that both figures are not at the same scale/ (see also the animation at: https://www.geogebra.org/m/ypqcuqcs)

Program listing:

```
01  LBL "ZFIB"
02  ZRPL^*
03  1.618033989
04  X^Z
05  ZENTER^*
06  0
07  LASTX
08  1/X
09  CHS
10  ZRUP
11  W^Z
12  5
13  SQRT
14  ST/ Z
15  /
16  ZAVIEW
17  END
```

Example:

ZFIB (1.5)
0, ENTER^*, 1.5, ΣZ$ “ZFIB”

Result:

`0.9207 + j0.217`
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.**

<table>
<thead>
<tr>
<th>Index</th>
<th>Function</th>
<th>Formula</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ZWANG</td>
<td>Arg(ZW) = Arg(Z) – Arg(W)</td>
<td>Angle between 2 vectors</td>
</tr>
<tr>
<td>2</td>
<td>ZWDIST</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Distance between 2 points</td>
</tr>
<tr>
<td>3</td>
<td>ZWDOT</td>
<td>Z<em>W = Zx</em>Wx + Zy*Wy</td>
<td>2D vector Dot product</td>
</tr>
<tr>
<td>4</td>
<td>ZWCROSS</td>
<td>Z x W =</td>
<td>z</td>
</tr>
<tr>
<td>5</td>
<td>ZWDET</td>
<td></td>
<td>2D determinant</td>
</tr>
<tr>
<td>6</td>
<td>ZWLINE</td>
<td>a=(Y1-Y2) / (X1-X2) b=Y2 – a*X2</td>
<td>Equation of line through two points</td>
</tr>
</tbody>
</table>

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
```

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

![Image](image1)

The angle will be expressed in the selected angular unit.

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

![Image](image2)

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

![Image](image3)

(*) Note that despite having a simpler formula, ZWDET shows less precision than ZWCROSS.
Alternate Displaying: Quads and Tones.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZDISP</td>
<td>Compacted LCD view</td>
<td>Positive RE, IM</td>
</tr>
<tr>
<td>ZQUAD</td>
<td>Shows Quad message</td>
<td>Sets User flag</td>
</tr>
<tr>
<td>ZTONE</td>
<td>Sounds Tone using Y,X</td>
<td>Duration and frequency</td>
</tr>
</tbody>
</table>

These three functions provide additional user feedback on the complex value in the stack level Z (i.e. stack registers Y,X). Use them as a complement to the main ZAVIEW, each has interesting aspects but cannot be a full replacement to ZAVIEW given their shortcomings.

**ZDISP** main value is that it only uses the LCD to display a compacted version of the complex number. This leaves the ALPHA register undisturbed, in cases it needs to maintain its contents through a visualization of the Z result.

```
2.07 + 0.02
2 + 0.02
```

or:

```
2.07 + j4
2 + j4
```

- Each Real and Imaginary parts are allowed six LCD characters,
- Each is split as follows: three for the mantissa, one for the exponent sign and two more for the exponent itself.
- The two strings of six characters are separated by a comma to tell them apart
- No scrolling is supported, as all values are represented using the equivalent to a SCI 02 format.

Needless to say its main shortcoming is that it does not support negative values in the real or imaginary parts. This can be partially palliated using **ZQUAD** prior to **ZDISP**, as this will set the corresponding user flag depending on the complex location, leaving the other three flags from F1-F4 cleared:

- F1 set if bth Re(z)>0 and Im(z)>0
- F2 set if Re(z)<0 and Im(z)>0
- F3 set if both Re(z) and Im(z)<0
- F4 set if Re(z)?0 and Im(z)<0

**ZQUAD** will briefly show an informative message with the quadrant number, then it’ll revert to the standard ZAVIEW output to end. It’ll also reset the user flags 1-2-3-4 corresponding to the quadrant.

**ZTONE** will sound an acoustic tone using the information in the Y,X registers for frequency and duration; thus at least in theory each complex value is associated to its unique sound. In practice however the typical values make all sounds rather alike so it is more of a curiosity than of practical value. - for instance all real values (Im(z)=0) will have the same “blank” tone.
9. **Polynomial Roots and Evaluation**

A classic in calculator history just got improved. The 41Z Deluxe adds to the set new fast MCODE functions for the evaluation of polynomials with complex coefficients, as well as their primitive and their first and second derivatives.

**Table 9.1. Polynomial Evaluations group.**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Control word in X</th>
<th>Does LastZ</th>
<th>Control word in X</th>
<th>FOCAL Routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZPL</td>
<td>Polynomial Evaluation</td>
<td>Control word in X</td>
<td>Does LastZ</td>
<td>Control word in X</td>
<td>FOCAL Routine</td>
</tr>
<tr>
<td>ZPLI</td>
<td>Primitive of Polynomial</td>
<td>Control word in X</td>
<td>Does LastZ</td>
<td>Control word in X</td>
<td>FOCAL Routine</td>
</tr>
<tr>
<td>ZPD1</td>
<td>Pol. First derivative</td>
<td>Control word in X</td>
<td>Does LastZ</td>
<td>Control word in X</td>
<td>FOCAL Routine</td>
</tr>
<tr>
<td>ZPD2</td>
<td>Pol. Second Derivative</td>
<td>Control word in X</td>
<td>Does LastZ</td>
<td>Control word in X</td>
<td>FOCAL Routine</td>
</tr>
<tr>
<td>ZINPT</td>
<td>Data Input Routine</td>
<td>Control word in X</td>
<td>Does LastZ</td>
<td>Control word in X</td>
<td>FOCAL Routine</td>
</tr>
<tr>
<td>ZOUPT</td>
<td>Data Output Routine</td>
<td>Control word in X</td>
<td>Does LastZ</td>
<td>Control word in X</td>
<td>FOCAL Routine</td>
</tr>
</tbody>
</table>

Besides the evaluation point \( z_0 \), the evaluation functions require a control word as input parameter. This control word defines the complex register range used to store the polynomial coefficients, in the usual form “bbb.eee”, *with the highest term coeff. stored in ZRbbb*. If the degree of the polynomial is “n” there should be \( n+1 \) complex registers in the range, i.e. \( (\text{ee}-\text{bbb}) = n \).

Like the other hybrid functions in the module, you need to enter the complex value first \( z_0 \) and then the real value (control word) in the X-register – which will push \( z_0 \) one level up in the REAL stack. The result will be returned in the complex-Z register, with \( z_0 \) saved in LastZ – but the control word is lost (i.e. not saved in LastX).

The utility routines ZINPT and ZOUPT come very handy to enter the polynomial coefficients in the complex registers. They too use the same control word bbb.eee to define the complex register range used for the input/output action. Let’s see one example next.

**Example:** Evaluate the polynomial, main primitive and derivatives in the point \( z_0=1+i \) for the 4\textsuperscript{th} degree polynomial: \( P(z) = (1+i)z^4 - (2-3i)z^3 + (-1+2i)z^2 + z - (1+i) \)

First we introduced the coefficients in the complex data registers ZR00 to ZR04 as follows. Note that the index in the prompts refers to the complex register to use, *and not to the polynomial term*:

\[
\begin{align*}
0.004, & \text{ZF}^\$ \text{"ZINPT"}, & \text{"Z0=?"} \\
1, & \text{ENTER}^\^, \text{R/S} & \text{"Z1=?"} \\
3, & \text{ENTER}^\^, 2, \text{CHS}, \text{R/S} & \text{"Z2=?"} \\
2, & \text{ENTER}^\^, 1, \text{CHS}, \text{R/S} & \text{"Z3=?"} \\
0, & \text{ENTER}^\^, 1, \text{R/S} & \text{"Z4=?"} \\
1, & \text{ENTER}^\^, \text{ZNEG}, \text{R/S} & \text{shows Z-level again}
\end{align*}
\]

With the data safely stored in \{ZR00 – ZR04\} we proceed to calculate the results. First we enter the evaluation point in the complex Z register, followed by the control word in X:

\[
\begin{align*}
1, & \text{ENTER}^\^, \text{ENTER}^\^, 0.004, & \text{XEQ "ZPL"} & \Rightarrow -10-116 \\
& \text{LASTZ, 0.004,} & \text{ZF}^\$ \text{"ZPD1"} & \Rightarrow -39-110 \\
& \text{LASTZ, 0.004,} & \text{ZF}^\$ \text{"ZPD2"} & \Rightarrow -56+134 \\
& \text{LASTZ, 0.004} & \text{ZF}^\$ \text{"ZPLI"} & \Rightarrow 1.333-17.600
\end{align*}
\]

Note how the result for the polynomial and derivatives have integer real and imaginary parts (i.e. are Gaussian numbers) – but the primitive is not. We’ll revisit these results when we cover the Complex Derivative Engine in the next chapters.
Preview: Polynomial Roots and Values Launchers - both together now.

A convenient grouping of the polynomial functions provides access to the individual choices from a common prompt. To access it you can use its dedicated launcher from the complex keyboard shortcut - just press:

\[ [Z], [A], [L], [SHIFT] \]

Options “I” / “O” will trigger \texttt{ZINPUT} and \texttt{ZOUPT} respectively. Note that there are other functions also included here ("F" and "D"), which are related to the first derivative and continued fractions. They will be covered in another chapter later in the manual.

Note also that pressing [SHIFT] again this launcher toggles with the corresponding for the Complex Roots, as shown below:

\[ \begin{array}{c}
\text{ZPR}T \quad 1:2:3:N:S:O:M \\
\text{USER} \quad \text{RAD}
\end{array} \quad \begin{array}{c}
\text{ZPL} \quad 0:1:2:P:F:I:I:O \\
\text{USER}
\end{array} \]

A convenient grouping of the root-finding applications provides access to the root finders for the first, second, third and n-th. degree polynomials, as well as the general-purpose \texttt{ZSOLVE}. To access it you can call the sub-function \texttt{ZPRT}, or using the complex keyboard shortcut just press:

\[ [Z], [A], [L] \]

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

For \texttt{ZQRT} and \texttt{ZCRT} the coefficients are expected to be in the complex stack prior to the execution – whilst \texttt{ZPROOT} and \texttt{ZSOLVE} will prompt for the required entries.
**Solution of Quadratic and Cubic equations.**

<table>
<thead>
<tr>
<th>ZQRT</th>
<th>Roots of 2nd. Degree Eq.</th>
<th>Coefs. in Z-Stack</th>
<th>All MCODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZCRT</td>
<td>Roots of 3rd. Degree Eq.</td>
<td>Coefs. In Z-Stack</td>
<td>FOCAL program</td>
</tr>
<tr>
<td>ZQUDR</td>
<td>Driver for ZCRT</td>
<td>Prompts for values</td>
<td>FOCAL Routine</td>
</tr>
</tbody>
</table>

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

\[ C_1 \cdot z^2 + C_2 \cdot z + C_3 = 0; \]  
where \( C_1, C_2, C_3, \) and \( z \) are complex numbers

By applying the general formula:

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

**Example 1.** Find out the roots of \((1+i)z^2 + (-1-i)z + (1-i) = 0\)

1, ENTER , ZENTER 
1, CHS, ENTER , ZENTER 
1, CHS, ENTER , 1, XEQ "ZQRT"
"RUNNING..." followed by: " 1,300+j0,625" 
Z<>W " -0,300-j0,625"

We see that contrary to the real coefficients case, here the roots are NOT conjugated of one another.

**ZQRT** is entirely written in MCODE. It expects the three complex coefficients stored in levels \( V, W, \) and \( Z \) of the complex stack. The driver program below is an example using FOCAL instructions instead. 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.

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

Solving the Cubic Equation.

**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" → \( z_1 = -0,117-J0,910 \)
ZRDN → \( z_2 = -0,922+J1,047 \)
ZRDN → \( z_3 = 1,039-J0,136 \)
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.

<table>
<thead>
<tr>
<th></th>
<th>LBL &quot;ZCRT&quot;</th>
<th>Main version</th>
<th>LBL &quot;ZCRT2&quot;</th>
<th>Alternative Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>ZRUP</td>
<td>a3</td>
<td>ZRUP</td>
<td>a3</td>
</tr>
<tr>
<td>3</td>
<td>Z/</td>
<td>a0/a3</td>
<td>Z/</td>
<td>a0/a3</td>
</tr>
<tr>
<td>4</td>
<td>ZSTO (00)</td>
<td>a0'</td>
<td>ZSTO (00)</td>
<td>a0'</td>
</tr>
<tr>
<td>5</td>
<td>Z&lt;&gt;W</td>
<td>a1</td>
<td>Z&lt;=&gt;W</td>
<td>a1</td>
</tr>
<tr>
<td>6</td>
<td>LASTZ</td>
<td>a3</td>
<td>LASTZ</td>
<td>a3</td>
</tr>
<tr>
<td>7</td>
<td>Z/</td>
<td>a1/a3</td>
<td>Z/</td>
<td>a1/a3</td>
</tr>
<tr>
<td>8</td>
<td>ZSTO 01</td>
<td>a1'</td>
<td>ZSTO 01</td>
<td>a1'</td>
</tr>
<tr>
<td>9</td>
<td>ZRUP</td>
<td>a2</td>
<td>ZRUP</td>
<td>a2</td>
</tr>
<tr>
<td>10</td>
<td>LASTZ</td>
<td>a3</td>
<td>LASTZ</td>
<td>a3</td>
</tr>
<tr>
<td>11</td>
<td>Z/</td>
<td>a2/a3</td>
<td>Z/</td>
<td>a2/a3</td>
</tr>
<tr>
<td>12</td>
<td>Z</td>
<td>3</td>
<td>Z</td>
<td>3</td>
</tr>
<tr>
<td>13</td>
<td>ST/ Z</td>
<td>ST/ Z</td>
<td>ST/ Z</td>
<td>ST* Z</td>
</tr>
<tr>
<td>14</td>
<td>ST* Z</td>
<td>a2^2/3</td>
<td>ST* Z</td>
<td>a2^2/3</td>
</tr>
<tr>
<td>15</td>
<td>*</td>
<td>a1-a2^2/3</td>
<td>*</td>
<td>a1-a2^2/3</td>
</tr>
<tr>
<td>16</td>
<td>Z-</td>
<td>a2/3</td>
<td>Z-</td>
<td>a2/3</td>
</tr>
<tr>
<td>17</td>
<td>ZRCL 02</td>
<td>a2^3/27</td>
<td>ZRCL 02</td>
<td>a2^3/27</td>
</tr>
<tr>
<td>18</td>
<td>Z^3</td>
<td>a2^3/27</td>
<td>Z^3</td>
<td>a2^3/27</td>
</tr>
<tr>
<td>19</td>
<td>ZDBL</td>
<td>a2^3/27</td>
<td>ZDBL</td>
<td>a2^3/27</td>
</tr>
<tr>
<td>20</td>
<td>ZRCR 01</td>
<td>a1</td>
<td>ZRCR 01</td>
<td>a1</td>
</tr>
<tr>
<td>21</td>
<td>ZRCR 02</td>
<td>a2/3</td>
<td>ZRCR 02</td>
<td>a2/3</td>
</tr>
<tr>
<td>22</td>
<td>Z</td>
<td>a1*a2/3</td>
<td>Z</td>
<td>a1*a2/3</td>
</tr>
<tr>
<td>23</td>
<td>Z-</td>
<td>(a2^3/27)-(a1*a2/3)</td>
<td>Z-</td>
<td>(a2^3/27)-(a1*a2/3)</td>
</tr>
<tr>
<td>24</td>
<td>ZRCL (00)</td>
<td>a0</td>
<td>ZRCL (00)</td>
<td>a0</td>
</tr>
<tr>
<td>25</td>
<td>Z+</td>
<td>q = a0 + (a2^3/27)-(a1*a2/3)</td>
<td>Z+</td>
<td>q = a0 + (a2^3/27)-(a1*a2/3)</td>
</tr>
<tr>
<td>26</td>
<td>ZHALF</td>
<td>a/2</td>
<td>ZHALF</td>
<td>a/2</td>
</tr>
<tr>
<td>27</td>
<td>Z</td>
<td>p</td>
<td>Z</td>
<td>p</td>
</tr>
<tr>
<td>28</td>
<td>Z&lt;&gt;W</td>
<td>a2^3/27</td>
<td>Z&lt;=&gt;W</td>
<td>a2^3/27</td>
</tr>
<tr>
<td>29</td>
<td>ZRCL 02</td>
<td>a2/3</td>
<td>ZRCL 02</td>
<td>a2/3</td>
</tr>
<tr>
<td>30</td>
<td>ZRCR 02</td>
<td>a1*a2/3</td>
<td>ZRCR 02</td>
<td>a1*a2/3</td>
</tr>
<tr>
<td>31</td>
<td>ZRCR 01</td>
<td>a1</td>
<td>ZRCR 01</td>
<td>a1</td>
</tr>
</tbody>
</table>

32| 3           | 3            | 3           | 3                   |
| 33| ST/ Z       | ST/ Z        | ST/ Z       | ST/ Z               |
| 34| /           | /            | /           | /                   |
| 35| Z/          | 3q/2p        | Z/          | 3q/2p               |
| 36| LASTZ       | p/3          | LASTZ       | p/3                 |
| 37| ZSQRT       | sqrt(p/3)    | ZSQRT       | sqrt(p/3)           |
| 38| ZSTO (00)   | ZSTO (00)    | ZSTO (00)   | ZSTO (00)           |
| 39| Z/          | 3q/2p / sqrt(p/3) | Z/      | 3q/2p / sqrt(p/3)  |

40| ZHASIN       | ZASIN        | ZHASIN      | ZASIN               |

41| 3           | 3            | 3           | 3                   |
| 42| ST/ Z       | ST/ Z        | ST/ Z       | ST/ Z               |
| 43| /           | /            | /           | /                   |
| 44| ZRPL^       | Fill complex stack | ZRPL^     | Fill complex stack |
| 45| ,002        | ,002         | ,002        | ,002                |
| 46| STO 02      | STO 02       | STO 02      | STO 02              |
| 47| RDN         | RDN          | RDN         | RDN                 |

48| LBL 02      | Data output loop | LBL 02     | Data output loop    |
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 many FOCAL programs by replacing the standard functions one-to-one 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.
Roots of Complex Polynomials. \{ZPROOT, ZPLRT\}

<table>
<thead>
<tr>
<th>ZPROOT</th>
<th>Roots of Polynomials</th>
<th>Data entry/output</th>
<th>Valentín Albillo</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZPLRT</td>
<td>Polynomial Roots</td>
<td>Uses Newton method</td>
<td>Martin - Baillard</td>
</tr>
</tbody>
</table>

These programs calculate 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.

- The first one 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. The program includes data entry and output, simply answer the prompts as they're presented. See the program listing in the appendix below.

- The second is a direct implementation of the Newton method combined with a deflation technique for each root found using the iterative process. It is based on JM Baillard’s example for real roots (see paragraph #1.f at: [http://hp41programs.yolasite.com/polynomials.php](http://hp41programs.yolasite.com/polynomials.php)), simply replacing the standard HP-41 functions with 41Z equivalents – to make it valid in the complex domain. This method takes advantage of the polynomial evaluation and first derivative MCODE functions \(ZPL\) and \(ZPLD1\), which should reduce considerably the execution time provided that a good initial guess is provided.

The routine assumes the polynomial coefficients are stored in Complex Data registers \(ZR(bbb)\) to \(ZR(eee)\) - the initial guess is the \((Z,Y)\) stack registers, and the polynomial control word “\(bbb.eee\)” in the X-register (using Complex Data register indexes). You can automate the data entry process using sub-function ZINPT, make sure that the first complex register used is no lower than \(ZR03\) (i.e. real registers \{R06 and R07\}).

```
01 LBL "ZPRT" bbb,eee 16 Z/ 31 LBL 02
02 STO 02 17 ZST- (00) 32 ZRC* (00)
03 STO 03 reg range 18 ZRCL (00) 33 ZST+ IND 05
04 STO 04 19 Z=0? 34 ZRCL IND 05
05 RDN 20 SIGN 35 ISG 05
06 ZSTO (00) 21 Z/ 36 GTO 02
07 ISG 04 22 ZMOD 37 ZRCL (00)
08 LBL 01 23 E- 8 tolerance 38 ZSTO IND 05
09 ZRCL (00) 24 X<Y? 39 ISG 04
10 ZAVIEW 25 GTO 01 40 GTO 01
11 RCL 03 reg range 26 E-3 0,001 41 RCL 02
12 ZPL 27 ST- 03 deflate pol 42 E
13 LASTZ 28 RCL 03 reduced deg 43 + bbb+1,eee
14 RCL 03 reg range 29 STO 05 used as index 44 END
15 ZPLD1 (ZF# 47) 30 CLZ
```

Registers used by ZPLRT.

The program uses \(ZR00\) (i.e. \{R00-R01\}) to hold the current complex guess, and registers R02-R05 for scratch. Therefore the polynomial coefficients cannot be stored in complex registers below \(ZR03\) (i.e. \{R06-R07\}).

This method convergence is quite fast, which also contributes to the general good performance. This however is conditioned to a good initial guess as entered by the user.

(c) Ángel M. Martin – May 2021
**Example 1.** Calculate the roots of \( P(x) = 2x^5 + 3x^4 - 35x^3 - 10x^2 + 128x - 74 \)

Using `ZINPT` we introduce the six coefficients in registers \( \{ZR03 - ZR08\} \), then enter the guess \( z_0=(1+0i) \) and the control word for the polynomial, then execute the routine. The successive values are shown, and upon completion the control word of the roots is left in \( X \) – so you can use `ZOUPT` to review. The roots are all real, with values:

\[
\begin{align*}
  ZRCL 03 & \rightarrow -4.373739462 \\
  ZRCL 04 & \rightarrow -2.455070118 \\
  ZRCL 05 & \rightarrow 2.984066207 \\
  ZRCL 06 & \rightarrow 1.641131729 \\
  ZRCL 07 & \rightarrow 0.703611645
\end{align*}
\]

**Example 2.** Calculate the three roots of: \( x^3 + x^2 + x + 1 \)

\[
\begin{align*}
  \text{XEQ } "ZPROOT" & \rightarrow "\text{DEGREE=}?" \\
  3, \text{ R/S} & \rightarrow "\text{IM^RE (3)=?}" \\
  0, \text{ ENTER^, 1, R/S} & \rightarrow "\text{IM^RE (2)=?}" \\
  0, \text{ ENTER^, 1, R/S} & \rightarrow "\text{IM^RE (1)=?}" \\
  0, \text{ ENTER^, 1, R/S} & \rightarrow "\text{IM^RE (0)=?}" \\
  0, \text{ ENTER^, 1, R/S} & \rightarrow "\text{SOLVING...}"
\end{align*}
\]

-5,850E-14-j1 (that is, -i)  
5,850E-14+j1 (that is, i)  
-1+j1,170E-13 (that is, -1)

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

\[
\begin{align*}
  \text{XEQ } "ZPROOT" & \rightarrow "\text{DEGREE=}?" \\
  4, \text{ R/S} & \rightarrow "\text{IM^RE (4)=?}" \\
  2, \text{ ENTER^, 1, R/S} & \rightarrow "\text{IM^RE (3)=?}" \\
  2, \text{ CHS, ENTER^, 1, CHS, R/S} & \rightarrow "\text{IM^RE (2)=?}" \\
  3, \text{ CHS, ENTER^, CHS, R/S} & \rightarrow "\text{IM^RE (1)=?}" \\
  0, \text{ ENTER^, 1, R/S} & \rightarrow "\text{IM^RE (0)=?}" \\
  0, \text{ ENTER^, 1, CHS, R/S} & \rightarrow "\text{SOLVING...}"
\end{align*}
\]

-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:

\[
\begin{align*}
  z_1 & = 1,698 + 0,802 j \quad \text{or: } 1,878 < 25,27 \\
  z_2 & = -0,400 - 0,859 j \quad \text{or: } 0,948 <-114,976 \\
  z_3 & = 0,358 + 0,130 j \quad \text{or: } 0,381 < 9,941 \\
  z_4 & = -0,656 - 0,073 j \quad \text{or: } 0,660 <-173,676
\end{align*}
\]

(*) You can also use the Z-pad to input real coefficients, i.e. \( [Z] \), 1 instead of 0, ENTER^, 1.
Appendix. - Program Listing for ZPROOT

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* 2
4 PROMPT 47 FIX 2 90 STO IND 05 133 *
5 STO 2 48 RND 91 RCL 04 134 DSE 08
6 ST+X 49 FIX 6 92 STO IND 06 135 GTO 02
7 X 50 X07 93 DSE 00 136 RTN
8 + 51 GTO 01 94 GTO 06 137 LBL 00
9 PSIZE 52 SIGN 95 RCL 01 138 RCL 04
10 RCL Z 53 STO 04 96 INT 139 RCL 03
11 STO 00 54 RCL 00 97 E1 140 Z*
12 STO 03 55 STO 08 98 - 141 RCL IND 05
13 9,008 56 SF 01 99 E3 142 FS? 01
14 + 57 XEQ 11 100 / 143 RCL IND 06
15 X<>Y 58 R-P 101 ST- 05 144 RCL 08
16 STO 01 59 R-P 102 ST- 05 145 FS? 01
17 STO 05 60 1/X 103 FIX 3 146 * 147 +
18 X<>Y 61 STO 07 104 SF 21 148 FS? 00
19 X<>Y 62 X<>Y 105 RCL 01 149 STO IND 05
20 - 63 GTO 06 106 ISG 00 150 X<>Y
21 STO 02 64 STO 08 107 NOP 151 RCL IND 06
22 FIX 0 65 CF 01 108 RCL IND 06 152 RCL IND 05
23 CF 29 66 XEQ 11 109 RCL IND 05 153 LBL 08
24 LBL 05 67 ZENTER^ 110 ZAVIEW 154 FS? 01
25 "IM^RE(" 68 RCL 08 111 DSE 06 155 *
26 N 69 RCL 07 112 DSE 05 156 +
27 ARC03 70 P-R 113 GTO 10 157 FS? 00
28 "J=j?" 71 Z* 114 CF 21 158 STO IND 06
29 PROMPT 72 ST- 03 115 SF 29 159 X<>Y
30 STO IND 05 73 X<>Y 116 RTN 160 FS? 01
31 X<>Y 74 ST- 04 117 LBL 10 161 DSE 08
32 STO IND 06 75 ZRND 118 RCL 01 162 LBL 02
33 DSE 03 76 ZRND 119 STO 05 163 DSE 06
34 DSE 06 77 GTO 01 120 RCL 02 164 DSE 05
35 DSE 05 78 FIX 0 121 STO 06 165 GTO 00
36 "FOUND ROOT#" 79 122 FC? 01 166 END
37 GTO 05 80 ARCL 00 123 GTO 13
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

(c) Ángel M. Martin – May 2021

41Z Deluxe User Manual  Page 55 of 124
10. 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 six functions in total, three completely written in machine code and three as FOCAL programs, with a couple of example applications to complement it.

Table 10.1. Gamma function group.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZGAMMA</td>
<td>Complex Gamma function</td>
<td>for z#-k, k=integer Does LastZ</td>
</tr>
<tr>
<td>ZLNG</td>
<td>Gamma Logarithm</td>
<td>see below Does LastZ</td>
</tr>
<tr>
<td>ZPSI</td>
<td>Complex Digamma (Psi)</td>
<td>see below Does LastZ</td>
</tr>
<tr>
<td>ZIGAM</td>
<td>Inverse of Gamma</td>
<td>Iterative method FOCAL program</td>
</tr>
<tr>
<td>ZPSIN</td>
<td>Complex Poly-Gamma</td>
<td>See below FOCAL program</td>
</tr>
<tr>
<td>ZZETA</td>
<td>Complex Riemann Zeta</td>
<td>For z#1 FOCAL program</td>
</tr>
</tbody>
</table>

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. To remark that ZGAMMA is implemented completely in machine code, even for Re(z)<0 using the reflection formula for analytical continuation.

For complex numbers on the positive semi-plane [Re(z)>0], the formula used is as follows:

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

And the following identity (reflection formula) is used for numbers in the negative semi-plane: [Re(z)<0]: which can be re-written as:

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

For cases when the real part of the argument is negative [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 \(\Gamma(1/2) = SQR(\pi)\)

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. Also below the Stirling’s approximation for Gamma:

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

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

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

1, ENTER\(^\wedge\), ZGAMMA, LASTZ, ZLNG, ZEXP, Z- \( \rightarrow 2,400E-9+j3,000E-10 \)
Digamma and LogGamma Functions \{ ZPSI, ZLNG \}

Both the Digamma and LogGamma are implemented entirely in MCODE – fast execution and full LastZ support of the original argument. No data registers are used, no additional complex stack levels.

The formula used is the approximation for Digamma when \( x > 8 \):

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

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

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

\[
\Psi(x + 1) = \Psi(x) + \frac{1}{x}
\]

Equivalent 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).

\[
\begin{align*}
\text{01} & \quad \text{LBL "ZPSI"} \\
\text{02} & \quad \text{ZREPL}^\text{Z} \\
\text{03} & \quad 7 \ E^{-3} \\
\text{04} & \quad \text{STO O} \\
\text{05} & \quad \text{CLZ} \\
\text{06} & \quad \text{LBL 00} \\
\text{07} & \quad \text{Z<>W} \\
\text{08} & \quad \text{RCL O} \\
\text{09} & \quad \text{INT} \\
\text{10} & \quad + \\
\text{11} & \quad \text{ZINV} \\
\text{12} & \quad \text{Z}+ \\
\text{13} & \quad \text{ISG O} \\
\text{14} & \quad \text{GTO 00} \\
\text{15} & \quad \text{ZSTO} \\
\text{16} & \quad \text{E} \\
\text{17} & \quad \text{Z<>W} \\
\text{18} & \quad 8 \\
\text{19} & \quad + \\
\text{20} & \quad \text{ZINV} \\
\text{21} & \quad \text{ZSTO (00)} \\
\text{22} & \quad \text{Z}^2 \\
\text{23} & \quad \text{ZREPL} \\
\text{24} & \quad 20 \\
\text{25} & \quad \text{ZREAL}^\text{Z} \\
\end{align*}
\]

for \( x > 8 \)

\[
\Psi(x) = \log x - 1/(2x) - 1/(12x^2) + 1/(120x^4) - 1/(252x^6) + 1/(240x^8)
\]

together with the relationship: \( \Psi(x+1) = \Psi(x) + 1/x \)

\[
\begin{align*}
\text{01} & \quad \text{LBL "ZG"} \\
\text{02} & \quad \text{ZENTER}^\text{Z} \\
\text{03} & \quad \text{X}>Y \\
\text{04} & \quad \text{XH0?} \\
\text{05} & \quad \text{GTO 00} \\
\text{06} & \quad \text{X}>Y \\
\text{07} & \quad \text{X>0?} \\
\text{08} & \quad \text{GTO 00} \\
\text{09} & \quad \text{INT} \\
\text{10} & \quad \text{LASTX} \\
\text{11} & \quad \text{XHY?} \\
\text{12} & \quad \text{GTO 00} \\
\text{13} & \quad 0 \\
\text{14} & \quad \text{1/X} \\
\text{15} & \quad \text{LBL 00} \\
\text{16} & \quad \text{ZRDN} \\
\text{17} & \quad \text{CF 00} \\
\text{18} & \quad \text{X<0?} \\
\text{19} & \quad \text{SF 00} \\
\text{20} & \quad \text{FS? 00} \\
\text{21} & \quad \text{ZNEG} \\
\text{22} & \quad \text{FS? 00} \\
\text{23} & \quad \text{INCK} \\
\text{24} & \quad \text{ZGAMMA} \\
\text{25} & \quad \text{FC? 00} \\
\text{26} & \quad \text{GTO 01} \\
\text{27} & \quad \text{LASTZ} \\
\text{28} & \quad \text{ZGNZG} \\
\text{29} & \quad \text{Z<>W} \\
\text{30} & \quad \text{Z/} \\
\text{31} & \quad \text{LBL 01} \\
\text{32} & \quad \text{ZAVIEW} \\
\text{33} & \quad \text{END}
\end{align*}
\]
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: \( \ln \Gamma(z) = \ln \Gamma(z+7) - \ln[\prod(z+k)] | k=1,2..6 \)

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.

```
01 LBL "ZLNG"
 02 7
 03 +(z+7)
 04 ZSTO (00)
 05 \text{-Nop}
 06 6
 07 ZX
 08 810
 09 ST+Z
 10 *
 11 ZINV
 12 ZRCL (00)
 13 ZINV
 14 ZSINH
 15 ZRCL (00)
 16 Z*
 17 Z+
 18 ZLN
 19 ZRCL (00)
 20 ZLN
 21 ZDBL
 22 Z+
 23 2
 24 -
 25 ZRCL (00)
 26 Z*
 27 ZRCL (00)
 28 ZLN
 29 Z-
 30 PI
 31 ST+X
 32 LN
 33 +
 34 ZHALF
 35 ZRCL (00)
 36 \text{-Nop}
 37 7
 38 -z
 39 ZGPRD
 40 ZLN
 41 Z-
 42 ZAVIEW
 43 END
```

```
01 LBL "ZLNG2"
 02 1
 03 STO 02
 04 RDN
 05 ZSTO (00)
 06 XEQ 05
 07 LBL 00
 08 ZENTER^\[\text{-Nop}
 09 XEQ 05
 10 Z+
 11 Z=WR?
 12 GTO 02
 13 GTO 00
 14 LBL 02
 15 ZRCL (00)
 16 ZLN
 17 Z-
 18 ZRCL (00)
 19 0,5772156649
 20 ST+Z
 21 *
 22 Z-
 23 ZANVIEW
 24 RTN
 25 LBL 05
 26 Z*
 27 ZRCL (00)
 28 ZLN
 29 Z-
 30 PI
 31 ST+X
 32 LN
 33 +
 34 ZHALF
 35 ZRCL (00)
 36 \text{-Nop}
 37 7
 38 -z
 39 ZGPRD
 40 ZLN
 41 Z-
 42 ZAVIEW
 43 END
```

The table above shows the correspondence between the complex register number (ZRnn) and the required SIZE in the calculator. Note that a minimum of SIZE 002 is required for ZR00 to exist.
Poly-Gamma Function \{ \texttt{ZPSIN} \}

To complete the set of Gamma-related functions, here’s a 41Z version of JM Baillard’s program to calculate \( \text{PSIN} \) for a generic integer degree \( n \). The program listing is given below, notice the usage of the STO math functions as well as other 41Z fixtures (like the complex stack and data register management) showcasing the applicability of the function set.

```
1 LBL "ZPSIN"
02 STO 09
03 RDN
04 ZSTO 01
05 CLX
06 STO 04
07 STO 05
08 LBL 01
09 ZRCL 01
10 ZSTO 00
11 RCL 09
12 8
13 +
14 X<Y?
15 GTO 00
16 CLX
17 E
18 RCL 09
19 +
20 CHS
21 Z^X
22 ZST+ 02
23 E
24 ST+ 02
25 GTO 01
26 LBL 00
27 ZRCL 00
28 ZSTO 03
29 ZRCL 01
30 Z^2
31 ZINV
32 ZSTO 00
33 ZSTO 01
34 RCL 09
35 9
36 +
37 FACT
38 39.6
39 /
40 0
41 X<>Y
```

Examples: Calculate (\( n=2 \)) Tetra- and (\( n=3 \)) Penta-gamma of \( z=1+i \), and \( w=-1-i \):

1. \( \text{ENTER}^\wedge, \text{ENTER}^\wedge, 2, \text{ZF$ \{ \text{ZPSIN} \} \} \Rightarrow \text{"RUNNING..."} \Rightarrow 0.369+J0.767 \)
2. \( \text{ENTER}^\wedge, \text{ENTER}^\wedge, 3, \text{LASTF} \Rightarrow \text{"RUNNING..."} \Rightarrow -1.523-30.317 \)
3. \( \text{CHS, ENTER}^\wedge, \text{ENTER}^\wedge, 2, \text{ZF$ \{ \text{ZPSIN} \} \} \Rightarrow \text{"RUNNING..."} \Rightarrow -0.131+J0.733 \)
4. \( \text{CHS, ENTER}^\wedge, \text{ENTER}^\wedge, 3, \text{LASTF} \Rightarrow \text{"RUNNING..."} \Rightarrow 2.977+J0.317 \)
Inverse Gamma Function and Catalan Numbers \{ ZIGAM, ZCTLN \}

Here’s the extension to the complex realm of the Inverse Gamma function first introduced in the SandMath module. Like its real variable counterpart, this is not a very useful beyond the academic interest: after all, who needs to know what arguments yield a given gamma function result?

Well if you’d ever need to know, here’s where you can get *some* answers - and I deliberately say some because in the complex plane this is a multi-valued function, which it’s yet to be seen whether it has any formation rule for the different branches... but that, I guess, is another story altogether.

You can refer to the SandMath manual for a description of the algorithm used, which is applied directly here simply replacing the real functions with their complex counterparts.

The function is located in the -DELUXE section of the auxiliary FAT, and you can access it either by means of the sub-function launchers or via the extended “General Methods” launcher, ΣZL, [A], [R/S]

Example 1: Obtain a complex value `z` which yields \( \Gamma(z) = 1+i \)

```
1, ENTER^, [Z], ALPHA “ZIGAM” -> “RUNNING…”
0 0.412574972 – J 0.404915377
```

Note that the function follows an iterative process (Newton’s method actually). Each time an iteration is completed the program shows the module of the difference between the current and previous arguments, which when convergence exists it will be decreasing until it’s less that the 1 E-8 tolerance used.

Example 2: Use the ZLASTF feature to obtain which real value `x` yields \( \Gamma(x) = 2 \)

```
[ Z ], 2, [ Z ], [ , ], [ , ]
```

```
0.442877396 – J 9.0000000 E-24
```

Complex Catalan Numbers

Based on the classic combinatorian definition, one can extend the concept using the Gamma function instead of the factorials as follows:

\[
C_n = \frac{4^n \Gamma(n + 1/2)}{\sqrt{\pi} \Gamma(n + 2)},
\]

Where `n` is a complex number (not necessarily an integer). So we see it basically consists of two calculations of the Gamma function, which in the 41Z module is conveniently implemented as an MCODE function – so a trivial FOCAL routine does the trick.

Examples: Obtain the `C(n)` values for `n=1, 2, 3,` and `i+1`

\[
\begin{align*}
C(1) &= 1.000000001 + J 0 \\
C(2) &= 1.999999990 + J 0 \\
C(3) &= 5.000000011 + J 0 \\
C(1+i) &= 0.661301105 + J 0.443764974 \\
\end{align*}
\]
10.1. Riemann’s Zeta function. \{ ZZETA \}

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 \( \zeta(2) \)

\[
\begin{align*}
2, \text{ ZREAL}^{^\wedge}, \text{ZZETA} & \rightarrow 1,645+J0 \\
\text{FIX 9} & \rightarrow 1,644934066
\end{align*}
\]

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:

- **For the case Re(z)<0.5**, 2 formulas may be used

\[
\begin{align*}
\zeta(z) &= \zeta(1-z) \cdot 2^z \pi^{z-1} \sin(\pi z/2) \Gamma(1-z) \\
\zeta(z) &= \zeta(1-z) \pi^{z-1/2} \Gamma((1-z)/2) / \Gamma(z/2)
\end{align*}
\]

- **If Re(z) >=0.5**

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

where:

\[
\chi(z) = \sum((-1)^k/k^z), \quad k=0,1,2,...
\]

is calculated by:

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

where:

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

with an error:

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

Note that dk is calculated using the following approach:

\[
dk = e(0)+e(1)+...+e(k)
\]

where:

- e(0)=1 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"
02 X=0? ST+ X
03 GTO 00 LN1+X
04 .5 3 E10
05 CHS +
06 ZAVIEW 8
07 RTN
08 LBL 00 SQRT
09 CF 00 3
10 ZSTO 03 R06 - Re(z)
11 ZSTO 00 R07 - Im(z)
12 .5 +
13 X<=Y? INT
14 GTO 00 X^2
15 SF 00 114 ST/ 08
16 SIGN DSE 10
17 - DSE 09
18 ZNEG R06:  - Im(z)
19 ZSTO 03 R07: 0,5-Re(z)
20 XEQ 00 STO 05
21 ZRCL 00 STO 04
22 ZNEG CHS
23 E X<>Y
24 + Y^X
25 2 ST/ Z
26 ST/ Z 126 P-R
27 / CLX
28 ZGAMMA STO 08
29 Z 79 STO 09
30 ZRCL 00 129 E
31 ,5 ZRCL 03
32 - ZNEG
33 PI RCL 10
34 X*Z 2*X-1
35 Z* RCL 05
36 ZRCL 00 RCL 04
37 2 CHS
38 ST/ Z STO 04
39 / ST/ T
40 ZGAMMA ST+ Z
41 Z/ RCL 10
42 ZAVIEW ZST+ 04
43 RTN ZAVIEW
44 LBL 00 ENTER^
45 PI ST+ Y
46 2 ST* Y
47 / ST+ T
48 RCL 06 Re(z)
49 ABS ST*
50 ST* Y RCL 02
```
10.2 Lambert W function. \{ \texttt{ZWL}, \texttt{ZAWL} \}

<table>
<thead>
<tr>
<th>ZWL</th>
<th>Lambert W(z)</th>
<th>FOCAL program</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZAWL</td>
<td>Inverse of Lambert-W</td>
<td>( z^* e^{z} )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Does LastZ</td>
</tr>
</tbody>
</table>

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 \texttt{Z=WR?} Function on FIX 8 mode to determine that two consecutive iterations are equal.

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

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

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

Note that \texttt{ZWL} is a FOCAL program, and thus you won’t be able to use LASTZ to recover the initial argument. This is common to all the function implemented as FOCAL routines instead of full MCODE functions.

Examples. Calculate \( W(1+i) \) and trace back the original argument using the inverse function.

\[
\begin{align*}
1, \text{ENTER}^\wedge, \text{XEQ} \ "ZWL" & \rightarrow 0.657+0.325 \text{J} \\
\text{ZAWL} & \rightarrow 1.000+1.000 \text{J}
\end{align*}
\]

Another version using SOLVE is listed in section 12.2, with slightly more accurate results, but significantly slower execution and a few trouble spots (near 1/e and -1/e).
11. Complex Means, Elliptic integrals and DFT.

This section covers the MCODE functions to calculate Arithmetic, Harmonic and Geometric single Means on a set of data, plus the dual means AGM and GHM of two complex arguments. These are related to the Elliptic Integrals, also obtained via the Hypergeometric function and other dedicated formulas.

Table 10.1. Complex Means Functions.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Control word</th>
<th>Arguments</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZAMN</td>
<td>Complex Arithmetic Mean</td>
<td>Control word</td>
<td>ZRbbb to ZReee</td>
<td>bbbeee</td>
</tr>
<tr>
<td>ZHMN</td>
<td>Complex Harmonic Mean</td>
<td>Control word</td>
<td>ZRbbb to ZReee</td>
<td>bbbeee</td>
</tr>
<tr>
<td>ZGMN</td>
<td>Complex Geometric Mean</td>
<td>Control word</td>
<td>ZRbbb to ZReee</td>
<td>bbbeee</td>
</tr>
<tr>
<td>ZAGM</td>
<td>Complex Arithmetic-Geometric Mean</td>
<td>Arguments in Z, W</td>
<td>Does LastZ</td>
<td></td>
</tr>
<tr>
<td>ZGHM</td>
<td>Complex Geometric-Harmonic Mean</td>
<td>Arguments in Z, W</td>
<td>Does LastZ</td>
<td></td>
</tr>
</tbody>
</table>

For the single means the data is expected to be stored in a contiguous set of Complex Data registers, ZRbbb to ZReee. You can use the utility ZINPT to populate those registers. The functions require the control word in the X-register to define the register range for the calculation.

Example 1. Calculate the three single means for the set of complex values stored in the following data registers: ZR00 = -1 – i; ZR01 = 1 + i; ZR02 = 2 + 2i; ZR03 = 3 + 3i

\[
0.003, ZF$"ZAMN" => 1.250(1+J) \\
0.003, ZF$"ZHMN" => 4.800(1+J) \\
0.003, ZF$"ZGMN" => 2.213+J0
\]

For the dual means, the same definitions for real numbers hold in the complex plane. There’s no special considerations to the Arithmetic and harmonic means of complex arguments, but since the n-th root is used in the Geometric mean, it’d have a multi-value result. This becomes of singular importance in the calculation of the arithmetic-geometric mean of two values, as the convergence has many different paths – all leading to different final results.

The implementation uses the following criteria for chosen value of the geometric mean, c = sqr(ab) (see: [https://www.math.leidenuniv.nl/scripties/carls.pdf](https://www.math.leidenuniv.nl/scripties/carls.pdf)): if it is the “correct” square root for the geometric mean step, then Im[c/(a+b)] is strictly positive (i.e. > 0), otherwise replace c with -c. Furthermore this imposes the condition that Im(c) and Im(a+b) have the same sign.

Example 2. Calculate the dual means AGM and GHM for the complex pair: z = 2-4i and w = -3+i

\[
4, CHS, ENTER^, 2, ZENTER^  \quad 2-J4 \\
1, ENTER, 3, CHS, XEQ "ZAGM" \quad => -1.343-J2.146
\]

Exact result:

\[\text{agm}(-3+i, 2-4i)\]

Decimal approximation:

\[-1.24323272827549983418068907380475842661951392973543524755... \quad -2.145601106378194819897575380210627218579039692969301655... i\]

\[
4, CHS, ENTER^, 2, ZENTER^  \quad 2-J4 \\
1, ENTER, 3, CHS, ZF$"ZGHM" \quad => -4.268-J3.604
\]

Which verifies the known relationship:

\[M(x, y) = \frac{1}{AG\left(\frac{1}{x}, \frac{1}{y}\right)}\]
11.1 Complex Elliptic Integrals.

Table 10.2. Complex Elliptic Integrals.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Arguments</th>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZELIP1</td>
<td>Incomplete Ellipt. 1\textsuperscript{st} kind</td>
<td>Complex Argument in {Z,Y}, real modulus in X</td>
<td>Needs SandMath</td>
</tr>
<tr>
<td>ZELIP2</td>
<td>Incomplete Ellipt. 2\textsuperscript{nd} kind</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZELPE</td>
<td>Complete Ellipt. 1\textsuperscript{st} kind</td>
<td>Complex Modulus in ”Z”</td>
<td>Uses ZHGF</td>
</tr>
<tr>
<td>ZELPK</td>
<td>Complete Ellipt. 2\textsuperscript{nd} kind</td>
<td></td>
<td>Uses ZAGM</td>
</tr>
<tr>
<td>ZELK</td>
<td>Uses ZAGM for Complete Ellipt. Integral</td>
<td></td>
<td>JM Baillard</td>
</tr>
<tr>
<td>ZELPKE</td>
<td>Both Complete Integrals</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The Elliptic integrals are covered in several FOCAL programs as shown in the table above. Note that:

- For the Incomplete types the amplitude can be a complex number but the modulus is expected to be a real value. This method uses dedicated formulas that apply the real expressions on a repeated basis according to changes of variable, \textit{and it requires the SandMath module to be plugged in as well}. Here the function name ZELIP1 corresponds to F(z; m), and ZELIP2 corresponds to E(z; m).

- for the Complete types (where the amplitude is therefore 90 degrees) the modulus can be a complex number. Here two methods are available, one based on the hypergeometric function (slower and requires |modulus|<1), and another based on the complex AGM – faster and without that restriction.

- No provision is made for the case where both amplitude and modulus are complex numbers. To check the results you can use the syntax “EllipticF” and “EllipticE” on WolframAlpha using two arguments for incomplete cases or just one argument for complete cases.

Let’s see a few examples next. Be aware that the execution time can range from long to very long depending on the case. You can abort the execution pressing the R/S key at any time.

Example 1: calculate the complete Elliptic integrals for \( a = 2 + 3i \)

The first thing we notice is that \(|z|\)\textgreater{}1, thus the hypergeometric method is not going to converge – so discard using ZELIPE and ZELIPK. Being based on the AGM method, function ZELK is the faster way to obtain the 1\textsuperscript{st}. kind result - but using ZELPKE we can get both results on a single execution as follows:

\[
3, \text{ENTER}\^\wedge, 2, \text{ZF$\$}"\text{ZELPKE}" \rightarrow 1.043+0.630 \\
\text{Z<>W} \rightarrow 1.473-1.232
\]

Example 2. Calculate the incomplete Elliptic integrals for \( a = 1-i, \ m=0.5 \)

\[
1, \text{CHS, ENTER}\^\wedge, \text{CHS, ENTER}\^\wedge, .5, \text{ZF$\$"ZELIP1"} \rightarrow 0.804+11.163 \\
\text{EllipticF}(1\text{-}i, .5): \text{http://www.wolframalpha.com/input/?i=EllipticF%281-i,+.5%29}
\]

\[
1, \text{CHS, ENTER}\^\wedge, \text{CHS, ENTER}\^\wedge, .5, \text{ZF$\$"ZELIP2"} \rightarrow 1.128+0.789 \\
\text{EllipticE}(1\text{-}i, .5): \text{http://www.wolframalpha.com/input/?i=EllipticE%281-i,+.5%29}
\]

Example 3. Calculate the incomplete Elliptic integrals for \( a = 1+i, \ m=0.5 \)

\[
1, \text{CHS, ENTER}\^\wedge, \text{CHS, ENTER}\^\wedge, .5, \text{ZF$\$"ZELIP1"} \rightarrow 0.804+11.163 \\
\text{EllipticF}(1+i, .5): \text{http://www.wolframalpha.com/input/?i=EllipticF%281+i,+.5%29}
\]

\[
1, \text{CHS, ENTER}\^\wedge, \text{CHS, ENTER}\^\wedge, .5, \text{ZF$\$"ZELIP2"} \rightarrow 1.128+0.789 \\
\text{EllipticE}(1+i, .5): \text{http://www.wolframalpha.com/input/?i=EllipticE%281+i,+.5%29}
\]
Formulas used (from Abramowitz-Stegun, Section 14.4)

Writing $z = (\phi + i \psi)$ then we have for the first kind:

$$F(\phi + i \psi | m) = F(\lambda | m) + iF(\mu | 1 - m)$$

Where $\cot^2(\lambda)$ is the positive root of the quadratic equation:

$$x^2 - [\cot^2 \varphi + m \sinh^2 \psi \csc^2 \varphi - m_1]x - m_1 \cot^2 \varphi = 0$$

and $m \tan^2 \mu = \tan^2 \varphi \cot^2 \lambda - 1$.

And similarly for the second kind integral:

$$E(\varphi + i \psi \alpha) = E(\lambda \alpha) - iE(\mu 90^\circ - \alpha) + iF(\mu 90^\circ - \alpha) + \frac{b_1 + ib_2}{b_3}$$

where now:

$$b_1 = \sin^2 \alpha \sin \lambda \cos \lambda \sin^2 \mu (1 - \sin^2 \alpha \sin^2 \lambda)$$

$$b_2 = (1 - \sin^2 \alpha \sin^2 \lambda) (1 - \cos^2 \alpha \sin^2 \mu) \sin \mu \cos \mu$$

$$b_3 = \cos^2 \mu + \sin^2 \alpha \sin^2 \lambda \sin^2 \mu$$

as you can see an elaborate set of equations that requires a relatively long FOCAL program even if some functions from the SandMath really expedite things significantly. Refer to next page for the FOCAL program listing as a reference.

The solution is therefore expressed as a linear combination of the real-variable case for the Elliptic integrals, which are also included in the SandMath as functions ELIPF and LEI1 and LEI2.

The elliptic integrals have a complicated structure in the complex plane.

In[83]:= Plot3D[Im[EllipticF[px + I py, 2]], {px, 0.5, 2.5}, {py, -1, 1}, PlotPoints -> 60]
Program Listing: Incomplete Elliptic Integrals. *(SandMath required.)*

Data Registers: R00-R08 ; User flag: F1

```
01  LBL "ZELIP1"  46  RCL 02  91  SIN
02  SF 01  47  SQRT  92  *
03  GTO 00  48  1/X  93  X^2
04  LBL "ZELIP2"  49  ATAN  94  CHS
05  CF 01  50  ELIPF  95  E
06  LBL 00  51  STO 00  96  +
07  RAD  52  ZRCL 00  97  SQRT
08  STO 00  53  ZAVIEW  98  RCL 03
09  RDN  54  RTN  99  *
10  STO 01  55  LBL 00  100  RCL 01
11  RDN  56  STO 08  101  E
12  HSIN  57  RCL 00  102  P-R
13  X^2  58  RCL 02  103  *
14  /  59  SQRT  104  *
15  RCL 00  60  1/X  105  RCL 02
16  E  61  ATAN  106  STO 05
17  -  62  STO 02  107  SIN
18  *  63  SIN  108  *
19  E  64  X^2  109  X^2
20  STO T(0)  65  *  110  RCL 00
21  RDN  66  E  111  *
22  QROOT  67  -  112  RCL 01
23  X<Y?  68  CHS  113  COS
24  X<>Y  69  STO 03  114  X^2
25  STO 02  70  SQRT  115  +
26  RCL 01  71  RCL 00  116  ST/ 06
27  TAN  72  *  117  ST/ 07
28  X^2  73  RCL 02  118  RCL 08
29  *  74  E  119  ST+ 07
30  E  75  P-R  120  E
31  -  76  *  121  RCL 00
32  RCL 00  77  *  122  STO 08
33  /  78  RCL 01  123  -
34  SQRT  79  SIN  124  RCL 01
35  ATAN  80  X^2  125  LE12
36  STO 01  81  *  126  ST- 07
37  E  82  RCL 01  127  RCL 08
38  RCL 00  83  SIN  128  RCL 05
39  -  84  X^2  129  LE12
40  RCL 01  85  STO 06  130  ST+ 06
41  ELIPF  86  RCL 00  131  ZRCL 03
42  FS? 01  87  SQRT  132  ZAVIEW
43  GTO 00  88  ASIN  133  END
44  STO 01  89  COS
45  RCL 00  90  RCL 01
```

Granted, this listing doesn’t have much of a 41Z flavor to it since it really operates on real variable functions. Pulling all stops with the aid of the SandMath we deflect the complex variable with linear combinations as per the formulas shown before.
Program Listing: Complete Elliptic integrals

Data registers: R00-R08 ; no user flags.

```
01  LBL "ZELPKE"                34  ZRCL 00
02  ZNEG                       35  Z*
03  ZSTO 02                   36  ZSQRT  Geometric Mean, GM
04  E                          37  ZSTO 01
05  STO 08  counter           38  ZRCL 03
06  STO 00  real part         39  ZSTO 00
07  +  1-z                    40  CLX
08  ZSQRT  sqrt(1-z)          41  SIGN
09  ZSTO 01  initial value     42  ST+ 08
10  CLX                       43  RCL 08
11  STO 01  initial z0 = (1+0i) 44  8
12  LBL 01                    45  X>Y?
13  ZRCL 00                   46  GTO 01  -63 bytes
14  ZRCL 01                   47  ZRCL 00
15  Z-                         48  0
16  2                         49  2
17  ST/ Z                     50  ST* Z
18  /  arithmetic mean        51  *  doubles it
19  Z^2  AM^2                  52  ZINV
20  2                          53  ZPI*  more accurate
21  RCL 08  k                 54  ZSTO 00
22  Y^X  2^k                  55  ZRCL 02
23  ST* Z                     56  2
24  *                         57  ST/ Z
25  ZST- 02                   58  /  halves it
26  ZRCL 00                   59  E
27  ZRCL 02  -z - (2^k *AM^2) 60  +
28  Z+                         61  Z*
29  2                          62  ZSTO 01
30  ST/ Z                     63  ZRCL 00
31  /  halves it              64  ZAVIEW
32  ZSTO 03                   65  END
33  ZRCL 01
```

Upon completion both complete integrals of the 1st and 2nd kinds are left in the complex stack levels Z and W. They're also saved in ZR00 and ZR01 respectively.

---

Note: Many 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 however cannot be copied into main memory using COPY. Another drawback is that frequently 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.
11.2 Complex Discrete Fourier Transform {[ZDFT], [ZIDFT]}

An interesting subject on its own right, the Discrete Fourier Transform has had little coverage on the 41 platform – perhaps the single exception being JM Baillard’s Spectral Analysis pages. A reason for this scarcity may be the slow CPU speed, rendering the applicability to just academic cases for small sets of data. The advent of the 41CL and or course SW emulators make this less of an issue, as the examples below will show.

On the 41Z Deluxe the direct and inverse DFT are implemented entirely as MCODE functions. The “n” data points are expected to be in contiguous Complex Data registers, starting with ZRbbb to ZReee. Then you enter the control word “bbb.eee” - complex indexes - in the X register and call the function. You can use ZINPT to enter those values in memory.

When the execution completes the transformed data values are placed in the following block of Complex data registers {ZR(eee+1) to ZR(eee+n)}, and the new control word is left in X – so you can use ZOUPT to review the results.

This implementation just scratches the surface of the topic. It uses the straight-forward definition for the transform (not fast algorithms like in the FFT case). The code however has several shortcuts to accelerate the calculations when any of the indexes are zero – which results in an exponential value equal to one. See the formulas below for the direct (left) and inverse (right) cases.

\[
X_k \overset{\text{def}}{=} \sum_{n=0}^{N-1} x_n \cdot e^{-2\pi i kn/N}, \quad k \in \mathbb{Z} \quad x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k \cdot e^{2\pi i kn/N}, \quad n \in \mathbb{Z}
\]

Unavoidably rounding errors are the reason that some result values won’t show as integers. This is an inherent limitation of the 10-digit accuracy, which unfortunately can’t be extended to 13-digit in many areas despite being written in MCODE.

**Example.** Calculate the DFT for the set of values in the left column below. (see application at: http://calculator.vhex.net/calculator/fast-fourier-transform-calculator-fft/1d-discrete-fourier-transform).

<table>
<thead>
<tr>
<th>i</th>
<th>(a_i)</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 + 2j</td>
<td>64 + 72j</td>
</tr>
<tr>
<td>2</td>
<td>3 + 4j</td>
<td>-27.313708 + 11.313708j</td>
</tr>
<tr>
<td>3</td>
<td>5 + 6j</td>
<td>-16 + 0j</td>
</tr>
<tr>
<td>4</td>
<td>7 + 8j</td>
<td>-11.313708 - 4.686292j</td>
</tr>
<tr>
<td>5</td>
<td>9 + 10j</td>
<td>-8 - 8j</td>
</tr>
<tr>
<td>6</td>
<td>11 + 12j</td>
<td>-4.686292 - 11.313708j</td>
</tr>
<tr>
<td>7</td>
<td>13 + 14j</td>
<td>0 - 16j</td>
</tr>
<tr>
<td>8</td>
<td>15 + 16j</td>
<td>11.313708 - 27.313708j</td>
</tr>
</tbody>
</table>

To obtain the original sample (assuming complex regs C01 – C08 were used) you can do:
18,002016, REGMOVE, 1.008, XEQ “ZIDFT”

A sample size of 8 complex values takes about 25 seconds to complete on a normal-speed HP-41, and just shy of 1 second on the 41CL at Turbo-50; not bad for such a venerable machine.
The FOCAL program below is a rough equivalent of the MCODE function. Execution times for this program are about four to five times longer than the MCODE counterpart.

```
01 LBL "ZDFT"
02 CF 01
03 GTO 00
04 LBL "ZIDFT"
05 SF 01
06 GTO 00
07 STO 00     N
08 E3/E+      j,00N
09 STO M(5)   outer loop
10 RCL 00     N
11 VIEW M(5)
12 RCL 00     N
13 E3/3+      k,00N
14 RCL 5(M)   j,00N
15 INT        j
16 ST+ N(6)   dest: ZR(N+j)
17 E
18 -          j-1
19 PI
20 *          j-1
21 ST+ X(3)   2p.(j-1)
22 RCL 00     N
23 /           2p.(j-1)/N
24 STO 01
25 CLZ
26 ZSTO IND N(6) reset destination
27 *LBL 02 inner loop
28 RCL 07 k,00N
29 INT k
30 E - k-1
31 RCL 01 2p.(j-1)/N
32 FC? 01 2p.(j-1)(k-1)/N
33 * CHS
34 ZRC* IND O(7)
35 ZST+ IND N(6)
36 ISG O(7) next k
37 GTO 02 loop back
38 FC? 01
39 P-R
40 ZRCL IND 01
41 RCL 00
42 ST/ Z
43 STO IND 01
44 RCL 00
45 ISG M(5) next j
46 END
```

The functions will check that enough data registers are available. If not, the “NONEXISTENT” message will be presented; adjust the size and try again. Make sure complex data register ZR00 is not used to store the sample – which must start at ZR01. This is because (real) data registers R00 and R01 are used for scratch calculations by these functions.

Most of the following functions are complex versions of general methods, included either to illustrate actual programming of the complex number functions of the module or to provide a parallel environment to the real-variable case.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZMTV</td>
<td>Multi-valued functions</td>
<td>ÁM</td>
</tr>
<tr>
<td>ZSOLVE</td>
<td>Solves f(z)=0 by secant method</td>
<td>ÁM</td>
</tr>
<tr>
<td>ZNWT</td>
<td>Complex Step (Real) Differentiation</td>
<td>ÁM</td>
</tr>
<tr>
<td>ZHALL</td>
<td>Solves f(z)=0 by Halley's method</td>
<td>ÁM</td>
</tr>
<tr>
<td>ZDERV</td>
<td>Complex 1st &amp; 2nd Derivatives</td>
<td>Greg McClure</td>
</tr>
<tr>
<td>ZCF2V</td>
<td>Complex Continued Fractions</td>
<td>Greg McClure</td>
</tr>
<tr>
<td>ZCSX</td>
<td>Fresnel Integrals.</td>
<td>JM Baillard</td>
</tr>
<tr>
<td>ZKLV1</td>
<td>Weber &amp; Anger functions</td>
<td>JM Baillard</td>
</tr>
</tbody>
</table>

12.0 Real Functions as Complex Extensions \{ZCSX, ZKLV1\}

Here's an interesting approach to the calculation of some real-variable functions, treated as the real and imaginary parts of a complex extension that uses complex-variable arguments. Two examples are included:

1. The Kelvin functions of 1st kind, \( \text{ber}(n, x) \) & \( \text{bei}(n, x) \); and
2. The Fresnel Integrals, \( C(x) \) and \( S(x) \)

The expressions are based on the hypergeometric function, which also in the complex variable becomes a real power horse of high applicability for the programming of the routines.

\[
\text{ber}_n(x) + i \text{bei}_n(x) = \left( \frac{x (i - 1)}{\sqrt{8}} \right)^n \; _0F_1\left( n+1 ; i \frac{x^2}{4} \right) / \Gamma(n+1)
\]

\[
c(x) + i s(x) = x \; _1F_1\left( 1/2 ; 3/2 ; i \frac{x^2}{2} \right)
\]

Note that the input parameters are real values, and thus are expected to be in the real stack X- and Y- registers. The output will show a complex number, where it's to be understood it reflects the two solutions arranged as real and imaginary parts.

Example1: Calculate the Kelvin functions for \( x = \pi \) and \( n = \sqrt{2} \)

\[
2, \text{SQRT, PI, ZF$\text{ZKLV1}" \Rightarrow " \text{RUNNING...} \Rightarrow -0.674-\text{J1.597}
\]

\[
\text{FIX 9 for ber}_{\sqrt{2}}(\pi) \Rightarrow -0.674095956
\]

\[
\text{X<>Y for bei}_{\sqrt{2}}(\pi) \Rightarrow -1.597357212
\]

Example2: Calculate the Fresnel Integrals for \( x = 1.4 \)

\[
1.4, \text{ZF$\text{ZCSX}" \Rightarrow " \text{RUNNING...} \Rightarrow 0.543+\text{J0.714}
\]

\[
\text{FIX 9 for C(1,4) \Rightarrow 0.543095784}
\]

\[
\text{X<>Y for S(1.4) \Rightarrow 0.713525077}
\]

Note that \( x \) must remain "small", say \( x < 2 \). For \( x = 3 \), the errors are of the order of \( 10^{-6} \) and the results are meaningless with \( x = 4 \).
12.1 Multi-valued Functions. \{ZMTV\}

This program calculates all possible values for the multi-valued functions, including the \(n\) different \(N^{th}\) 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 \texttt{ZNEXT} prompt, followed by the \texttt{XEQ} key – i.e: \texttt{[Z]}, ["A"], [SHIFT], ["K"]

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

\begin{itemize}
  \item \texttt{A} – \texttt{ASIN}
  \item \texttt{B} – \texttt{ACOS}
  \item \texttt{C} – Nth. Root
  \item \texttt{D} – \texttt{ATAN}
  \item \texttt{E} – Ln
  \item \texttt{a} – \texttt{HSIN}
  \item \texttt{b} – \texttt{HACOS}
  \item \texttt{d} – \texttt{HATAN}
\end{itemize}

Or more succinctly:

\begin{itemize}
  \item \texttt{S:H}
  \item \texttt{C:H}
  \item \texttt{N:}
  \item \texttt{T:H}
  \item \texttt{L:}
\end{itemize}

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 \texttt{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 \texttt{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 \texttt{NXT} function set as well.

Note that in version 9L the FAT entry for \texttt{ZMTV} was removed – the same functionality exists accessed via the launcher menus. Refer to the following sections for details.

\textbf{Example:} Obtain all values of ASIN [Sin(1+j)]

\begin{verbatim}
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...
\end{verbatim}

\textit{Alternatively}, using the \texttt{NXTASN} function:

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

\begin{verbatim}
1, ENTER^, NXTASN -> 7,238+j1
Z<>W -> 2,142-j1
NXTASN -> 8,425-j1
NXTASN -> 14,708-j1
\end{verbatim}
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.
12.2 Solution to \( f(z)=0 \). \{ \text{ZSOLVE}, \text{ZHALL} \}

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):

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.
**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:

- ZR03 = Initial estimation z1,
- ZR04 = initial estimation z2
- R12 = Function’s name,
- FIX set manually to required precision.

**Example 1.**- Calculate one root of the equation: \( \text{Sinh}(z) + z^2 + \pi = 0 \)

Which we easily program using 41Z functions as follows:

```
LBL "ZT", ZHSIN, LASTZ, Z^2, 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: \( e^z = z \)

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:

\[
\begin{align*}
P2 &= (1+i)*z^2 + (-1-i)*z + (1-i) \\
P3 &= z^3 + z^2 + z + 1 \\
P4 &= (1+2i)*z^4 + (-1-2i)*z^3 + (3-3i)*z^2 + z - 1
\end{align*}
\]

Re-written using the Honer’s method as follows:

\[
\begin{align*}
P2 &= z \left[ (1-i) - z(1+i) \right] + (1-i) \\
P3 &= z \left[ 1 + z(1+z) \right] + 1 \\
P4 &= z \left\{ 1 + z \left[ (3-3i) - z \left[ (1+2i) - z(1+2i) \right] \right] \right\} - 1
\end{align*}
\]

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 complex registers ZR03 – ZR06 (i.e. 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 complex registers ZR00 – ZR02 (i.e. R00 - R05) . This removes the need to use cumbersome REGMOVE program steps with its memory-hungry control words.

The programs below can be used to obtain the roots as per the examples given before:
\[(1+i)*z^2 + (-1-i)*z + (1-i) = 0\]
\[(1+2i)*z^4 + (-1-2i)*z^3 + (3-3i)*z^2 + z - 1\]

<table>
<thead>
<tr>
<th>1</th>
<th>LBL &quot;Z2&quot;</th>
<th>1</th>
<th>LBL &quot;Z4&quot;</th>
<th>1</th>
<th>LBL &quot;Z4&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>ZREPL</td>
<td>2</td>
<td>ZREPL</td>
<td>2</td>
<td>ZREPL</td>
</tr>
<tr>
<td>3</td>
<td>E</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>ENTER^</td>
<td>4</td>
<td>ENTER^</td>
<td>4</td>
<td>Z^X</td>
</tr>
<tr>
<td>5</td>
<td>Z*</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>ZENTER^</td>
</tr>
<tr>
<td>6</td>
<td>ZENTER^</td>
<td>6</td>
<td>Z*</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>-1</td>
<td>7</td>
<td>LASTZ</td>
<td>7</td>
<td>ENTER^</td>
</tr>
<tr>
<td>8</td>
<td>ENTER^</td>
<td>8</td>
<td>Z-</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>Z+</td>
<td>9</td>
<td>Z*</td>
<td>9</td>
<td>Z*</td>
</tr>
<tr>
<td>10</td>
<td>Z*</td>
<td>10</td>
<td>ZENTER^</td>
<td>10</td>
<td>Z&lt;W</td>
</tr>
<tr>
<td>11</td>
<td>ZENTER^</td>
<td>11</td>
<td>-3</td>
<td>11</td>
<td>3</td>
</tr>
<tr>
<td>12</td>
<td>-1</td>
<td>12</td>
<td>ENTER^</td>
<td>12</td>
<td>Z^X</td>
</tr>
<tr>
<td>13</td>
<td>ENTER^</td>
<td>13</td>
<td>CHS</td>
<td>13</td>
<td>ZENTER^</td>
</tr>
<tr>
<td>14</td>
<td>CHS</td>
<td>14</td>
<td>Z*</td>
<td>14</td>
<td>-2</td>
</tr>
<tr>
<td>15</td>
<td>Z+</td>
<td>15</td>
<td>Z*</td>
<td>15</td>
<td>ENTER^</td>
</tr>
<tr>
<td>16</td>
<td>END</td>
<td>16</td>
<td>1</td>
<td>16</td>
<td>-1</td>
</tr>
<tr>
<td>17</td>
<td>+</td>
<td>17</td>
<td>Z*</td>
<td>17</td>
<td>Z*</td>
</tr>
<tr>
<td>18</td>
<td>Z*</td>
<td>18</td>
<td>Z*</td>
<td>18</td>
<td>Z*</td>
</tr>
<tr>
<td>19</td>
<td>LBL &quot;Z3&quot;</td>
<td>19</td>
<td>1</td>
<td>19</td>
<td>Z&lt;W</td>
</tr>
<tr>
<td>20</td>
<td>ZREPL</td>
<td>20</td>
<td>-</td>
<td>20</td>
<td>Z^2</td>
</tr>
<tr>
<td>21</td>
<td>END</td>
<td>21</td>
<td>ZENTER^</td>
<td>21</td>
<td>-3</td>
</tr>
<tr>
<td>22</td>
<td>+</td>
<td>22</td>
<td>Z*</td>
<td>22</td>
<td>ZENTER^</td>
</tr>
<tr>
<td>23</td>
<td>ENTER^</td>
<td>23</td>
<td>ZENTER^</td>
<td>23</td>
<td>ZENTER^</td>
</tr>
<tr>
<td>24</td>
<td>ZENTER^</td>
<td>24</td>
<td>ZENTER^</td>
<td>24</td>
<td>ZENTER^</td>
</tr>
<tr>
<td>25</td>
<td>Z*</td>
<td>25</td>
<td>ZENTER^</td>
<td>25</td>
<td>ZENTER^</td>
</tr>
<tr>
<td>26</td>
<td>Z*</td>
<td>26</td>
<td>ZENTER^</td>
<td>26</td>
<td>ZENTER^</td>
</tr>
<tr>
<td>27</td>
<td>Z+</td>
<td>27</td>
<td>ZENTER^</td>
<td>27</td>
<td>ZENTER^</td>
</tr>
<tr>
<td>28</td>
<td>+</td>
<td>28</td>
<td>ZENTER^</td>
<td>28</td>
<td>ZENTER^</td>
</tr>
<tr>
<td>29</td>
<td>ZENTER^</td>
<td>29</td>
<td>ZENTER^</td>
<td>29</td>
<td>ZENTER^</td>
</tr>
<tr>
<td>30</td>
<td>END</td>
<td>30</td>
<td>ZENTER^</td>
<td>30</td>
<td>ZENTER^</td>
</tr>
</tbody>
</table>

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
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:

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

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

<table>
<thead>
<tr>
<th>1</th>
<th>LBL &quot;ZWL&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>ZSTO</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>ZLN</td>
</tr>
<tr>
<td>5</td>
<td>ZENTERA</td>
</tr>
<tr>
<td>6</td>
<td>E</td>
</tr>
<tr>
<td>7</td>
<td>+</td>
</tr>
<tr>
<td>8</td>
<td>SF 06</td>
</tr>
<tr>
<td>9</td>
<td>&quot;WL&quot;</td>
</tr>
<tr>
<td>10</td>
<td>ASTO 06</td>
</tr>
<tr>
<td>11</td>
<td>ZSOLVE</td>
</tr>
<tr>
<td>12</td>
<td>ZAVIEW</td>
</tr>
<tr>
<td>13</td>
<td>RTN</td>
</tr>
<tr>
<td>14</td>
<td>LBL &quot;W&quot;</td>
</tr>
<tr>
<td>15</td>
<td>ZEXP</td>
</tr>
<tr>
<td>16</td>
<td>LASTZ</td>
</tr>
<tr>
<td>17</td>
<td>Z+</td>
</tr>
<tr>
<td>18</td>
<td>ZRCL</td>
</tr>
<tr>
<td>19</td>
<td>4</td>
</tr>
<tr>
<td>20</td>
<td>Z-</td>
</tr>
<tr>
<td>21</td>
<td>END</td>
</tr>
</tbody>
</table>

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:

\[
W(0) = 0 \\
W(1) = \Omega \approx 0.56714329 \ldots \\
W(e) = 1 \\
W(-1) \approx -0.31813 - 1.33723i
\]

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://mathworld.wolfram.com/LambertW-Function.html
12.3. Newton’s Method with Complex Step Differentiation.

This method is used to calculate real function derivatives, just as a quasi-magical application of complex variables. Complex step differentiation is a technique that employs complex arithmetic to obtain the numerical value of the first derivative of a real valued analytic function of a real variable, avoiding the loss of precision inherent in traditional finite differences. This is then used in Newton’s method in the usual way.

We’re concerned with an analytic function. Mathematically, that means the function is infinitely differentiable and can be smoothly extended into the complex plane. Computationally, it probably means that it is defined by a single "one line" formula, not a more extensive piece of code with if statements and for loops.

Let $F(z)$ be such a function, let $x_0$ be a point on the real axis, and let $h$ be a real parameter. Expand $F(z)$ in a Taylor series off the real axis.

$$F(x_0 + ih) = F(x_0) + ihF'(x_0) - h^2F''(x_0)/2! - ih^3F'''(x_0)/3! + ...$$

Take the imaginary part of both sides and divide by $h$

$$F'(x_0) = \frac{\text{Im}(F(x_0 + ih))}{h} + O(h^2)$$

Armed with the 41Z arsenal of functions it’s very likely that your real function can be programmed as an equation in the complex variable too. Then all it takes is to calculate the value of said complex function in a complex point close to the real argument $x_0$, offset by a very small amount in the imaginary axis $ih$. The program expects the program name in ALPHA and the values of $h$ and $x_0$ in the Y,X stack registers, and it returns the real derivative value in X. It uses data registers R00 to R02.

```
01 LBL "ZNWT"
02 ASTO 02
03 ZSTO (00)
04 LBL 00
05 FS? 10
06 VIEW 00
07 ZRCL (00)
08 XEQ IND 02
09 X<>Y
10 /
11 RCL 01
12 *
13 ST- 00
14 RND
15 X#0?
16 GTO 00
17 RCL 00
18 END
```

What’s remarkable is that with just one execution of the complex function we calculate both the real function’s value (the real part) and its derivative (the imaginary part with correction) at the same time. Note also the clever use of complex data register C00 to store $z_0 = x_0 + ih$, and then how it keeps calculating the complex function value until two successive iterations are equal for the current FIX selected in the calculator.

Something’s remarkable when the root-finding routine is almost shorter than the equation use to program the function!

Time for some examples. The first one just a simple polynomial to try our hand with the new method, taken from the MoHPC forum: [https://www.hpmuseum.org/forum/thread-6667.html](https://www.hpmuseum.org/forum/thread-6667.html)
Calculate the three roots of the third degree polynomial: $x^3 - x^2 - x + 0.5 = 0$

We program the equation as shown below:

```
01 LBL "Z3"
02 Z^3
03 LASTZ
04 Z^2
05 Z+
06 .5
07 +
08 END
```

And type:
```
ALPHA, "Z1", ALPHA
.01, ENTER^, 0, XEQ "ZNWT"   =>  0.40301587
.01, ENTER^, 2, XEQ "ZNWT"   =>  1.45174468
.01, ENTER^, -2, XEQ "ZNWT"  => -0.85476055
```

And then a more elaborate example adapted from the seminal reference:

The blog uses the function $F(x)$ given below, which does not have any real roots:

$$F(x) = \frac{e^x}{(\cos x)^3 + (\sin x)^3}$$

For our purposes let’s calculate the roots of, say $g(x) = F(x) - \pi$

```
1. LBL "Z2"
2. ZEXP
3. LASTZ
4. ZSIN
5. LASTZ
6. ZCOS
7. 3
8. Z^X
9. Z<>W
10. 3
11. Z^X
12. Z+
13. Z/
14. PI
15. -
16. END
```

And type:
```
ALPHA, "Z2", ALPHA
.01, ENTER^, 1, XEQ "ZNWT"   =>  0.79830245
```
12.4 Successive Approximations Method. \{ZSAM\}

The next application uses the successive approximation method to obtain the roots of a system of \(n\) non-linear equations, provided that the equations can be written as an explicit form of each variable. This is usually doable, but not always possible, and even when it is the method is slow – but should be a reliable approach provided that sensible initial guesses are provided.

The program includes data entry and results output routines, i.e. a classic "driver" structure for additional convenience. The core routine is adapted from JM Baillard’s FNZ posted at: [http://hp41programs.yolasite.com/approx.php](http://hp41programs.yolasite.com/approx.php)

Some modifications to the original core routine FNZ were required to adjust the register mapping to the 41Z convention. Using native 41Z functions also resulted in a code reduction, which is always a good thing.

Example. Let’s solve the system of the two equations below:

\[
\begin{align*}
z_1 &= (z_1^2 - z_2)^{1/3}, \\
z_2 &= (z_2^2 - z_1)^{1/4}
\end{align*}
\]

Programmed as follows:

```
01 LBL "Z1=" 11 LBL "Z2=" \\
02 ZRCL 01 12 ZRCL 02 \\
03 Z^2 13 Z^2 \\
04 ZRCL 02 14 ZRCL 01 \\
05 Z- 15 Z- \\
06 3 16 4 \\
07 Z^1/X 17 Z^1/X \\
08 RTN 18 END
```

Using \((1+i)\) as initial guesses for both \(z_1\) and \(z_2\), the results are obtained in a few seconds on the 41-CL, or with an emulator in Turbo mode.

\[
\begin{align*}
z_1 &= R02 + i R03 = 1.038322757 + 0.715596476 i \\
z_2 &= R04 + i R05 = 1.041713085 - 0.462002405 i
\end{align*}
\]

The program listing is provided below.

<table>
<thead>
<tr>
<th>LBL &quot;ZSAM&quot;</th>
<th>*LBL 01</th>
<th>XEQ IND T</th>
<th>E3/E+</th>
<th>&quot;F#&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIZE?</td>
<td>VIEW 02</td>
<td>ZENTER</td>
<td>ZOUPT</td>
<td>ARCLI 01</td>
</tr>
<tr>
<td>&quot;N=?”</td>
<td>CLA</td>
<td>ZRCL IND M</td>
<td>RTN</td>
<td>&quot; /? &quot;</td>
</tr>
<tr>
<td>PROMPT</td>
<td>RCL 00</td>
<td>Z-</td>
<td>LBL 00</td>
<td>ARCL IND 02</td>
</tr>
<tr>
<td>STO 00</td>
<td>ST+ X</td>
<td>ZMOD</td>
<td>RCL 00</td>
<td>STOP</td>
</tr>
<tr>
<td>3</td>
<td>STO M</td>
<td>+</td>
<td>ST+ X</td>
<td>FS?C 23</td>
</tr>
<tr>
<td>*</td>
<td>RCL 02</td>
<td>2.1</td>
<td>DSE M(5)</td>
<td>ASTO IND 02</td>
</tr>
<tr>
<td>E</td>
<td>RCL 00</td>
<td>+</td>
<td>GTO 02</td>
<td>ISG 02</td>
</tr>
<tr>
<td>+</td>
<td>ST+ X</td>
<td>STO 02</td>
<td>X&lt;&gt;Y</td>
<td>ISG 01</td>
</tr>
<tr>
<td>X&gt;Y?</td>
<td>E</td>
<td>RCL 00</td>
<td>E-8</td>
<td>GTO 05</td>
</tr>
<tr>
<td>PSIZE</td>
<td>RCL M</td>
<td>X&lt;&gt;Y</td>
<td>RCL 00</td>
<td>AOFF</td>
</tr>
<tr>
<td>XEQ 00</td>
<td>+</td>
<td>STO 01</td>
<td>X&lt;Y?</td>
<td>END</td>
</tr>
<tr>
<td>RCL 00</td>
<td>RCL IND X</td>
<td>AON</td>
<td>GTO 01</td>
<td></td>
</tr>
<tr>
<td>E3/E+</td>
<td>RDN</td>
<td>CF 23</td>
<td>RCL 00</td>
<td></td>
</tr>
<tr>
<td>ZINPT</td>
<td>*LBL 05</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\begin{align*}
z_1 &= R02 + i R03 = 1.038322757 + 0.715596476 i \\
z_2 &= R04 + i R05 = 1.041713085 - 0.462002405 i
\end{align*}
\]
Comments.

E3/E+ is a shortcut for the sequence \{ 1E3, /, 1, + \} and ARCLI 01 is the short form for \{ CF 29, FIX 0, ARCL 01, FIX 3 SF 29 \} – or other combination using functions like AIP, ARCLI, or AINT.

Note how ZRCL is happy using indirect stack arguments – written as non-merged program steps, which are automatically added by the function itself when entered in the program.

Because registers M and N are used, the execution should not be done using the single-step – as that will overwrite these registers with the intermediate results of the complex functions (which is not done in running mode).

Note that to call the respective equations, we first get the global label name in the X-register using the RCL IND X step; then do RDN and XEQ IND T. Could we have used XEQ IND X directly? It turns out not really, because surely the equation routines will use the complex Z-Stack, and that will complain if the current content of the \{X,Y\} registers cis Alpha data. There’s no real reason for this behavior, sunt so far that’s how the complex buffer reacts – thus the work-around using the T register for the call.

See below the graphics of both functions (real and imaginary parts). The solution of the system would represent where both real parts and both imaginary parts intersect.
The **ZDERV** function calculates the 1st and 2nd derivatives of a global function defined by the user (and thus visible via Catalog 1). The function needs to be continuous thru the range around the value at which the derivatives of the function are desired. The program uses data registers (R00-R08) as follows:

- **ZR00** (R00/R01) is the summing register for the first derivative partials calculated each pass. It should be left alone (read only).
- **ZR01** (R02/R03) is the summing register for the second derivative partials calculated each pass. It should be left alone (read only).
- **ZR02** (R04/R05) is the current Z for calculation by routine pointed to by alpha for XEQA. It was initialized to the value entered by the user in X, Y (and also in complex Z). It is modified by **ZDERV** each step, so it is the next value for the user routine when called. It is up to the user program to decide when to use the value (it is not required to be saved by the user if not needed at the beginning of the user program this way). It should be considered read only.
- **ZR03** (R06/R07) is the complex step size ZS entered by the user in complex stack level W. It should be left alone (read only).
- **R08** is initialized to 0 and contains the current step number (0 to 10). It is used by the logic to know when to go from Z+5*StepSize to Z-5*StepSize (right after handling step #5) and when to stop (right after step#9, when it increments to 10). So, for example, if \( z = 1+0i \) and \( ZS = 0.3+0i \), the sampling will be: 1, 1.03, 1.06, 1.12, 1.15, 0.85, 0.88, 0.91, 0.94, 0.97 (each with h +0i) for the 10 points. Again it should be left alone (read only).
- So any of registers R00 thru R08 shouldn’t be disturbed by the user program. As long as the user program name is 6 or less characters, it can be ASTO’d / ARCL’d by the program if required. The user program can use ANY of the stack registers and any of the complex stack registers, as long as the final result ends up in X,Y. Never mind that it is duplicated in complex Z, as it should be there if followed 41Z protocol for the program.
- Note that if the FOCAL user program contains high-level math complex functions (such as **ZGAMMA**) then its LBL name should also be stored in a separate data register, say R09. This is needed because the more complex functions make internal usage of the ALPHA registers, which therefore would be compromised. Should that occur you’re likely to get a “NONEXISTENT” error message when attempting the execute the user program from within **ZDERV**.

Besides the user function name in ALPHA, the program takes two input values, both of them complex: the point where the derivatives are to be evaluated and the complex step size to use for the derivative evaluation formula (this is a measure of the distance between points sampled). When developing this program, many formulas were available to use... this program uses the 10-point formulas developed by Jean-Marc Baillard.

The formulas used are exact for any complex polynomial of degree < 11 : - \( f(x+k.h) \) is denoted \( f_k \) to simplify these expressions -

\[
\frac{df}{dx} = \left( \frac{1}{2520.h} \right)[ 2100.( f_1 - f_{-1} ) - 600.( f_2 - f_{-2} ) + 150.( f_3 - f_{-3} ) - 25.( f_4 - f_{-4} ) + \\
+ 2.( f_5 - f_{-5} ) ] + O(h^{10})
\]

\[
\frac{d^2f}{dx^2} = \left( \frac{1}{25200.h^2} \right)[ -73766 f_0 + 42000.( f_1 + f_{-1} ) - 6000.( f_2 + f_{-2} ) + 1000.( f_3 + f_{-3} )]
\]
- 125.( f_4 + f_4 ) + 8.( f_5 + f_5 ) ] + O(h^{10})

The implementation of ZDERV also makes use of a hidden function, ZDRTN. It is NOT designed to be used in the user function created, which only need RTN or END to terminate the FOCAL code that defines them. Why then is XQRTN needed? The operating system normally does not allow returning to MCODE from FOCAL programs. So to overcome this restriction ZDERV jumps to a mini-FOCAL program that contains ZDRTN to execute the user function and return back to the ZDERV MCODE after doing a real RTN.

All this is transparent to the user, who needs only to provide the function name in ALPHA and the input values in the W- and Z- complex stack levels as described above. The execution ends with the first derivative value in both complex stack level Z and ZR00, and the second derivative value in both complex stack W and ZR01.

Example 1. Derivatives of SIN

Let’s say we want to find the derivative of f(z) = sin(z) at z=1. First we need to create a Global label program to define the function (as it cannot use mainframe function names). Note that there’s no need to preserve the routine name in R09 as ZSIN does not use the ALPHA registers internally.

```
01 LBL "SINZ"
02  ZSIN
03  END
```

Let’s try a step value of .03 (so the points sampled will be (.85, .88, .91, …, 1.12, 1.15).

Type: 0, ENTER^, .03, ZENTER^ => 0.030+j0
0, ENTER^, 1, XEQ "ZDERV_"SINZ" ALPHA = > "RUNNING..."

On return, both ZR00 and Z contain 0.540302302 (the actual 1st derivative is 0.54032306) and ZR01 and W contains -0.841470900 (the actual 2nd derivative is -0.841470985).

Testing the sine function for other values and step sizes is easy if you use the explicit derivatives, f'(sin(z)) = cos(z), and f''(cos(z)) = -sin(z), that is to say, you can test the values obtained by this program for this example by taking the cos(z) and -sin(z) for the actual 1st and 2nd derivative values.

Example 2.- Calculate f'(1+i) & f''(1+i) for: f(z) = exp(-z^2)

We program the function using any global LBL , 6 characters or less

```
01  LBL "EX2"
02  Z^2
03  ZNEG
04  ZEXP
05  END
```

If we choose h = 0.03(1+i) as step-size we type:

```
0.03, ENTER^, ZENTER^ => 0.003(1+j)
1, ENTER^, 1, XEQ "ZDERV_"EX2" ALPHA => "RUNNING..."
```

f'(1+i) = -0.986301184 + j2.650888353; and Z<>W
f''(1+i) = 8.106657849 - j1.510648148 ;

Choosing the best h-value is not easy but h ~ 0.03 (in both axis) "often" produces good results. Be aware that unfortunately the better step-size for the first derivative may not be a good one for the second, and vice-versa.
Example 3. Cubic Polynomial Derivatives.

With the following coefficients stored in ZR06-ZR09 (i.e. R12-R19), calculate the derivatives in \( z=1+j \) of the cubic polynomial. Use \( \text{Zstep}=0.1+0.j \). The results should be \(-3+j17\) and \(4+j16\) for the 1st and 2nd derivatives respectively, as calculated by ZDP1 and ZPD2.

\[
\begin{align*}
ZR06 &= 1+j \quad \text{- third degree coeff.} \\
ZR07 &= 2+2j \quad \text{- second degree coeff.} \\
ZR08 &= 3+3j \quad \text{- first degree coeff.} \\
ZR09 &= 4+4j \quad \text{- independent term.}
\end{align*}
\]

We start by programming the function under the user label "ZP69", taking advantage of the ZPL function to do the polynomial evaluation. Note that this uses the ALPHA register M internally for scratch, thus we need to preserve the global program name in another data register and restore it after the evaluation is done. We’ll use R09 for this purpose. Note as well that the usage of storage registers must be compatible with ZDERV requirements, which uses ZR00 to ZR03.

01 LBL “ZP69”  
02 ASTO 09 \( \text{LBL name preserved} \)  
03 \textbf{ZRCL 02} \( \text{initial argument} \)  
04 NOP \( \text{to separate numeric steps} \)  
05 6.009 \( \text{control word} \)  
06 ZPL \( \text{evaluates polyn} \)  
07 CLA \( \text{clear scratch} \)  
08 ARCL 09 \( \text{routine LBL restored} \)  
09 END

Then we enter the function parameters as usual:

\[
\begin{align*}
0.1, \ &\text{ENTER, } \text{ZENTER}^\wedge, \quad \Rightarrow \ 0.100(1+J) \\
1, \ &\text{ENTER}^\wedge, \ 1, \ \text{XEQ } \text{"ZDERV_ "ZP69" ALPHA} \quad \Rightarrow \ "\text{RUNNING...}"
\end{align*}
\]

Which shortly returns the exact solutions in the complex stack: \(-3+j17\); \(Z<>W\) \(4+j16\)


Let’s now do a high-level math example using ZGAMMA, which also messes with the ALPHA registers thus we need to save the global label in R09 in this case as well. Let’s calculate the derivatives in the point \( z_0 = 1+i \), also using a step size \( \text{zh} = 0.1 \) (1+j).

01 LBL “GAM”  
02 ASTO 09  
03 ZGAMMA  
04 CLA  
05 ARCL 09  
06 END  

\[
\begin{align*}
0.1, \ &\text{ENTER}^\wedge, \ \text{ZENTER}^\wedge, \quad \Rightarrow \ 0.100(1+J) \\
1, \ &\text{ENTER}^\wedge, \ \text{XEQ } \text{"ZDERV_ "GAM” ALPHA} \quad \Rightarrow \ 0.2140+J0.5215 \\
\text{Z<>W} \quad \Rightarrow \ -0.4338-J0.1875
\end{align*}
\]

The first derivative should equal ZPSI * ZGAMMA, and it does!

\[
\begin{align*}
1, \ &\text{ENTER}^\wedge, \ \text{ZGAMMA}, \ \text{ZSTO 02}, \ \text{LASTZ}, \ \text{ZPSI}, \ \text{ZRC}^* \ 02 \quad \Rightarrow \ 0.2140+J0.5215
\end{align*}
\]
Which can also be verified using WolframAlpha, see: 
http://www.wolframalpha.com/input/?i=gamma%281%2Bi%29*digamma%281%2Bi%29

\[ \Gamma(1 + i) \psi^{(n)}(1 + i) \]

\( \psi^{(n)}(x) \) is the \( n \)th derivative of the digamma function.

Decimal approximation:

\[
\begin{align*}
0.213967801454691705299993903701421610650192802705792900... + \\
0.52153449146659590332290379309443139938150009129698405858... i
\end{align*}
\]

Polar coordinates:

\[ r = 0.56372 \ (\text{radius}), \ \theta = 07.6933^\circ \ (\text{angle}) \]

And similarly for the second derivative using the tri-gamma function: 
http://www.wolframalpha.com/input/?i=gamma%281%2Bi%29*%28trigamma%281%2Bi%29+%2B digamma%281%2Bi%29%29^2

\[ \Gamma(1 + i) \left(\psi^{(0)}(1 + i)^2 + \psi^{(1)}(1 + i)\right) \]

Decimal approximation:

\[
\begin{align*}
-0.4337555823419010464380234637159711937527225979305979333... - \\
0.1875455480059837529787466671208797968376132790964719447... i
\end{align*}
\]

Polar coordinates:

\[ r = 0.472565 \ (\text{radius}), \ \theta = -156.617^\circ \ (\text{angle}) \]

Example 5: Halley’s Method.

This example clearly illustrates the usefulness of \texttt{ZDERV} – applied to the Halley’s method to obtain the roots of a function. Contrary to the secant algorithm, the Halley’s method only needs one initial estimation, and the convergence is meant to be faster - reducing so the execution time.

The following FOCAL program lists the code (set FIX as needed for precision):

```
01  LBL “ZHALL”
02  “FNAME?”
03  AON
04  PROMPT
05  ASTO 09
06  AOFF
07  “Z0=?”
08  PROMPT
09  ZSTO 02
10  .1
11  ENTER^  
12  ZSTO 03
13  LBL 00
14  CLA
15  ARCL 09
16  ZRCL 03
17  ZRCL 02
18  ZDERV
19  ZRCL 02
20  XEQ IND 09
21  ZRPL^ 
22  ZRCL (00)
23  ZDBL
24  ZRCL (00)
25  Z^2 
26  ZDBL
27  ZRCL 01
28  ZRUP
29  Z* 
30  Z- 
31  Z/ 
32  ZNEG
33  ZRC+ 02
34  ZENTER^ 
35  Z<> 02
36  Z=WR? 
37  GTO 01
38  GTO 00
39  LBL 01
40  ZAVIEW
41  END
```
12.6 Continued Fractions. \{ZCF2V\} (by Greg McClure)

Continued Fractions are expressions of the form:

\[
B(0) + \frac{A(1)}{B(1) + \frac{A(2)}{B(2) + \frac{A(3)}{B(3) + \ddots}}}
\]

The use of + in the denominator indicates that the remainder of the terms actually are part of that denominator. So the above expression means \(B(0) + A(1) / [B(1) + A(2) / [B(2) + A(3) / [...]]]\).

This can be mathematically abbreviated as \(B(0) + [A(1), A(2), A(3), ... ; B(1), B(2), B(3), ...]\) which will be used here. The number of expressions may or may not be infinite.

Many values are easily expressed as continued fractions. Some examples are:

- \(\text{Tanh}(x) = \left[x, x^2, x^2, x^2, x^2, ... ; 1, 3, 5, 7, ...\right]\)
- \(\pi = \left[4, 1^2, 3^2, 5^2, 7^2, ... ; 1, 2, 2, 2, 2, ...\right]\) (one of MANY representations of \(\pi\))
- \(\frac{1}{e-1} = \left[1, 2, 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8, ...\right]\) (again one of MANY representations of \(e\))

The simpler form of continued fractions often used are expressions with \(A(n) = 1\), therefore of the form: \(B(0) + 1/(B(1) + 1/(B(2) + ... 1/(B(n) + ...))\) mathematically abbreviated as: \([B(0); B(1), B(2), B(3), ...]\). For example: \(e = [2; 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8, ...]\)

The \(ZCF2V\) function is designed to calculate a complex continued fraction value. It requires a user created subroutine that calculates \(A(n)\) and \(B(n)\) for \(n \geq 1\). The function assumes \(z\) is available in ZR01 and \(n\) available in R12 for this program, and should leave \(A(n)\) in complex stack level “Z” and \(B(n)\) in complex stack level “W” on completion of the user subroutine. The subroutine must be callable by a global label (of up to 7 characters). The program uses R00 thru R12.

To execute \(ZCF2V\), put the value of \(B(0)\) in complex stack level “W”, and the value of evaluation point \(z\) in complex stack level “Z”. Execute \(ZCF2V\) to evaluate the continued fraction – which will prompt for the name of the routine that calculates both \(A(N)\) and \(B(N)\) and will write it into the alpha register to evaluate the continued fraction. In a program execution (no prompting) you need to enter the user program name in ALPHA prior to the ZCF2V step.

Here is an example of use of \(ZCF2V\). Let’s say we want to evaluate the Tanh function mentioned above. We would create the following program in memory (assume we use the label TT):

```
01 LBL "ZTH"                        11 LBL 01
02 RCL 12 ; get n from R12          09 - ; (n-1) in X
03 1 ; Is it 1?                     10 RCL 12 ; get n again
04 X#Y?                             11 + ; (2n-1) in X
05 GTO 01 ; No, skip to LBL 01      12 0
06 CLX ; make it complex            13 X<>Y ; make it complex
07 X<=Y ; make it complex           14 ZRCL 01
09 ZRCL 01 ; B(1) = 1+j0 in “W”,    15 Z^2 ; B(n) = (2n-1)+j0 in “W”,
10 RTN ; A(1) = z in “Z”           16 END ; A(n) = z^2 in “Z”
```

To evaluate Tanh(1) with \(B(0)=0\) enter the following:

```
0 , ENTER^, ZENTER^, ENTER^, 1, ZCF2V_“ZTH” ALPHA -> “RUNNING...”
```

The answer of 0.761594156 (assuming FIX 9) is displayed in a few seconds. The value returned should be accurate to at least 9 significant digits.
Try now to evaluate $\tanh(1+i)$, which answer is $1.083923328 + j \cdot 0.271752586$:

$$
\begin{align*}
\text{Z0, ZENTER^, 1, ENTER^, XEQ "ZCF2V $_$ZTH \text{ ALPHA} =>
\text{RUNNING...", => 1.0839+J0.2718}
\end{align*}
$$

Example 2. If $f(z)$ is defined by: $b_0 = 0.2 + 0.3i$; $a_n = 2z + n$; $b_n = z^2 + n^2$ ($n > 0$); evaluate $f(1+2i)$. We program the function components as follows:

```
01 LBL "FZ"
02 ZRCL 01
03 Z^2
04 RCL 12
05 X^2
06 +
07 ZRCL 01
08 2
09 ST* Z
10 *
11 RCL 12
12 +
13 END
```

Calculated as:

$$
0.3, \text{ ENTER^, 0.2, ZENTER^, 2, ENTER^, 1, XEQ "ZCF2V $_$FZ" => 1.0846-J0.7498}
$$

Register usage for ZCF2V.

First, name of function must be in Alpha (up to 7 chars allowed). The prompting makes that easy for keyboard usage!

Like it was the case for ZDERV, the user program name must be in the ALPHA register each time the function is to be evaluated. If the user program contains functions that alter the contents of ALPHA then you'll need to restore said user program name as part of the user program itself. Typically you save it on entry (say in an available data register like R15) and restore it upon completion of the continued fraction.

- Register ZR00 is the continuing estimate of $F(N)$ and hopefully gets closer and closer to the real solution (or we wind up with an infinite loop). It should be considered read only.
- Register ZR01 is the saved value of $z$. The user program that calculates the next $A(N)$ and $B(N)$ terms can use this value. It should be considered read only.
- Register ZR02 is $C(N)$ from the modified Lentz formula. It should be considered read only.
- Register ZR03 is $D(N)$ from the modified Lentz formula. It should be considered read only.
- Register ZR04 is $A(N)$ saved from the user program. It can be reused by the user program but will be replaced on reentry to the ZCF2V calculation loop.
- Register ZR05 is $B(N)$ saved from the user program. It can be reused by the user program but will be replaced on reentry to the ZCF2V calculation loop.
- Register R12 is the current loop count ($N$). The user program that calculates the next $A(N)$ and $B(N)$ terms can use this value. It should be considered read only.

If an infinite loop is occurring, pressing R/S should stop the program on the next entry to the user program. If no infinite loop is occurring, the answer should eventually show up in $X,Y$ (Complex Z).
Example 2. Bessel Functions $J_n(x)$ and $Y_n(x)$. {\texttt{JYNX}}

This example showcases the use of continued fractions to calculate the Bessel functions of first and second kinds, $J_n(x)$ and $Y_n(x)$, for real values of order and argument. It is a very interesting application that has the benefit to avoid the limitations of the direct methods when the order and/or argument are large. Therefore, unlike the counterpart functions in the SandMath, the following program produces accurate results for large arguments.

You should note that this approach involves solving two continued fractions, one in the complex domain and another in the real domain – therefore both the 41Z and SandMath modules need to be plugged in the calculator.

Formulae:

Let ZCF be the complex continued fraction defined by:

$$ZCF = \left[ \frac{0.5^2 - n^2}{2x + 2i + \left( 1.5^2 - n^2 \right)/(2x + 4i + \ldots)} \right]$$

And CF be the real continued fraction defined by:

$$CF = -1/((2n + 2)/x - 1/((2n + 4)/x - \ldots))$$

Let: $p + i. q = -1/(2x) + i. \left( 1 + (1/x) [ZCF] \right)$

And: $s = (p - CF - n/x)$

then we have the following expressions:

$$J_n(x) = \text{sign}(D). \sqrt{\left( \frac{2q}{\pi \cdot x} / \left( (q^2 + s^2) \right) \right)}$$

$$Y_n(x) = \left[ \frac{s}{q} \right] \cdot J_n(x)$$

Numeric application:

```
10 ENTER^ XEQ "JYNX" => J10(10) = 0.207486107
X<>Y Y10(10) = -0.359814151 (in 2mn27s)
```

```
3.14, ENTER^, 100, XEQ "JYNX" => J3.14(100) = 0.079535723
X<>Y Y3.14(100) = 0.006582327 (in 4mn14s)
```

The method doesn't work if $n$ is a negative integer. However in that case, if $n < 0$ we can use the relations

$$J_n = J_{-n} \cos n.\pi + Y_{-n} \sin n.\pi \, , \text{ and}$$

$$Y_n = -J_{-n} \sin n.\pi + Y_{-n} \cos n.\pi$$

If $x < 0$ the results are generally complex and won't be properly calculated by this program.

Data Register Usage.

"JYNX" needs data registers R00 to R13. \{R00 - R12\} are used by ZCF2V, plus one additional register (R13) is needed to save the value of the order "N".

Note that both ZCF2V and CF2V have slightly different conventions as to where the arguments are stored: in CR01 for ZCF2V, which translated to R02 and R03 for the real and imaginary parts. All this is transparent to the user for this example.
The Program Listing is shown below. Note the calculation for the p and q factors takes advantage of the complex result returned by ZCFV, transposing the real and imaginary parts as per the multiplication by "i" in the definition formulae:

\[p = -\frac{1}{x} \cdot \left[ \frac{1}{2} + \text{Im}(ZCF) \right]\]
\[q = \left[ 1 + \text{Re}(ZCF/x) \right]\]

Credits: The original program was written by Jean-Marc Bailalrd, and has been adapted to use the MCODE implementations of the continued fractions routines. Thanks also to Greg McClure for his assistance provided for the adaptation.

```
01 LBL "JYNX" 30 RCL 09  p 59 LBL "ZCF"
02 STO 01  x 31 + 60 RCL 12  N
03 X<>Y 32 RCL 13  N 61 ST+ X  2N
04 STO 13 N 33 RCL 01  x 62 RCL 02  x
05 "ZCF“ 34 / 63 ST+ X  2x
06 CLST 35 –
07 ZENTER^ 36 STO 11  s 64 ZENTER^ 74 LBL "CF"
08 0 37 RCL 10  q 75 X<>Y
09 RCL 01  38 R-P 76 STO 05  Bn
10 ZCF2V 39 LASTX  q 77 X<>Y
11 RCL 02  x 40 ST+ X  2q 78 RCL 02  n
12 STO 01  41 PI 79 RCL 13 N
13 ST/ Z 42 RCL 01  x 80 +
14 / 43 * 81 ST+ X
15 E 44 / 82 RCL 01  x
16 + 45 SQRT 83 /
17 STO 10  q 46 X<>Y 84 -1
18 X<>Y 47 / 85 END
19 CHS 48 RCL 05  20 RCL 01  x
20 RCL 01  x 49 SIGN 50 *
21 ST+ X 51 STO 12  22 1/X
23 – 52 RCL 11 s 24 STO 09  p
25 "CF“ 53 * 26 0 54 RCL 10  27 RCL 01  x
28 CF2V 55 / 56 RCL 12  29 CHS
30 RCL 12 N
```

Note that this program is not available in the 41Z Module, but it has been included in the "Advantage_Math" ROM, a collection of applications using the advanced modules like the 41Z, SandMath and SandMatrix, sometimes used together.

Bessel functions for complex variable are covered in the next sections of the manual.
12.7 **Bessel and Hankel 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.

<table>
<thead>
<tr>
<th>Index</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ZJBS</td>
<td>Complex Bessel J function First kind</td>
</tr>
<tr>
<td>2</td>
<td>ZIBS</td>
<td>Complex Bessel I function First kind</td>
</tr>
<tr>
<td>3</td>
<td>ZKBS</td>
<td>Complex Bessel K function Second kind</td>
</tr>
<tr>
<td>4</td>
<td>ZYBS</td>
<td>Complex Bessel Y function Second kind</td>
</tr>
<tr>
<td>5</td>
<td>EIZ/IIZ</td>
<td>Spherical Hankel first kind order zero SHK1 (0, z)</td>
</tr>
<tr>
<td>6</td>
<td>ZSHK1</td>
<td>Spherical Hankel first kind SHK1 (n, z)</td>
</tr>
<tr>
<td>7</td>
<td>ZSHK2</td>
<td>Spherical Hankel second kind SHK2 (n, z)</td>
</tr>
<tr>
<td>8</td>
<td>ZANGJ</td>
<td>Anger Function J (n, z); n real</td>
</tr>
<tr>
<td>9</td>
<td>ZWEBE</td>
<td>Weber Function W (n, z); n real</td>
</tr>
</tbody>
</table>

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:

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

\[
U(k) = -U(k-1) * (z/2)^2 / k(k+n)
\]

\[
U(0) = 1
\]

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

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

\[
n \# \ldots -3 ; -2 ; -1 ; 0 ; 1 ; 2 ; 3..
\]

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”)

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

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
2 CF 00
3 GTO 00
4 LBL ZIBS
5 SF 00
6 LBL 00
7 CF 01
8 Z<>W
9 ZINT? is n integer?
10 XEQ 05
11 Z<>W
12 ZHALF
13 XROM “ZBS”
14 FS? 01
15 RCL 01
16 FS? 01
17 ZCHSX 
18 LBL 04
19 ZAVIEW
20 RTN
21 LBL 05
22 X<0?
23 SF 01
24 AB S
25 RTN
26 LBL “ZBS”
27 Z#0?
28 GTO 00
29 Z=W?
30 E
31 GTO 04
32 LBL 00
33 -ZSTACK running...
34 ZSTO 01 (z/2)
35 Z<>W n
36 ZSTO 00 n
37 E 1
38 ZREAL 1+i0
39 ZSTO 02 1+i0
40 ZSTO 03 1+i0
41 STO M k=1
42 LBL 02 CR00 - n
43 ZRCL 01 CR01 - Z/2
44 Z^2 (z/2)^2
45 ZRCL 02 Uk-1
46 FC? 00 CR04 - result
47 ZNEG
```

**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

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LBL &quot;ZB1&quot;</td>
<td>SUM{f(n,x)}</td>
</tr>
<tr>
<td>2</td>
<td>CLZ</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ZSTO 02</td>
<td>Jn / In</td>
</tr>
<tr>
<td>4</td>
<td>ZSTO 04</td>
<td>SUM</td>
</tr>
<tr>
<td>5</td>
<td>STO 01</td>
<td>k=0</td>
</tr>
<tr>
<td>6</td>
<td>LBL 02</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>XEQ 10</td>
<td>summing term</td>
</tr>
<tr>
<td>8</td>
<td>Z=0?</td>
<td>x=0?</td>
</tr>
<tr>
<td>9</td>
<td>GTO 01</td>
<td>ignore term</td>
</tr>
<tr>
<td>10</td>
<td>ZRCL 04</td>
<td>S(k-1)</td>
</tr>
<tr>
<td>11</td>
<td>Z+</td>
<td>S(k)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>ZENTER^</td>
<td>increase index</td>
</tr>
<tr>
<td>13</td>
<td>Z&lt; 04</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Z=W?</td>
<td>are they equal?</td>
</tr>
<tr>
<td>15</td>
<td>RTN</td>
<td>Final result(s)</td>
</tr>
<tr>
<td>16</td>
<td>LBL 01</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>ST+ 01</td>
<td>k=k+1</td>
</tr>
<tr>
<td>19</td>
<td>GTO 02</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>LBL 10</td>
<td>Function to Sum</td>
</tr>
<tr>
<td>21</td>
<td>ZRCL 01</td>
<td>x/2</td>
</tr>
<tr>
<td>22</td>
<td>RCL 01</td>
<td>k</td>
</tr>
<tr>
<td>23</td>
<td>ST+ X</td>
<td>2k</td>
</tr>
<tr>
<td>24</td>
<td>RCL 00</td>
<td>n</td>
</tr>
<tr>
<td>25</td>
<td>+</td>
<td>2k+n</td>
</tr>
<tr>
<td>26</td>
<td>Z^X</td>
<td>(x/2)^{(2k+n)}</td>
</tr>
<tr>
<td>27</td>
<td>ZENTER^</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>RCL 01</td>
<td>k</td>
</tr>
<tr>
<td>29</td>
<td>FACT</td>
<td>k!</td>
</tr>
<tr>
<td>30</td>
<td>LASTX</td>
<td>k</td>
</tr>
<tr>
<td>31</td>
<td>RCL 00</td>
<td>n</td>
</tr>
<tr>
<td>32</td>
<td>+</td>
<td>k+n</td>
</tr>
<tr>
<td>33</td>
<td>FACT</td>
<td>(k+n)!</td>
</tr>
<tr>
<td>34</td>
<td>*</td>
<td>k! * (k+n)!</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>ZREAL</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>Z/</td>
<td>k-th. Term</td>
</tr>
<tr>
<td>37</td>
<td>FS? 00</td>
<td>is it Kn?</td>
</tr>
<tr>
<td>38</td>
<td>GTO 00</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>RCL 01</td>
<td>k</td>
</tr>
<tr>
<td>40</td>
<td>ZCHSX</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>LBL 00</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>Z&lt; 02</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>ZRCL 02</td>
<td>SUM(k-1)</td>
</tr>
<tr>
<td>44</td>
<td>Z&gt; 02</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>ZENTER^</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>RCL 01</td>
<td>k</td>
</tr>
<tr>
<td>47</td>
<td>HARMN</td>
<td>H(k)</td>
</tr>
<tr>
<td>48</td>
<td>LASTX</td>
<td>k</td>
</tr>
<tr>
<td>49</td>
<td>RCL 00</td>
<td>n</td>
</tr>
<tr>
<td>50</td>
<td>+</td>
<td>k+n</td>
</tr>
<tr>
<td>51</td>
<td>HARMN</td>
<td>H(k+n)</td>
</tr>
<tr>
<td>52</td>
<td>+</td>
<td>H(k)+H(k+n)</td>
</tr>
<tr>
<td>53</td>
<td>ZREAL</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>Z*</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>END</td>
<td></td>
</tr>
</tbody>
</table>

Note: functions DECX and INCX can be replaced by standard FOCAL sequences:
The formulae used for integer orders are as follows:

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

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

\[
g_k(n,x) = (x/2)^{2k-n} \left[\frac{(n-k-1)!}{k!}\right] ; k=0,2,...(n-1)
\]

\[
f_k(n,x) = (x/2)^{2k+n} \left[\frac{H(k) + H(n+k)}{k! (n+k)!}\right] ; k=0,1,2,....
\]
Example: Calculate \( KBS(-0.5+i; 1-0.5i) \)

\[
1, \text{ ENTER}^\wedge, 0.5, \text{ CHS, ZENTER}^\wedge, \text{ ENTER}^\wedge, 1, \text{ XEQ "ZKBS"} \rightarrow 0.348 + j 0.104
\]

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

\[
0, \text{ ENTER}^\wedge, 1, \text{ CHS, ZENTER}^\wedge, \text{ XEQ "ZYBS"} \rightarrow -0.781 + j 0.880
\]

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

Example: Calculate \( JBS \) and \( IBS \) for \((1+2i, -1-3i)\)

\[
2, \text{ ENTER}^\wedge, 1, \text{ ZENTER}^\wedge, 3, \text{ CHS, ENTER}^\wedge, 1, \text{ CHS, XEQ "ZIBS"} \rightarrow 35.813 - j 191.737
\]

\[
2, \text{ ENTER}^\wedge, 1, \text{ ZENTER}^\wedge, 3, \text{ ENTER}^\wedge, 1, \text{ ZNEG, XEQ "ZJBS"} \rightarrow -257.355 - j 12.633
\]

12.7. **Hankel and Spherical Hankel functions.** \{\text{ZSHK1, ZSHK2, EIZ/IZ}\}

With the Bessel functions in the pocket it takes a little more than a trivial exercise to write a few short routines to calculate the Hankel and Spherical Hankel functions – both of the first and second kind. Their defining expressions are as follows:

\[
H^{(1)}_\alpha(x) = J_\alpha(x) + iY_\alpha(x) \quad \quad H^{(2)}_\alpha(x) = J_\alpha(x) - iY_\alpha(x) \quad \quad h^{(1)}_n(x) = j_n(x) + iy_n(x) \quad \quad h^{(2)}_n(x) = j_n(x) - iy_n(x).
\]

These linear combinations are also known as Bessel functions of the third kind, and it’s just an association of the previous two kinds together. Here the spherical analogues of the Hankel functions are based on the Spherical Bessel functions as follows:

\[
j_n(x) = \sqrt{\frac{\pi}{2x}} J_{n+1/2}(x), \quad y_n(x) = \sqrt{\frac{\pi}{2x}} Y_{n+1/2}(x) = (-1)^{n+1} \sqrt{\frac{\pi}{2x}} J_{-n-1/2}(x).
\]

Example: Calculate \( HK1 \) and \( HK2 \) of zero order for \( z=(1+i) \)

\[
\[ Z \], [0], \text{ ZENTER}^\wedge, [1], \text{ ENTER}^\wedge, \text{ XEQ "ZSHK1"} \Rightarrow 0.055-30.254
\]

\[
\[ Z \], [0], \text{ ZENTER}^\wedge, [1], \text{ ENTER}^\wedge, \text{ XEQ "ZSHK2"} \Rightarrow 1.878-30.409
\]

Note that for the zero-th order \( SHK1 \) we can also use the \text{EIZ/IZ} function, which uses the direct exponentials formula and therefore comes to the same result in a much shorter time (shown below with 9 decimal digits):

\[
\text{Re} = 0.055396883; \quad \text{Im} = -0.254162993
\]
These functions are also valid for the non-integer order cases, for example: \( n= (1 + i) \) and \( z= (1+i) \):

\[
1, \text{ ENTER}^\wedge, \text{ ZENTER}^\wedge, \text{ XEQ } "\text{ZSHK1}" \Rightarrow -0,434\text{-}30,874
\]

Which has a 9-digit accuracy when compared to the Wolfram Alpha result – astonishing if you consider the long and winding process needed to get to their result – all done behind the scenes.

The FOCAL programs below list the simple code snippets to program the regular (\( \text{ZH}K1 \) and \( \text{ZH}K2 \)) and spherical pairs \( \text{ZSH}K1 \) and \( \text{ZSH}K2 \). Note that \( J \) is obtained during the \( Y \) calculation, thus there’s no need to repeat the execution for it – we retrieve its value from complex register ZR02. Note how the complex stack performs a vital role in these programs – storing the intermediate results unaffected by the complex calculations that take place.

```
01 LBL "ZSHK1"
02 CF 03
03 GTO 03
04 LBL "ZSHK2"
05 SF 03
06 LBL 03
07 Z<>W
08 ,5
09 +
10 Z<>W
11 ZYBS
12 FS? 03
13 ZNEG
14 Z*I
15 ZRCL 02
16 Z+
17 ZRCL 01
18 4
19 ST\(^\ast\) Z
20 *
21 ZINV
22 ZPI*
23 ZSQRT
24 Z*
25 ZAVIEW
26 END
```
The plots below show the Spherical Hankel-1 function for orders 1 and 2, for a short range of the real argument $x$. Obviously the results are complex as well, thus the real and imaginary parts are plotted separately.

$$\begin{align*}
\hat{h}_0^{(1)}(z) &= -i e^{i z} \frac{1}{z} \\
\hat{h}_1^{(1)}(z) &= -e^{i z} \frac{z + i}{z^2} \\
\hat{h}_2^{(1)}(z) &= i e^{i z} \frac{z^2 + 3 i z - 3}{z^3} \\
\hat{h}_3^{(1)}(z) &= e^{i z} \frac{z^3 + 6 i z^2 - 15 z - 15 i}{z^4}.
\end{align*}$$

**Complex Keyboard shortcuts.** - The Bessel and Hankel functions 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^Z and 1/Z in the [NEXT] group – so you can quickly compare them with the main functions for accuracy and speed.
12.8. Weber and Anger Functions. \{\text{ZANGJ}, \text{ZWEBE}\}

In mathematics, the Anger function, introduced by C. T. Anger (1855), is a function defined as

\[ J_\nu(z) = \frac{1}{\pi} \int_0^\pi \cos(\nu \theta - z \sin \theta) \, d\theta \]

The Weber function introduced by H. F. Weber (1879), is a closely related function defined by:

\[ E_\nu(z) = \frac{1}{\pi} \int_0^\pi \sin(\nu \theta - z \sin \theta) \, d\theta \]

The Anger and Weber functions are related by:

\[ \sin(\pi \nu) J_\nu(z) = \cos(\pi \nu) E_\nu(z) - E_{-\nu}(z) \]
\[ - \sin(\pi \nu) E_\nu(z) = \cos(\pi \nu) J_\nu(z) - J_{-\nu}(z) \]

so in particular if \( \nu \) is not an integer they can be expressed as linear combinations of each other. If \( \nu \) is an integer then Anger functions \( J_\nu \) are the same as Bessel functions \( J_\nu \), and Weber functions can be expressed as finite linear combinations of Struve functions (\( H_n \) and \( L_n \)).

The expressions used in the 41Z module are based on the Hypergeometric function, therefore use the ascending series method - as follows:

\[ J_n(z) = (z/2) \sin(90^\circ n) \text{ _1F_2}( 1 \quad (3-n)/2 , (3+n)/2 \quad -z^2/4 ) / \Gamma((3-n)/2) / \Gamma((3+n)/2) \]
\[ + \cos(90^\circ n) \text{ _1F_2}( 1 \quad (2-n)/2 , (2+n)/2 \quad -z^2/4 ) / \Gamma((2-n)/2) / \Gamma((2+n)/2) \]

and:

\[ E_n(z) = (z/2) \cos(90^\circ n) \text{ _1F_2}( 1 \quad (3-n)/2 , (3+n)/2 \quad -z^2/4 ) / \Gamma((3-n)/2) / \Gamma((3+n)/2) \]
\[ + \sin(90^\circ n) \text{ _1F_2}( 1 \quad (2-n)/2 , (2+n)/2 \quad -z^2/4 ) / \Gamma((2-n)/2) / \Gamma((2+n)/2) \]

Note that even if the argument \( z \) can be a complex number, this implementation requires the order \( \nu \) to be a real value so the dual-complex case is not supported. The input parameters are expected in the real registers \( Z, Y, X \), with the order in the \( X \)-register as per the standard 41Z conventions.

Examples. Calculate the Weber and Anger functions for \( \nu = \pi \), and \( z=1+i \)

```
1, ENTER, 1, PI, ZF$ "ZANGJ" -> "RUNNING..." => -0.064+J0.041
1, ENTER^, 1, PI, ZF$ "ZWEBE" -> "RUNNING..." => 0.211+J0.077
```

Which can be checked in WolframAlpha using the syntax shown in the link:

http://www.wolframalpha.com/input/?i=AngerJ+%28pi,+1%2Bi%29
http://www.wolframalpha.com/input/?i=WeberE+%28pi,+1%2Bi%29
Program Listing for Weber and Anger routines.

01 LBL "ZWEBE" 32 SIN 63 *
02 SF 00 33 FC? 00 64 FS? 00
03 GTO 00 34 COS 65 ZNEG
04 LBL "ZANGJ" 35 ST* Z 66 Z<>W
05 CF 00 36 * 67 ZRDN
06 LBL 00 37 ZENTER^ 68 Z+
07 RAD 38 RCL 00 69 ZAVIEW
08 STO 00 39 3 70 RTN
09 RDN 40 + 71 LBL 00
10 2 41 2 72 ZRCL 02
11 ST/ Z 42 / 73 Z^2
12 / 43 STO 03 74 ZNEG
13 ZSTO 02 44 3 75 RCL 01
14 E 45 RCL 00 76 2
15 STO 01 46 - 77 RDN
16 RCL 00 47 2 78 RDN
17 2 48 / 79 ZHGF
18 / 49 STO 02 80 ZENTER^
19 - 50 XEQ 00 81 0
20 STO 02 51 ZRCL 02 82 RCL 02
21 LASTX 52 Z* 83 ZGAMMA
22 E 53 RCL 00 84 Z/
23 + 54 PI 85 ZENTER^
24 STO 03 55 * 86 0
25 XEQ 00 56 2 87 RCL 03
26 RCL 00 57 / 88 ZGAMMA
27 PI 58 FS? 00 89 Z/
28 * 59 COS 90 END
29 2 60 FC? 00
30 / 61 SIN
31 FS? 00 62 ST* Z

Registers used: R00-R05
Flags used: F0
12.9 Dilogarithm and Polylogarithm. \{ ZLIN, ZLI2 \}

The Polylogarithm (also known as Jonquière's function) is a special function \( L_s(z) \) that is defined by the infinite sum, or power series

\[
L_s(z) = \sum_{k=1}^{\infty} \frac{z^k}{k^s} = z + \frac{z^2}{2^s} + \frac{z^3}{3^s} + \cdots.
\]

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](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](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 \( L_i(2; 0.3+0.4i) \)

\[
0, \text{ ENTER}^\wedge, 2, \text{ ZENTER}^\wedge \quad \rightarrow \quad 2+0j
\]

(the Z-keypad version: \[ Z \], 2 does the same easier)

\[
.4, \text{ ENTER}^\wedge, .3, \text{ XEQ "ZLIN"} \quad \rightarrow \quad 0,266+0,461
\]

or with FIX 9 settings:

Re = 0.266596867

Im = 0.461362892

**Example 2.** Calculate \( L_i(1+i; 0.3+0.4i) \)

\[
1, \text{ ENTER}^\wedge, \text{ ZENTER}^\wedge \quad \rightarrow \quad 1(1+j)
\]

\[
.4, \text{ ENTER}^\wedge, .3, \text{ XEQ "ZLIN"} \quad \rightarrow \quad 0,326+0,565
\]

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 and ZLI2 are FOCAL programs, and therefore the argument \( z \) won’t be saved in the LastZ complex register.
12.10. Lerch Transcendent Function. \{ ZLRCH \}

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:

\[\text{Li}(s, z) \approx \text{Lerch} (z, s, 0)\]

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

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

4, ENTER^, 3, ZENTER^ \( \rightarrow 3+J4\)
2, ENTER^, 1, ZENTER^ \( \rightarrow 1+J2\)
.4, ENTER^, .3, XEQ “ZLRCH” \( \rightarrow 7,658-J1,515,\)

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 LBL “ZLRCH”
02 “|Z|>1”
03 ZOUT?
04 PROMPT
05 ZSTO 01
06 CLZ
07 SIGN
08 ZSTO 00\( x^{0} = 1\)
09 ZRDN
10 ZSTO 02
11 Z<>W
12 ZSTO 03\( s\)
13 ZNEG
14 W^Z\( 1/\alpha^{s}\)
15 LBL 01
16 ZRCL 01
17 ZRCL 00\( x^{(k-1)}\)
18 Z^\( k\)
19 ZSTO 00
20 ZRCL 02\( \alpha + k - 1\)
21 E
22 ST+ 05
23 +\( \alpha + k\)
24 ZRCL 03\( s\)
25 W^Z\( (\alpha + k)^s\)
26 Z/\( x^{k}/(\alpha + k)^{s}\)
27 Z^+
28 ZWR?
29 GTO 01
30 ZAVIEW
31 END
12.11. Exponential Integrals.

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

<table>
<thead>
<tr>
<th>Index</th>
<th>Function</th>
<th>Description</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ZHGF</td>
<td>Complex Hypergeometric function</td>
<td>Jean-Marc Baillard</td>
</tr>
<tr>
<td>2</td>
<td>ZEI</td>
<td>Complex Exponential Integral</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ZCI</td>
<td>Complex Cosine Integral</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>ZHCI</td>
<td>Complex Hyperbolic Cosine Integral</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>ZSI</td>
<td>Complex Sine integral</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>ZHSI</td>
<td>Complex Hyperbolic Sine Integral</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>ZERF</td>
<td>Complex Error function</td>
<td></td>
</tr>
</tbody>
</table>

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] \rightarrow ZHGF E5:H:C:I:R:F
```

**Examples.**

Calculate erf(1+i) and Ei(1+i)

```
1, ENTER^, [Z], [A], [H], ["R"] \rightarrow 1,316+J0,190
1, ENTER^, [Z], [A], [H], ["E"] \rightarrow 1,765+J2,388
```

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

```
1, ENTER^, [Z], [A], [H], ["S"] \rightarrow 1,104+J0,882
1, ENTER^, [Z], [A], [H], ["H"] \rightarrow 0,882+J1,104
1, ENTER^, [Z], [A], [H], ["C"] \rightarrow 0,882+J0,287
1, ENTER^, [Z], [A], [H], ["I"] \rightarrow 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 "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 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 "ZHSI" 
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
```

All this many functions sure enough will benefit from having “theme” launchers grouping them, for easier access and logical segregation. The usability is enhanced and doesn't require overlays for the most frequente options within the groups.

The first one combines the Exponential Integrals and the Complex Means. Use the key sequence below to access it, and then the [SHIFT] key to toggle between its two parts:

![Image of the first launcher with keys: Z H G F E S H C I R F]

![Image of the second launcher with keys: Z M N A G H M N K I O]

See below the function correspondence for each launcher:

<table>
<thead>
<tr>
<th>Exponential Integrals</th>
<th>Complex Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>[E]ZEI</td>
<td>[A]ZAMN</td>
</tr>
<tr>
<td>[S]ZSI</td>
<td>[G]ZGMN</td>
</tr>
<tr>
<td>[H]ZHSI</td>
<td>[H]ZHMN</td>
</tr>
<tr>
<td>[C]ZCI</td>
<td>[M]ZAGM</td>
</tr>
<tr>
<td>[I]ZHCI</td>
<td>[N]ZGHM</td>
</tr>
<tr>
<td>[R]ZERF</td>
<td>[K]ZELK</td>
</tr>
<tr>
<td>[F]ZHG</td>
<td>[I]ZINPT</td>
</tr>
<tr>
<td></td>
<td>[O]ZOUPT</td>
</tr>
</tbody>
</table>

Finally the remaining Launchers deal with Elliptical Functions and Complex Methods. You access these groups using the keyword combinations shown below:

![Image of the third launcher with keys: Z E L P I Z M E K P]

![Image of the fourth launcher with keys: Z D F T I Z M E K P]

<table>
<thead>
<tr>
<th>Elliptic Functions</th>
<th>DFT/Other Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]ZELIP1</td>
<td>[I]ZIDFT</td>
</tr>
<tr>
<td>[L]ZELK</td>
<td>[S]“ZSAM”</td>
</tr>
<tr>
<td>[E]ZELPE</td>
<td>[C]ZCTLN</td>
</tr>
<tr>
<td>[K]ZELPK</td>
<td>[P]ZPSIN</td>
</tr>
<tr>
<td>[P]ZELPKE</td>
<td>[M]ZIGAM</td>
</tr>
</tbody>
</table>
Appendix. Saving & Restoring the Z-Stack in X-Memory. \{ SAVEZS, GETZS \}

Two functions are included in the Deluxe 41Z Module to save and restore the complete complex stack buffer in extended memory. This includes all complex levels, the real stack and the current POLAR/REC settings. The functions are SAVEZS and GETZS.

In program execution, the file name is expected by these functions to be in ALPHA. For convenience, in RUN mode the functions will prompt for the file name automatically - remember that ALPHA is constantly being updated with the complex number values, so without this automated prompting feature you would need to re-write the file name in-between operations.

You can use them to preserve their contents in a permanent X-Mem file. Only one active complex buffer is allowed in the calculator, but you can choose from several X-Mem files holding different complex stacks, to upload their contents on demand. Therefore prior to executing GETZS you need to ensure that there’s no buffer#8 in memory – you can use function CLB in the AMC_OS/X module for that. Failure to do so will generate the error message “DUP BUF”

The X-mem file has a custom type “Z”, with code=8. The file size is always 12 registers. If you’re using the AMC_OS/X Module the CAT“4 enumeration includes support for this file type, which will be properly shown as a “Z” type:

Where here the complex stack file name is “ZSTACK”.

Note 3.- The Hypergeometric Function is also 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
Appendix.- Delta-Wye Transformation.

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 :-)

<table>
<thead>
<tr>
<th>Delta &lt;-&gt; Why conversions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LBL &quot;D-Y&quot;</strong></td>
</tr>
<tr>
<td>SF 00</td>
</tr>
<tr>
<td>GTO 00</td>
</tr>
<tr>
<td><strong>LBL &quot;Y-D&quot;</strong></td>
</tr>
<tr>
<td>CF 00</td>
</tr>
<tr>
<td>LBL 00</td>
</tr>
<tr>
<td>“Za”</td>
</tr>
<tr>
<td>FS? 00</td>
</tr>
<tr>
<td>“J-b”</td>
</tr>
<tr>
<td>“J-a”</td>
</tr>
<tr>
<td>PROMPT</td>
</tr>
<tr>
<td>ZSTO 00</td>
</tr>
<tr>
<td>“Zb”</td>
</tr>
<tr>
<td>FS? 00</td>
</tr>
<tr>
<td>“J-c”</td>
</tr>
<tr>
<td>“J-c”</td>
</tr>
<tr>
<td>PROMPT</td>
</tr>
<tr>
<td>ZSTO 01</td>
</tr>
<tr>
<td>“Zc”</td>
</tr>
<tr>
<td>FS? 00</td>
</tr>
<tr>
<td>“J-a”</td>
</tr>
<tr>
<td>PROMPT</td>
</tr>
<tr>
<td>ZSTO 02</td>
</tr>
<tr>
<td>XEQ &quot;DYD&quot;</td>
</tr>
<tr>
<td>ZSTO 02</td>
</tr>
<tr>
<td>ZRDN</td>
</tr>
<tr>
<td>ZSTO 01</td>
</tr>
<tr>
<td>ZRDN</td>
</tr>
<tr>
<td>ZSTO 00</td>
</tr>
<tr>
<td>ZRDN</td>
</tr>
<tr>
<td>ZVIEW 00</td>
</tr>
<tr>
<td>ZVIEW 01</td>
</tr>
<tr>
<td>ZVIEW 02</td>
</tr>
<tr>
<td>RTN</td>
</tr>
<tr>
<td>Z+</td>
</tr>
<tr>
<td>LASTZ</td>
</tr>
<tr>
<td>Z+</td>
</tr>
<tr>
<td>ZRCL 01</td>
</tr>
<tr>
<td>Z+</td>
</tr>
<tr>
<td>ZRCL 00</td>
</tr>
<tr>
<td>ZRCL 01</td>
</tr>
<tr>
<td>Z/</td>
</tr>
<tr>
<td>ZRCL 02</td>
</tr>
<tr>
<td>Z+</td>
</tr>
<tr>
<td>LASTZ</td>
</tr>
<tr>
<td>Z+</td>
</tr>
<tr>
<td>ZRCL 00</td>
</tr>
<tr>
<td>Z+</td>
</tr>
<tr>
<td>RTN</td>
</tr>
</tbody>
</table>

\[
Z_1 = \frac{Z_a Z_c}{Z_a + Z_b + Z_c} \\
Z_2 = \frac{Z_c Z_a}{Z_a + Z_b + Z_c} \\
Z_3 = \frac{Z_a Z_b}{Z_a + Z_b + Z_c}
\]
Appendix 1.- Complex Buffer functions.

This appendix lists the buffer handling functions included in the 41Z DIAGNOSTICS module, 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!).

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

(*) 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.
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 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 \( \text{Re}(z) \) first - as an alternative method for the previous example. Note that because \(^\text{IMG} \) does stack lift, it's not necessary to use \(^\text{ZENTER}^\text{^\text{up}}\):

\[
1, \ ^\text{IMG}, 1, \text{R/S}, \ ^\text{ZLN}, \ 2, \ ^\text{IMG}, 1, \text{CHS}, \text{R/S}, \ ^\text{Z+} \rightarrow 2,347-0,215
\]

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{L1} = \text{XY}? )</td>
<td>Tests for the first buffer level and XY registers</td>
</tr>
<tr>
<td>( \text{XY} &gt; \text{L1} )</td>
<td>Copies X,Y into level L1</td>
</tr>
<tr>
<td>( \text{L2} = \text{ZT}? )</td>
<td>Tests for second buffer level and Z,T registers</td>
</tr>
<tr>
<td>( \text{L2} &gt; \text{ZT} )</td>
<td>Copies L2 into registers Z,T</td>
</tr>
<tr>
<td>( \text{ST} &gt; \text{ZB} )</td>
<td>Copies real stack ( \text{XYZT} ) to buffer levels L1 &amp; L2</td>
</tr>
<tr>
<td>( \text{ZB} &gt; \text{ST} )</td>
<td>Copies L1 &amp; L2 to the real stack ( \text{XYZT} )</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{ZB} &gt; \text{RG} )</td>
<td>Copies complex buffer to memory registers</td>
</tr>
<tr>
<td>( \text{RG} &gt; \text{ZB} )</td>
<td>Copies memory registers to complex buffer</td>
</tr>
</tbody>
</table>

Note that \( \text{RG} > \text{ZB} \) won’t check for valid header data, thus it expects the contents to be correct – like with a previously execution of \( \text{ZB} > \text{RG} \). 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:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{L1} \leftrightarrow \text{L} )</td>
<td>Swaps buffer level L1 and level given by prompt</td>
</tr>
<tr>
<td>( \text{L1} \leftrightarrow \text{LX} )</td>
<td>Swaps buffer level L1 and level input in X</td>
</tr>
<tr>
<td>( \text{XY} &gt; \text{L0} )</td>
<td>Copies registers X,Y into buffer level L0 (used to save arguments into LastZ)</td>
</tr>
<tr>
<td>( \text{XY} &gt; \text{L} )</td>
<td>Copies registers X,Y into buffer level given by prompt</td>
</tr>
<tr>
<td>( \text{ZBDROP} )</td>
<td>Drops contents of complex buffer one level (used during ( \text{ZRDN} ))</td>
</tr>
<tr>
<td>( \text{ZBLIFT} )</td>
<td>Lifts contents of complex buffer one level (used by ( \text{ZRUP}, \ ^\text{ZENTER}^\text{^\text{up}} ) and others)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{LVVIEW} )</td>
<td>Prompts for level number (0 – 4)</td>
</tr>
<tr>
<td>( \text{LVVIEWX} )</td>
<td>Expects level number in X</td>
</tr>
<tr>
<td>( \text{ZBSHOW} )</td>
<td>Lists the contents of all buffer levels</td>
</tr>
<tr>
<td>( \text{ZBHEAD} )</td>
<td>Shows in Alpha the decoded buffer header</td>
</tr>
</tbody>
</table>

Note that with these functions all complex level contents will be shown transposed, that is: \( \text{Im}(z) + j \text{Re}(z) \).
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.
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 ΣZL complex keyboard function launcher.

<table>
<thead>
<tr>
<th>Level</th>
<th>Function</th>
<th>Level</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z 1/X</td>
<td>ZINV</td>
<td>Z Y¹X</td>
<td>-HP 41Z</td>
</tr>
<tr>
<td>Z SQRT</td>
<td>ZSQRT</td>
<td>Z W¹Z</td>
<td></td>
</tr>
<tr>
<td>Z LOG</td>
<td>ZLOG</td>
<td>Z Z²</td>
<td></td>
</tr>
<tr>
<td>Z LN</td>
<td>ZLN</td>
<td>Z Z¹0</td>
<td>ZALOG</td>
</tr>
<tr>
<td>Z X&lt;&gt;Y</td>
<td>Z&lt;&gt;W</td>
<td>Z e¹X</td>
<td>ZEXP</td>
</tr>
<tr>
<td>Z RDN</td>
<td>ZRDN</td>
<td>Z X&lt;&gt;Y</td>
<td>ZTRP</td>
</tr>
<tr>
<td>Z SIN</td>
<td>ZSIN</td>
<td>Z ZRDN</td>
<td>ZRUP</td>
</tr>
<tr>
<td>Z COS</td>
<td>ZCOS</td>
<td>Z ASIN</td>
<td>ZASIN</td>
</tr>
<tr>
<td>Z TAN</td>
<td>ZTAN</td>
<td>Z ACOS</td>
<td>ZACOS</td>
</tr>
<tr>
<td>Z XEQ</td>
<td>^IMG_</td>
<td>Z ZTAN</td>
<td>ZTAN</td>
</tr>
<tr>
<td>Z STO</td>
<td>ZSTO_</td>
<td>Z ASN</td>
<td>ZK?YN</td>
</tr>
<tr>
<td>Z RCL</td>
<td>ZRCL_</td>
<td>Z LBL</td>
<td>ZSIGN</td>
</tr>
<tr>
<td>Z SST</td>
<td>Z&lt;&gt;_</td>
<td>Z GTO</td>
<td>Z¹?</td>
</tr>
<tr>
<td>Z ENT^</td>
<td>ZENTER^</td>
<td>Z CAT</td>
<td>^IMG_</td>
</tr>
<tr>
<td>Z CHS</td>
<td>ZNEG</td>
<td>Z ISG</td>
<td>ZCONJ</td>
</tr>
<tr>
<td>Z EEX</td>
<td>Z²X</td>
<td>Z RTN</td>
<td>ZPOL</td>
</tr>
<tr>
<td>Z +</td>
<td>ZX=Ý?</td>
<td>Z Z=W?</td>
<td></td>
</tr>
<tr>
<td>Z *</td>
<td>ZSF</td>
<td>Z ZNORM</td>
<td></td>
</tr>
<tr>
<td>Z /</td>
<td>ZCF</td>
<td>Z ZMOD</td>
<td></td>
</tr>
<tr>
<td>0-9</td>
<td>Z0-Z9</td>
<td>Z FS?</td>
<td>ZARG</td>
</tr>
<tr>
<td>Z R/S</td>
<td>ZAVIEW</td>
<td>Z X=Ý?</td>
<td>Z?W?</td>
</tr>
<tr>
<td>Z 0-9</td>
<td>ZJ0-ZJ9</td>
<td>Z BEEP</td>
<td>ZTONE</td>
</tr>
<tr>
<td>Z Z 1/X</td>
<td>W¹/ÍZ</td>
<td>Z P-R</td>
<td>ZREC</td>
</tr>
<tr>
<td>Z Z SQRT</td>
<td>ZPSI</td>
<td>Z R-P</td>
<td>ZPOL</td>
</tr>
<tr>
<td>Z Z LN</td>
<td>e¹Z</td>
<td>Z RX</td>
<td>ZRND</td>
</tr>
<tr>
<td>Z Z X&lt;&gt;Y</td>
<td>Z&lt;&gt;V</td>
<td>Z SCI</td>
<td>ZINT</td>
</tr>
<tr>
<td>Z Z RDN</td>
<td>ZQRT</td>
<td>Z ENG</td>
<td>ZFRC</td>
</tr>
<tr>
<td>Z Z XEQ</td>
<td>ZIMAG^</td>
<td>Z X=Ý0?</td>
<td>Z?0?</td>
</tr>
<tr>
<td>Z Z STO</td>
<td>ZREAL^</td>
<td>Z PI</td>
<td>ZGAMMA</td>
</tr>
<tr>
<td>Z Z RCL</td>
<td>ZI</td>
<td>Z LASTX</td>
<td>ZLASTZ</td>
</tr>
<tr>
<td>Z Z SST</td>
<td>CLSTZ</td>
<td>Z VIEW</td>
<td>ZVIEW_</td>
</tr>
<tr>
<td>Z Z ENT^</td>
<td>ZRPL</td>
<td>Z SIN</td>
<td>ZSINH</td>
</tr>
<tr>
<td>Z Z EEX</td>
<td>Z¹/ÍX</td>
<td>Z COS</td>
<td>ZCOSH</td>
</tr>
<tr>
<td>Z Z -</td>
<td>Z#W?</td>
<td>Z TAN</td>
<td>ZTANH</td>
</tr>
<tr>
<td>Z Z 7</td>
<td>ZWDDET</td>
<td>Z SIN</td>
<td>ZASINH</td>
</tr>
<tr>
<td>Z Z 8</td>
<td>ZWDIST</td>
<td>Z COS</td>
<td>ZACOSH</td>
</tr>
<tr>
<td>Z Z 9</td>
<td>ZWANG</td>
<td>Z TAN</td>
<td>ZTANH</td>
</tr>
<tr>
<td>Z Z +</td>
<td>ZREAL?</td>
<td>Z SQRT</td>
<td>ZNXTNRTR</td>
</tr>
<tr>
<td>Z Z 4</td>
<td>ZIN?</td>
<td>Z ZLN</td>
<td>ZNXTLN</td>
</tr>
<tr>
<td>Z Z 5</td>
<td>ZWCROSS</td>
<td>Z SIN</td>
<td>ZNXTASN</td>
</tr>
<tr>
<td>Z Z *</td>
<td>ZIMAG?</td>
<td>Z COS</td>
<td>ZNXTACSN</td>
</tr>
<tr>
<td>Z Z 1</td>
<td>ZUNT?</td>
<td>Z Z TAN</td>
<td>ZNXTATN</td>
</tr>
<tr>
<td>Z Z 2</td>
<td>ZWLINE</td>
<td>Z LOG</td>
<td>ZKBS</td>
</tr>
<tr>
<td>Z Z /</td>
<td>Z#0?</td>
<td>Z Z LN</td>
<td>ZYBS</td>
</tr>
<tr>
<td>Z Z 0</td>
<td>ZOUT?</td>
<td>Z Z COS</td>
<td>ZIBS</td>
</tr>
<tr>
<td>Z Z .</td>
<td>ZWDOT</td>
<td>Z Z TAN</td>
<td>ZJBS</td>
</tr>
<tr>
<td>Z Z Z</td>
<td>Z&lt;&gt;U</td>
<td>Z Z SIN</td>
<td>ZWL</td>
</tr>
<tr>
<td>Z Z Z</td>
<td>Z&lt;&gt;U</td>
<td>Z Z SQRT</td>
<td>EB/IZ</td>
</tr>
</tbody>
</table>
Appendix 3. - Formula Compendium.

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

\[ j = \sqrt{-1} = e^{\pi i} = 1 \angle 90^\circ \quad j^2 = e^{\pi i} = 1 \angle 180^\circ = -1 \quad -j = j^4 \]

\[ Z = \text{Re}(Z) + j\text{Im}(Z) = x + jy = re^{\theta i} = r \cos \theta + j r \sin \theta \quad r = |Z| = \sqrt{x^2 + y^2} \quad \theta = \tan^{-1}(y/x) \]

\[ Z^* = x - jy = re^{-\theta i} = r \angle -\theta \quad Z + Z^* = 2\text{Re}(Z) \quad Z - Z^* = j2\text{Im}(Z) \]

\[ (Z, Z^*) = (z_1, z_2)^* = (z_2, z_1)^* \quad (Z, Z^*)^* = (Z^*, Z)^* \quad (Z_1 + Z_2)^* = Z_1^* + Z_2^* \quad (Z_1 - Z_2)^* = Z_1^* - Z_2^* \]

\[ |Z| = |Z_1| \quad |Z_1 + Z_2| = |Z_1| + |Z_2| \quad |Z_1 - Z_2| = |Z_1| - |Z_2| \]

\[ |Z^*|^2 = (Z + Z^*)(Z + Z^*) = Z^*Z + Z^*Z + Z_1Z_2 + Z_1Z_2^* = |Z|^2 + 2\text{Re}(ZZ^*) + |Z|^2 \]

\[ Z + Z_2 = (x_1 + x_2) + j(y_1 + y_2) \quad Z_1 - Z_2 = (x_1 - x_2) + j(y_1 - y_2) \quad |Z_1 + Z_2| \leq |Z_1| + |Z_2| \]

\[ Z_1Z_2 = (x_1x_2 - y_1y_2) + j(x_1y_2 + y_1x_2) = r_1r_2 \angle (\theta_1 + \theta_2) = r_1r_2e^{i(\theta_1 - \theta_2)} \quad \text{Re}(1/Z^*) = \text{Re}(1/Z) \]

\[ Z_i/Z_2 = (x_1x_2 + y_1y_2) + j(y_1x_2 - x_1y_2)/(x_2^2 + y_2^2) = r_1/r_2 \angle (\theta_1 - \theta_2) \quad Z^* = (x - jy)/(x^2 + y^2) = e^{-\theta i} \quad r_1 = |Z_1| \]

\[ Z_1Z_2 = (x_1x_2 + y_1y_2) + j(y_1x_2 - x_1y_2) = r_1r_2 \angle (\theta_1 - \theta_2) \]

\[ (Z_1, Z_2^*) = 2D \text{vectors} \]

\[ Z^2 = x^2 - y^2 + 2jxy \quad r^2 = e^{2\theta} \quad (\text{principal}) \]

\[ \pi = 3.141592653589793238462643 \quad e = 2.718281828459045235360287 \]

\[ e^z = e^{x+y} = e^x e^y = e^{x\cos \theta + y \sin \theta} \quad e^{x+iy} = e^{x+i\tan \theta} \quad e^{x+iy} = e^{x+i\tan \theta} \quad (\pi, \theta_2, \leq \pi) \]

\[ e^{x+y} = e^x e^y = \cos(Z) + j \sin(Z) \quad e^{-jZ} = (\sin Z + j \cos Z)/Z = k_0^2(Z) \]

\[ \ln Z = \ln r + j \theta = \ln(\sqrt{x^2 + y^2}) + j \tan \theta \quad \ln Z = \ln r + j \theta = \ln(\sqrt{x^2 + y^2}) + j \tan \theta \]

\[ Z_1 \ln Z_2 = \ln Z_2 = \ln Z_1 + \ln Z_1 = \ln(Z_1Z_2) \quad \ln 0 = \infty \quad \ln Z = \ln e^Z \]

\[ e^z = e^{z/0} = \text{Log}_Z = \ln Z/a \quad \ln j = 0 + j\pi/2 \quad \ln(-1) = 0 + j\pi \]

\[ \frac{\partial}{\partial Z} Z^a = aZ^{a-1} \quad \frac{\partial}{\partial Z} e^{az} = ae^{az} \quad \frac{\partial}{\partial Z} e^{az} = e^{az} \ln a \quad \frac{\partial}{\partial Z} \ln Z = \frac{1}{Z} \quad \int e^{az} dZ = \frac{e^{az}}{a} \quad \frac{dz}{Z} = \ln Z \]

\[ e^z = 1 + \frac{z}{1!} + \frac{z^2}{2!} + \frac{z^3}{3!} + ... \quad \ln Z = \frac{z}{Z+1} + \frac{z(1-Z)^3}{3Z+1} + ... \quad (\text{Re}(Z) \geq 0) \]

\[ \sin Z = (-j/2)(e^{jz} - e^{-jz}) = (-j/2)(e^{jz} - 1/e^{jz}) \quad \sin^{-1} Z = -j \ln(Z + \sqrt{1 - Z^2}) \]

\[ \cos Z = (1/2)(e^{jz} + e^{-jz}) = (1/2)(e^{jz} + 1/e^{jz}) \quad \cos^{-1} Z = -j \ln(Z + \sqrt{Z^2 - 1}) \]

\[ \tan Z = -j \frac{e^{jz} - e^{-jz}}{e^{jz} + e^{-jz}} = -j \frac{e^{jz} - 1}{e^{jz} + 1} \quad \tan^{-1} Z = -j \ln \left(1 + jZ \right) \quad \tan^{-1} Z = -j \ln \left(1 + jZ \right) \]

\[ \frac{\partial}{\partial Z} \cos Z = -\sin Z \quad \frac{\partial}{\partial Z} \sin Z = \cos Z \quad \int \cos Z dZ = \sin Z \quad \int \sin Z dZ = -\cos Z \]

\[ \sin Z = Z - \frac{Z^3}{3!} + \frac{Z^5}{5!} + ... \quad \sin^{-1} Z = \ln \left(Z + \sqrt{Z^2 + 1} \right) \]

\[ \cos Z = (1/2)(e^{jz} - e^{-jz}) = (1/2)(e^{jz} + 1/e^{jz}) \quad \cos^{-1} Z = \ln \left(Z + \sqrt{Z^2 + 1} \right) \]

\[ \sinh Z = (1/2)(e^{jz} - e^{-jz}) = -j \sinh(jZ) \quad \sinh^{-1} Z = \ln \left(Z - \sqrt{Z^2 - 1} \right) - \ln \left(1 - Z \right) \]

\[ \cosh Z = (1/2)(e^{jz} + e^{-jz}) = \cos(jZ) \quad \cosh^{-1} Z = \ln \left(Z + \sqrt{Z^2 - 1} \right) \]

\[ \tanh Z = \frac{e^{jz} - e^{-jz}}{e^{jz} + e^{-jz}} = \frac{e^{jz} - 1}{e^{jz} + 1} \quad \tanh^{-1} Z = \frac{1}{2} \ln \left(1 + jZ \right) \quad \tanh^{-1} Z = \frac{1}{2} \ln \left(1 + jZ \right) \]

\[ \sin Z = \sin x \cosh y - j \sin x \sinh y \quad \sin Z = \sin x \cosh y + j \cos x \sinh y \]
### Appendix 4. - Quick Reference Guide.

The tables in the following six pages list all 41Z functions in alphabetical order. New functions in the Deluxe edition have pink background. Sub-functions are in brown font color.

<table>
<thead>
<tr>
<th>#</th>
<th>Function</th>
<th>Description</th>
<th>Formula</th>
<th>Input</th>
<th>Output</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-HP 41Z</td>
<td>Initializes Complex Stack</td>
<td>Z=XY; W=ZT</td>
<td>none</td>
<td>Initializes Z buffer &amp; ZAVIEW</td>
<td>runs on CALC ON</td>
</tr>
<tr>
<td>2</td>
<td>W^1/Z</td>
<td>Complex Y^1/X</td>
<td>w^1/z = exp(Ln w / Z)</td>
<td>w in W; z in Z (XY)</td>
<td>w^1/z in Z (XY)</td>
<td>Drops Buffer</td>
</tr>
<tr>
<td>3</td>
<td>W^2</td>
<td>Complex Y^X</td>
<td>w^2z = exp(z *Ln w)</td>
<td>w in W; z in Z (XY)</td>
<td>w^2z in Z (XY)</td>
<td>Drops Buffer</td>
</tr>
<tr>
<td>4</td>
<td>ZF#</td>
<td>Launcher by index</td>
<td>n/a</td>
<td>Sub-function index</td>
<td>Executes Sub-function</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>ZFS</td>
<td>Launcher by Name</td>
<td>n/a</td>
<td>Sub-function Name</td>
<td>Executes Sub-function</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Z+</td>
<td>Complex addition</td>
<td>(x1+x2) + i (y1+y2)</td>
<td>w in W; z in Z (XY)</td>
<td>w+z in Z (XY)</td>
<td>Drops Buffer, LastZ</td>
</tr>
<tr>
<td>7</td>
<td>Z-</td>
<td>Complex subtraction</td>
<td>w-z = w + (-z)</td>
<td>w in W; z in Z (XY)</td>
<td>w-z in Z (XY)</td>
<td>Drops Buffer, LastZ</td>
</tr>
<tr>
<td>8</td>
<td>Z*</td>
<td>Complex multiplication</td>
<td>(x1<em>y2 - y1</em>x2) + i (x1<em>y2 + y1</em>x2)</td>
<td>w in W; z in Z (XY)</td>
<td>w*z in Z (XY)</td>
<td>Drops Buffer, LastZ</td>
</tr>
<tr>
<td>9</td>
<td>Z/</td>
<td>Complex division</td>
<td>w/z = w * (1/z)</td>
<td>w in W; z in Z (XY)</td>
<td>w/z in Z (XY)</td>
<td>Drops Buffer, LastZ</td>
</tr>
<tr>
<td>10</td>
<td>Z^1/X</td>
<td>Hybrid Y^X</td>
<td>z^1/n = r^n * exp(i<em>n</em>Arg/n)</td>
<td>x in X reg; z in Y,Z regs</td>
<td>z^1/x in Z (XY)</td>
<td>does LastZ</td>
</tr>
<tr>
<td>11</td>
<td>Z^2</td>
<td>Complex X^2</td>
<td>z^2 = r^2 * exp(2i*Arg)</td>
<td>z in Z (XY)</td>
<td>z^2 in Z (XY)</td>
<td>does LastZ</td>
</tr>
<tr>
<td>12</td>
<td>Z^3</td>
<td>Cubic power</td>
<td>z=z^3</td>
<td>z in Z (Im in Y, Re in X)</td>
<td>result in Z (XY)</td>
<td>more accurate than Z^X</td>
</tr>
<tr>
<td>13</td>
<td>Z^X</td>
<td>Hybrid Y^X</td>
<td>z^n = r^n * exp(i<em>n</em>Arg)</td>
<td>x in X reg; z in Y,Z regs</td>
<td>z^n in Z (XY)</td>
<td>does LastZ</td>
</tr>
<tr>
<td>14</td>
<td>Z=0?</td>
<td>Is z=0?</td>
<td>is z=0?</td>
<td>z in Z (XY)</td>
<td>YES/NO (skips if false)</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Z=1?</td>
<td>Is z=1?</td>
<td>is z=1?</td>
<td>z in Z (XY)</td>
<td>YES/NO (skips if false)</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Z=W?</td>
<td>Is z=w?</td>
<td>is z=w?</td>
<td>w in W; z in Z (XY)</td>
<td>YES/NO (skips if false)</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Z=WR?</td>
<td>are z &amp; w equal if rounded?</td>
<td>is Rnd(z)=Rnd(w)?</td>
<td>w in W; z in Z (XY)</td>
<td>YES/NO (skips if false)</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>Z#0?</td>
<td>is z equal to zero?</td>
<td>is #0?</td>
<td>z in Z (XY)</td>
<td>YES/NO (skips if false)</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>Z#W?</td>
<td>is z equal to w?</td>
<td>is z=w?</td>
<td>w in W; z in Z (XY)</td>
<td>YES/NO (skips if false)</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>ZACOS</td>
<td>Complex ACOS</td>
<td>acos z = pi/2 - asin z</td>
<td>z in Z (XY)</td>
<td>acos(z) in Z (XY)</td>
<td>does LastZ</td>
</tr>
<tr>
<td>21</td>
<td>ZALOG</td>
<td>Complex 10^X</td>
<td>e^[z*ln(10)]</td>
<td>z in Z (XY)</td>
<td>10^z in Z (XY) and ALPHA</td>
<td>does LastZ</td>
</tr>
<tr>
<td>22</td>
<td>ZASIN</td>
<td>Complex ASIN</td>
<td>asin z = -i * asinh (iz)</td>
<td>z in Z (XY)</td>
<td>asinh(z) in Z (XY)</td>
<td>does LastZ</td>
</tr>
<tr>
<td>23</td>
<td>ZATAN</td>
<td>Complex ATAN</td>
<td>atan z = -i * atanhz (iz)</td>
<td>z in Z (XY)</td>
<td>atan(z) in Z (XY)</td>
<td>does LastZ</td>
</tr>
<tr>
<td>24</td>
<td>ZCOS</td>
<td>Complex COS</td>
<td>cos z = cosh (iz)</td>
<td>z in Z (XY)</td>
<td>cos(z) in Z (XY)</td>
<td>does LastZ</td>
</tr>
<tr>
<td>25</td>
<td>ZEXP</td>
<td>Complex e^X</td>
<td>e^n * e^(iy)</td>
<td>z in Z (XY)</td>
<td>e^z in Z (XY) and ALPHA</td>
<td>does LastZ</td>
</tr>
<tr>
<td>26</td>
<td>ZHACOS</td>
<td>Complex Hyp. ACOS</td>
<td>acosh z = Ln[z + SQ[z^2 - 1]]</td>
<td>z in Z (XY)</td>
<td>acosh(z) in Z (XY)</td>
<td>does LastZ</td>
</tr>
<tr>
<td>27</td>
<td>ZHASIN</td>
<td>Complex Hyp. ASIN</td>
<td>asinh z = Ln[z + SQ[z^2 + 1]]</td>
<td>z in Z (XY)</td>
<td>asinh(z) in Z (XY)</td>
<td>does LastZ</td>
</tr>
<tr>
<td>28</td>
<td>ZHATAN</td>
<td>Complex Hyp. ATAN</td>
<td>atanh z = 1/2 * Ln<a href="1-1/z">1/z</a></td>
<td>z in Z (XY)</td>
<td>atanh(z) in Z (XY)</td>
<td>does LastZ</td>
</tr>
<tr>
<td>29</td>
<td>ZHCOS</td>
<td>Complex Hyp. COS</td>
<td>cosh z = 1/2 * [e^n + e^-n]</td>
<td>z in Z (XY)</td>
<td>cosh(z) in Z (XY)</td>
<td>does LastZ</td>
</tr>
<tr>
<td>30</td>
<td>ZHSIN</td>
<td>Complex Hyp. SIN</td>
<td>sinh z = 1/2 * [e^n - e^-n]</td>
<td>z in Z (XY)</td>
<td>sinh(z) in Z (XY)</td>
<td>does LastZ</td>
</tr>
<tr>
<td>#</td>
<td>Function</td>
<td>Description</td>
<td>Formula</td>
<td>Input</td>
<td>Output</td>
<td>Comments</td>
</tr>
<tr>
<td>----</td>
<td>----------------</td>
<td>-------------------------------------------------</td>
<td>------------------------------------------------------------------------</td>
<td>--------</td>
<td>-------------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>31</td>
<td>ZHTAN</td>
<td>Complex Hyperbolic TAN</td>
<td>tanh z = (e^z-e^-z)/(e^z+e^-z)</td>
<td>z in (XY)</td>
<td>tanh(z) in (XY)</td>
<td>does LastZ</td>
</tr>
<tr>
<td>32</td>
<td>ZCF2V</td>
<td>Complex Continued Fractions</td>
<td>f(z) = B(0) + A1/B1 + A2/B2 + A3/[B3+...]</td>
<td>User program with Bn and An</td>
<td>F(Z) if convergence</td>
<td>Prompts for Prgm Name</td>
</tr>
<tr>
<td>33</td>
<td>ZDERV</td>
<td>Complex Function Derivatives</td>
<td>df/dz and d2f/dz</td>
<td>User program w/ f(z)</td>
<td>f&quot;(z) in &quot;W&quot;, f'(z) in &quot;Z&quot;</td>
<td>Prompts for Prgm Name</td>
</tr>
<tr>
<td>34</td>
<td>ZINT?</td>
<td>Checks if Z is an integer number</td>
<td>are Im(z)=0 and FRC(Re(z))=0?</td>
<td>z in Z</td>
<td>Im(z in Y, Re in X) YES/NO (skips if false)</td>
<td>used in Bessel fncs</td>
</tr>
<tr>
<td>35</td>
<td>ZINV</td>
<td>Complex Inversion</td>
<td>x/(x^2+ y^2) - i/y/(x^2 + y^2)</td>
<td>z in (XY)</td>
<td>1/z in (XY) and ALPHA</td>
<td>does LastZ</td>
</tr>
<tr>
<td>36</td>
<td>ZLN</td>
<td>Complex LN</td>
<td>ln(z) = ln(r) + i*Arg</td>
<td>z in (XY)</td>
<td>Ln(z) in (XY)</td>
<td>does LastZ</td>
</tr>
<tr>
<td>37</td>
<td>ZLOG</td>
<td>Complex LOG</td>
<td>log(z) = ln(z)/ln(10)</td>
<td>z in (XY)</td>
<td>Log(z) in (X,Y)</td>
<td>does LastZ</td>
</tr>
<tr>
<td>38</td>
<td>ZNEG</td>
<td>Complex CHS</td>
<td>-z = -x - iy</td>
<td>z in (XY)</td>
<td>-z in (XY)</td>
<td>does LastZ</td>
</tr>
<tr>
<td>39</td>
<td>ZOUT?</td>
<td>Is z outside the unit circle?</td>
<td>is</td>
<td>z &gt; 1?</td>
<td>z in (XY) YES/NO (skips if false)</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>ZPI*</td>
<td>Product by pi</td>
<td>z*p</td>
<td>z in (XY)</td>
<td>result in (XY)</td>
<td>more accurate than FOCAL</td>
</tr>
<tr>
<td>41</td>
<td>ZGSS?</td>
<td>Is z Gaussian?</td>
<td>Re(z) and Im(z) integers?</td>
<td>z in (XY)</td>
<td>YES/NO (skips if false)</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>ZRND</td>
<td>Rounds Z to display settings</td>
<td>rounded values to display</td>
<td>z in (XY)</td>
<td>Rounded Re &amp; Im in (XY)</td>
<td>does LastZ</td>
</tr>
<tr>
<td>43</td>
<td>ZSIN</td>
<td>Complex SIN</td>
<td>sin z = -i*sinh (iz)</td>
<td>z in (XY)</td>
<td>sin(z) in (XY)</td>
<td>does LastZ</td>
</tr>
<tr>
<td>44</td>
<td>ZSQRT</td>
<td>Complex SQRT (Direct)</td>
<td>sqrt(z) = sqrt(r) * e^(i*Arg/2)</td>
<td>z in (XY)</td>
<td>main value of z^1/2 in (XY)</td>
<td>does LastZ</td>
</tr>
<tr>
<td>45</td>
<td>ZTAN</td>
<td>Complex TAN</td>
<td>tan z = -i * tanh (iz)</td>
<td>z in (XY)</td>
<td>tan(z) in (XY)</td>
<td>does LastZ</td>
</tr>
<tr>
<td>46</td>
<td>ZUNIT?</td>
<td>Is z on the unit circle?</td>
<td>is</td>
<td>z &gt; 1?</td>
<td>z in (XY) YES/NO (skips if false)</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>-ZSTACK</td>
<td>Section Header</td>
<td>n/a</td>
<td>none</td>
<td>Shows &quot;Running...&quot; msg</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>CLZ</td>
<td>Clears Z</td>
<td>Re(z)=0=Im(z)</td>
<td>none</td>
<td>Z level (XY) cleared</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>CLZST</td>
<td>Clears Z-Stack</td>
<td>n/a</td>
<td>none</td>
<td>Z-Stack Cleared</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>LASTZ</td>
<td>Complex LASTX</td>
<td>n/a</td>
<td>none</td>
<td>Last z in X,Y regs;</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>ZAVIEW</td>
<td>Shows Complex Z</td>
<td>z in (XY)</td>
<td>none</td>
<td>Shows z in ALPHA</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>ZENTER^</td>
<td>Copies Z into the W register</td>
<td>z in (XY)</td>
<td>none</td>
<td>Pushes z one level Up</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>Z&lt;&gt;</td>
<td>Complex Exchange</td>
<td>Reg# as suffix</td>
<td>none</td>
<td>Exchanges Z with reg's contents</td>
<td>Prompting</td>
</tr>
<tr>
<td>54</td>
<td>Z&lt;&gt;ST</td>
<td>Exchanges Z and Level#</td>
<td>z in (XY); level# in prompt</td>
<td>none</td>
<td>z in L#, L# in L1 &amp; T, W in L1 &amp; X,Y</td>
<td>Prompting</td>
</tr>
<tr>
<td>55</td>
<td>Z&lt;&gt;W</td>
<td>Exchange Z and W (L2)</td>
<td>w in W, z in (XY)</td>
<td>none</td>
<td>z in L2 &amp; Z, T, w in L1 &amp; X,Y</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>ZIMAG^</td>
<td>Enter imaginary number</td>
<td>Im(z) in X</td>
<td>none</td>
<td>zero in X; Im(z) in Y</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>ZRCL^</td>
<td>Complex RCL</td>
<td>Reg# as suffix</td>
<td>none</td>
<td>z in X,Y - lifts stack</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>ZRDN</td>
<td>Z-Stack Roll Down</td>
<td>n/a</td>
<td>n/a</td>
<td>Rolls Down stack</td>
<td>Drops Buffer</td>
</tr>
<tr>
<td>59</td>
<td>ZREAL^</td>
<td>Enter Real number in Z</td>
<td>n/a</td>
<td>n/a</td>
<td>Re(z) in X</td>
<td>Lifts Buffer</td>
</tr>
<tr>
<td>60</td>
<td>ZRPL^</td>
<td>Replicates z in all levels</td>
<td>L4=L3=L2=L1</td>
<td>n/a</td>
<td>z in (XY) z in all levels</td>
<td>Lifts Buffer</td>
</tr>
<tr>
<td>61</td>
<td>ZRUP</td>
<td>Z-Stack Roll Up</td>
<td>n/a</td>
<td>n/a</td>
<td>Stack Levels</td>
<td>Lifts Buffer</td>
</tr>
<tr>
<td>62</td>
<td>ZSTO^</td>
<td>Complex STO</td>
<td>n/a</td>
<td>n/a</td>
<td>Reg# as suffix</td>
<td>Stores z in consecutive reg's</td>
</tr>
<tr>
<td>63</td>
<td>ZVIEW^</td>
<td>Complex View</td>
<td>n/a</td>
<td>n/a</td>
<td>Reg# as suffix</td>
<td>Shows z in ALPHA</td>
</tr>
<tr>
<td>64</td>
<td>ZK?YN</td>
<td>Block Key Assignments</td>
<td>n/a</td>
<td>n/a</td>
<td>prompt-driven</td>
<td>Makes / Removes assignments may do PACKING</td>
</tr>
<tr>
<td>1</td>
<td>^IM/AG</td>
<td>Natural Data Entry</td>
<td>Re ^ IM or r ^ arg</td>
<td>z in (XY)</td>
<td>stack lifted</td>
<td>Prompting, Lifts Buffer</td>
</tr>
<tr>
<td>2</td>
<td>GETSZ</td>
<td>Get z=Stack file from X-Mem</td>
<td>n/a</td>
<td>File Name in Alpha</td>
<td>Copies file to Buffer #8</td>
<td>Includes REC/POLAR</td>
</tr>
</tbody>
</table>

(c) Ángel M. Martin - May 2021
<table>
<thead>
<tr>
<th>#</th>
<th>Function</th>
<th>Description</th>
<th>Formula</th>
<th>Input</th>
<th>Output</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>NXTACS</td>
<td>Next AOS Value</td>
<td>z_{1,2} = +/- z_0 + 2p</td>
<td>z_0 \text{ in } Z (XY)</td>
<td>z_1 \text{ in } W, z_2 \text{ in } Z (XY)</td>
<td>does LastZ</td>
</tr>
<tr>
<td>4</td>
<td>NXTASN</td>
<td>Next ASN Value</td>
<td>z_{1,2} = +/- z_0 + 2p/2</td>
<td>z_0 \text{ in } Z (XY)</td>
<td>z_1 \text{ in } W, z_2 \text{ in } Z (XY)</td>
<td>does LastZ</td>
</tr>
<tr>
<td>5</td>
<td>NXTATN</td>
<td>Next ATAN value</td>
<td>z_{1,2} = z_0 +/- p</td>
<td>z_0 \text{ in } Z (XY)</td>
<td>z_1 \text{ in } W, z_2 \text{ in } Z (XY)</td>
<td>does LastZ</td>
</tr>
<tr>
<td>6</td>
<td>NXLN</td>
<td>Next LN(z)</td>
<td>\text{next}(k) = \text{Ln}(z) + 2zp \text{ j}</td>
<td>\text{LN(z)} \text{ in } Z (XY) \text{ regs}</td>
<td>z_1 \text{ in } W, z_2 \text{ in } Z (XY)</td>
<td>does LastZ</td>
</tr>
<tr>
<td>7</td>
<td>NXXRN</td>
<td>Next XRN</td>
<td>\text{next}(k) = z^n \text{ j} * e^{n}(2kp/n)</td>
<td>n \text{ in } X \text{ reg.}; z^n \text{ j} \text{ in } Z, Y \text{ regs}</td>
<td>z_1^{i\text{h}} * e^{n}(2p/n) \text{ j} \text{ in } Z (XY)</td>
<td>does LastZ</td>
</tr>
<tr>
<td>8</td>
<td>SAVEZS</td>
<td>Saves z Buffer to X-Mem</td>
<td>n/\text{a}</td>
<td>File Name in ALPHA</td>
<td>Copies buffer #8 to File Includes REC/POLAR</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>ZCHXS</td>
<td>Sign Change by X</td>
<td>(-1)^n * z</td>
<td>x \text{ in } X \text{ reg}; z \text{ in } Y, Z \text{ regs}</td>
<td>((-1)^n * z) \text{ in } Z (XY)</td>
<td>does LastZ</td>
</tr>
<tr>
<td>10</td>
<td>ZGEU</td>
<td>Euler's gamma constant</td>
<td>y_0 = 0,577215665</td>
<td>none</td>
<td>g constant as complex</td>
<td>Lifts Buffer</td>
</tr>
<tr>
<td>11</td>
<td>ZKL</td>
<td>Complex keyboard launcher</td>
<td>n/\text{a}</td>
<td>Prompt-driven</td>
<td>Launches function</td>
<td>prompting, launcher</td>
</tr>
<tr>
<td>12</td>
<td>ZPL</td>
<td>Complex Polynomial Evaluation</td>
<td>P(z) = \sum ak z^n</td>
<td>Control word bbb.eee</td>
<td>Polynomial result</td>
<td>Coeffs. Expected in ZRegs</td>
</tr>
<tr>
<td>13</td>
<td>ZRC+</td>
<td>RCL addition</td>
<td>Z = \text{z} + \text{cR}</td>
<td>z \text{ in } Z (XY), data in cR</td>
<td>Adds cR to z</td>
<td>does LastZ</td>
</tr>
<tr>
<td>14</td>
<td>ZRC-</td>
<td>RCL subtraction</td>
<td>Z = \text{z} - \text{cR}</td>
<td>z \text{ in } Z (XY), data in cR</td>
<td>Subtracts cR from z</td>
<td>does LastZ</td>
</tr>
<tr>
<td>15</td>
<td>ZRC*</td>
<td>RCL product</td>
<td>Z = \text{z} \times \text{cR}</td>
<td>z \text{ in } Z (XY), data in cR</td>
<td>Multiplies x by cR</td>
<td>does LastZ</td>
</tr>
<tr>
<td>16</td>
<td>ZRC/</td>
<td>RCL division</td>
<td>Z = \text{z} / \text{cR}</td>
<td>z \text{ in } Z (XY), data in cR</td>
<td>Divides z by cR</td>
<td>does LastZ</td>
</tr>
<tr>
<td>17</td>
<td>ZST+</td>
<td>STO Addition</td>
<td>cR = \text{cR} + z</td>
<td>z \text{ in } Z (XY), data in cR</td>
<td>Adds cR to complex register</td>
<td>prompting</td>
</tr>
<tr>
<td>18</td>
<td>ZST-</td>
<td>STO Subtraction</td>
<td>cR = \text{cR} - z</td>
<td>z \text{ in } Z (XY), data in cR</td>
<td>Subtracts z from complex register</td>
<td>prompting</td>
</tr>
<tr>
<td>19</td>
<td>ZST*</td>
<td>STO Multiply</td>
<td>cR = \text{cR} \times z</td>
<td>z \text{ in } Z (XY), data in cR</td>
<td>Multiplies z to complex register</td>
<td>prompting</td>
</tr>
<tr>
<td>20</td>
<td>ZST/</td>
<td>STO Divide</td>
<td>cR = \text{cR} / z</td>
<td>z \text{ in } Z (XY), data in cR</td>
<td>Divides complex register by z</td>
<td>prompting</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section Header</th>
<th>n/\text{a}</th>
<th>none</th>
<th>Displays Revision Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>21 -ZVECTOR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>POLAR</td>
<td>Sets POLAR mode on</td>
<td>sets the Polar flag in Buffer</td>
</tr>
<tr>
<td>23</td>
<td>RECT</td>
<td>Sets RECT mode on</td>
<td>clears the Polar flag in Buffer</td>
</tr>
<tr>
<td>24</td>
<td>ZAGM</td>
<td>Arithmetic-Geometric Mean</td>
<td>AGM</td>
</tr>
<tr>
<td>25</td>
<td>ZARG</td>
<td>Argument of Z</td>
<td>atan(y/x)</td>
</tr>
<tr>
<td>26</td>
<td>ZMOD</td>
<td>Module of Z</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>ZNORM</td>
<td>Norm of Z (i.e. square of Module)</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>ZPOL</td>
<td>Converts to Polar notation</td>
<td>R-P</td>
</tr>
<tr>
<td>29</td>
<td>ZREC</td>
<td>Converts to Rectangular notation</td>
<td>P-R</td>
</tr>
<tr>
<td>30</td>
<td>ZWANG</td>
<td>Angle between Z and W</td>
<td>\text{arg}(z_1) = \text{Arg}(z) - \text{Arg}(w)</td>
</tr>
<tr>
<td>31</td>
<td>ZWCROSS</td>
<td>Cross product of Z and W</td>
<td>x \times w =</td>
</tr>
<tr>
<td>32</td>
<td>ZWDET</td>
<td>Determinant of Z and W</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>ZWDIST</td>
<td>Distance between Z and W</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>ZWDOT</td>
<td>Dot product of Z and W</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>ZWLINE</td>
<td>Line equation defined by Z and W</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>ZWLOG</td>
<td>Base-w Logarithm</td>
<td>base \text{ w in } W, \text{ arg. in } Z</td>
</tr>
</tbody>
</table>

(c) Ángel M. Martin – May 2021
<table>
<thead>
<tr>
<th>#</th>
<th>Function</th>
<th>Description</th>
<th>Formula</th>
<th>Input</th>
<th>Output</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>38</td>
<td>ZAWL</td>
<td>Inverse of Lambert W</td>
<td>$z^* e^{*z}$</td>
<td>$z$ in Z (XY)</td>
<td>result in Z (XY)</td>
<td>does LastZ</td>
</tr>
<tr>
<td>39</td>
<td>ZBS#</td>
<td>Bessel subroutine 1st./2nd. Kind</td>
<td>see manual, Flag 6 controls case</td>
<td>$w$ in W, $z/2$ in Z</td>
<td>$w$ in ZR00, $z/2$ in ZR01</td>
<td>FOCAL</td>
</tr>
<tr>
<td>40</td>
<td>ZC1</td>
<td>Cosine Integral</td>
<td>$C_i(z) = -i^2 z^2/4 F_2(1, 1; 2, z; 2/3, z^2/4)$</td>
<td>$z$ in Z (XY)</td>
<td>result in Z (XY)</td>
<td>FOCAL</td>
</tr>
<tr>
<td>41</td>
<td>ZCRT</td>
<td>Complex Cubic Eq. Roots</td>
<td>Cubic equation roots</td>
<td>A,B,C,D in Z-Stack</td>
<td>roots in V, W, and Z (XY) levels</td>
<td>FOCAL</td>
</tr>
<tr>
<td>42</td>
<td>ZEI</td>
<td>Exponential Integral</td>
<td>$E_i = z^{\gamma} + \ln(z^{\gamma} + z^{\gamma} F_2(1, 1; 2, z; z)$</td>
<td>$z$ in Z (XY)</td>
<td>result in Z (XY)</td>
<td>FOCAL</td>
</tr>
<tr>
<td>43</td>
<td>ZERF</td>
<td>Error Function</td>
<td>erf(z) = $2z/sqrt(z) e^{(-z^2)} F_1(1, 1; 2, z^2)$</td>
<td>$z$ in Z (XY)</td>
<td>result in Z (XY)</td>
<td>FOCAL</td>
</tr>
<tr>
<td>44</td>
<td>ZGAMMA</td>
<td>Complex G(z) for $z\neq 0, -1, -2$</td>
<td>Lanczos approximation</td>
<td>$z$ in Z (XY)</td>
<td>G(z) in Z (XY)</td>
<td>uses reflection for Re(z)&lt;0</td>
</tr>
<tr>
<td>45</td>
<td>ZHCI</td>
<td>Hyperbolic Cosine Integral</td>
<td>$C_i(z) = (z^2/4) F_2(1, 1; 2, z; 2/3, z^2/4)$</td>
<td>$z$ in Z (XY)</td>
<td>result in Z (XY)</td>
<td>FOCAL</td>
</tr>
<tr>
<td>46</td>
<td>ZHGCF</td>
<td>Hypergeometric Function</td>
<td>See manual</td>
<td>see manual</td>
<td>result in Z (XY)</td>
<td>by Jean-Marc Baillard</td>
</tr>
<tr>
<td>47</td>
<td>ZHSI</td>
<td>Hyperbolic Sine Integral</td>
<td>$S_i(z) = z^* F_{12}(1/2, 3/2, 3/2, z^2/4)$</td>
<td>$z$ in Z (XY)</td>
<td>result in Z (XY)</td>
<td>FOCAL</td>
</tr>
<tr>
<td>48</td>
<td>ZIBS</td>
<td>Bessel I function</td>
<td>see manual</td>
<td>$w$ in W, $z$ in Z (XY)</td>
<td>I(w,z) in Z (XY)</td>
<td>FOCAL</td>
</tr>
<tr>
<td>49</td>
<td>ZIBS</td>
<td>Bessel J function</td>
<td>see manual</td>
<td>$w$ in W, $z$ in Z (XY)</td>
<td>J(w,z) in Z (XY)</td>
<td>FOCAL</td>
</tr>
<tr>
<td>50</td>
<td>ZKB5</td>
<td>Bessel K function</td>
<td>see manual</td>
<td>$w$ in W, $z$ in Z (XY)</td>
<td>K(w,z) in Z (XY)</td>
<td>FOCAL</td>
</tr>
<tr>
<td>51</td>
<td>Z12I</td>
<td>Dilogarith</td>
<td>$L_1(z) = \sum(z^{n} / (n+k)^2)$; $k=1,2,...$</td>
<td>$z$ in Z (XY)</td>
<td>result in Z (XY)</td>
<td>by Jean-Marc Baillard</td>
</tr>
<tr>
<td>52</td>
<td>Z12N</td>
<td>Polylogarith</td>
<td>$L_2(z) = (z^2/4) F_2(1, 3/2, z^2/4)$</td>
<td>order w in W; arg. z in Z</td>
<td>result in Z (XY)</td>
<td>FOCAL</td>
</tr>
<tr>
<td>53</td>
<td>Z12NG</td>
<td>Gamma Logarithm function</td>
<td>Stirling method with correction</td>
<td>$z$ in Z (XY)</td>
<td>result in Z (XY)</td>
<td>FOCAL</td>
</tr>
<tr>
<td>54</td>
<td>ZLRCH</td>
<td>Lerch Transcendent</td>
<td>$F(z, s, a) = \sum(z^{n} / (n+a)^s)$; $k=0,1,...$</td>
<td>$a, z$ in U, W, and Z (XY)</td>
<td>result in Z (XY)</td>
<td>FOCAL</td>
</tr>
<tr>
<td>55</td>
<td>ZPROOT</td>
<td>Roots of complex polynomials</td>
<td>Iterative Prompt-driven</td>
<td>roots in W and Z (XY) levels</td>
<td>by Valentin Albillo</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>ZPSI</td>
<td>Complex Digamma</td>
<td>Approximation</td>
<td>$z$ in (XY)</td>
<td>Psi(z) in X,Y,_REGS and ALPHA</td>
<td>FOCAL</td>
</tr>
<tr>
<td>57</td>
<td>ZQRT</td>
<td>Complex Quadratic Eq. Roots</td>
<td>Quadratic equation roots</td>
<td>A,B,C in Z-Stack</td>
<td>Calculates roots of equation</td>
<td>FOCAL</td>
</tr>
<tr>
<td>58</td>
<td>ZSHK1</td>
<td>Spherical Hankel h1</td>
<td>$h_{11}(w, z)$</td>
<td>order w in W; arg. z in Z</td>
<td>result in Z (XY)</td>
<td>FOCAL</td>
</tr>
<tr>
<td>59</td>
<td>ZSHK2</td>
<td>Spherical Hankel h2</td>
<td>$h_{22}(w, z)$</td>
<td>order w in W; arg. z in Z</td>
<td>result in Z (XY)</td>
<td>FOCAL</td>
</tr>
<tr>
<td>60</td>
<td>ZSI</td>
<td>Sine Integral</td>
<td>$S_i(z) = z^* F_{12}(1/2, 3/2, 3/2, z^2/4)$</td>
<td>$z$ in Z (XY)</td>
<td>result in Z (XY)</td>
<td>FOCAL</td>
</tr>
<tr>
<td>61</td>
<td>ZSOLVE</td>
<td>Solves for $f(z)=0$</td>
<td>Newton’s method</td>
<td>Fnc. name in R06</td>
<td>Calculates one root for $f(z)$</td>
<td>FOCAL</td>
</tr>
<tr>
<td>62</td>
<td>ZWL</td>
<td>Lambert W function</td>
<td>see manual</td>
<td>$z$ in Z (XY)</td>
<td>W(z) in Z (XY)</td>
<td>FOCAL</td>
</tr>
<tr>
<td>63</td>
<td>ZYSB</td>
<td>Bessel Y function</td>
<td>see manual</td>
<td>$w$ in W, $z$ in Z (XY)</td>
<td>Y(w,z) in Z (XY)</td>
<td>FOCAL</td>
</tr>
<tr>
<td>64</td>
<td>ZZETA</td>
<td>Riemann Zeta Function</td>
<td>Borwein Algorithm</td>
<td>$z$ in Z (XY)</td>
<td>by Jean-Marc Baillard</td>
<td></td>
</tr>
</tbody>
</table>

(c) Ángel M. Martin - May 2021

118
<table>
<thead>
<tr>
<th>#</th>
<th>Function</th>
<th>Description</th>
<th>Formula</th>
<th>Input</th>
<th>Output</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>ZBLS</td>
<td>Bessel Funcs. Sub-Launcher</td>
<td>n/a</td>
<td>Prompts for Function</td>
<td>Executes Function</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>ZCONJ</td>
<td>Complex Conjugate</td>
<td>conj = x - iy</td>
<td>z in Z (XY)</td>
<td>Inverts sign of Im(z)</td>
<td>does LastZ</td>
</tr>
<tr>
<td>11</td>
<td>ZDISP</td>
<td>Displays Z in LCD</td>
<td>Z = Re:Im</td>
<td>Values in Y, X</td>
<td>String in LCD</td>
<td>No negative values!</td>
</tr>
<tr>
<td>12</td>
<td>ZDDBL</td>
<td>Doubles z</td>
<td>2*z</td>
<td>z in Z (XY)</td>
<td>2z in Z (XY)</td>
<td>does LastZ</td>
</tr>
<tr>
<td>13</td>
<td>ZFRC</td>
<td>Makes Re(z), Im(z) fractional</td>
<td>Int(Re(z)) = Int(Im(z)) = 0</td>
<td>z in Z (XY)</td>
<td>Result in Z (XY)</td>
<td>does LastZ</td>
</tr>
<tr>
<td>14</td>
<td>ZHALF</td>
<td>Halves z</td>
<td>z/2</td>
<td>z in Z (XY)</td>
<td>z/2 in Z (XY)</td>
<td>Does LastZ</td>
</tr>
<tr>
<td>15</td>
<td>ZHGF</td>
<td>Hypergeometric Launcher</td>
<td>n/a</td>
<td>Prompts for choice</td>
<td>Executes function</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>ZHYP</td>
<td>Hyperbolic Launcher</td>
<td>n/a</td>
<td>Prompts for choice</td>
<td>Executes function</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>ZIMAG?</td>
<td>Is Im(z)=0?</td>
<td>z in Z (XY)</td>
<td>YES/NO (skips if false)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>ZIN?</td>
<td>Is z inside the unit circle?</td>
<td></td>
<td>z in Z (XY)</td>
<td>YES/NO (skips if false)</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>ZINT</td>
<td>Makes Re(z) and Im(z) integers</td>
<td>Re(z) = Int(Re(z)); Im(z) = Int(Im(z))</td>
<td>z in Z (XY)</td>
<td>Result in Z (XY)</td>
<td>Does LastZ</td>
</tr>
<tr>
<td>20</td>
<td>ZMTV</td>
<td>Multi-functions Launcher</td>
<td>n/a</td>
<td>Prompts for choice</td>
<td>Executes function</td>
<td>FOCAL</td>
</tr>
<tr>
<td>21</td>
<td>ZNXT</td>
<td>NEXT function Launcher</td>
<td>n/a</td>
<td>Prompts for choice</td>
<td>Executes functions</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>ZPI</td>
<td>Pi as a complex number</td>
<td>Zpi = pi + j0</td>
<td>none</td>
<td>Pi in Z(XY)</td>
<td>Lifts Buffer</td>
</tr>
<tr>
<td>23</td>
<td>ZPRT</td>
<td>Poly-roots functions Launcher</td>
<td>n/a</td>
<td>Prompts for choice</td>
<td>Executes function</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>ZREAL?</td>
<td>Is Re(z)=0?</td>
<td>z in Z (XY)</td>
<td>YES/NO (skips if false)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>ZQUAD</td>
<td>Shows quadrant for z</td>
<td>Quad# as function of location</td>
<td>z in Z (XY)</td>
<td>Sets corresponding user flag 1-4</td>
<td>Clears other flags 1-4</td>
</tr>
<tr>
<td>26</td>
<td>ZSIGN</td>
<td>Complex SIGN</td>
<td>Sign = z/</td>
<td>z</td>
<td></td>
<td>z in Z (XY)</td>
</tr>
<tr>
<td>27</td>
<td>ZTONE</td>
<td>Makes a sound</td>
<td>Frequency and duration</td>
<td>z in Z (XY)</td>
<td>Makes sound</td>
<td>Shows Z at end</td>
</tr>
<tr>
<td>28</td>
<td>ZTRP</td>
<td>Exchanges Re(Z) and Im(Z)</td>
<td>zTrp = y + iX</td>
<td>z in Z (XY)</td>
<td>Im(z) in X, Re(z) in Y</td>
<td>does LastZ</td>
</tr>
<tr>
<td>29</td>
<td>-DELUXE-</td>
<td>Section Header</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>Data expected in ZRegs</td>
</tr>
<tr>
<td>30</td>
<td>ZAMN</td>
<td>Complex Arithmetic Mean</td>
<td>AM = ( \Sigma zk/n )</td>
<td>Control word bbb.eee in X</td>
<td>Result in Z (XY)</td>
<td>Data expected in ZRegs</td>
</tr>
<tr>
<td>31</td>
<td>ZANGJ</td>
<td>Anger J(n,z) Function</td>
<td>See manual</td>
<td>z in Z(Y), n in Z</td>
<td>Result in Z (XY)</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>ZCRF</td>
<td>Carlson Integral 1st. kind</td>
<td>See manual</td>
<td>n,n,p,z in stack</td>
<td>Result in X</td>
<td>Complex conjugate</td>
</tr>
<tr>
<td>33</td>
<td>ZCRJ</td>
<td>Carlson Integral 3rd. kind</td>
<td>See manual</td>
<td>n,n,p,z in stack</td>
<td>Result in X</td>
<td>Complex conjugate</td>
</tr>
<tr>
<td>34</td>
<td>ZCSX</td>
<td>Fresnel Integrals C(x) &amp; S(x)</td>
<td>See manual</td>
<td>X in X</td>
<td>S(x) in Y, C(x) in X</td>
<td>FOCAL</td>
</tr>
<tr>
<td>35</td>
<td>ZELIP1</td>
<td>Incomplete Elliptic integral 1st kind</td>
<td>Complex amplitude, real modulus</td>
<td>a in (Y,Z), m in X</td>
<td>Result in Z (XY)</td>
<td>FOCAL</td>
</tr>
<tr>
<td>36</td>
<td>ZELIP2</td>
<td>Incomplete Elliptic Integral 2nd kind</td>
<td>Complex amplitude, real modulus</td>
<td>a in (Y,Z), m in X</td>
<td>Result in Z (XY)</td>
<td>FOCAL</td>
</tr>
<tr>
<td>37</td>
<td>ZELIP3</td>
<td>Complete Elliptic Integral 2nd kind</td>
<td>Uses Hypergeometric functions</td>
<td>Complex m in Z (XY)</td>
<td>Result in Z (XY)</td>
<td>Requires</td>
</tr>
<tr>
<td>38</td>
<td>ZELIPK</td>
<td>Complete Elliptic Integral 1st kind</td>
<td>Uses Hypergeometric functions</td>
<td>Complex m in Z (XY)</td>
<td>Result in Z (XY)</td>
<td>Requires</td>
</tr>
<tr>
<td>39</td>
<td>ZELK</td>
<td>Complete Elliptic Integral 1st kind</td>
<td>Uses AGM</td>
<td>Complex m in Z (XY)</td>
<td>Result in Z (XY)</td>
<td>FOCAL</td>
</tr>
<tr>
<td>40</td>
<td>ZELPKE</td>
<td>Comlete Elliptic Intg. 1st &amp; 2nd kinds</td>
<td>Uses AGM and AGM2</td>
<td>Complex m in Z (XY)</td>
<td>Results in W and Z (XY)</td>
<td>FOCAL</td>
</tr>
<tr>
<td>41</td>
<td>ZGHM</td>
<td>Geometric-Harmonic Mean</td>
<td>GHM</td>
<td>w in W, z in Z (XY)</td>
<td>Result in Z (XY)</td>
<td>does LastZ</td>
</tr>
<tr>
<td>42</td>
<td>ZGMN</td>
<td>Geometric Complex mean</td>
<td>GM = ( [I1 Zk]^*1/k )</td>
<td>Control word bbb.eee in X</td>
<td>Geometric mean in Z (XY)</td>
<td>Data expected in ZRegs</td>
</tr>
<tr>
<td>#</td>
<td>Function</td>
<td>Description</td>
<td>Formula</td>
<td>Input</td>
<td>Output</td>
<td>Comments</td>
</tr>
<tr>
<td>----</td>
<td>------------</td>
<td>-------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>----------------------------------------------------</td>
<td>----------------------------------------------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>43</td>
<td>ZHMN</td>
<td>Complex Harmonic Mean</td>
<td>$HM = \sum 1 / [1/z_k]$</td>
<td>Control word bbb.eee in X</td>
<td>Harmonic mean in $Z$ (XY)</td>
<td>Data expected in ZRegs</td>
</tr>
<tr>
<td>44</td>
<td>ZINPT</td>
<td>Enters complex data in ZRegs</td>
<td>n/a</td>
<td>Control word bbb.eee in X</td>
<td>Data is stored sequentially</td>
<td>FOCAL</td>
</tr>
<tr>
<td>45</td>
<td>ZKLV1</td>
<td>Kelvin Functions 1st kind</td>
<td>Uses Hypergeometric Function</td>
<td>x in X</td>
<td>bei(x) in Y, ber(x) in X</td>
<td>FOCAL</td>
</tr>
<tr>
<td>46</td>
<td>ZOUPT</td>
<td>Shows complex data</td>
<td>n/a</td>
<td>Control word bbb.eee in X</td>
<td>Data is shown sequentially</td>
<td>SF 21 to stop each value</td>
</tr>
<tr>
<td>47</td>
<td>ZPD1</td>
<td>Complex Polynomial 1st derivative</td>
<td>$P'(z) = \Sigma k ak z^{*k-1}</td>
<td>k=1,2..n$</td>
<td>$z_0$ in $(Y,Y)$ ; bbb.eee in X</td>
<td>$dP(z)/dz$ in “Z” (XY)</td>
</tr>
<tr>
<td>48</td>
<td>ZPD2</td>
<td>Complex Polynomial 2nd derivative</td>
<td>$P''(z) = \Sigma k (k-1) ak z^{k-2}</td>
<td>k=2,3..n$</td>
<td>$z_0$ in $(Y,Y)$ ; bbb.eee in X</td>
<td>$d2P(z)/dz^2$ in “Z” (XY)</td>
</tr>
<tr>
<td>49</td>
<td>ZPLI</td>
<td>Complex Polynomial Primitive</td>
<td>$IT[P(z)] = \Sigma ak z^{k+1} / (k+1)</td>
<td>k=0,1..n$</td>
<td>$z_0$ in $(Y,Y)$ ; bbb.eee in X</td>
<td>Result in $Z$ (XY)</td>
</tr>
<tr>
<td>50</td>
<td>ZPSIN</td>
<td>Complex Poly-Gamma function</td>
<td>See manual</td>
<td>$z$ in $(Y,Y)$ ; $n$ in X</td>
<td>Result in $Z$ (XY)</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>ZSJB</td>
<td>Complex Spherical Bessel J(w,z)</td>
<td>$j(w,z) = \sqrt{\pi/2z} J(w+1/2, z)$</td>
<td>w in $W$, z in $Z$ (XY)</td>
<td>Result in $Z$ (XY)</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>ZSYB</td>
<td>Complex Spherical Bessel Y(w,z)</td>
<td>$y(w,z) = \sqrt{\pi/2z} Y(w+1/2, z)$</td>
<td>w in $W$, z in $Z$ (XY)</td>
<td>Result in $Z$ (XY)</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>ZWEBE</td>
<td>Weber function $E(n,z)$</td>
<td>See manual</td>
<td>$z$ in $(Y,X)$ ; $n$ in X</td>
<td>Result in $Z$ (XY)</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>CAT+ _</td>
<td>Sub-function Catalog</td>
<td>n/a</td>
<td>none</td>
<td>Sequential Enumeration</td>
<td>XEQ executes function</td>
</tr>
<tr>
<td>55</td>
<td>(c)</td>
<td>Copyright Message</td>
<td>n/a</td>
<td>none</td>
<td>Shows copyright in ALPHA</td>
<td>&quot;(c) AMC 2016&quot;</td>
</tr>
<tr>
<td>56</td>
<td>LASTF</td>
<td>Executes last function</td>
<td>n/a</td>
<td>Previous call by launcher</td>
<td>Re-executes function</td>
<td>Includes sub-functions</td>
</tr>
</tbody>
</table>

(c) Ángel M. Martin – May 2021
<table>
<thead>
<tr>
<th></th>
<th>Section Header</th>
<th>n/a</th>
<th>None</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>C2ZB</td>
<td>n/a</td>
<td>None</td>
<td>buffer cleared</td>
</tr>
<tr>
<td>3</td>
<td>L1=XY?</td>
<td>n/a</td>
<td>None</td>
<td>Y/N, skip if false</td>
</tr>
<tr>
<td>4</td>
<td>L1&lt;&gt;L _ _</td>
<td>n/a</td>
<td>Level# as suffix</td>
<td>levels exchanged</td>
</tr>
<tr>
<td>5</td>
<td>L1&lt;&gt;L2</td>
<td>n/a</td>
<td>None</td>
<td>levels exchanged</td>
</tr>
<tr>
<td>6</td>
<td>L1&lt;&gt;L3</td>
<td>n/a</td>
<td>None</td>
<td>levels exchanged</td>
</tr>
<tr>
<td>7</td>
<td>L1&lt;&gt;L4</td>
<td>n/a</td>
<td>None</td>
<td>levels exchanged</td>
</tr>
<tr>
<td>8</td>
<td>L1&lt;&gt;LX</td>
<td>n/a</td>
<td>level in X</td>
<td>levels exchanged</td>
</tr>
<tr>
<td>9</td>
<td>L2=ZT?</td>
<td>n/a</td>
<td>None</td>
<td>Y/N, skip if false</td>
</tr>
<tr>
<td>10</td>
<td>L2&lt;&gt;ZT</td>
<td>n/a</td>
<td>L2 copied to ZT</td>
<td>Prompting</td>
</tr>
<tr>
<td>11</td>
<td>VIEW _</td>
<td>n/a</td>
<td>Level# as suffix</td>
<td>Transposed value!</td>
</tr>
<tr>
<td>12</td>
<td>VIEWX</td>
<td>n/a</td>
<td>level in X</td>
<td>Transposed value!</td>
</tr>
<tr>
<td>13</td>
<td>PREMON</td>
<td>n/a</td>
<td>Re(z) in X; Im(z) in Y</td>
<td>none</td>
</tr>
<tr>
<td>14</td>
<td>PSTMON</td>
<td>n/a</td>
<td>Re(z) in X; Im(z) in Y</td>
<td>None</td>
</tr>
<tr>
<td>15</td>
<td>RG&gt;ZB _ _</td>
<td>n/a</td>
<td>Reg# as suffix</td>
<td>data copied from registers</td>
</tr>
<tr>
<td>16</td>
<td>ST&gt;ZB</td>
<td>n/a</td>
<td>None</td>
<td>stack copied to buffer</td>
</tr>
<tr>
<td>17</td>
<td>KY=L _</td>
<td>n/a</td>
<td>Level# as suffix</td>
<td>XY copied to LEVEL</td>
</tr>
<tr>
<td>18</td>
<td>KY=L0</td>
<td>n/a</td>
<td>Re(z) in X; Im(z) in Y</td>
<td>XY copied to L0</td>
</tr>
<tr>
<td>19</td>
<td>KY=L1</td>
<td>n/a</td>
<td>Re(z) in X; Im(z) in Y</td>
<td>XY copied to L1</td>
</tr>
<tr>
<td>20</td>
<td>ZB&gt;RG _ _</td>
<td>n/a</td>
<td>Reg# as suffix</td>
<td>data copied to registers</td>
</tr>
<tr>
<td>21</td>
<td>ZB&gt;ST</td>
<td>n/a</td>
<td>None</td>
<td>buffer copied to Stack</td>
</tr>
<tr>
<td>22</td>
<td>ZBDROP</td>
<td>n/a</td>
<td>None</td>
<td>levels dropped</td>
</tr>
<tr>
<td>23</td>
<td>ZBHEAD</td>
<td>n/a</td>
<td>None</td>
<td>header register in ALPHA</td>
</tr>
<tr>
<td>24</td>
<td>ZBLIFT</td>
<td>n/a</td>
<td>None</td>
<td>buffer lifted</td>
</tr>
<tr>
<td>25</td>
<td>ZBVIEW</td>
<td>n/a</td>
<td>None</td>
<td>shows header &amp; all levels</td>
</tr>
<tr>
<td>26</td>
<td>-B UTILS</td>
<td>n/a</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>27</td>
<td>B?</td>
<td>n/a</td>
<td>buffer id# in X</td>
<td>YES/NO (skips if false)</td>
</tr>
<tr>
<td>28</td>
<td>LIST</td>
<td>n/a</td>
<td>none</td>
<td>list in Alpha</td>
</tr>
<tr>
<td>29</td>
<td>BLNG?</td>
<td>n/a</td>
<td>buffer id# in X</td>
<td>buffer size in X</td>
</tr>
<tr>
<td>30</td>
<td>BX&gt;RG</td>
<td>n/a</td>
<td>buffer id# in X</td>
<td>data copied into R00 to end</td>
</tr>
<tr>
<td>31</td>
<td>C2B</td>
<td>n/a</td>
<td>buffer id# in X</td>
<td>clears buffer from memory</td>
</tr>
<tr>
<td>32</td>
<td>FNDDBX</td>
<td>n/a</td>
<td>buffer id# in X</td>
<td>buffer address in X</td>
</tr>
<tr>
<td>33</td>
<td>MAKEBX</td>
<td>n/a</td>
<td>(id#,size) in X</td>
<td>buffer created</td>
</tr>
<tr>
<td>34</td>
<td>RG&gt;BX</td>
<td>n/a</td>
<td>Data in R00 to Rnn</td>
<td>Copied to Buffer</td>
</tr>
</tbody>
</table>

(*) Buffer functions have been moved to the BUFFERLAND Module, under a dedicated section for the 41Z case.
### Appendix 5.- Buffer logic function table.

<table>
<thead>
<tr>
<th>Pre-Exec</th>
<th>Post-Exec</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Alpha in XY</strong></td>
<td><strong>XY to L0</strong></td>
</tr>
<tr>
<td><strong>Buffer DROP</strong></td>
<td><strong>XY into L1</strong></td>
</tr>
<tr>
<td>1</td>
<td>- HP-41 Z</td>
</tr>
<tr>
<td>2</td>
<td>W^Z</td>
</tr>
<tr>
<td>3</td>
<td>Z+</td>
</tr>
<tr>
<td>4</td>
<td>Z-</td>
</tr>
<tr>
<td>5</td>
<td>Z*</td>
</tr>
<tr>
<td>6</td>
<td>Z/</td>
</tr>
<tr>
<td>7</td>
<td>ZWANG</td>
</tr>
<tr>
<td>8</td>
<td>ZWCROSS</td>
</tr>
<tr>
<td>9</td>
<td>ZWDET</td>
</tr>
<tr>
<td>10</td>
<td>ZWDIST</td>
</tr>
<tr>
<td>11</td>
<td>ZWDOT</td>
</tr>
<tr>
<td>12</td>
<td>ZWLINE</td>
</tr>
<tr>
<td>13</td>
<td>Z=W?</td>
</tr>
<tr>
<td>14</td>
<td>Z=WR?</td>
</tr>
<tr>
<td>15</td>
<td>Z#W?</td>
</tr>
<tr>
<td>16</td>
<td>Z=0?</td>
</tr>
<tr>
<td>17</td>
<td>Z#0?</td>
</tr>
<tr>
<td>18</td>
<td>Z=i?</td>
</tr>
<tr>
<td>19</td>
<td>ZREAL?</td>
</tr>
<tr>
<td>20</td>
<td>ZIMAG?</td>
</tr>
<tr>
<td>21</td>
<td>ZIN?</td>
</tr>
<tr>
<td>22</td>
<td>ZOUT?</td>
</tr>
<tr>
<td>23</td>
<td>ZUNIT?</td>
</tr>
<tr>
<td>24</td>
<td>X^2</td>
</tr>
<tr>
<td>25</td>
<td>Z^2</td>
</tr>
<tr>
<td>26</td>
<td>Z^X</td>
</tr>
<tr>
<td>27</td>
<td>ZACOS</td>
</tr>
<tr>
<td>28</td>
<td>ZACOSH</td>
</tr>
<tr>
<td>29</td>
<td>ZALOG</td>
</tr>
<tr>
<td>30</td>
<td>ZASIN</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>ASINH</td>
<td>asinh</td>
</tr>
<tr>
<td>ATAN</td>
<td>atan</td>
</tr>
<tr>
<td>ATANH</td>
<td>atanh</td>
</tr>
<tr>
<td>X-Yj</td>
<td>atan(yj)</td>
</tr>
<tr>
<td>COS</td>
<td>cos</td>
</tr>
<tr>
<td>COSH</td>
<td>cosh</td>
</tr>
<tr>
<td>2^Z</td>
<td>exp(Z)</td>
</tr>
<tr>
<td>E^Z</td>
<td>exp(Z)</td>
</tr>
<tr>
<td>Z/2</td>
<td>z/2</td>
</tr>
<tr>
<td>1/Z</td>
<td>z^-1</td>
</tr>
<tr>
<td>Ln(Z)</td>
<td>ln(z)</td>
</tr>
<tr>
<td>Log(Z)</td>
<td>log(z)</td>
</tr>
<tr>
<td>-Z</td>
<td>z^-1</td>
</tr>
<tr>
<td>rounded Z</td>
<td>z = Re(z) + Im(z)</td>
</tr>
<tr>
<td>Sign(z)</td>
<td>sgn(z)</td>
</tr>
<tr>
<td>SIN</td>
<td>sin</td>
</tr>
<tr>
<td>SINO</td>
<td>sinh</td>
</tr>
<tr>
<td>Square Root</td>
<td>sqrt(z)</td>
</tr>
<tr>
<td>TAN</td>
<td>tan</td>
</tr>
<tr>
<td>TANH</td>
<td>tanh</td>
</tr>
<tr>
<td>Rec-&gt;Im</td>
<td>Re(z) + Im(z)</td>
</tr>
<tr>
<td>Zarg</td>
<td>zarg</td>
</tr>
<tr>
<td></td>
<td>z</td>
</tr>
<tr>
<td>Rectangular</td>
<td>yes</td>
</tr>
<tr>
<td>Polar Notation</td>
<td>yes</td>
</tr>
<tr>
<td>alternate ZEXP</td>
<td>yes</td>
</tr>
<tr>
<td>function</td>
<td>exp(Z)</td>
</tr>
<tr>
<td>hybrid power</td>
<td>yes</td>
</tr>
<tr>
<td>rotation</td>
<td>yes</td>
</tr>
<tr>
<td>Next ASIN</td>
<td>yes</td>
</tr>
<tr>
<td>Next ACOS</td>
<td>yes</td>
</tr>
<tr>
<td>Next ATAN</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>-------</td>
</tr>
<tr>
<td>67</td>
<td>NXTLOG</td>
</tr>
<tr>
<td>68</td>
<td>NXTNRT</td>
</tr>
<tr>
<td>69</td>
<td>ZAVIEW</td>
</tr>
<tr>
<td>70</td>
<td>CLZ</td>
</tr>
<tr>
<td>71</td>
<td>ZIMAG</td>
</tr>
<tr>
<td>72</td>
<td>ZREAL</td>
</tr>
<tr>
<td>73</td>
<td>CLZST</td>
</tr>
<tr>
<td>74</td>
<td>Z&lt;&gt;</td>
</tr>
<tr>
<td>75</td>
<td>Z&lt;&gt;W</td>
</tr>
<tr>
<td>76</td>
<td>Z&lt;&gt;R</td>
</tr>
<tr>
<td>77</td>
<td>Z&lt;&gt;S</td>
</tr>
<tr>
<td>78</td>
<td>LASTZ</td>
</tr>
<tr>
<td>79</td>
<td>ZR^</td>
</tr>
<tr>
<td>80</td>
<td>ZRCL</td>
</tr>
<tr>
<td>81</td>
<td>IMAGINE</td>
</tr>
<tr>
<td>82</td>
<td>ZENTER^</td>
</tr>
<tr>
<td>83</td>
<td>ZREAL^</td>
</tr>
<tr>
<td>84</td>
<td>ZIMAG^</td>
</tr>
<tr>
<td>85</td>
<td>ZRDN</td>
</tr>
<tr>
<td>86</td>
<td>ZREPL</td>
</tr>
<tr>
<td>87</td>
<td>ZSTO</td>
</tr>
</tbody>
</table>