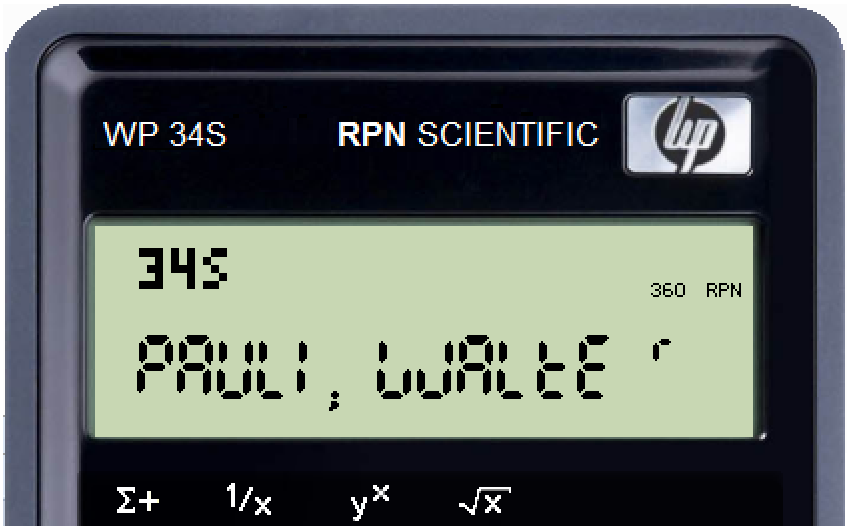
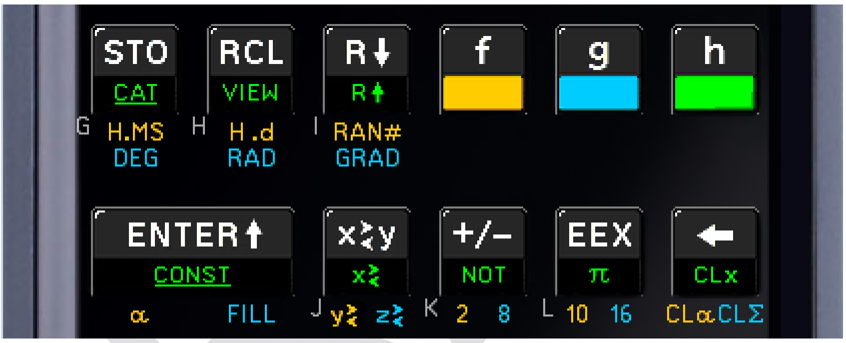


3.0 Work in Progress



First aid for those complaining about getting trapped in an unexpected or unwanted calculator mode while playing around before reading: H.d (i.e. f RCL) will bring you back to floating point mode.

For those who don't even read this: Sorry, we can't help you.

TABLE OF CONTENTS

JUST IN CASE …

... you still have your *HP-20b Business Consultant* or your *HP-30b Business Professional* sitting on your desk unchanged as produced by HP, please turn to *Appendix A* for some instructions how to convert it into a full fledge WP 34S yourself. Alternatively, if you don't want to bother with cables on your desk connecting it to your computer, with flashing the calculator firmware and attaching a sticky overlay, you may purchase e.g. a *HP-30b*-based WP 34S readily in the internet:

http://www.thecalculatorstore.com/epages/eb9376.sf/en_GB/?ObjectPath=/Shops/eb9376/Products/%22WP34s%20Pack%22
(We apologize for the small font – it allows this hyperlink fitting into one print line).

The first way may just cost your time, the second will cost you some money at the store. If you choose buying your WP 34S at the address mentioned, we (the developers) will get a modest fraction of the price. Both ways, however, are proven to work – it is your choice.

For the following, we assume the flashing is done and you hold a WP 34S in your hands.

WELCOME

Dear user, now you have got it: your own WP 34S. It uses the mechanics and hardware of the *HP-20b Business Consultant* or the new *HP-30b Business Professional*, so you benefit from their unexcelled processor speed. And with the *HP-30b* you get the famous rotate-and-click keys in addition, giving the tactile feedback appreciated in vintage *Hewlett-Packard* calculators for decades.

On the other hand, the firmware and user interface of the WP 34S were thoroughly thought through and discussed by us, newly designed and written from scratch, loaded with functions, pressed into the little memory provided, and tested over and over again to give you **a fast and compact programmable <u>scientific</u> calculator like you have never had before.**

The WP 34S function set is based on the famous *HP-42S RPN Scientific*, the most powerful programmable RPN calculator built so far [1]. We expanded this set, incorporating the functionality of the renowned programmer's calculator *HP-16C*, the fraction mode of the *HP-32SII*, probability distributions as featured by the *HP-21S*, and added **many more useful functions for mathematics, statistics, physics, engineering, programming etc.** like

- Euler's Beta function, Fibonacci numbers, Lambert's W (all of these in real and complex domains), the error function, incomplete regularized Beta and Gamma, Riemann's Zeta, the most 'popular' orthogonal polynomials, testing for primality,
- many statistical distributions and their inverses like Poisson, Binomial, Geometric as well as Cauchy-Lorentz, Exponential, Logistic, Weibull for reliability analysis, Lognormal and Gaussian with arbitrary means and standard deviations,
- programmable sums and products, first and second derivatives,
- extended date and time calculations based on a real time clock,

[1] Though the *HP-42S* was sold in 1988 already, this statement holds still. – Due to hardware restrictions, the matrix math of the *HP-42S* cannot be supported by the WP 34S. Matrices are covered, however, by a package of basic commands.

- integer computing in arbitrary bases from binary to hexadecimal,
- financial operations like mean rate of return and margin calculations,
- 80 conversions, mainly between universal SI and old Imperial units,
- 50 fundamental physical constants as precise as known today by national standards institutes like NIST or PTB, plus some more out of mathematics, astronomy, and surveying,
- complete Greek and extended Latin letter fonts covering many languages on this planet (upper and lower case in two font sizes each).

The WP 34S is the first RPN calculator overcoming the limits of a 4-level stack – forget worries about stack overflow in calculations. It features a choice of two stack sizes expanded by a complex LASTx register: traditional four stack levels for HP compatibility, eight levels for convenient calculations in complex domain, advanced real calculus, vector algebra, or for whatever application you have in your mind. You find a full set of commands for stack handling and navigation in either size.

Furthermore, your WP 34S features over 100 general purpose registers, 104 user flags, 506 program steps in working memory, more than 4000 in flash, a 31 byte alpha register for message generation, and 4 programmable hotkeys for your favorite functions or routines. And you may backup your work in battery-fail-safe memory.

Your WP 34S is the result of a long range collaboration of two individuals, an Australian and a German. We did this in our free time, so you may call it our hobby (though some people close to us found different names for this). From its very beginning, our project was discussed on the forum of the *Museum of HP Calculators* (www.hpmuseum.org), so we want to express our gratitude to all the international contributors there who taught us a lot and brought their ideas and support in several stages of our project. Special thanks go to Marcus von Cube (Germany) supporting us in bringing the WP 34S to life, starting with an emulator for v1.14, allowing widespread use and convenient testing. From v1.17 on, the software runs on the real hardware as well. A very useful assembler / disassembler is supplied by Neil Hamilton (Canada) since v1.18 and even a symbolic preprocessor was added with v2.1.

We baptized our baby WP 34S in honor of one of the most powerful LED pocket calculators, the *HP-34C* of 1979. The WP 34S is our humble approach – with the hardware given – to a future 43S we can only dream of becoming the successor of the *HP-42S* once. May the WP 34S help in convincing those having access to more resources than us: covering the market of serious scientific instruments is worthwhile.

Firmware-wise, we have carefully checked everything we could think of to our best knowledge, so our hope may be justified the WP 34S is free of bugs. Anyway, we promise we will continue improving the WP 34S whenever it turns out being necessary – so if you discover any strange result, please report it to us, and if it is revealed to be an internal error we will provide you with an update as soon as we have got one ourselves. We did show short response times so far, and we will continue this way.

Enjoy!

Paul Dale and Walter Bonin

PRINT CONVENTIONS

- Throughout this manual, standard font is Arial. Emphasis is added by underlining. *Specific terms, names or titles* are printed in italics, *hyperlinks* in blue underlined italics. Bold italic letters like ***n*** are used for variables. Calculator commands – e.g. ENTER – are generally called by their names, printed in capitals in running text for easy recognition. Each and every command featured is listed in the *Index of Operations* below.
- This CPX font is taken for explicit references to calculator keys.
- Register addresses are printed using **bold Times New Roman**, while lower case italic letters of this font are employed for register contents. So, for example, ***y*** lives in stack level **Y**, ***r45*** in general purpose register **R45**, and ***alpha*** in the alpha register, respectively. Overall stack contents are quoted in the order [***x***, ***y***, ***z***, …] generally.

All this holds unless stated otherwise explicitly.

GETTING STARTED

If you know how to deal with a good old HP RPN scientific calculator, you can start with your WP 34S right away. Use the following as a reference manual.

Else we recommend you get an *HP-42S Owner's Manual*. It is available at low cost on the DVD distributed by the *Museum of Hewlett-Packard Calculators* (*www.hpmuseum.org*). There are also other sources in the internet.

Please read Part 1 of said manual as a starter. This part includes an excellent introduction to RPN. This RPN is a very effective method making (,), [,], {, } and = keys obsolete in calculations. Once you got used to it you will most probably never employ a calculator featuring = again.

Part 2 of said manual will support you when you are heading for programming your WP 34S for easy handling of repeated or iterative computations. Further documentation, also about the other calculators mentioned above and in the following text, will add valuable information – it is all readily accessible on a single DVD from said source.

Most traditional commands on your WP 34S will work as they did on the *HP-42S*. This little manual here is meant as a supplement mainly showing you all the new features. It contains all the necessary information including some formulas and technical explanations but is not intended to replace textbooks about mathematics, statistics, physics, programming, or the like.

The following text starts presenting the keyboard as it will be active in various modes, so you know where to find what you are looking for. It continues explaining the memory, addressing items therein, the display and indicators used to give you feedback what is going on. Then the major part of this booklet is taken by the index of all the operations, catalog contents, constants and conversions featured. It closes with a list of messages the WP 34S will display if special conditions prevent it from executing your command as expected.

WHAT'S ON THE KEYBOARD AND HOW TO ACCESS IT?

Let us investigate your WP 34S in default state. Take off the battery cover, locate the little RESET hole between the batteries, and use a paper clip to reset. This will erase all user contents and give you a fresh start.

As usual, white labels execute the *default primary function* of the respective key.

There are further (secondary) functions provided for 34 keys. Their labels are printed next to the white ones in golden, blue, green or grey color.

Green labels are placed on the slanted faces of 34 keys. Golden and blue labels are printed below of the respective key on the *key plate* of the WP 34S. Grey letters are put bottom left of 26 keys.

Labels printed underlined open *catalogs*.

To access a golden, blue, or green label, use the *prefix* f, g, or h, respectively.

E.g. the key 5 preceded by

- f will calculate the arithmetic mean values of the data accumulated in the statistic registers via x̄ ,
- g will return the standard deviations for the same data via s ,
- h will open a catalog of supplementary statistic functions via STAT .
- The grey letter R will become relevant in *alpha mode*, e.g. for input of text.

These prefixes allow for easily accessing a multiple of the 37 primary functions the keyboard can take. You may keep the respective prefix pressed if you want to call several functions in sequence showing the same label color. Any numeric entry will just fill the display and is interpreted when completed, not earlier.

Time for a little example. Please take your WP 34S and press

ON (i.e. the bottom left key) to turn your calculator on. You will get

Unless specified otherwise, we shall quote the numeric results only in the following, i.e. 0. here.

Now let us assume you want to fence a little patch of land 40 yards long [2] and 30 yards wide. You have set the first corner post (A) already, and also the second (B) in a distance of 30 yards from A. Where do you place the third post (C) to be sure setting up the fence forming a proper rectangle? Simply enter:

4 **0**	40	
ENTER↑	40.	(this key is for separating two numbers in input here)
3 **0**	30	
→P	50.	(**→P** is reached by pressing **g** first, then **→**)

So, just take a 90 yards rope, nail its one end on post A and its other end on B, fetch the loose loop and walk 40 yards away. As soon as both parts of the rope are tightly stretched, stop and place post C there. You may set the fourth post the same way.

This method works for arbitrary rectangles. Your WP 34S does the calculation of $\sqrt{40^2 + 30^2}$ (or whatever lengths apply for you) automatically. You just care for the land, the rope, hammer and nails. And it will be up to you to set the posts!

As in this example, we will generally refer to shifted functions like **→P** by just printing the colored label in this text and omit the prefix key of corresponding color, since redundant.

By the way, by pressing **→P** the function →POL is called, converting rectangular to polar coordinates. Most labels printed on your WP 34S simply call operations carrying the same name as the respective label. There are, however, also a number of cases like **→P**. Thus, let us introduce them, starting top left on the keyboard:

[2] Though this manual is written for an international readership and we very well know the SI system of units agreed on internationally and adopted by almost all countries on this planet, we use Imperial units here making it easier for our US-American readers to follow. But see point 1 at the bottom of next page.

- A, B, C, and D are called *hotkeys*, since they immediately call the user programs carrying these labels if defined. If the respective labels are not (yet) defined, these keys act as Σ+, 1/x, y^x, or $\sqrt{x}$, respectively.
- HYP is the prefix for hyperbolic functions, as HYP^{-1} is for their inverses (see SINH, COSH, TANH, ASINH, ACOSH, and ATANH). In analogy, SIN^{-1} stands for ASIN, etc.
- ➡ is the prefix for five immediate conversions: ➡ trailed by H.MS, H.d, DEG, RAD, or GRAD will convert x , i.e. the value currently displayed. The respective function names all begin with →. Additionally, ➡ is employed for indirect addressing.
- R← calls →REC converting polar to rectangular coordinates in two dimensions. So the pair R←→P takes care of the two classic coordinate transformations.
- CPX is mainly employed as a prefix for calling complex operations. See the respective paragraph *below* for more.
- a b/c and d/c enter the fraction mode for proper and improper fractions, respectively (see PROFRC and IMPFRC).
- H.MS and H.d represent the two time modes, where H.d stands for decimal hours, but also for floating point numbers in general (see DECM).
- α enters *alpha mode*, while 2, 8, 10, or 16 will enter integer modes for calculating with binary, octal, decimal, or hexadecimal numbers (see BASE…).
- ! calls X! in default floating point mode.
- ./, toggles radix marks (see RDX**,** and RDX**.**), P/R programming mode, ⬆ upper and lower case in alpha mode, and |x| calls ABS.

These were all the special labels featured. You will find each and every command provided on your WP 34S below in the *index of operations* for your reference, together with the necessary individual explanation.

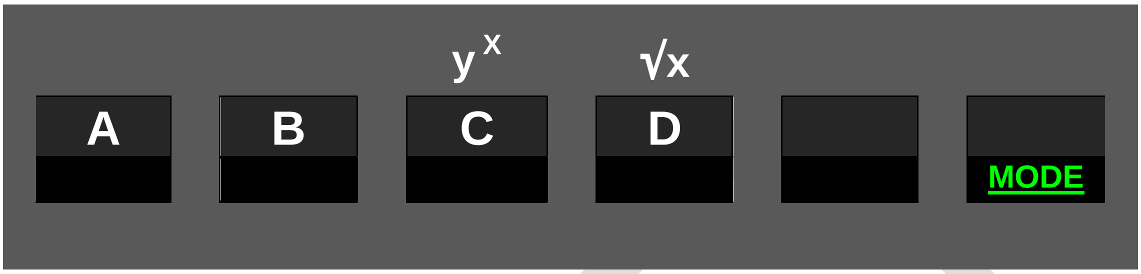
In four decades of pocket calculators, a wealth of nice to sophisticated application examples were invented and described by different authors – more and better than we can ever create ourselves. Also it is not our intention to copy these old examples. Instead, we recommend the DVD mentioned *above* once more: it contains all the user guides, handbooks, and manuals of vintage Hewlett Packard calculators. Be assured that almost everything described there for any scientific calculator can be done on your WP 34S significantly faster and sometimes even in a more elegant way.

Let us return to our introductory example for two remarks:

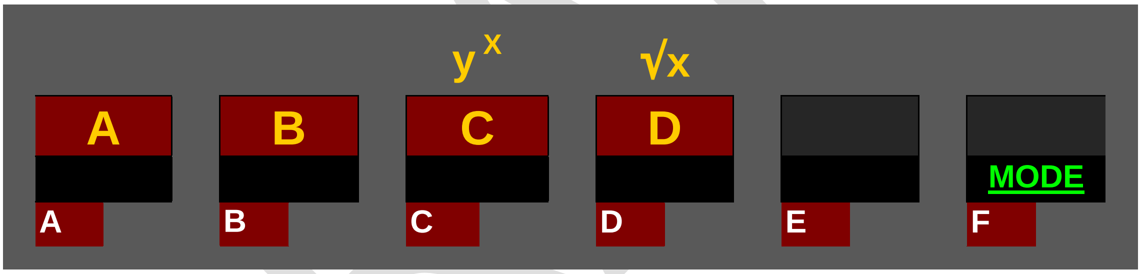
1. There is no need to enter any units. The example will work with meters as well, for example.
2. Although we entered *integer* numbers only for both sides of our little ground, the calculation was executed in default floating point mode of your WP 34S. This calculator mode allows for decimal fractions of e.g. feet in input and output as well. Another mode lets you key in proper fractions like e.g. 6 ¼.

Before you suffer from feet fractions, however, we want to briefly show you some additional modes your WP 34S features (you will find a complete list of all modes provided in a separate chapter further below).

Integer modes are meant to deal with integers only – in input, output, and calculations. This is useful for computer logic and similar applications. Your WP 34S allows for binary, ternary, etc. through hexadecimal computing. In these modes, operations like SIN don't make sense for obvious reasons. Thus, for integer bases up to ten, the top row of keys on your WP 34S will effectively work as shown here:



In hexadecimal integer mode, primary functions of these top keys will be switched automatically becoming numeric input, so f will be used for accessing their default primary functions:



The dark red background is used to highlight changed key functionality here. Prefix f will access the default primary functions wherever they aren't primary anymore after reassignment.

Calculating in bases 11 … 15, those keys not needed for numeric input will work as shown in the first picture above. In any integer base, attempts to enter an illegal digit – like e.g. 4 in binary – will be blocked.

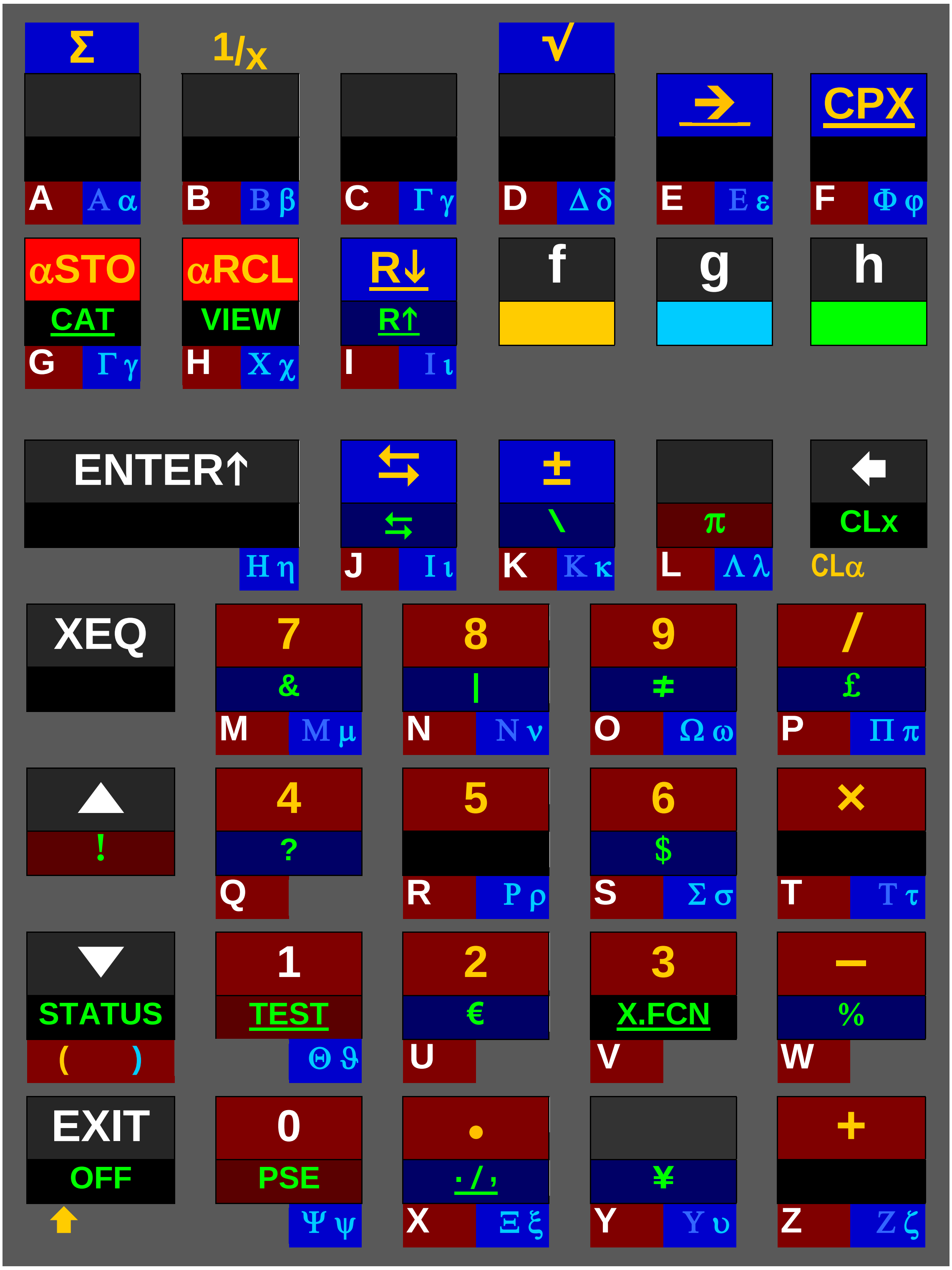
Alpha mode is designed for text entry, e.g. for prompts. In this mode, the alpha register is displayed in the upper part of the LCD, and the numeric line (kept from your last calculation) is accessible by commands only. The display may look like this:

In alpha mode, almost all the mathematical operations are neither needed nor applicable. So the keyboard is redefined automatically when you enter alpha mode, as shown overleaf.

All labels printed on dark red background here append characters to ***alpha*** immediately or via alpha catalogs. Primary function of most keys is appending the letter printed bottom left of this key – grey on the key plate. Alpha mode starts with capitals, and ⇧ toggles upper and lower case. PSE appends a space. As in integer modes, f will access default primary functions wherever necessary [3].

[3] The digits 0 and 1 may also be called using f 0 or f 1, respectively.

Looking at the standard labels on the keyboard, we can safely offer more:



All labels printed on dark blue background here append characters to ***alpha*** as well, but deviate from the labels printed on your WP 34S keyboard at these locations.

Prefix g leads to homonymic Greek letters where applicable [4]. And h gives access to logic symbols via the Boolean operations, to '!', to '?' at the letter Q, as well

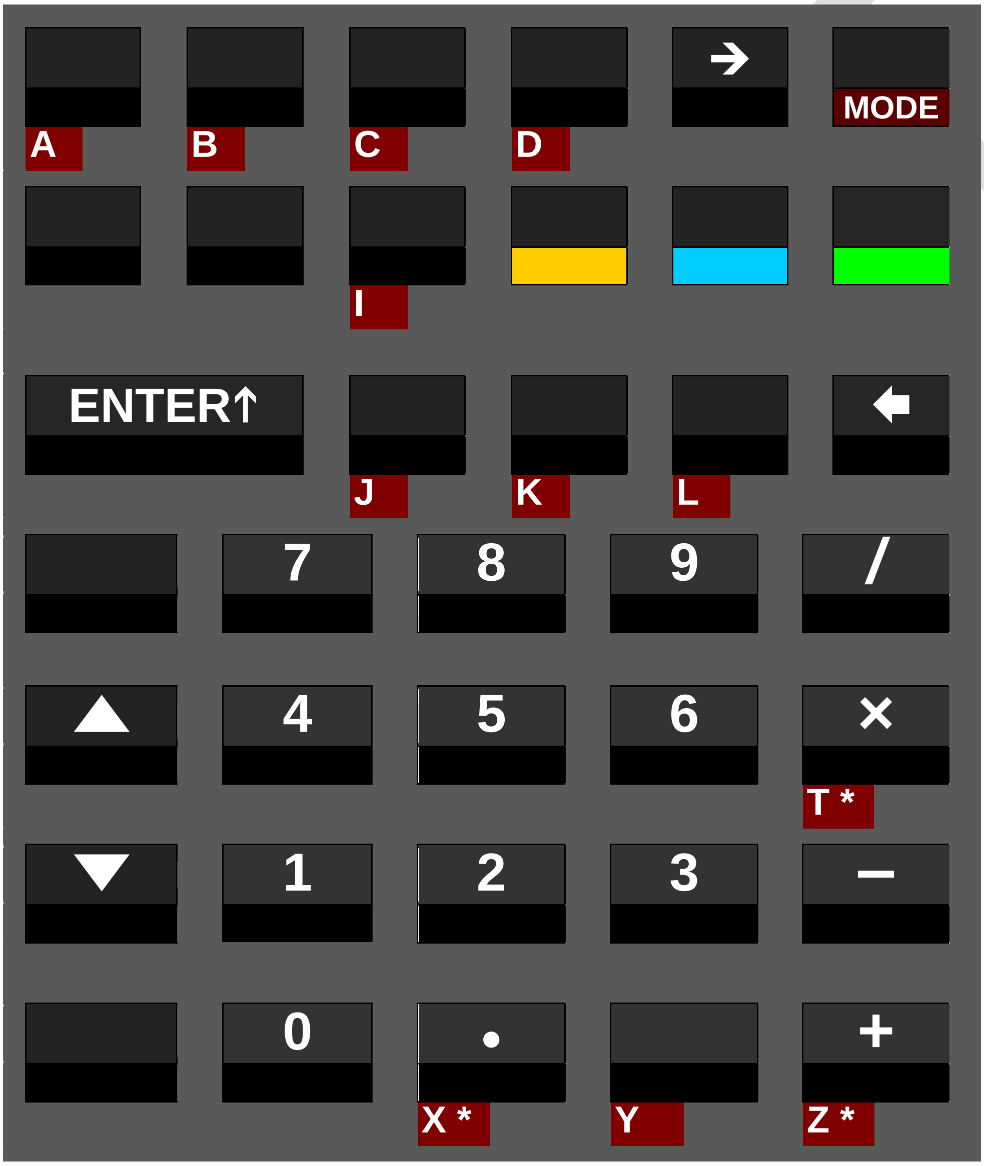
[4] "Homonymic" according to ancient Greek pronunciation. And we assigned **Gamma** also to **C** due to the alphabet, **Chi** to **H** since this letter comes next in pronunciation, and **Iota** also to **J**. Three Greek

as to four currency symbols located next to the %-command as follows: **$** at the letter **S** – **€** at **U** for Euro – **£** at **P** – **¥** at **Y** for Yen or Yuan – and **%** at −.

The catalogs accessible via f →, f CPX, f R↓, R↑, TEST, and ./, feature even more characters (see *below*). See the *index of operations* for αSTO, αRCL, and more alpha commands.

When ***alpha*** exceeds 31 characters, the leftmost character(s) are discarded.

A **temporary alpha mode** is entered during input processing in comparisons and in memory addressing, e.g. during storing. Examples are shown *below*. See the respective virtual keyboard here:



This mode is left automatically when sufficient characters are put in for the respective command.

Special rules apply for **T**, **X**, and **Z** – see *below*.

letters require special handling: **Psi** is accessed via g 0 (below PSE), **Theta** via g 1 (below TEST and following 'T'), and **Eta** via g ENTER↑. **Omicron** is not featured since looking exactly like the Latin letter 'O' in either case. – Where we printed Greek capitals with lower contrast, they look like the respective Latin letters in our fonts. Greek professors, we count on your understanding.

REAL AND INTEGER OPERATIONS

Most of the commands your WP 34S features are mathematical operations or functions in real domain. “Real domain” means these functions use real numbers like 1 or 2.34 or π or 5.6E-7, and work with them. Please note integer numbers like 8, 9, 10, or -1 are just a subset of real numbers.

Most real number functions provided operate on one number only – the number currently displayed. For example, key in

[.] [4] [9]   0.49

and press

[√x]   0.7   since $0.7^2 = 0.49$

Generally, such functions replace x (i.e. the number currently displayed) by the result **f(x)** , that’s all they do.

Some of the most popular mathematical functions, however, operate on two numbers. Think of + and – , for example. On your WP 34S, such a two-number real function replaces x by the result **f(x, y)** . Now the stack enters the game. Think of it like a pile of numbers. For subtracting two numbers, you need to know them first, then you can execute the subtraction. That’s the essence of RPN.

So having an account of 1,234 US$ and taking 56.7 US$ from it is solved as follows:

[1] [2] [3] [4]	1234	enter first number
[ENTER↑]	1234.	separates the two numbers in input as in the very first example above
[5] [6] [.] [7]	56.7	enter second number
[−]	1177.3	subtract it from the first

By [ENTER↑], the first number is “pushed on the stack” so a second separate number can be entered in sequence. The operation [−] takes its input from the lowest two stack levels **X** and **Y** but needs only **X** to put its result in. Knowing your WP 34S features more than only two stack levels, level **Y** is then filled with the content of the next higher level, i.e. z . This goes on for higher levels, as shown *below*. Please note the top stack level content is repeated then (since there is nothing else available for filling). You may use this top level repetition for some nice tricks.

There are also a few three-number real functions included – e.g. Iβ and %MRR – replacing x by the result **f(x, y, z)** . Then **Y** is filled with t and so on, and the content of the top level is repeated twice.

Some real functions (e.g. DECOMP) operate on one number but return two. Other operations (like RCL or SUM) do not consume any stack input at all but just return one or two numbers. Then these extra number(s) will be pushed on the stack, taking one level per real number.

STATISTICAL DISTRIBUTIONS, PROBABILITIES ETC.

You will find a lot of statistics in your WP 34S, going far beyond the Gaussian distribution. Many preprogrammed functions are implemented here for the first time in an RPN calculator – we packed all in what we always had missed. All of these functions have a few features in common:

- Discrete statistical distributions (e.g. Poisson, Binomial) are confined to integers. Whenever we sum up a probability mass function (*pmf*[5]) $p(n)$ to get a cumulated distribution function (*cdf*) $F(m)$ we start at $n = 0$. Thus,

$$F\ (m) = \sum_{n=0}^{m} p(n) = P(m)\ .$$

- Whenever we integrate a function, we start at the left end of the integration interval. Thus, integrating a continuous probability density function (*pdf*) $f(x)$ to get a *cdf* $F(x)$ typically works as

$$F(x) = \int_{-\infty}^{x} f(\xi)d\xi = P(x)\ .$$

- Typically, F starts with a very shallow slope, becomes steeper then, and runs out with a decreasing slope while slowly approaching 100%. Obviously you get the most precise results on the left side of the *cdf* using P . On its right side, however, the "error probability" $Q = 1 - P$ is more precise: since P comes very close to 100% there, you may see 1.0000 displayed while e.g. $P = 0.99996$ in reality.

- On your WP 34S, with an arbitrary *cdf* named ***XYZ*** you find the name ***XYZ*** $^{-1}$ for its inverse (the so-called *quantile function*) and ***XYZ*** $_P$ for the *pdf* or *pmf*, unless stated otherwise explicitly.

- For calculating confidence limits for the "true value" based on a sample evaluation, employing a particular confidence level (e.g. 95%), you must know your objective:
  - Do you want to know the upper limit, under which the "true value" will lie with a probability of 95%? Then take 0.95 as the argument of the *inverse cdf* to get said limit, and remember there is an inevitable chance of 100% – 95% = 5% for the "true value" being greater than it.

---

[5] In a nutshell, discrete statistical distributions deal with "events" governed by a known mathematical model. The *pmf* then tells the probability to observe a certain number of such events, e.g. 7. And the *cdf* tells the probability to observe up to 7 such events, but not more.

For doing statistics with continuous statistical variables – e.g. the heights of three-year-old toddlers – similar rules apply: Assume we know the applicable mathematical model. Then the respective *cdf* tells the probability for their heights being less than an arbitrary limit value, for example less than 1m. And the corresponding *pdf* tells how these heights are distributed in a sample of let's say 1000 children of this age.

WARNING: This is a very coarse sketch of this topic only – please turn to textbooks about statistics to learn dealing with it properly.

The terms *pmf* and *pdf* translate to German „Dichtefunktion" or „Wahrscheinlichkeitsdichte", *cdf* to „Verteilungsfunktion" or „Wahrscheinlichkeitsverteilung".

- Do you want an upper <u>and</u> a lower limit confining the "true value"? Then there is an inevitable chance of 5% / 2 = 2.5% for said value being less than the lower limit and an equal chance for it being greater than the upper limit. So you shall use 0.025 and 0.975 as arguments in two subsequent calculations using the *inverse cdf* to get both limits.

We strongly recommend you turn to a good statistics textbook for more information, also about the terminology used and the particular distributions provided.

MATRICES

Numbers arranged in a flat grid like in a table are called matrices by the mathematicians. If you do not know matrices, feel free to leave them aside – you can use your WP 34S perfectly without them.

Else please note your WP 34S features a set of operations for adding, multiplying, inverting and transposing matrices, as well as for manipulating rows in such matrices. In general, the respective commands are building blocks designed to provide the low level support routines for creating more useful matrix functions as keystroke programs. I.e. they represent the basic linear algebra subprograms of the WP 34S matrix support. There are, however, also functions featured for computing determinants as well as for solving systems of linear equations.

A matrix is represented within your WP 34S by its *descriptor*, formatted `bb.rrcc` with

> ***rr*** being the number of its rows and
>
> ***cc*** the number of its columns. Thus the matrix has ***rr*** × ***cc*** elements.
>
> These elements are stored in consecutive registers starting at base address |***bb***| . See *below* to learn about the registers of your WP 34S.

Example: A descriptor 7.0203 represents a 2×3 matrix – let us call it (*M*). As you know, its six elements are arranged in two rows and three columns, and are numbered as follows:

$$(M) = \begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \end{pmatrix}$$

The matrix descriptor tells us now where to find the values of these elements:

> $\boldsymbol{m_{11} = r07}$, $\boldsymbol{m_{12} = r08}$, $\boldsymbol{m_{13} = r09}$, $\boldsymbol{m_{21} = r10}$, $\boldsymbol{m_{22} = r11}$, and $\boldsymbol{m_{23} = r12}$.

If ***cc*** is omitted in a descriptor, it is set to ***rr*** so a square matrix is assumed. The maximum number of matrix elements is 100 – it is the number of general purpose registers available. A vector descriptor looks like `bb.01cc` or `bb.rr01` .

COMPLEX OPERATIONS

Mathematicians know more complicated items than real numbers. The next step are complex numbers. If you do not know them, leave them aside – you can use your WP 34S perfectly without them.

Else please note your WP 34S supports many operations in complex domain as well. The key CPX is employed as a prefix for calling complex functions. E.g. CPX f COS calls the complex cosine, and it is displayed and listed as ^{C}COS (the elevated C is the signature for complex functions on your WP 34S). All such functions operating on complex numbers do so in Cartesian coordinates exclusively. Each complex number occupies two adjacent registers: the lower one for its real part and the higher one for its imaginary part.

Generally, if an arbitrary real function **f** operates on …

- … one real number x only, then its complex sibling $^{C}\mathbf{f}$ will operate on the complex number $x_c = x + i\,y$.
- … one register, e.g. **R12**, then $^{C}\mathbf{f}$ will operate on **R12** and **R13**.
- … x and y, then $^{C}\mathbf{f}$ will operate on x, y, z and t .

Where one-number real functions replace x by the result $\mathbf{f}(x)$, one-argument complex functions replace x by the real part and y by the imaginary part of the complex result $^{C}\mathbf{f}(x_c)$. Higher stack levels remain unchanged. Such functions are ^{C}1/x, ^{C}ABS, ^{C}ANGLE, ^{C}CUBE, ^{C}CUBERT, ^{C}FIB, ^{C}FP, ^{C}IP, ^{C}RND, ^{C}SIGN, ^{C}W, $^{C}W^{-1}$, ^{C}x!, $^{C}x^{2}$, $^{C}\sqrt{\ }$, ^{C}**+/–**, $^{C}\Gamma$(x), the logarithmic and exponential functions with bases 10, 2 and **e**, as well as hyperbolic, trigonometric, and their inverses.

Two-number real functions replace x by the result $\mathbf{f}(x, y)$. Analogously, two-argument complex functions replace x by the real part and y by the imaginary part of the complex result $^{C}\mathbf{f}(x_c, y_c)$. The next stack levels are filled with the complex contents of higher levels, and the complex number contained in the top two stack levels is repeated as shown *below*. Such complex functions are $^{C}LOG_{X}$, $^{C}y^{x}$, $^{C}\beta$(x,y), ^{C}// , and the basic arithmetic operations in complex domain.

Where complex operations (like ^{C}RCL) do not consume any stack input at all but just return a complex number, this will be pushed on the stack taking two levels.

MEMORY

Stack registers
D *
C *
B *
A *
T
Z
Y
X

Mode	
Alpha (31 bytes)	
Display	
L	I **

General purpose registers
R00
R01
R02
…
…
R95
R96
R97
R98
R99
R.00 = R112
R.01 = R113
…
…
R.15 = R127
R128
…
…
R210
R211

Program steps
000
001
002
…
…
508
509
510

X = R100
Y = R101
Z = R102
T = R103
A = R104
B = R105
C = R106
D = R107
L = R108
I = R109
J = R110
K = R111

User flags
00
01
02
…
…
95
96
97
98
99
X = 100
Y = 101
Z = 102
T = 103
A = 104
B *Big, overflow*
C *Carry*
D *Danger*
L = 108
I = 109
J = 110
K = 111

For the first time ever in a calculator, your WP 34S offers a choice of 4 or 8 stack levels. So either **T** or **D** will be the top level. Registers **A** - **D** will be allocated as stack registers if required.

Please see *below* for top level repetition and stack contents in complex calculations. While register **L** takes the real part of the last argument, **I** takes the imaginary part when a complex function was executed (see ^{C}LASTx).

Registers **J** and **K** may carry parameters of statistical distributions.

Unless required for the purposes mentioned, **A, B, C, D, I, J,** and **K** are available as additional g. p. registers. Direct numeric addressing works up to number 99. For *indirect addressing*, the stack levels etc. carry the numbers 100 … 211 as shown at right. **R112** – **R211** may be used for *local registers* (see *below*).

For information about the flags, please turn overleaf.

Flags 00 … 99 are free to use for whatever purpose you like. Further flags are addressed using letters as shown on previous page. They may be used the same way like the other flags. Flags A, B, C and D, however, are also checked by the system. Flag A lights the big '=' symbol in display. In integer modes, flags B and C will be set by the system in analogy to the overflow and carry bits of the *HP-16C*. Some integer operations (like shift and rotate) also read flag C. Flag D may be set by the user to allow special results (infinities and non-numeric results) without getting an error. The system only reads D.

For *indirect addressing*, flags X … K carry the numbers 100 … 111 as shown on previous page.

Please note you will not get 510 program steps and 211 registers and 112 flags all together at the same time – see *Appendix B* for details of memory management.

In addition to the RAM provided, your WP 34S allows you to access **flash memory** for voltage-fail safe storage of user programs and data. Flash memory features up to ten segments (regions, banks) of 1 kB each. Segment 0 is the backup region, holding the image of the entire program memory, registers and calculator state as soon as you completed a SAVE. The other segments (1, 2, 3, etc.) hold programs only. Alphanumeric labels (see below) in flash can be called via XEQ like in RAM. This allows creating program libraries in flash. Use CAT to see the labels defined already.

Flash memory is ideal for backups or other long-living data, but shall not be used for repeated transient storage like in programmed loops (since it will not survive more than some 10,000 flashes). Registers and standard user program memory, residing in RAM on the opposite, are designed for frequent data changes but will not hold data with the batteries removed. So both kinds of memory have specific advantages and disadvantages you shall take into account for optimum benefit and long lasting joy with your WP 34S. Find more about flash memory in *Appendix A* below.

Furthermore, there is a memory section called **XROM** (for "extended ROM"), where some additional routines live. Though written in user code, these are read only and thus can be called, executed, but not edited. For you, it makes no difference whether a preprogrammed routine executes in ROM or XROM.

Structuring program memory and jumping around in it is eased by **labels** you may tag to any program steps – as known from previous programmable pocket calculators. Your WP 34S features a full set of alphanumeric labels as described *below*.

When a command like e.g. GTO ***xy*** is encountered, with ***xy*** representing one, two or three characters (like A, BC, 12, Tst, Pg3, x1µ, etc.), your WP 34S will search this label ***xy*** using the following method:

1. If ***xy*** is purely numeric, it will be searched forward from the current position of the program pointer. When the end of the program space is reached without finding ***xy***, the quest will continue at the start of the current segment. No other segments will be searched. This is as known from vintage HP calculators.
2. Else, i.e. if ***xy*** is an alpha label of up to three characters of arbitrary case, searching will start at program step 000 and cover the entire memory in the order RAM, flash segments from their greatest number down, and XROM, independent of the position of the program pointer.

STACK MECHANICS

The following assumes you are familiar with RPN – else please turn to the *HP-42S Owner's Manual* first.

The fate of particular stack register contents depends on the operation executed, its domain (real or complex) and the stack size chosen. Real functions in a 4-level stack work as known for decades. In a larger stack, everything works alike on your WP 34S – just with more levels for intermediate results. Please note only the contents of **X** are displayed in any case. See below for details of the stack mechanics:

	Level	Assumed stack contents at the beginning:	Stack contents after executing … … the real stack register operations ENTER	FILL	DROP	x⇆y	R↓	R↑	LASTx	… real functions of … one number like x^2	… two numbers like /
With **4** stack levels	T	*t*	*z*	*x*	*t*	*t*	*x*	*z*	*z*	*t*	*t*
	Z	*z*	*y*	*x*	*t*	*z*	*t*	*y*	*y*	*z*	*t*
	Y	*y*	*x*	*x*	*z*	*x*	*z*	*x*	*x*	*y*	*z*
	X	*x*	*x*	*x*	*y*	*y*	*y*	*t*	*last x*	x^2	*y / x*
With **8** stack levels	D	*d*	*c*	*x*	*d*	*d*	*x*	*c*	*c*	*d*	*d*
	C	*c*	*b*	*x*	*d*	*c*	*d*	*b*	*b*	*c*	*d*
	B	*b*	*a*	*x*	*c*	*b*	*c*	*a*	*a*	*b*	*c*
	A	*a*	*t*	*x*	*b*	*a*	*b*	*t*	*t*	*a*	*b*
	T	*t*	*z*	*x*	*a*	*t*	*a*	*z*	*z*	*t*	*a*
	Z	*z*	*y*	*x*	*t*	*z*	*t*	*y*	*y*	*z*	*t*
	Y	*y*	*x*	*x*	*z*	*x*	*z*	*x*	*x*	*y*	*z*
	X	*x*	*x*	*x*	*y*	*y*	*y*	*d*	*last x*	x^2	*y / x*

Calculating formulas from inside out stays a wise strategy in either stack. With more levels, however, stack overflow will hardly ever happen, even with the most advanced formulas you compute in your life as a scientist or engineer.

Calculating with complex numbers uses two registers or stack levels for each such number as explained above and shown here:

	Level	Assumed stack contents at the beginning:	Stack contents after executing … … the complex stack register operations ^{C}ENTER	^{C}FILL	^{C}DROP	^{C}x⇆y	^{C}R↓	^{C}R↑	^{C}LASTx	… complex functions of … one number like $^{C}x^2$	… two numbers like $^{C}/$
With **4** stack levels	T	**Im(y_c) = Im(t_c)**	**Im(x_c)**		$y_c = t_c$	**Im(x_c)**	x_c		x_c	$y_c = t_c$	$y_c = t_c$
	Z	**Re(y_c) = Re(t_c)**	**Re(x_c)**			**Re(x_c)**					
	Y	**Im(x_c)**	**Im(x_c)**		y_c	**Im(y_c)**	y_c		*last* x_c	**Im($(x_c)^2$)**	**Im(y_c / x_c)**
	X	**Re(x_c)**	**Re(x_c)**			**Re(y_c)**				**Re($(x_c)^2$)**	**Re(y_c / x_c)**
With **8** stack levels	D	**Im(t_c)**	z_c	x_c	t_c	t_c	x_c	z_c	z_c	t_c	t_c
	C	**Re(t_c)**									
	B	**Im(z_c)**	y_c	x_c	t_c	z_c	t_c	y_c	y_c	z_c	t_c
	A	**Re(z_c)**									
	T	**Im(y_c)**	x_c	x_c	z_c	x_c	z_c	x_c	x_c	y_c	z_c
	Z	**Re(y_c)**									
	Y	**Im(x_c)**	x_c	x_c	y_c	y_c	y_c	t_c	*last* x_c	$(x_c)^2$	y_c / x_c
	X	**Re(x_c)**									

So, an 8-level stack gives you the same flexibility in complex domain you are used to with a 4-level stack in real domain.

PROGRAMMING

The WP 34S is a keystroke-programmable calculator. If this statement makes you smile with delight, this paragraph is for you. Else please turn to the *HP-42S Owner's Manual* for an introduction into keystroke-programming – then continue reading here.

The basic building blocks within program memory are routines (or programs). Generally, each of them starts with a label (see *above*) and ends with a return statement. In between, you may store any sequence of instructions (or commands) for repeated use. Choose any operation featured – only very few commands are not programmable. The operations stored in your program may use each and every register provided – there are (almost) no limits. You are the sole and undisputed master of the memory!

This freedom, however, has a prize: you must take care that your programs do not interfere in their quest for data storage space. So it is good practice keeping a list of the registers used by a particular program, and documenting the purposes or contents of those for later reference.

If – after some time – you have a number of different programs stored, keeping track of their memory requests may become a challenge. Most of nowadays programming languages take care of this problem by declaring *local variables*, i.e. memory space allocated from general data memory and accessible for the current routine only – when the routine is finished, the respective memory is released again. We cannot feature variables on the WP 34S, but only registers – so we offer you *local registers* allocated for your routine. They are taken from the heap of general purpose registers (below of the statistics registers, i.e. from **R85** on decrementing) at the very moment you request them.

Example: Let us assume you write a program labeled 'P1':

1. You just enter the command LOCR in your program specifying the number of local registers you need (e.g. 5),
2. then you may access these registers most easily using local numbers 00 … 04 throughout *P1*.

Now, if you call another routine *P2* from *P1*, also *P2* may contain a step LOCR requesting one or more local registers. They will then carry local register numbers 00 etc. again, but local register 00 of *P2* will be different from local register 00 of *P1*, so no interference will happen. As soon as the return statement is executed, the local registers of the corresponding routine are released and given back to the heap mentioned above.

This construction allows e.g. for recursive programs, since every time such a program is called again it will get a new set of local registers being different from the ones it got before. Nevertheless, since you remain the sole and undisputed master of the memory, you may also employ registers identified by their traditional absolute number – and this usage may overlap with the registers requested via LOCR. So proper programming and care-taking persist being your job.

See *Appendix B* and the commands LOCR, LOCR?, MEM?, and POPLR in the *index of operations* below for more information.

COMPARING AND ADDRESSING REAL NUMBERS

1	User input	x= ?, x≠ ?, x< ?, x≤ ?, x≈ ?, x≥ ?, or x> ?				RCL, STO, RCLM, STOM, RCLS, STOS, αRCL, αSTO, VIEW, VWα+, x≷, DSE, ISG, DSZ, ISZ, FIX, SCI, ENG, DISP, BASE, KEY?, bit or flag commands, etc.		
	Dot matrix display	**OP _** (with temporary alpha mode set), e.g. x> _				**OP _** (with temporary alpha mode set), e.g. RCL _ [6]		
2	User input	0 or 1	*Stack level or named reg.* Y, Z, ...	ENTER↑ [7] leaves temp. alpha mode.	→ opens indirect addressing.	*Stack level or named register* X, Y, Z, .. , K [8]	***Number** of register or flag or bit(s) or decimals* [9]	→ [10] opens indirect addressing.
	Dot matrix display	**OP *n*** e.g. x≤ 0 ?	**OP *x*** e.g. x≥ y ?	**OP r_**	**OP →_**	**OP *x*** e.g. SCI Z	**OP *nn*** e.g. SF 15	**OP →_**
3	User input	Compares ***x*** with the number **0**.	Compares ***x*** with the number on stack level **Y**.	*Register no.* 00 … 99	Look right for more about indirect addressing.	Sets scientific display with the number of decimals specified in stack level **Z**.	*Stack level etc.* X, Y, Z, ... , K	*Register number* 00 … 99
	Dot matrix display			**OP r *nn*** e.g. x≠ r23?			**OP → *x*** e.g. VIEW →L	**OP → *nn*** e.g. STO →45
				Compares ***x*** with the number stored in **R23**.			Shows the content of the register where **L** is pointing to.	Stores ***x*** into the location where **R45** is pointing to.

---

[6] For RCL and STO, any of +, −, ×, /, ▲, or ▼ may precede step 2, except in RCLM and STOM. VIEW ENTER↑ calls αVIEW, And ENG ENTER↑ calls ENGOVR, SCI ENTER↑ calls SCIOVR. See the index of operations.

[7] You may skip this for register numbers >19.

[8] Exceptions: RCL T, RCL× T, RCL X, RCL Z, RCL+ Z require an ENTER↑, e.g. RCL + ENTER↑ Z for the latter. This holds for STO as well.

[9] Legal register numbers are 00 … 211 (00 … 99 may be specified directly). Valid flag numbers are 00 … 111, with the twelve top flags directly addressed via X … K . Legal numbers of decimals are 0 … 11, accepted integer bases 2 … 16, bit numbers 0 to 63, and integer word size up to 64 bits. For numbers <10, you may key in e.g. 5 ENTER↑ instead of 0 5 . – Please take into account some registers may be allocated to special applications.

[10] Works for all commands taking a parameter or argument except DEL$_P$.

COMPARING AND ADDRESSING COMPLEX NUMBERS

1 User input	CPX x= ? or x≠ ?				CPX RCL, STO, or x⪋		
Dot matrix display	OP _ (with temporary alpha mode set) e.g. ▪x= _				OP _ (with temporary alpha mode set) e.g. ▪RCL _ [11]		
2 User input	0 or 1	*Stack level or named register* X, Z, A, C, L, or J	ENTER↑ [12] leaves temp. alpha mode	→ opens indirect addressing.	*Stack level or named register* Z [13], A, C, L, or J	*Register number* 0 0 .. 9 8 [14]	→ opens indirect addressing.
Dot matrix display	**OP *n*** e.g. ▪x= 0 ?	**OP *x*** e.g. ▪x≠ z ?	**OP r_**	**OP →_**	**OP *x*** e.g. ▪RCL L	**OP *nn*** e.g. ▪STO 18	**OP →_**
3 User input	Compares ***x*** + *i* ***y*** with the real number **0**.	Compares ***x*** + *i* ***y*** with ***z*** + *i* ***t***.	*Register number* 0 0 … 9 8	Look right for more about indirect addressing.	This is ᶜLASTx.	*Stack level or named register* X, Y, … , K	*Register number* 0 0 … 9 9
Dot matrix display			**OP r *nn*** e.g. ▪x≠ r26?			**OP → *x*** e.g. ▪x<> →Z	**OP → *nn*** e.g. ▪STO →45
			Compares ***x*** + *i* ***y*** with ***r26*** + *i* ***r27***.			Swaps ***x*** with the contents of the register where **Z** is pointing to, and ***y*** with the contents of the next one.	Stores ***x*** + *i* ***y*** into 2 consecutive registers, starting with the one where **R45** is pointing to.

[11] For RCL and STO, any of +, −, ×, or / may precede step 2. See the index of operations.

[12] You may skip this keystroke for register numbers >19.

[13] Exceptions: ᶜRCL Z, ᶜRCL + Z, ᶜSTO Z, and ᶜSTO + Z require an ENTER↑ preceding Z, e.g. CPX STO + ENTER↑ Z for the latter.

[14] You may key in e.g. 8 ENTER↑ instead of 0 8 . Take care of pairs, since a complex operation will always affect two registers: the one specified and the one following this. We strongly recommend storing complex numbers with their real parts at even register numbers. – Please take into account some registers may be allocated to special applications.

ADDRESSING LABELS

1	User input Dot matrix display	A, B, C, or D **XEQ *label*** e.g. `XEQ C`	XEQ, GTO, LBL, LBL?, SLV, ∫, π, Σ, αGTO or αXEQ **OP _** e.g. `GTO _`				
2	User input Dot matrix display	Calls the function labeled **C**.	A, B, C, or D **OP *label*** e.g. `Σ B`	ENTER↑ sets alpha mode. **OP '_**	→ [15] opens indirect addressing and sets temporary alpha mode. **OP →_**		*2-digit numeric label* 0 0 … 9 9 **OP *nn*** e.g. `LBL 07`
3	User input Dot matrix display		Sums up the function labeled **B**.	***Alphanumeric label*** (1 … 3 characters [16]) **OP '*label*'** e.g. `SLV'F1µ'`	*Stack level or named register* X, Y, Z, … , K **OP → *x*** e.g. `∫ →T`	*Register number* 0 0 … 9 9 [17] **OP → *nn*** e.g. `XEQ →44`	
				Solves the function **F1µ** (with **F1µ** keyed in as explained in footer).	Integrates the function whose label is on stack level **T**.	Executes the routine whose label is in **R44**.	

Additionally, see *above* for the way your WP 34S searches labels, and look up GTO in the *index of operations* for two special cases applying to this command exclusively.

---

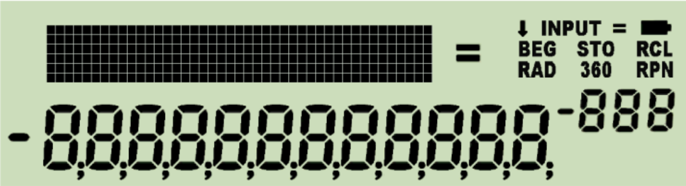
[15] Works with all these operations except LBL.

[16] The 3rd character terminates entry and closes alpha mode again – shorter labels need a closing ENTER↑. For the example given here you just press f 2 ENTER↑ CPX 1 f EXIT g 7 and you are done. Statements including alpha labels decrement the number of free program steps by 2. – **WARNING:** LBL A and LBL'A' are different animals! The latter is entered in alpha mode, the first via the hotkey directly.

[17] Some registers may be allocated to special applications. Please check the memory table above.

<u>**DISPLAY AND MODES**</u>

The display features three sections: numeric, dot matrix and fixed symbols. The numeric section features a minus sign and 12 digits for the mantissa, as well as a minus sign and 3 digits for the exponent. The dot matrix is 6 dots high and 43 dots wide, allowing for some 7 to 12 characters, depending on their widths. The fixed symbols (except the big "=") are called *annunciators*, and are for indicating modes.



The dot matrix section above is used for

1. indicating some more modes than the annunciators allow,
2. passing additional information to the user.

The numeric section in the lower part of the LCD is used for displaying numbers in different formats, for status, or messages.

If two or more requests concur for display space, the items will be shown according to their priorities as follows:

1. error messages as described in a *paragraph further below*,
2. special information as explained below,
3. information about the modes the calculator is running in.

The *annunciators* or specific characters in the LCD signal the modes:

Integer base or mode name	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	DECM
Signaled by … in the exponent	**b**	**3**	**4**	**5**	**6**	**7**	**o**	**9**	**d**	**-1**	**-2**	**-3**	**-4**	**-5**	**h**	
Set by …	2	BASE3, … , BASE7, 8, BASE9, 10, … , BASE15													16	.d
Cleared by …	any other BASE setting, FRACT, a b/c, d/c . ALL, FIX, SCI, ENG, and TIME will set DECM															

Mode name	PRG	α				FRC
Signaled by …	*STO*	*INPUT*	*360*	*RAD*	G	
Set by …	P/R	α αON	DEG	RAD	GRAD	d/c, a b/c 2^{nd} . in input BASE1, FRACT
Cleared by …	P/R EXIT	ENTER αOFF EXIT	GRAD RAD	DEG GRAD	DEG RAD	BASE ≠ 1 H.MS, TIME, → H.MS ALL, FIX, SCI, ENG

BEG indicates the program pointer standing at step 000 of program memory. A running program is signaled by ***RCL*** flashing. ***RPN*** is lit permanently unless a temporary message is shown. Time modes (12h / 24h) are seen in the time string directly. The numeric format of fraction mode is unambiguous as well. Further settings are signaled in the dot matrix section, like the different date modes being indicated there by **Y.MD** or **M.DY**. Defaults D.MY and DECM are not indicated. Please check the examples below.

Some mode and display settings may be stored and recalled collectively by STOM and RCLM. These are stack depth and contrast set, complete decimal display settings, trig mode, choices for date and time display, the parameters of integer and fraction mode, curve fitting model and rounding mode selected. STOM stores this information in the register you specify. RCLM recalls the contents of such a register and sets the calculator modes accordingly. Please note the user is responsible for recalling valid mode data – else your WP 34S may be driven into a lockup state! See the *index of operations* for more information about changing modes and the individual commands employed.

Some regional combinations may be set at once using a single command:

- SETCHN sets 24h, Y.MD, decimal point, and E3OFF;
- SETEUR sets 24h, D.MY, decimal comma, E3ON, and JG1582 (these settings apply also to South America);
- SETIND sets 24h, D.MY, decimal point, E3OFF, and JG1752;
- SETUK sets 12h, D.MY, decimal point, E3ON, and JG1752.
- SETUSA sets 12h, M.DY, decimal point, E3ON, and JG1752;

Please note the people living in the area of the former Soviet Union, in South Africa, Indonesia, and Vietnam use the decimal comma as well, but have different settings for dates and times.

Especially the angular modes deserve a closer look: there are three of them, DEG, RAD, and GRAD. And degrees (DEG) may be displayed in decimal numbers as well as in hours, minutes, seconds and hundredth of seconds (H.MS). Conversions are provided for going from one to the other:

From to	degrees H.MS	decimal degrees	radians	gon (grad)	current angular mode
degrees H.MS	—	→H.MS	—	—	—
decimal degrees	→H .d	—	rad→°	G→°	→DEG
radians	—	°→rad	—	G→rad	→RAD
gon/grad	—	°→G	rad→G	—	→GRAD
current angular mode	—	DEG→	RAD→	GRAD→	—

Please see the *index of operations* for the commands printed on white background, and the *catalog of unit conversions* for those printed on yellow.

Some commands and modes use the display in a special way. They are listed below in order of falling priority:

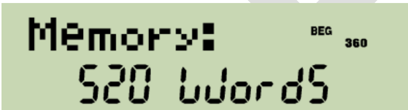
1. **VERS** generates a display similar to the one shown on the title page of this manual. This temporary message will vanish with the next key pressed. ⬅ will just clear the message, any other key will be executed.

2. **SHOW** displays the full mantissa of x, i.e. all sixteen digits present internally. E.g. π SHOW returns

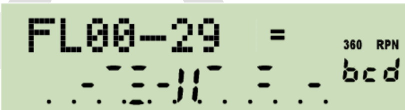
   as a temporary message.

3. **STATUS** shows the amount of free memory first, e.g.:

   ,

   then – after ▼ – the status of 30 user flags. They are shown very concisely in one display, allowing an immediate status overview after some training. If e.g. flags 2, 3, 5, 7, 11, 13, 14, 17, 19, and 23 are set, and labels B, C, and D are defined in program memory, STATUS ▼ will display this:

   Within the numeric section, each row of horizontal bars in the mantissa shows the status of 10 flags. When a flag is set, the respective bar turns black. So here the top row of bars indicates flags 0 and 1 are clear, 2 and 3 set, and flag 4 clear. Then, the divider II separates the first group of five flags from the next. Top row bars on its right side indicate flags 5 and 7 are set. Next row of bars shows flags 11, 13, 14, 17, 19 are set, and in the lowest row only flag 23 is set. All other flags in the range from 10 to 29 are clear.

   Scrolling down by ▼ will display flags 10 - 39, then 20 - 49 etc. until 70 - 99, 80 - 99, 90 - 99, and X:T A:D L:K. Scrolling up by ▲ reverts this. Alternatively, pressing a digit, e.g. 5, will show up to 30 flags starting with 10 times this digit, e.g. flags 50 - 79. The numeric exponent always indicates the status of the hotkeys top left on the keyboard – if all four labels are used in program memory then **ALL** will be shown there.

   The status will be displayed until any key is pressed but ▼, ▲, or a digit.

4. During **command input**, the dot matrix displays the command chosen until input is completed, i.e. until all required trailing parameters are entered. The prefixes f, g,

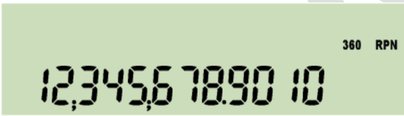
and h are shown until they are resolved. If you pressed any of f, g, or h erroneously, recovery is as easy as follows:

- f f = NOP = g g = h h = CPX CPX = → →
- g f = h f = f
  f g = h g = g
  f h = g h = h

In addressing, progress is recorded as explained in the *tables above* in detail. You may cancel such pending operations by EXIT as described *below*.

5. In **programming mode**, the numeric display indicates the program step (000 – 505) in the mantissa and the number of free steps in the exponent, while the dot matrix shows the command contained in the respective step, e.g.:

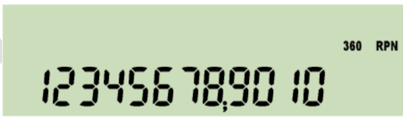
6. For **floating point decimal numbers**, the mantissa will be displayed adjusted to the right, the exponent to the left. Within the mantissa, either points or commas may be selected as radix marks [18], and additional marks may be chosen to separate thousands. Assume the display set to FIX 4, then 12.345678901 millions may look like:



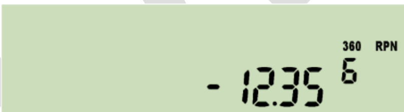
or

with thousands separators on, and without them like:

or



These separators may also be beneficial in integer or fraction modes described below. – With ENG 3 and after changing the sign, the same number will look like this:



or

If the last operation executed was a complex one, a capital **C** is displayed top left in the dot matrix pointing to the fact that you find the result of this function in **X** and **Y**.

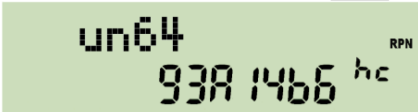
Floating point decimal numbers within $10^{-383} < |x| < 10^{+385}$ may be entered easily. Using a decimal mantissa, even numbers down to 10^{-394} can be keyed in. The calculator works with numbers down to 10^{-398} correctly. Smaller values are set to zero. For results $|x| \geq 10^{+385}$, error 4 or 5 will appear (see *below*).

[18] Starting here, decimal input is written using a point as radix mark throughout this manual, although significantly less visible, unless specified otherwise explicitly. By experience, the „comma people“ are more capable to read radix points and interpret them correctly than vice versa.

7. In **integer modes**, numbers are displayed adjusted to the right as well. Word size and complement setting are indicated in the dot matrix using a format ***xx.ww***, with ***xx*** being **1c** or **2c** for 1's or 2's complement, respectively, **un** for unsigned, or **sm** for sign-and-mantissa mode. Sign and first digit of the exponent show the base, a "c" in the second digit signals a carry bit set, an "o" in the third an overflow. Integer bases are indicated as follows:

Base	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Sign and 1st digit of exponent displayed	**b**	**3**	**4**	**5**	**6**	**7**	**o**	**9**	**d**	**-1**	**-2**	**-3**	**-4**	**-5**	**h**

The example shows the WP 34S displaying an arbitrary number in unsigned hexadecimal mode with word size 64 and carry set:



After changing to binary mode, this number will need 28 digits, being 1001001110100001010010110110. The 12 least significant digits will be displayed initially together with an indication that there are three display windows in total with the rightmost shown:

Now press ◀ and you will get the next 12 digits in the middle window:

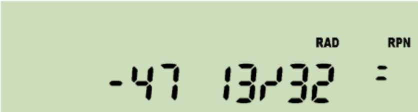
Press ◀ again to show the most significant digits:

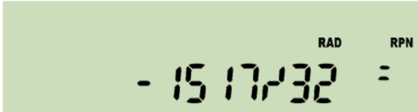
If leading zeros were turned on, there will be six display windows in this case, with the three "most significant" containing only zeros.

Please note numeric input is limited to 12 digits in any integer base.

8. **Fraction mode** works similar to HP-35S. In particular, DENMAX sets the maximum allowable denominator (see the *index of operations*). Display will look like in the examples below. If the fraction is exactly equal, slightly less, or greater than the floating point number converted, "=", "Lt", or "Gt" is indicated in the exponent, respectively. This mode can handle numbers with absolute values < 100,000 and > 0.0001. Maximum denominator is 9999. Underflows as well as overflows will be displayed in the format set before fraction mode was entered.

Now assume your WP 34S being reset. Key in -47.40625 a b/c and you will see:

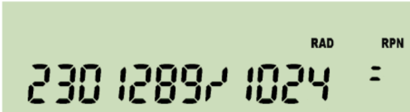


or after d/c : 

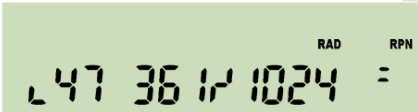
.

Please note integers like 123 will be displayed as “123 0/1” or “123/1” in fraction mode, respectively, to indicate this mode.

Squaring the improper fraction shown above results in



Now, enter a b/c for converting this result into a proper fraction. You will get



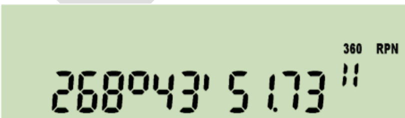
with a little hook left of the first digit shown. This indicates the leading number is displayed incompletely – there are at least two digits preceding 47 but no more display space. Press SHOW to unveil the integer part of this proper fraction is 2247.

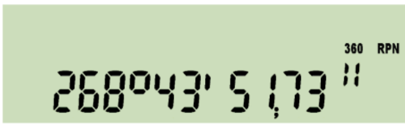
Input in fraction mode is straightforward and logically coherent:

Key in:	and get in proper fraction mode:
1 2 . 3 . 4 ENTER↑	12 3/4
1 . 2 ENTER↑	1 1/5
. 1 . 2 ENTER↑	1/2
. 1 2 ENTER↑	3/25 (= 0.12)
1 . . 2 ENTER↑	1 0/1 (= 1 $^{0}/_{2}$!)

For comparison, please note the *HP-32SII* reads the last input here as ½ – which is, however, not consistent with its other input interpretations in fraction mode.

9. In **H.MS display mode**, format is `hhhh°mm'ss.dd"` with the number of hours or degrees limited to 9000. Output may look like this:



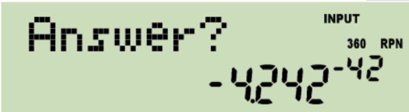
or 

depending on the radix setting. For decimal times less than 5ms or 0.005 angular seconds but greater than zero, an “u” for underflow will be lit in the exponent section. For times or angles exceeding the upper limit, an “o” will be shown there signaling an overflow, and the value is displayed modulo 9000.

10. Output of the function **WDAY** will look as follows for an input of 1.13201 in M.DY mode (equivalent to inputs of 13.01201 in D.MY or 2010.0113 in Y.MD):

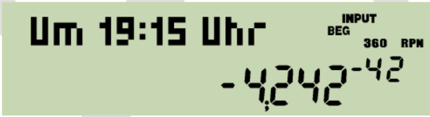
Expect similar displays after DAYS+. – Dates before the year 8 may be indicated differently to what they really were due to the inconsistent application of the leap year rule before this.

11. In **alpha mode**, the alpha register is displayed in the dot matrix, showing the last characters it is containing, while the numeric section keeps the result of the last numeric operation, e.g.:



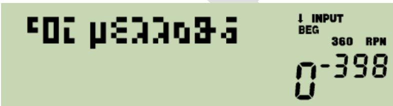
Different information may be appended to ***alpha***. See the commands starting with "α" in the index of operations below. E.g. αTIME allows creating texts like

or



depending on time mode setting (12h / 24h). And αDATE will append – depending on date format setting – either 2011-04-16 or 16.04.2011 or 04/16/2011 to ***alpha***.

Please note ***alpha*** may contain up to 31 characters. And your WP 34S features a rich set of special letters. So you may easily store a message like



Use ▲ and ▼ for browsing it in steps of 6 characters. Browsing to the left will stop with the very first characters shown, browsing to the right stops showing the right end completely, i.e.

in this very special case.

All keyboard input will be interpreted according to the mode set at input time.

FONTS

Your WP 34S features a large and a small font. Both are based on Luiz Viera's fonts as distributed in 2004. Some letters were added and some modified for better legibility, since the dot matrix is only 6 pixels high here. The following tables show the characters directly accessible through the keyboard:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
ABCDEFGHIJKLM NOPQRSTUVWXYZ
ABCDEFGHIJKLM NOPQRSTUVWXYZ

a b c d e f g h i j k l m n o p q r s t u v w x y z
abcdefghijklm nopqrstuvwxyz
abcdefghijklm nopqrstuvwxyz

Α Β Γ Δ Ε Ζ Η Θ Ι Κ Λ Μ Ν Ξ Ο Π Ρ Σ Τ Υ Φ Χ Ψ Ω
ΑΒΓΔΕΖΗΘΙΚΛΜ ΝΞΟΠΡΣΤΥΦΧΨΩ
ΑΒΓΔΕΖΗΘΙΚΛΜ ΝΞΟΠΡΣΤΥΦΧΨΩ

α β γ δ ε ζ η ϑ ι κ λ μ ν ξ ο π ρ σ τ υ φ χ ψ ω
αβγδεζηϑικλμ νξοπρστυφχψω
αβγδεζηϑικλμ νξοπρστυφχψω

0 1 2 3 4 5 6 7 8 9	( ) + - × / ± . ! ?	↔ % √ \ & | ≠ $ € £ ¥
0123456789	()+−×/±.!?	↔%√\&|≠$€£¥
0123456789	()+−×/±.!?	↔%√\&|≠$€£¥

More characters live in the alpha catalogs you find *below*.

INDEX OF OPERATIONS

All commands available are found below with their *names* and *keystrokes* necessary. Names printed in **bold** face in this list belong to functions directly accessible on the key-board, the other commands may be picked from catalogs. The command names will show up identically catalogs and in program listings unless specified otherwise explicitely. Sorting in index and catalogs is case insensitive and works in the following order:

␣, 0…9, A…Z, α…ω, ( ) + – × / ± , . ! ? : ; ‘ “ # * @ _ ~
→ ← ↑ ↓ ↔ < ≤ = ≠ ≥ > % $ € £ ¥ √ ∫ ∞ & \ ^ | G [ ] { }

Super- and subscripts are handled like normal characters in sorting. The fifth last item in the sorting order list above is the indicator for the angular mode GRAD.

Generally, functions and keystroke programming will work as on *HP-42S*, bit and integer functions as on *HP-16C*, unless stated otherwise under remarks. Especially, all **tests** will return “true” or “false” in the dot matrix if called from the keyboard; if called in a program, they will skip the next program line if the test is false. Please refer to the manuals of the vintage calculators mentioned for additional information about traditional commands.

Functions available on the WP 34S for the first time on an RPN calculator are high-lighted yellow under remarks, while operations carrying a familiar name but deviating in their functionality here are marked light red.

Parameters will be taken from the lowest stack level(s) unless mentioned explicitly in the 2nd column – then they must follow the command. If ***underlined***, they may also be specified using indirect addressing, as shown in the *tables* above. Some parameters of statistical distributions must be given in registers **J** and **K** if specified.

In the following, each function is listed stating the mode(s) it will work in, abbreviated by their *indicators*. In this column an “&” stands for a Boolean AND, a comma for an OR, and a backslash for “not”. So e.g. 2^x works in all modes but alpha. All operations may also be entered in mode PRG unless stated otherwise explicitly.

Name	Keys to press	in modes	Remarks
c…	CPX …	DECM	Indicates an operation allowing complex input(s) and/or complex results (see *above*). The prefix CPX may be heading all functions whose *names are printed in italics in this list*.
10^x	f 10^x	DECM	
12h	h MODE …	\α	Sets 12h time display mode meaning 1:23 becomes 1:23 AM and 13:45 becomes 1:45 PM. This makes a difference in αTIME only.
1COMPL	h MODE …	\α	Sets 1’s complement mode like in *HP-16C*.
1/x	f 1/x	DECM	
	B	DECM	Shortcut as long as label B is not defined yet.

Name	Keys to press	in modes	Remarks
24h	h MODE …	\α	Sets 24h time display mode meaning 1:23 AM becomes 1:23, and 1:45 PM becomes 13:45.
2COMPL	h MODE …	\α	Sets 2's complement mode like in *HP-16C*.
2^x	f 2^x	\α	
ABS	f |x|	\α	Returns the absolute value.
	CPX f |x|	DECM	Returns $r=\sqrt{x^2+y^2}$ in **X** and clears **Y**.
ACOS	g COS⁻¹	DECM	Returns $\arccos(x)$.
ACOSH	g HYP⁻¹ COS	DECM	Inverse hyperbolic cosine, known as *arcosh*. Note there is no need for pressing f here.
AGM	h X.FCN …	DECM	Returns the arithmetic-geometric mean of x and y .
ALL	h ALL ***n***	\α	ALL 00 works like ALL in *HP-42S*. For $x > 10^{13}$, however, display will switch to SCI or ENG with the maximum number of digits necessary (see SCIOVR / ENGOVR). The same will happen if $x < 10^{-n}$ <u>and</u> more than 12 digits are required to show x completely.
AND	h AND	Integer	Works bitwise as in *HP-16C*.
		DECM	Works like AND in *HP-28S*, i.e. x and y are interpreted before executing this operation. 0 is "false", any other real number is "true".
ANGLE	h X.FCN …	DECM	Returns the angle between positive x-axis and the straight line from the origin to the point (x, y) , i.e. $arctan(y/x)$. This is a two-number function, it consumes y.
ASIN	g SIN⁻¹	DECM	Returns $\arcsin(x)$.
ASINH	g HYP⁻¹ SIN	DECM	Inverse hyperbolic sine, known as *arsinh*.
ASR	h X.FCN ASR ***n***	Integer	Works like ***n*** (up to 63) consecutive ASR commands in *HP-16C*, corresponding to a division by 2^n . ASR 0 executes as NOP, but loads **L**.
ATAN	g TAN⁻¹	DECM	Returns $\arctan(x)$.
ATANH	g HYP⁻¹ TAN	DECM	Inverse hyperbolic tangent, known as *artanh*.

Name	Keys to press	in modes	Remarks
BACK	h P.FCN BACK ***n***	PRG	Jumps ***n*** program steps backwards (1 ≤ ***n*** ≤ 99). So e.g. BACK 01 goes to the previous step. Reaching step 000 stops program execution.
BASE	h MODE BASE ***n***	\α	Sets the base for integer calculations, with 2 ≤ ***n*** ≤ 16. Popular bases are directly accessible on the keyboard. Current integer base setting is indicated in the exponent as explained above. Furthermore, BASE0 equals DECM, and BASE1 calls FRACT. See below. **ATTENTION:** Going from DECM to any integer mode, the current stack contents will be truncated. Going from integer to DECM, the current stack contents will be converted. Other register contents will not!
BASE10	g 10		
BASE16	g 16		
BASE2	f 2		
BASE8	f 8		
BATT	h X.FCN …	DECM	Measures the battery voltage in the range between 1.9V and 3.4V and returns this value.
		Integer	As above but returns the voltage in 0.1V units.
BC?	h TEST BC? ***n***	Integer	Tests the specified bit in x .
BestF	h CFIT …	DECM	Selects the best curve fit model, maximizing the correlation like BEST does in *HP-42S*.
Binom	h PROB …	DECM	Binomial distribution with the number of successes ***g*** in **X**, the probability of a success $\boldsymbol{p_0}$ in **J** and the sample size ***n*** in **K**: Binom$_P$ [19] returns $p_B(g;n;p_0)=\binom{n}{g}\cdot p_0^g\cdot(1-p_0)^{n-g}$. Binom returns $F_B(m;n;p_0)=\sum_{g=0}^{m}p_B(g;n;p_0)$, with the maximum number of successes ***m*** in **X**. Binom $^{-1}$ returns ***m*** for given probabilities $\boldsymbol{F_B}$ in **X** and ***p*** in **J** with sample size ***n*** in **K**.
Binom$_P$			
Binom $^{-1}$			
B_n	h X.FCN …	DECM	Returns the Bernoulli number for an integer ***n*** > 0 given in **X**: $B_n=(-1)^{n+1}n\cdot\zeta(1-n)$. See below for ζ. This lives in XROM, may be less accurate.

[19] Binom$_P$ equals BINOMDIST(***g; n; p$_0$;*** **0**) and Binom equals BINOMDIST(***m; n; p$_0$;*** **1**) in MS Excel.

Name	Keys to press	in modes	Remarks
B_n*	h X.FCN …	DECM	Returns the Bernoulli number according to its old definition for integer $n > 0$ given in **X**: $B_n^* = \frac{2 \cdot (2n)!}{(2\pi)^{2n}} \cdot \zeta(2n)$. See below for ζ. This lives in XROM, may be less accurate.
BS?	h TEST BS? ***n***	Integer	Tests the specified bit in ***x*** .
Cauch	h PROB …	DECM	Cauchy-Lorentz distribution (also known as Lorentz or Breit-Wigner distribution) with the location x_0 specified in **J** and the shape γ in **K** :
CauchP			CauchP returns $f_{Ca}(x) = \frac{1}{\pi\gamma} \cdot \frac{1}{1+\left(\frac{x-x_0}{\gamma}\right)^2}$,
Cauch $^{-1}$			Cauch returns $F_{Ca}(x) = \frac{1}{2} + \frac{1}{\pi}\arctan\left(\frac{x-x_0}{\gamma}\right)$. Cauch $^{-1}$ returns ***x*** for a given probability F_{Ca} in **X**, with location x_0 in **J** and shape γ in **K**.
CB	h X.FCN CB ***n***	Integer	Clears the specified bit in ***x*** .
CEIL	h X.FCN …	DECM	Returns the smallest integer ≥ ***x*** .
CF	h P.FCN CF ***n***	\α	Clears the flag specified.
CLALL	h X.FCN …	\PRG	Clears all registers and programs if confirmed.
CLFLAG	h P.FCN …	\α	Clears all user flags.
CLP	h P.FCN …	\PRG	Positions the program pointer to step 000 and clears the subroutine return stack.
		PRG	Clears all the program memory if confirmed. Not programmable.
CLREG	h X.FCN …	All	Clears all general purpose registers. The stack and its contents are kept.
CLSTK	0 g FILL h P.FCN …	\α	Clears all stack registers, i.e. **X** through **T** or **X** through **D**, respectively. All other register contents are kept.
CLx	h CLx	All	Clears **X** only, disabling stack lift as usual.
CLα	f CLα	All	Clears the alpha register like CLA in *HP-42S*.
CLΣ	g CLΣ	DECM	Clears all statistical sums in the respective general purpose registers.

Name	Keys to press	in modes	Remarks
COMB	f Cy,x	DECM	Returns the number of possible <u>sets</u> of y items taken x at a time. No item occurs more than once in a set, and different orders of the same x items are <u>not</u> counted separately. Formula: $C_{y,x} = \binom{y}{x} = \frac{y!}{x! \cdot (y-x)!}$
CONJ	CPX X.FCN …	DECM	Changes the sign of y, thus returning the complex conjugate of x_c.
CORR	g r	DECM	Returns the correlation coefficient for the current statistical data and curve fitting model.
COS	f COS	DECM	Returns the cosine of the angle in X.
COSH	f HYP COS	DECM	Returns the hyperbolic cosine of x.
COV	h CFIT …	DECM	Returns the population covariance for two data sets. It depends on the fit model selected. For LinF, it calculates $COV_{xy} = \frac{1}{n^2}\left(n\sum x_i y_i - \sum x_i \sum y_i\right)$ See s_{xy} for the sample covariance.
CUBE	h X.FCN …	\α	Returns x^3.
CUBERT	h X.FCN …	\α	Returns $\sqrt[3]{x}$.
DATE	h P.FCN …	DECM	Recalls the date from the real time clock and displays it in the numeric section in the format selected. See D.MY, M.DY, and Y.MD. The function DATE of *HP-12C* corresponds to DAYS+ in your WP 34S (see below).
DAY	h X.FCN …	DECM	Assumes x containing a date in the format selected and extracts the day.
DAYS+	h X.FCN …	DECM	Works like DATE in *HP-12C*, adding x days on a date in $\mathbf{Y}$ in the format selected and displaying the resulting date including the day of week in the same format as WDAY does.
DBLR DBL × DBL /	h X.FCN …	Integer	Double precision commands for remainder, multiplication and division like in *HP-16C*.

Name	Keys to press	in modes	Remarks
DEC	h P.FCN DEC *r*	\α	Decrements ***r*** by one, equivalent to 1 STO– ***r*** , but without modifying the stack.
DECM	f H.d	\α	Sets default decimal mode for calculations.
DECOMP	h X.FCN …	FRC	Decomposes x (after converting it into an improper fraction, if applicable), resulting in a stack [***numerator(x), denominator(x), y, z***] or [***num(x), den(x), y, z, t, a, b, c***] , respectively. Reversible by division.
DEG	g DEG	DECM	Sets angular mode to degrees.
DEG→	h X.FCN …	DECM	Takes x as degrees and converts them to the angular mode currently set.
DENANY	h MODE …	\α	Sets default fraction format like in *HP-35S*, allowing maximum precision. Any denominator up to the value set by DENMAX may appear.
DENFAC	h MODE …	\α	Sets "factors of the maximum denominator". With e.g. DENMAX = 60, possible denominators are 2, 3, 4, 5, 6, 10, 12, 15, 20, 30, and 60.
DENFIX	h MODE …	\α	Sets fixed denominator format, i.e. the denominator equaling DENMAX always.
DENMAX	h MODE …	\α	Works like /c in *HP-35S*, but maximum denominator settable is 9999. It will be set to this value if $x < 1$ or $x > 9999$ at execution time. For $x = 1$ the current setting is recalled.
DET	h MATRIX …	\α	Takes a descriptor of a square matrix in **X** and returns the determinant of the matrix. The matrix is not modified.
DISP	h MODE DISP *n*	DECM	Changes the number of decimals shown while keeping the basic display format (FIX, SCI, ENG) as is. With ALL set, DISP will change the switchover point (see ALL).
DROP	h X.FCN …	\α	Drops x . See above for details and ᶜDROP.
DSE	f DSE *r*	PRG	Given `cccccc.fffii` in ***r*** , DSE decrements ***r*** by `ii`, skipping next program line if then `cccccc` ≤ `fff` . If ***r*** features no fractional part then `fff` is 0 and `ii` is set to 1. Note that neither `fff` nor `ii` can be negative, and DSE makes only sense with `cccccc` > 0.

Name	Keys to press	in modes	Remarks
DSL	h P.FCN DSL ***r***	PRG	Works like DSE but skips if `ccccccc` < `fff` .
DSZ	h P.FCN DSZ ***r***	PRG	Decrements ***r*** by one, and skips if $\|r\|<1$ thereafter. Known from the *HP-16C*.
D.MY	h MODE …	\α	Sets the format for date display.
D→J	h X.FCN …	DECM	Takes ***x*** as a date in the format selected and converts it to a Julian day number according to JG…
D→R		DECM	Please see the *catalog of conversions below* for conversions from degrees to radians.
E3OFF E3ON	h MODE …	\α	Toggle the thousands separator (either a point or a comma depending on the radix setting).
ENG	h ENG ***n***	\α	Sets engineering display format.
ENGOVR	h ENG ENTER↑	\α	Numbers exceeding the range displayable in ALL or FIX will be shown in engineering format. See SCIOVR.
ENTER↑	ENTER↑	\α	See *above* for details.
ENTRY?	h TEST …	All	Checks the entry flag. This internal flag is set if: • any character is entered in alpha mode, or • any command is accepted for entry (be it via ENTER↑, a function key, or R/S with a partial command line).
erf erfc	h X.FCN …	DECM	Returns the error function or its complementary: $erf(x)=\frac{2}{\sqrt{\pi}}\int_0^x e^{-\tau^2}d\tau$ and $erfc(x)=1-erf(x)$
ERR	h P.FCN ERR ***n***	PRG	Raises the error specified and clears the return stack. See *below* for the respective error codes.
EVEN?	h TEST …	\α	Checks if ***x*** is integer and even.
e^x	f e^x	DECM	
ExpF	h CFIT …	DECM	Selects the exponential curve fit model $y=a_0e^{a_1x}$.

Name	Keys to press	in modes	Remarks
Expon Expon$_P$ Expon $^{-1}$	h PROB …	DECM	Exponential distribution with the rate λ in **J** : Expon$_P$ [20] returns $f_{Ex}(x) = \lambda \cdot e^{-\lambda x}$, Expon returns $F_{Ex}(x) = 1 - e^{-\lambda x}$. Expon $^{-1}$ returns the survival time t_s for a given probability F_{Ex} in **X** and rate λ in **J**.
EXPT	h X.FCN	DECM	Returns the exponent h of the number displayed $x = m \cdot 10^h$. Compare MANT.
e^x -1	h X.FCN e^x-1	DECM	Returns more accurate results for the fractional part of e^x with $x \approx 0$.
FAST	h MODE …	All	Sets the processor speed to “fast”. This is start-up default and is kept for fresh batteries.
FB	h X.FCN FB ***n***	Integer	Inverts (“flips”) the specified bit in x .
FC? FC?C FC?F FC?S	h TEST FC? ***n*** etc.	\α	Tests if the flag specified is clear. Clears, flips, or sets this flag after testing, if applicable.
FF	h P.FCN FF ***n***	\α	Flips the flag specified.
FIB	h X.FCN …	\α	Returns the Fibonacci number F_x .
FILL	g FILL	\α	Copies x to all stack levels. See details *above*.
FIX	h FIX ***n***	\α	Sets fixed point display format.
FLOOR	h X.FCN …	DECM	Returns the largest integer $\leq x$.
FP	g FP	DECM	Returns the fractional part of x .
FP?	h TEST …	\α	Tests x for having a nonzero fractional part.
FRACT	h MODE …	\α	Sets fraction mode like in HP-35S, but keeps display format as set by PROFRC or IMPFRC.
FS? FS?C FS?F FS?S	h TEST FS? ***n*** etc.	\α	Tests if the flag specified is set. Clears, flips, or sets this flag after testing, if applicable.

[20] The pdf corresponds to EXPONDIST($x; \lambda;$ **0**) and the cdf to EXPONDIST($x; \lambda;$ **1**) in MS Excel.

Name	Keys to press	in modes	Remarks
$F_P(x)$ $F(x)$ $F^{-1}(p)$	h PROB …	DECM	F-distribution. The *cdf* F(x) equals 1 - Q(F) in *HP-21S*. The degrees of freedom are specified in **J** and **K**.
f'(x)	h P.FCN f'(x) ***label***	DECM	Return the first or second derivative of ***f(x)***, respectively, with the function ***f(x)*** being specified in a routine starting with LBL ***label***. The return stack will have ***y***, ***z***, and ***t*** cleared and the position ***x*** in **L**. Either command will attempt to call a user routine labeled 'δx' to provide a fixed step size ***dx***. If that routine is not defined, a step size of 0.1 is employed instead.
f"(x)	h P.FCN f"(x) ***label***		
GCD	h X.FCN …	\α	Returns the Greatest Common Divisor of ***x*** and ***y*** .
Geom $Geom_P$ Geom $^{-1}$	h PROB …	DECM	Geometric distribution: $Geom_P$ returns $f_{Ge}(m) = p_0(1-p_0)^m$, Geom returns $F_{Ge}(m) = 1-(1-p_0)^{m+1}$, being the probability for a first success after ***m*** = ***x*** Bernoulli experiments. The probability $\boldsymbol{p_0}$ for a success in each such experiment must be specified in **J**. Geom $^{-1}$ returns the number of failures ***f*** before the first success for given probabilities $\boldsymbol{F_{Ge}}$ in **X** and $\boldsymbol{p_0}$ in **J**.
GRAD	g GRAD	DECM	Sets angular mode to gon or grads.
GRAD→	h X.FCN …	DECM	Takes ***x*** as given in gon or grads and converts them to the angular mode currently set.
GTO	h GTO ***label***	PRG	Inserts an unconditional branch to ***label***.
		\PRG, \α	Positions the program pointer to ***label***.
	h GTO . A , B , C , or D	\α	Positions the program pointer … (not programmable) … to one of these labels, if defined.
	h GTO . ***nnn***	\α	Positions the program pointer … (not programmable) … to step ***nnn*** .
	h GTO . .	\α	Positions the program pointer … (not programmable) … to step 000 .
GTOα	h P.FCN …	\α	Takes the first three characters of ***alpha*** (or less if there are less available) as a label and positions the program pointer to it.

Name	Keys to press	in modes	Remarks
H_n	h X.FCN …	DECM	Hermite's polynomials for probability: $H_n(x) = (-1)^n \cdot e^{x^2/2} \cdot \frac{d^n}{dx^n}\left(e^{-x^2/2}\right)$ with ***n*** in **Y**, solving the differential equation $f''(x) - 2x \cdot f'(x) + 2n \cdot f(x) = 0$.
H_{np}	h X.FCN …	DECM	Hermite's polynomials for physics: $H_{np}(x) = (-1)^n \cdot e^{x^2} \cdot \frac{d^n}{dx^n}\left(e^{-x^2}\right)$ with ***n*** in **Y**.
H.MS	f H.MS	DECM	Assumes **X** containing *decimal* hours or degrees, and displays them converted in the format `hhhh°mm′ss.dd″` as shown in the paragraph above. Will return to the previous decimal display with the next keystroke thereafter.
H.MS+ **H.MS–**	h P.FCN …	DECM	Assumes **X** and **Y** containing times or degrees in the format `hhhh.mmssdd` , and adds or subtracts them, respectively.
IBASE?	h TEST …	\α	Returns the integer base set (see BASE).
IMPFRC	g d/c	\α	Sets fraction mode allowing improper fractions in display (i.e. $^5/_3$ instead of $1\ ^2/_3$). Converts ***x*** according to the settings by DEN… Absolute decimal equivalents of ***x*** must not exceed 100,000. Compare PROFRC.
		FRC	Allows displaying improper fractions. Thus converts a proper fraction in **X** into the equivalent improper fraction, if applicable.
INC	h P.FCN INC ***r***	\α	Increments ***r*** by one, equivalent to 1 STO+ ***r*** , but without modifying the stack.
INTM?	h TEST …	\α	Tests if your WP 34S is in an integer mode.
INT?	h TEST …	\α	Tests ***x*** for being an integer, i.e. having a fractional part equal to zero. Compare FP?.
IP	f IP	DECM	Returns the integer part of ***x*** .
ISE	h P.FCN ISE ***r***	PRG	Works like ISG but skips if `ccccccc` ≥ `fff` .

Name	Keys to press	in modes	Remarks
ISG	g ISG ***r***	PRG	Given `cccccc.fffii` in ***r***, this function increments ***r*** by `ii`, skipping next program line if then `ccccccc` > `fff`. If ***r*** features no fractional part then `ii` is set to 1. Note that neither `fff` nor `ii` can be negative, but `cccccc` can.
ISZ	h P.FCN ISZ ***r***	PRG	Increments ***r*** by one, skipping next program line if then $\lvert r \rvert < 1$. Known from the *HP-16C*.
I β	h X.FCN …	DECM	Returns the regularized incomplete beta function $\frac{\beta_x(x,y,z)}{\beta(y,z)} = \frac{1}{\beta(y,z)} \cdot \int_0^x t^{y-1}(1-t)^{z-1}dt$ with β_x being the incomplete beta function and β being Euler's beta (see below).
I Γ	h X.FCN …	DECM	Returns the regularized incomplete gamma function $\frac{\gamma(x,y)}{\Gamma(x)}$ with $\gamma(x,y) = \int_0^y t^{x-1}e^{-t}dt$ being the lower incomplete gamma function. For Γ see below.
JG1582 JG1752	h X.FCN …	DECM	These two commands reflect different dates the Gregorian calendar was introduced in different large areas of the world. D→J and J→D will be calculated accordingly.
J→D	h X.FCN …	DECM	Takes ***x*** as a Julian day number and converts it to a date according to JG... in the format selected
KEY?	h TEST KEY? ***a***	All	Tests if a key was pressed while a program was running or paused. If <u>no</u> key was pressed, the next program step after KEY? will be executed, else it will be skipped and the code of said key will be found in address ***a***. Key codes start top left and correspond to the rows and columns on the keyboard – so e.g. A corresponds to 11, CPX to 16, STO to 21, and + to 75.

Name	Keys to press		in modes	Remarks
KTY?	[h] [TEST] KTY? ***a***		All	Assumes a key code in address ***a*** . Checks this code and returns • 0 … 9 if it corresponds to a digit [0] … [9] , • 10 if it corresponds to [.], [EEX], or [+/−] , • 11 if it corresponds to [f], [g], or [h] , • 12 if it corresponds to any other key. May help in user interaction with programs.
LASTx	[RCL] [L]		\α	See *above* for details.
LBL	[f] [LBL] ***label***		PRG	Identifies programs and routines for execution and branching. See opportunities for specifying ***label*** in the table *above*.
LBL?	[h] [TEST] LBL? ***label***		All	Tests for the existence of the label specified, anywhere in program memory. See opportunities for specifying ***label*** in the table *above*.
LCM	[h] [X.FCN] …		\α	Returns the Least Common Multiple of x and y.
LEAP?	[h] [TEST] …		DECM	Takes x as a date in the format selected, extracts the year, and tests for a leap year.
LgNrm	[h] [PROB] …		DECM	Lognormal distribution with $\mu = \ln \bar{x}_g$ specified in $\mathbf{J}$ and $\sigma = \ln \varepsilon$ in $\mathbf{K}$. See x̄g and ε below.
LgNrmₚ				LgNrmₚ returns $f_{Ln}(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{(\ln x - \mu)^2}{2\sigma^2}}$,
LgNrm $^{-1}$				LgNrm returns $F_{Ln}(x) = \Phi\left(\frac{\ln x - \mu}{\sigma}\right)$ with $\Phi(z)$ denoting the standard normal *cdf*. LgNrm $^{-1}$ returns x for a given probability F_{Ln} in $\mathbf{X}$, μ in $\mathbf{J}$, and σ in $\mathbf{K}$.
LINEQS	[h] [X.FCN] …		\α	Takes a base register in $\mathbf{X}$, a vector *descriptor* in $\mathbf{Y}$, and a descriptor of a square matrix in $\mathbf{Z}$. Solves the system of linear equations $(Z) \cdot \vec{x} = \vec{y}$ and returns the filled out vector descriptor in $\mathbf{X}$.
LinF	[h] [CFIT] …		DECM	Selects the linear curve fit model $y = a_0 + a_1 x$.
LJ	[h] [X.FCN] …		Integer	Left adjust as in *HP-16C*.
LN	[g] [LN]		DECM	Returns the natural logarithm of x , i.e. the logarithm of x for base *e*.

Name	Keys to press	in modes	Remarks
L_n	h X.FCN …	DECM	Laguerre's polynomials (compare $L_n\alpha$ below): $L_n(x)=\frac{e^x}{n!}\cdot\frac{d^n}{dx^n}\left(x^n e^{-x}\right)=L_n^{(0)}(x)$ with ***n*** in **Y**, solving the differential equation $x\cdot y''+(1-x)y'+ny=0$.
LN1+x	h X.FCN …	DECM	Natural logarithm of values close to zero. Returns $\ln(1+x)$, providing a much higher accuracy in the fractional part of the result.
$L_n\alpha$	h X.FCN …	DECM	Laguerre's generalized polynomials with ***n*** in **Y** and **α** in **Z**: $L_n^{(\alpha)}(x)=\frac{x^{-\alpha}e^x}{n!}\cdot\frac{d^n}{dx^n}\left(x^{n+\alpha}e^{-x}\right)$.
LN β	h X.FCN …	DECM	Returns the natural logarithm of Euler's β function. See there.
LN Γ	h X.FCN …	DECM	Returns the natural logarithm of Γ(***x***) . See there.
LOAD	h P.FCN …	\α	Restore the entire backup. Compare SAVE.
LocR	h P.FCN LocR ***n***	PRG	Allocates ***n*** + *1* local registers and 16 local flags for the current program. See *above*.
LocR?	h TEST LocR?	PRG	Returns the number of local registers allocated.
LOG₁₀	g LG	DECM	Returns the logarithm of ***x*** for base 10.
LOG₂	g LB	\α	Returns the logarithm of ***x*** for base 2.
LogF	h CFIT …	DECM	Selects the logarithmic curve fit model $y=a_0+a_1\ln x$.
Logis Logis$_P$ Logis $^{-1}$	h PROB …	DECM	Logistic distribution with ***μ*** given in **J** and ***s*** in **K** . Logis$_P$ returns $f_{Lg}(x)=e^{-\frac{x-\mu}{s}}\Big/ s\cdot\left(1+e^{-\frac{x-\mu}{s}}\right)^2$, Logis returns $F_{Lg}(x)=\left(1+e^{-\frac{x-\mu}{s}}\right)^{-1}$. Logis $^{-1}$ returns $F_{Lg}^{-1}(p)=\mu+s\cdot\ln\left(\frac{p}{1-p}\right)$ for a probability ***p*** given in **X**, ***μ*** in **J**, and ***s*** in **K**.

Name	Keys to press	in modes	Remarks
LOGx	g LOGx	DECM	Returns the logarithm of y for base x .
	CPX g LOGx	DECM	Returns the complex logarithm of $z + i\,t$ for the complex base $x + i\,y$.
LZOFF LZON	h MODE …	\α	Toggles leading zeros like flag 3 does in *HP-16C*. Relevant in integer modes only.
L.R.	h CFIT …	DECM	Returns the parameters a_1 and a_0 of the fit curve through the data points accumulated, according to the model selected, and pushes them on the stack. For a straight regression line, a_0 is the y-intercept and a_1 the slope.
MANT	h X.FCN …	DECM	Returns the mantissa m of the number displayed $x = m \cdot 10^h$. Compare EXPT.
MASKL MASKR	h X.FCN MASKL $\underline{n}$ etc.	Integer	Work like MASKL and MASKR on *HP-16C*, but with the mask length following the command instead of taken from **X**.
MAX	h X.FCN …	\α	Returns the maximum of x and y .
MEM?	h TEST …	All	Returns the number of free words in program memory, taking into account the local registers allocated.
MIN	h X.FCN …	\α	Returns the minimum of x and y .
MIRROR	h X.FCN …	Integer	Reflects the bit pattern in x (e.g. 000101 becomes 101000 for word size 6).
MONTH	h X.FCN …	DECM	Assumes x containing a date in the format selected and extracts the month.
MROW+×	h MATRIX …	DECM	Takes a matrix descriptor in **X**, a destination row number in **Y**, a source row number in **Z**, and a real number in **T**. It multiples each element m_{zi} by t and adds it to m_{yi} . The stack is unchanged. M.ROW+× is similar to *PPC M3*.
MROW×	h MATRIX …	DECM	Takes a matrix descriptor in **X**, a row number in **Y**, and a real number in **Z**. It multiples each element m_{yi} by z. The stack is unchanged. M.ROW× is similar to *PPC M2*.
MROW⇆	h MATRIX …	DECM	Takes a matrix descriptor in **X** and two row numbers in **Y** and **Z**. It swaps the contents of rows y and z. The stack is unchanged. M.ROW⇆ is similar to *PPC M1*.

Name	Keys to press	in modes	Remarks
MSG	h P.FCN MSG ***n***	PRG	Throws the error message specified. It will vanish with the next keystroke. See *below* for the respective error codes. Compare ERR.
M+×	h MATRIX …	DECM	Takes two matrix *descriptors* in **X** and **Y**, and a real number ***z***. Returns $(X)+(Y)\cdot z=(X)$. Thus a scalar multiple of one matrix is added to another matrix. The multiply adds are done internally in high precision and results should be exactly rounded.
M $^{-1}$	h MATRIX …	DECM	Takes a *descriptor* of a square matrix in **X** and inverts the matrix in-situ. Doesn't alter the stack.
M-ALL	h MATRIX …	DECM	Takes a matrix *descriptor* in **X** , saves it in **L**, and returns a value suitable for ISG or DSL looping in **X**. The loop processes all elements in the matrix. The loop index is DSL if the descriptor is negative and ISG else.
M-COL	h MATRIX …	DECM	Takes a matrix *descriptor* in **X** and a column number in **Y**. Returns a loop counter in **X**, dropping the stack. The matrix descriptor is saved in **L**. The loop processes all elements m_{iy} only. The loop index is DSL if the descriptor is negative and ISG else.
M-DIAG	h MATRIX …	DECM	Takes a matrix *descriptor* in **X**, saves it in **L**, and returns a loop counter in **X**. The loop processes all elements along the matrix diagonal, i.e. all elements m_{ii} . The loop index is DSL if the descriptor is negative and ISG else.
M-ROW	h MATRIX …	DECM	Takes a matrix *descriptor* in **X** and a row number in **Y**. Returns a loop counter in **X**, dropping the stack and setting last **L** like all two-argument commands. The loop processes all elements m_{yi} only. The loop index is DSL if the descriptor is negative and ISG else.

Name	Keys to press	in modes	Remarks
M×	h MATRIX …	DECM	Takes two matrix *descriptor*s in **Y** and **Z** and the integer part of ***x*** as the base address of the result. Returns $(Z) \cdot (Y) = (X)$. The fractional part of ***x*** is updated to match the resulting matrix – no overlap checking is performed. All calculations are done internally in high precision, although it would still be possible to trick the code up and produce bad results. It would be very difficult to get the same degree of accuracy in RPN since the best that can easily be achieved there is $a{\cdot}b+c{\cdot}d$ and a matrix multiply adds more terms than this.
M.COPY	h MATRIX …	DECM	Takes a matrix *descriptor* in **Y** and a register number in **X**. Copies the matrix into registers starting at **X**. Returns a properly formed matrix descriptor in **X**.
M.DY	h MODE …	\α	Sets the format for date display.
M.IJ	h MATRIX …	DECM	Takes a matrix *descriptor* in **X** and a register number in **Y**. Returns the column that register represents in **Y** and the row in **X**. The descriptor is saved in **L**. M.IJ is similar to *PPC M4*.
M.LU	h MATRIX …	DECM	Takes *descriptor* of a square matrix in **X**. Transforms the matrix into its LU decomposition. The matrix is modified in-situ. The value in **X** is replaced by a pivot descriptor that defines the pivots that were required to calculate the decomposition. The most significant digit is the pivot for the first diagonal entry, the next most the second and so forth.
M.REG	h MATRIX …	DECM	Takes a matrix *descriptor* in **X**, a row number in **Y**, and a column number in **Z**. The descriptor is saved in **L**. M.REG returns the register number in **X** (popping the stack twice). It is similar to *PPC M5*.
M.SQR?	h TEST …	DECM	Takes a matrix *descriptor* in **X** and tests it. Returns “true” if the matrix is square.
NAND	h X.FCN …	\α	Works in analogy to AND.
NaN?	h TEST …	\α	Tests ***x*** for being “Not a Number”.
nBITS	h X.FCN …	Integer	Counts bits set in ***x*** like #B does on *HP-16C*.

Name	Keys to press	in modes	Remarks
nCOL	h MATRIX …	DECM	Takes a matrix descriptor in **X**, saves it in **L**, and returns the number of columns in this matrix.
NEXTP	h X.FCN …	\α	Returns the next prime number > x.
NOP	h P.FCN …	PRG	“Empty” step FWIW.
NOR	h X.FCN …	\α	Works in analogy to AND.
Norml Norml$_P$	h PROB …	DECM	Normal distribution with an arbitrary mean ***μ*** specified in **J** and standard deviation ***σ*** in **K** : Norml$_P$ [21] returns $f_N(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$, Norml returns $F_N(x) = \Phi\left(\frac{x-\mu}{\sigma}\right)$. See below for the standard normal distribution Φ.
Norml $^{-1}$	h PROB …	DECM	Returns ***x*** for a given probability F_N in **X**, mean ***μ*** in **J**, and standard deviation ***σ*** in **K** [22].
NOT	h NOT	Integer	Works in analogy to AND.
nROW	h MATRIX …	DECM	Takes a matrix descriptor in **X**, saves it in **L**, and returns the number of rows in this matrix.
nΣ	h STAT …	DECM	Recalls the number of accumulated data points. Necessary for basic statistics.
ODD?	h TEST …	\α	Checks if ***x*** is integer and odd.
OFF	g OFF	PRG	Inserts a step to turn your WP 34S off under program control.
OR	h OR	\α	Works in analogy to AND.
PERM	g Py,x	DECM	Returns the number of possible arrangements of ***y*** items taken ***x*** at a time. No item occurs more than once in an arrangement, and different orders of the same ***x*** items are counted separately. Formula: $P_{y,x} = x! \cdot C_{y,x}$, compare COMB.

[21] Norml$_P$ corresponds to NORMDIST(***x; μ; σ;*** **0**) and Norml to NORMDIST(***x; μ; σ;*** **1**) in MS Excel.

[22] This corresponds to NORMINV(F_N***; μ; σ***) in MS Excel.

Name	Keys to press	in modes	Remarks
P_n	h X.FCN …	DECM	Legendre's polynomials: $P_n(x)=\frac{1}{2^n n!}\cdot\frac{d^n}{dx^n}\left[(x^2-1)^n\right]$ with ***n*** in **Y**, solving the differential equation $\frac{d}{dx}\left[(1-x^2)\cdot\frac{d}{dx}f(x)\right]+n(n+1)f(x)=0$.
Poiss Poiss_P Poiss^{-1}	h PROB …	DECM	Poisson distribution with the number of successes ***g*** in **X**, the gross error probability $\boldsymbol{p_0}$ in **J**, and the sample size ***n*** in **K**. Alternatively, Poisson's $\lambda=n\cdot p_0$ may be in **J** if ***k*** = 1: Poiss_P [23] returns $P_P(g;\lambda)=\frac{\lambda^g}{g!}e^{-\lambda}$, Poiss returns $F_P(m;\lambda)=\sum_{g=0}^{m}P_P(g;\lambda)$, with the maximum number of successes ***m*** in **X**. Poiss^{-1} returns ***m*** for given probabilities $\boldsymbol{F_P}$ in **X** and ***p*** in **J** with sample size ***n*** in **K**.
PopLR	h P.FCN …	PRG	Pops the local registers without returning. See LocR and RTN.
PowerF	h CFIT …	DECM	Selects the power curve fit model $y=a_0x^{a_1}$.
PRCL	h P.FCN PRCL ***n***	\α	Recall the user program space from flash segment ***n*** to RAM where it may be edited then (see *above*).
PRIME?	h TEST …	\α	Checks if the absolute value of the integer part of ***x*** is a prime. The method is believed to work for integers up to 9E18.
PROFRC	f a b/c	DECM	Sets fraction mode like in *HP-35S*, allowing only proper fractions or mixed numbers in display. Converts ***x*** according to the settings by DEN… Absolute decimal equivalents of ***x*** must not exceed 100,000. Compare IMPFRC.
		FRC	Allows displaying only proper fractions. Thus converts an improper fraction in **X**, if applicable, e.g. $^5/_3$ into 1 2/3.

[23] The pmf corresponds to POISSON(***g***; ***λ***; **0**) and the cdf to POISSON(***g***; ***λ***; **1**) in MS Excel.

Name	Keys to press	in modes	Remarks
PROMPT	h P.FCN …	PRG	Displays ***alpha*** and stops program execution (equaling αVIEW followed by STOP actually). See *below* for more.
PSE	h PSE ***nn***	PRG	Refreshes the display and pauses program execution for ***nn*** ticks, with $0 \le nn \le 99$. The pause will be terminated early as soon as a key is pressed.
PSTO	h P.FCN PSTO ***n***	\α	Stores the user program space in flash segment ***n*** (see *above*).
PUTK	h P.FCN PUTK ***a***	All	Assumes a key code in address ***a*** . Stops program execution, takes said code and puts it in the keyboard buffer resulting in immediate execution of the corresponding call. R/S is required to resume program execution. May help in user interaction with programs.
P⇆	h P.FCN P⇆ ***n***	\α	Exchanges the user program space with the contents of flash segment ***n*** (see *above*).
RAD	g RAD	DECM	Sets angular mode to radians.
RAD→	h X.FCN …	DECM	Takes ***x*** as radians and converts them to the angular mode currently set.
RAN#	f RAN#	DECM	Returns a random number between 0 and 1 like RAN in *HP-42S*.
		Integer	Returns a random bit pattern for the word size set.
RCF	h P.FCN RCF ***s*** h X.FCN RCF ***s***	\α	Works like RCL but recalls from a register in flash memory. Also the six recall arithmetic operations may be performed like with RCL.
RCF.RG	h P.FCN …	\α	Recovers all general purpose registers from the backup region (see SAVE and *above*).
RCF.ST	h P.FCN …	\α	Recovers the system state from the backup region (see SAVE and *above*).
RCL	RCL	\α	See the *addressing table above* for ᶜRCL.
RCLM	RCL MODE ***s*** h P.FCN RCLM ***s***	\α	Recalls mode settings stored via STOM as described *above*.
RCLS	h P.FCN RCLS ***s***	\α	Recalls 4 or 8 values from a set of registers starting at address ***s***, and pushes them on the stack. This is the converse command of STOS.

Name	Keys to press	in modes	Remarks
RCL+	RCL + ***s***	\α	Recalls the content of address ***s***, executes the specified operation on it and pushes the result on the stack. E.g. RCL–12 subtracts ***r12*** from ***x*** and displays the result (acting like RCL 12 –, but without losing a stack level). In analogy, ^{C}RCL–12 subtracts ***r12*** from ***x*** and ***r13*** from ***y***. See the *addressing table above* for ^{C}RCL.
RCL–	RCL – ***s***		
RCL×	RCL × ***s***		
RCL/	RCL / ***s***		
RCL↑	RCL ▲ ***s***	\α	RCL↑ (↓) recalls the maximum (minimum) of the values in ***s*** and **X**.
RCL↓	RCL ▼ ***s***		
RDX,	h MODE RDX,	\α	Sets the decimal mark to a comma.
	h P.FCN RDX,		
	h ./,		Toggles the radix mark.
RDX.	h MODE RDX.	\α	Sets the decimal mark to a point.
	h P.FCN RDX.		
REALM?	h TEST …	\α	Tests if your WP 34S is in real mode.
RECV	h P.FCN …	\α	Prepares your WP 34S for receiving data via serial I/O. See *Appendix A* for more.
REGS	h MODE REGS ***nn***	\α	Specifies the highest general purpose register accessible. With REGS 99 you get the default state, REGS 00 just leaves a single numbered register for use.
REGS?	h TEST …	\α	Returns the number of general purpose registers allocated (1 … 100).
RESET	h X.FCN …	All	Executes CLALL and resets all modes to start-up default, i.e. 24h, 2COMPL, ALL 00, DEG, DENANY, DENMAX 9999, DECM, LinF, PROFRC, RDX**.**, REGS 99, SCIOVR, SSIZE4, WSIZE 64, Y.MD. See these commands for more information. RESET is not programmable.
RJ	h X.FCN …	Integer	Right adjusts, in analogy to LJ on *HP-16C*.
RL	h X.FCN RL ***n***	Integer	Works like ***n*** consecutive RLs / RLCs on *HP-16C*. For RL, 1 ≤ ***n*** ≤ 63. For RLC, 1 ≤ ***n*** ≤ 64. RL 0 and RLC 0 execute as NOP.
RLC	h X.FCN RLC ***n***		

Name	Keys to press	in modes	Remarks
RM	h MODE …	\α	Sets the floating point rounding mode. This is for numerical mathematics geeks, since it is only used when converting from the high precision internal format to packed real numbers. It will not alter the display nor change the behavior of ROUND. The following modes are supported: 0: round half even: ½ = 0.5 rounds to next even number (default). 1: round half up: 0.5 rounds up ("business-man's rounding"). 2: round half down: 0.5 rounds down. 3: round up: away from 0. 4: round down: towards 0 (truncates). 5: ceiling: rounds towards +∞. 6: floor: rounds towards –∞.
RM?	h TEST …	\α	Returns the floating point rounding mode set. See RM for more.
RMDR	h RMDR	\α	Equals RMD on *HP-16C*.
ROUND	g RND	DECM	Rounds ***x*** using the current display format, like RND in *HP-42S*.
		FRC	Rounds ***x*** using the current denominator, like RND in *HP-35S* fraction mode.
ROUNDI	h X.FCN …	DECM	Rounds ***x*** to next integer. ½ rounds to 1.
RR	h X.FCN RR ***n***	Integer	Works like ***n*** consecutive RRs / RRCs on *HP-16C*. See RL / RLC for more.
RRC	h X.FCN RRC ***n***		
RTN	g RTN	PRG	Last command in a routine. Pops the local data (like PopLR) and returns control to the calling routine in program execution, i.e. moves the program pointer one step behind the most recent XEQ instruction encountered. If there is none, program execution halts and the program pointer is set to step 000.
RTN+1	h P.FCN …	PRG	Works like RTN, but moves the program pointer to the second line following the most recent XEQ instruction encountered. Halts if there is none.
R.CLR	h P.FCN …	DECM	Interprets ***x*** in the form `ss.nn`. Clears ***nn*** registers starting with number ***ss***. E.g. for ***x*** = 34.56, R-CLR will clear **R34** through **R89**.

Name	Keys to press	in modes	Remarks
R.COPY	h P.FCN …	DECM	Interprets ***x*** in the form `ss.nndd`. Takes ***nn*** registers starting with number ***ss*** and copies their contents to ***dd*** etc. E.g. for ***x*** = 7.0345678, ***r07, r08, r09*** will be copied into **R45, R46, R47**, respectively. For ***x*** < 0 , R-COPY will take ***nn*** registers from flash memory instead, starting with register number |***ss***| there.
R.SORT	h P.FCN …	DECM	Interprets ***x*** in the form `ss.nn`. Sorts the contents of ***nn*** registers starting with number ***ss*** . Assume ***x*** = 49.036 , ***r49*** = 1.2 , ***r50*** = –3.4 , and ***r51*** = 0 ; then R-SORT will return ***r49*** = -3.4 , ***r50*** = 0 , and ***r51*** = 1.2.
R.SWAP	h P.FCN …	DECM	Works like R-COPY but swaps the contents of source and destination registers.
R→D		DECM	Please see the *catalog of conversions below* for conversions of radians to degrees.
R↑	h R↑	\α	Rotates the stack contents one level up or down, respectively. See *above* for details.
R↓	R↓		
s	g s	DECM	Takes the statistical sums accumulated, calculates the sample standard deviations s_y and s_x and pushes them on the stack.
SAVE	h P.FCN …	\α	Saves user program space, registers and system state to flash memory. Program space is stored in segment 0. Registers and system state are in their own special region. **WARNING:** Do not use SAVE in program loops! Else you might kill your flash memory very fast (see *above*).
SB	h X.FCN SB ***n***	Integer	Sets the specified bit in ***x*** .
SCI	h SCI ***n***	\α	Sets scientific display format.
SCIOVR	h SCI ENTER↑	\α	Numbers exceeding the range displayable in ALL or FIX will be shown in scientific format (default as in vintage HP calculators). Compare ENGOVR.
SDL	h X.FCN SDL ***n***	DECM	Shifts digits left by ***n*** decimals, equivalent to multiplying ***x*** by 10^n .

Name	Keys to press	in modes	Remarks
SDR	h X.FCN SDR ***n***	DECM	Shifts digits right by ***n*** decimals, equivalent to dividing x through 10^n.
SEED	h STAT …	DECM	Stores a seed for random number generation.
SENDA	h P.FCN …	\α	Sends all RAM data … … via serial I/O to the device connected. See *Appendix A* below for more.
SENDL	h P.FCN SENDL ***n***	\α	Sends the library file of region ***n*** into RAM …
SENDP	h P.FCN …	\α	Sends the user program memory …
SENDR	h P.FCN …	\α	Sends the general purpose registers 00 to 99 …
SERR	h STAT …	DECM	Works like s but pushes the standard errors $s/\sqrt{n}$ on the stack (i.e. the standard deviations of $\bar{x}$ and $\bar{y}$).
SERRw	h STAT …	DECM	Works like sw but returns the standard error $s/\sqrt{\sum y_i}$ (i.e. the standard deviation of $\bar{x}$w).
SETCHN	h MODE …	\α	Sets some regional preferences (see above).
SETDAT	h X.FCN …	DECM	Sets the date for the real time clock (doesn't work with the emulator, since the emulator takes this information from the PC clock).
SETEUR SETIND	h MODE …	\α	Set some regional preferences (see above).
SETTIM	h X.FCN …	DECM	Sets the time for the real time clock (doesn't work with the emulator, since the emulator takes this information from the PC clock).
SETUK SETUSA	h MODE …	\α	Set some regional preferences (see above).
SF	h P.FCN SF ***n***	\α	Sets the flag specified.
SIGN	h X.FCN …	\α	Returns 1 for $x > 0$, –1 for $x < 0$, and 0 for $x = 0$ or non-numbers.
	CPX X.FCN …	DECM	Returns the unit vector of $x + i\,y$ in **X** and **Y**.
SIGNMT	h MODE …	\α	Sets sign-and-mantissa mode for integers.
SIN	f SIN	DECM	Returns the sine of the angle in **X**.

Name	Keys to press	in modes	Remarks
SINC	h X.FCN …	DECM	Returns $\frac{\sin(x)}{x}$.
SINH	f HYP SIN	DECM	Returns the hyperbolic sine of x.
SKIP	h P.FCN SKIP ***n***	PRG	Skips ***n*** program steps forwards ($1 \leq n \leq 99$). So e.g. SKIP 02 skips over the next two steps, going e.g. from step 123 to step 126. If the skip would land beyond the end of occupied program memory, the same will happen as if a RTN had been encountered.
SL	h X.FCN SL ***n***	Integer	Works like ***n*** (up to 63) consecutive SLs on *HP-16C*. SL 0 executes as NOP.
SLOW	h MODE …	All	Sets the processor speed to "slow". This is also entered for low battery voltage.
SLV	f SLV ***label***	DECM	Solves the equation $f(x) = 0$, with $f(x)$ calculated by the routine specified. Two initial estimates of the root must be supplied in **X** and **Y** when calling SLV. For the rest, the user interface is as in *HP-15C*. This also means SLV acts as a test, so the next program step will be skipped if SLV failed to find a root.
SLVQ	h X.FCN …	DECM	Solves the quadratic equation $ax^2 + bx + c = 0$, with the real parameters put on the stack [***c, b, a***, …] , and tests the result. • If $r := b^2 - 4ac \geq 0$, SLVQ returns $-\frac{b \pm \sqrt{r}}{2a}$ in **Y** and **X**. In a program, the step after SLVQ will be executed. • Else, SLVQ returns the real part of the first complex root in **X** and its imaginary part in **Y** (the 2nd root is the conjugate of the first – see CONJ). If run directly from the keyboard, the complex indicator is lit then – in a program, the step after SLVQ is skipped. In either case, ***r*** is returned in **Z**. Higher stack levels are kept unchanged. **L** contains ***c***.
SMODE?	h TEST …	\α	Returns the integer sign mode set, i.e. 2 (meaning "true") for 2's complement, 1 ("true" again) for 1's complement, 0 (i.e. "false") for unsigned, or -1 (i.e. "true") for sign and mantissa mode.
SPEC?	h TEST …	\α	True if x is special, i.e. infinity or NaN.

Name	Keys to press	in modes	Remarks
SR	h X.FCN SR ***n***	Integer	Works like ***n*** consecutive SRs on *HP-16C*. SR 0 executes as NOP.
SSIZE4	h MODE …	\α	Sets the stack size to 4 or 8 levels, respectively. See *above*. Please note register contents will remain unchanged in this operation. The same will happen if stack size is changed via RCLM.
SSIZE8			
SSIZE?	h TEST …	\α	Returns the number of stack levels accessible.
STATUS	h STATUS	\PRG	Shows the status of all user flags, similar to STATUS on *HP-16C*. See *above*.
STO	STO ***d***	\α	See the *addressing table above* for ^{C}STO.
STOM	STO MODE ***s*** h P.FCN STOM ***s***	\α	Stores mode settings for later use as described *above*. Take RCLM to recall them.
STOP	R/S	PRG	Stops program execution. May be used to wait for an input, for example.
STOS	h P.FCN STOS ***d***	\α	Stores all stack levels in a set of 4 or 8 registers, starting at destination ***d***.
STO+	STO + ***d***	\α	Executes the specified operation on the content of address ***d*** and stores the result into said address. E.g. STO–12 subtracts x from $r12$ like the sequence RCL 12 x⇄y − STO 12 does, but without touching the stack at all. See the *addressing table above* for ^{C}STO.
STO–	STO − ***d***		
STO×	STO × ***d***		
STO/	STO / ***d***		
STO↑	STO ▲ ***d***	\α	STO↑ (↓) takes the maximum (minimum) of the values in ***d*** and $\mathbf{X}$ and stores it.
STO↓	STO ▼ ***d***		
SUM	h STAT …	DECM	Recalls the linear sums Σy and Σx. Useful for elementary vector algebra in 2D.
sw	h STAT …	DECM	Returns the standard deviation for weighted data $$s_w = +\sqrt{\frac{\sum y_i \cdot \sum\left(y_i \cdot x_i^2\right) - \left[\sum\left(y_i \cdot x_i\right)\right]^2}{\left(\sum y_i\right)^2 - \sum y_i^2}}$$ with the weights entered in y via Σ+.

Name	Keys to press	in modes	Remarks
sxy	h CFIT …	DECM	Returns the sample covariance for two data sets. It depends on the fit model selected. For LinF, it returns $$s_{xy} = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{n\cdot(n-1)}.$$ See COV for the population covariance.
TAN	f TAN	DECM	Returns the tangent of the angle in **X**.
TANH	f HYP TAN	DECM	Returns the hyperbolic tangent of x.
TICKS	h P.FCN …	\α	Returns the number of ticks from the real time clock at execution time. With the quartz built in, 1 tick = 0.1 s. Without, it may be 10% more or less. So the quartz is inevitable prerequisite for the clock being useful in medium to long range.
TIME	h P.FCN …	DECM, α	Recalls the time from the real time clock at execution, displaying it in the format `hh.mmssdd` in 24h-mode. Chose FIX 6 for best results.
T_n	h X.FCN …	DECM	Chebychev's (a. k. a. Čebyšev, Tschebyschow, Tschebyscheff) polynomials of first kind $T_n(x)$ with n in **Y**, solving the differential equation $$\left(1-x^2\right)y''-x\cdot y'+n^2 y=0.$$
TOP?	h TEST …	PRG	Executes the next step only if it is called from a program that isn't a subroutine, i.e. if the program running flag is set and the return stack pointer points to an empty stack.
TRANSP	h MATRIX …	DECM	Takes a matrix *descriptor* in **X** and returns the descriptor of its transpose. The transpose is done in-situ and does not require any additional registers or storage.
$t_P(x)$ $t(x)$ $t^{-1}(p)$	h PROB …	DECM	Student's t distribution. t(x) equals 1 – Q(t) in *HP-21S*. The degrees of freedom are stored in **J**. $t_P(x)$ denotes the respective *pdf*.
U_n	h X.FCN …	DECM	Chebychev's polynomials of second kind $U_n(x)$ with n in **Y**, solving the differential equation $$\left(1-x^2\right)y''-3x\cdot y'+n(n+2)y=0.$$
UNSIGN	h MODE …	\α	Sets unsigned mode for integers.
VERS	h X.FCN …	\PRG	Shows your firmware version and build number.

Name	Keys to press	in modes	Remarks
VIEW	h VIEW <u>**s**</u>	\α	Displays the content of address ***s*** until the next key is pressed. See *below* for more.
VWα+	h VIEW <u>**s**</u>	α	Displays the alpha register in the top line plus the contents of address ***s*** in the bottom line until the next key is pressed. See *below* for more.
W	h X.FCN …	DECM	Returns Lambert's W for given $x \geq -1/e$.
WDAY	h X.FCN …	DECM	Takes ***x*** as a date in the format selected and returns the name of the day in the dot matrix and a corresponding integer in the numeric display (Monday = 1, Sunday = 7).
W^{-1}	h X.FCN …	DECM	Returns ***x*** for given W (≥ -1). See W above.
Weibl			Weibull distribution with the shape parameter ***b*** in **J** and the characteristic lifetime ***T*** in **K**:
Weibl$_P$	h PROB …	DECM	Weibl$_P$ [24] returns $f_W(t) = \frac{b}{T}\left(\frac{t}{T}\right)^{b-1} e^{-\left(\frac{t}{T}\right)^b}$,
Weibl $^{-1}$			Weibl returns $F_W(t) = 1 - e^{-\left(\frac{t}{T}\right)^b}$. Weibl $^{-1}$ returns the survival time t_s for given probability F_W , ***b*** in **J** and ***T*** in **K**.
WSIZE	h MODE WSIZE <u>***n***</u>	\α	Works like on *HP-16C*, but with the parameter following the command instead of taken from **X**. Reducing the word size truncates the values in the stack registers employed, including **L**. WSIZE 0 sets the word size to maximum, i.e. 64 bits.
WSIZE?	h TEST …	\α	Recalls the word size set.
x^2	g x^2	\α	
XEQ	XEQ <u>***label***</u>	PRG	Calls the respective subroutine.
		\PRG, \α	Executes the respective program.
	A , B , C , or D (you may need f for reaching these hotkeys in integer bases >10.)	PRG	Calls the respective subroutine, so e.g. XEQ C will be inserted when C is pressed.
		\PRG, \α	Executes the respective program if defined.

[24] The pdf equals WEIBULL(***x; b; T;* 0**) and the cdf WEIBULL(***x; b; T;* 1**) in MS Excel.

Name	Keys to press	in modes	Remarks
XEQα	h P.FCN …	\α	Takes the first three characters of ***alpha*** (or less if there are less) as a label and calls or executes the respective routine.
XNOR	h X.FCN …	\α	Works in analogy to AND.
XOR	h XOR	\α	Works in analogy to AND.
x̄	f x̄	DECM	Returns the arithmetic means, pushing $\bar{y} = \frac{1}{n}\sum y$ and $\bar{x} = \frac{1}{n}\sum x$ on the stack. See also s, SERR, and σ.
x̄g	h STAT …	DECM	Returns the geometric means, pushing $\bar{y}_g = \sqrt[n]{\prod y} = e^{\frac{1}{n}\sum \ln y}$ and $\bar{x}_g = \sqrt[n]{\prod x}$ on the stack. See also ε, ε_m, and ε_P.
x̄w	h STAT …	DECM	Returns the weighted arithmetic mean $\sum xy \big/ \sum y$. See also sw and SERRw.
$\sqrt[x]{y}$	h X.FCN $\sqrt[x]{y}$	\α	
x̂	h CFIT …	DECM	Returns a forecast ***x*** for a given ***y*** (in **X**) following the fit model chosen. See L.R. for more.
x !	h !	DECM	Return the factorial, equaling *Γ(x + 1)*.
x → α	h X.FCN …	α	Interprets ***x*** as character code. Appends the respective character to ***alpha***, similar to XTOA in *HP-42S*.
x⇆	h x⇆ ***r***	\α	Swaps the contents of **X** and ***r***, in analogy to ***x⇆y***.
x⇆y	x⇆y	\α	Swaps ***x*** and ***y***, performing Re⇆Im if a complex operation was executed immediately before. See above for details.

Name	Keys to press	in modes	Remarks
x < … ?	h TEST x < ? ***a***	\α	Compare ***x*** with ***a***. E.g. h TEST x < ? K will compare ***x*** with the contents of register **K** , and will be listed as x<K? in a program. See the examples given in the *addressing table above* for more. x ≈ ? will be true if the rounded values of ***x*** and ***a*** are equal (see ROUND). CPX f x = ? ***a*** and CPX g x ≠ ? ***a*** compare the complex number ***x*** + *i* ***y*** as explained in the *addressing table above*. The signed tests x = +0 ? and x = –0 ? are meant for integer modes 1COMPL and SIGNMT, and for DECM if flag **D** is set. Then, e.g. 0 divided by -7 will display -0.
x ≤ … ?	h TEST x ≤ ? ***a***		
x = … ?	f x = ? ***a***		
x = +0 ?	h TEST x=+0?		
x = –0 ?	h TEST x=–0?		
x ≈ … ?	h TEST x ≈ ? ***a***		
x ≠ … ?	g x ≠ ? ***a***		
x ≥ … ?	h TEST x ≥ ? ***a***		
x > … ?	h TEST x > ? ***a***		
YEAR	h X.FCN …	DECM	Assumes ***x*** containing a date in the format selected and extracts the year.
y^x	f y^x	\α	In integer modes ***x*** must be ≥ 0.
	C	\α & \(13, 14, 15, h)	Shortcut working as long as label C is not defined yet.
ŷ	f ŷ	DECM	Returns a forecast ***y*** (in **X**) for a given ***x*** following the fit model chosen. See L.R. for more.
Y.MD	h MODE …	\α	Sets the format for date display.
y⇆	f y⇆ ***r***	\α	Swaps the contents of **Y** or **Z** and ***r*** , in analogy to ***x***⇆.
z⇆	g z⇆ ***r***		
αDATE	h X.FCN …	\integer	Takes ***x*** as a date and appends it to ***alpha*** in the format set. See DATE. – To append a date stamp to ***alpha***, call DATE αDATE.
αDAY	h X.FCN …	\integer	Takes ***x*** as a date, recalls the name of the respective day and appends its first 3 letters to ***alpha***.
αGTO	h P.FCN αGTO ***nn***	\α	Takes the contents of **R*nn*** as character code. Takes the first three characters of the converted code (or less if there is only less) as an alpha label and positions the program pointer to it.
αIP	h X.FCN …	All	Appends the integer part of ***x*** to ***alpha***, similar to AIP in *HP-42S*.

Name	Keys to press	in modes	Remarks
αLENG	h X.FCN	All	Returns the number of characters found in ***alpha***, like ALENG in *HP-42S*.
αMONTH	h X.FCN …	\integer	Takes *x* as a date, recalls the name of the respective month and appends its first 3 letters to ***alpha***.
αOFF	h P.FCN ...	PRG & α	Work like AOFF and AON in *HP-42S*, turning alpha mode off and on.
αON	h P.FCN ...	PRG & \α	
αRCL	f RCL ***s***	α	Interprets the content of the source ***s*** as characters and appends them to ***alpha***.
	h X.FCN αRCL ***s***	\α	
αRC#	h X.FCN αRC# ***s***	All	Takes the content of ***s*** as a number, converts it to a string in the format set, and appends this to ***alpha***. If e.g. ***s*** = 1234 and ENG 2 and RDX**.** are set, then ␣1.23E3 will be appended.
αRL	h X.FCN αRL ***n***	All	Rotates ***alpha*** by ***n*** characters like AROT in *HP-42S*, but with ***n*** ≥ *0* and the parameter trailing the command instead of taken from **X**. αRL 0 executes as NOP.
αRR	h X.FCN αRR ***n***	All	Works like αRL but rotates to the right.
αSL	h X.FCN αSL ***n***	All	Shifts the ***n*** leftmost characters out of ***alpha***, like ASHF in *HP-42S*. αSL 0 equals NOP.
αSR	h X.FCN αSR ***n***	All	Works like αSL but takes the ***n*** rightmost characters instead.
αSTO	f STO ***d***	α	Stores the first (i.e. leftmost) 6 characters in the alpha register into destination ***d*** .
	h X.FCN αSTO ***d***	\α	
αTIME	h X.FCN ...	\integer	Takes *x* as a decimal time and appends it to ***alpha*** in the format `hh:mm:ss` according to the time mode selected. See TIME. – To append a time stamp to ***alpha***, call TIME αTIME.
αVIEW	h VIEW α	All	Displays ***alpha*** in the top line and - - - in the bottom line until the next key is pressed. See *below* for more.
	h P.FCN ...		
	h X.FCN ...		

Name	Keys to press	in modes	Remarks
αXEQ	h P.FCN αXEQ ***nn***	\α	Takes the contents of **R*nn*** as character code. Interprets the first three characters (or less if there are only less) of the converted code as an alpha label and calls or executes the respective routine.
α → x	h X.FCN ...	α	Returns the character code of the leftmost character in ***alpha*** and deletes this character, like ATOX in *HP-42S*.
β	h X.FCN β	DECM	Returns Euler's Beta $B(x,y)=\frac{\Gamma(x)\cdot\Gamma(y)}{\Gamma(x+y)}$ with $\mathrm{Re}(x)>0$, $\mathrm{Re}(y)>0$. Called β here for avoiding ambiguities.
Γ	h X.FCN Γ	DECM	Returns $\Gamma(x)$. Additionally, h ! calls $\Gamma(x+1)$.
ΔDAYS	h X.FCN ...	DECM	Assumes **X** and **Y** containing dates in the format chosen and calculates the number of days between them. Works like in *HP-12C*.
Δ%	g Δ%	DECM	Returns $100\cdot\frac{x-y}{y}$ like %CH in *HP-42S*.
ε	h STAT ε	DECM	Calculates the scattering factors (or geometric standard deviations) for lognormally distributed data $\ln(\varepsilon_y)=\sqrt{\frac{\sum\ln^2(y)-2n\cdot\ln(\bar{y}_G)}{n-1}}$ and $\ln(\varepsilon_x)$ and pushes them on the stack. This ε works for the geometric mean x̄g in analogy to s for the arithmetic mean x̄ but <u>multiplicative</u>.
εm	h STAT εm	DECM	Works like ε but pushes the scattering factors of the geometric means $\varepsilon_m=\varepsilon^{1/\sqrt{n}}$ on the stack.
ε p	h STAT ε p	DECM	Works like ε but with a denominator ***n*** instead of ***n – 1***, returning the scattering factors of the populations.
ζ	h X.FCN ζ	DECM	Returns Riemann's Zeta function for real arguments, with $\zeta(x)=\sum_{n=1}^{\infty}\frac{1}{n^x}$ for ***x*** > *1* and its analytical continuation for ***x*** < *1* : $\zeta(x)=2^x\pi^{x-1}\sin\left(\frac{\pi}{2}x\right)\cdot\Gamma(1-x)\cdot\zeta(1-x)$. This lives in XROM, may be less accurate.

Name	Keys to press	in modes	Remarks
π	h π	DECM	Complex version copies **π** in **X** and clears **Y**.
Π	f π ***label***	DECM	Computes a product with the routine specified by ***label***. Initially, **X** contains the loop control number in the format `cccccc.fffii` and the product is set to 1. Each run through the routine specified computes a factor. At its end, this factor is multiplied with said product; the operation then decrements `ccccccc` by `ii` and runs said routine again if then `ccccccc` ≥ `fff`, else returns the resulting product in **X**.
Σ	g Σ ***label***	DECM	Computes a sum with the routine specified by ***label***. Initially, **X** contains the loop control number in the format `cccccc.fffii` and the sum is set to 0. Each run through the routine specified computes a summand. At its end, this summand is added to said sum; the operation then decrements `ccccccc` by `ii` and runs said routine again if then `ccccccc` ≥ `fff`, else returns the resulting sum in **X**.
σ	h STAT ...	DECM	Works like s but returns the standard deviations of the populations instead.
Σln²x Σln²y Σlnx Σlnxy Σlny Σxlny Σylnx	h STAT …	DECM	Recall the respective statistical sums. These sums are necessary for curve fitting models beyond pure linear. Calling them by name enhances readability of programs significantly.
σw	h STAT ...	DECM	Works like sw but returns the standard deviation of the population instead. $\sigma_w = +\sqrt{\frac{\sum y_i (x_i - \bar{x}_w)^2}{\sum y_i}}$

Name	Keys to press	in modes	Remarks
Σx	h STAT …	DECM	Recall the respective statistical sums. These sums are necessary for basic statistics and linear curve fitting. Calling them by name enhances readability of programs significantly.
Σx^2			
Σx^2y			
Σxy			
Σy			
Σy^2			
Σ+	h Σ+	DECM	Adds a data point to the statistical sums.
	A	DECM	Shortcut as long as label A is not defined yet.
Σ–	h Σ–	DECM	Subtracts a data point from the statistical sums.
φ(x)	h PROB …	DECM	Standard normal *pdf*: $\varphi(x)=\frac{1}{\sqrt{2\pi}}e^{-x^2/2}$.
Φ(x)	f Φ	DECM	Standard normal *cdf* $\Phi(z)=\int_{-\infty}^{z}\varphi(x)dx$, equals 1 – Q in *HP-32E* and 1 – Q(z) in *HP-21S* with $z = x$.
Φ^{-1}(p)	g Φ^{-1}		
χ^2	h PROB χ^2 etc.	DECM	Chisquare distribution. The *cdf* χ^2 (with the degrees of freedom given in **J**) equals 1 – Q(χ^2) in *HP-21S*.
χ^2INV			
$\chi^2{}_P$			
(-1)^x	h X.FCN (-1)^x	\α	For x not being a natural number, this function will return $\cos(\pi \cdot x)$.
+	+	\α	Returns $y + x$.
–	–	\α	Returns $y - x$.
×	×	\α	Returns $y \cdot x$.
/	/	\α	Returns y / x .
+/–	+/–	\α	Unary minus like CHS in *HP-35*.
→DEG	→ DEG	DECM	Takes x as an angle in the angular mode currently set and converts it to degrees. Prefix g may be omitted.
→GRAD	→ GRAD	DECM	Like →DEG, but converts to gon or grads.

Name	Keys to press	in modes	Remarks
→H	➡ f H.d	DECM	Takes x as hours or degrees in the format `hhhh.mmssdd` and converts them into a decimal time or angle.
→H.MS	➡ f H.MS	DECM	Takes x as decimal hours or degrees and converts them into `hhhh.mmssdd` as in vintage HPs. For calculations, use H.MS+ or H.MS– then or reconvert to decimal values before.
→POL	g →P	DECM	Assumes **X** and **Y** containing 2D Cartesian coordinates (x, y) of a point and converts them to the respective polar coordinates (r, ϑ) with the radius $r = \sqrt{x^2 + y^2}$
→RAD	➡ RAD	DECM	Works like →DEG, but converts to radians.
→REC	f R←	DECM	Assumes **X** and **Y** containing 2D polar coordinates (r, ϑ) of a point and converts them to the respective Cartesian coordinates (x, y).
%	f %	DECM	Returns $\frac{x \cdot y}{100}$, leaving **Y** unchanged.
%MG	h X.FCN h % MG	DECM	Returns the margin [25] $100 \cdot \frac{x - y}{x}$ in % for a price x and cost y, like %MU-Price in *HP-17B*.
%MRR	h X.FCN h % MRR	DECM	Returns the mean rate of return in percent per period, i.e. $100 \cdot \left[\left(\frac{x}{y} \right)^{1/z} - 1 \right]$ with x = future value after z periods, y = present value. For $z = 1$, Δ% returns the same result easier.
%T	h X.FCN h % T	DECM	Returns $100 \cdot \frac{x}{y}$, interpreted as % of total.

[25] Margin corresponds to „Handelsspanne“ in German.

Name	Keys to press	in modes	Remarks
%Σ	h STAT h % Σ	DECM	Returns $100 \cdot \frac{x}{\sum x}$.
	h X.FCN h % Σ		
%+MG	h X.FCN h % +MG	DECM	Calculates a sales price by adding a margin of x % to the cost y, as %MU-Price does in *HP-17B*. Formula: $\frac{y}{1-\frac{x}{100}}$
√‾	f √x	\α	
	D	\α, \14, \15, \h	Shortcut working as long as label D is not defined yet.
∫	g ∫ ***label***	DECM	Integrates the function given in the routine specified. Lower and upper integration limits must be supplied in **Y** and **X**, respectively. Otherwise, the user interface is as in *HP-15C*.
∞?	h TEST ...	\α	Tests x for infinity.
||	g ||	DECM	Returns $\left(\frac{1}{x}+\frac{1}{y}\right)^{-1}$.

Alphanumeric input:

Character	Keys to press	in modes	Remarks
␣	h PSE	α	Appends a blank space to ***alpha***.
°	.	DECM	Separates degrees or hours from minutes and seconds, so input format is `hhhh.mmssdd`. The user has to take care where an arbitrary real number represents such an angle or time.
0 … 9	0 … 9	\α	Standard numeric input. For integer bases <10, input of illegal digits is blocked. Please note you cannot enter more than 12 digits in the mantissa.
		in addressing	Register input. See the *tables* above for more.
	0, 1, f 2, … , f 9	α	Appends the respective digit to ***alpha***.
A … F	A … F (grey print)	11, 12, 13,	Numeric input for digits >10. See page 6 for

Character	Keys to press	in modes	Remarks
		14, 15, h	more information.
A … Z	A … Z (grey print)	in addressing	Register input. See the *addressing tables* above for the letters applicable.
		α	Appends the respective Latin letter to ***alpha***. Use f ↑ to toggle cases.
EEX	EEX	DECM & \FRACT	Works like E in the Pioneers.
Α … Ω	g A … g O (grey print)	α	Appends the respective Greek letter to ***alpha***. f ↑ will toggle cases. See page 7 for more.
(	f ◄(	α	Appends the respective symbol to ***alpha***.
)	g)►		
+	f +		
–	f −		
x	f ×		
/	Second .	DECM	A persistent 2^{nd} . in input switches to fraction mode. It will be interpreted as explained below. Please note you cannot enter EEX after you entered . twice – but you may delete the 2^{nd} dot while editing the input line.
		FRC	First . is interpreted as a space, 2^{nd} as a fraction mark. E.g. input of 2 . 3 . 4 results in 2 ¾ in the display. Improper fractions may be entered starting with a . , e.g. . 3 . 2 .
	f /	α	Appends a slash to ***alpha***.
±	f +/–	α	Appends the respective symbol to ***alpha***.
,	h ./, XEQ		
.	f .		
'.' or ','	.	DECM	Inserts a radix mark as selected.
!	h !	α	Appends the respective symbol to ***alpha***.
?	h Q (grey print)		
⇆	h x⇆		
≠	h XOR		

Character	Keys to press	in modes	Remarks
%	h −		
$	h S (grey print)		
€	h U (grey print)		
£	h P (grey print)		
¥	h Y (grey print)		
√	f D		
&	h AND		
\	h NOT		
|	h OR		

Non-programmable Control, Clearing and Information Commands

Keys to press	in modes	Remarks
⬅ [26]	Input pending	Deletes the last digit or character put in.
	α	Deletes the rightmost character in ***alpha***.
	PRG	Deletes current step.
	Else	Acts like CLx.
▲ / ▼ [27]	Status open	Goes to previous / next set of flags.
	Catalog open	Goes to previous / next item in this catalog.
	α	Scrolls the display window six characters to the left / right in ***alpha*** if possible. If less than six characters are beyond the limits of the display window on the left / right side, the window will be positioned to the beginning / end of string. Useful for longer strings.
	Else	Acts like BST / SST in *HP-42S*. I.e. browses programs in PRG. Out of PRG, SST will also execute the respective program step.
f ◀ / g ▶	Integer	Shifts the display window to the left / right like in *HP-16C*. Helpful while working with small bases.
f ⇧	α	Toggles upper and lower case (indicated by ⬇).
	\α	Enters a memory browser.
h P.FCN DEL$_P$ ***label***	PRG	Deletes program steps from the current position downstream until, but excluding, the label specified. If said label is not found, “No such label” will be thrown. If the program pointer is on the step containing said label, nothing will be deleted.
ENTER↑	Catalog open	Selects the current item like XEQ below.
	α	Turns alpha mode off.
	Else	Acts like the command ENTER described above.

---

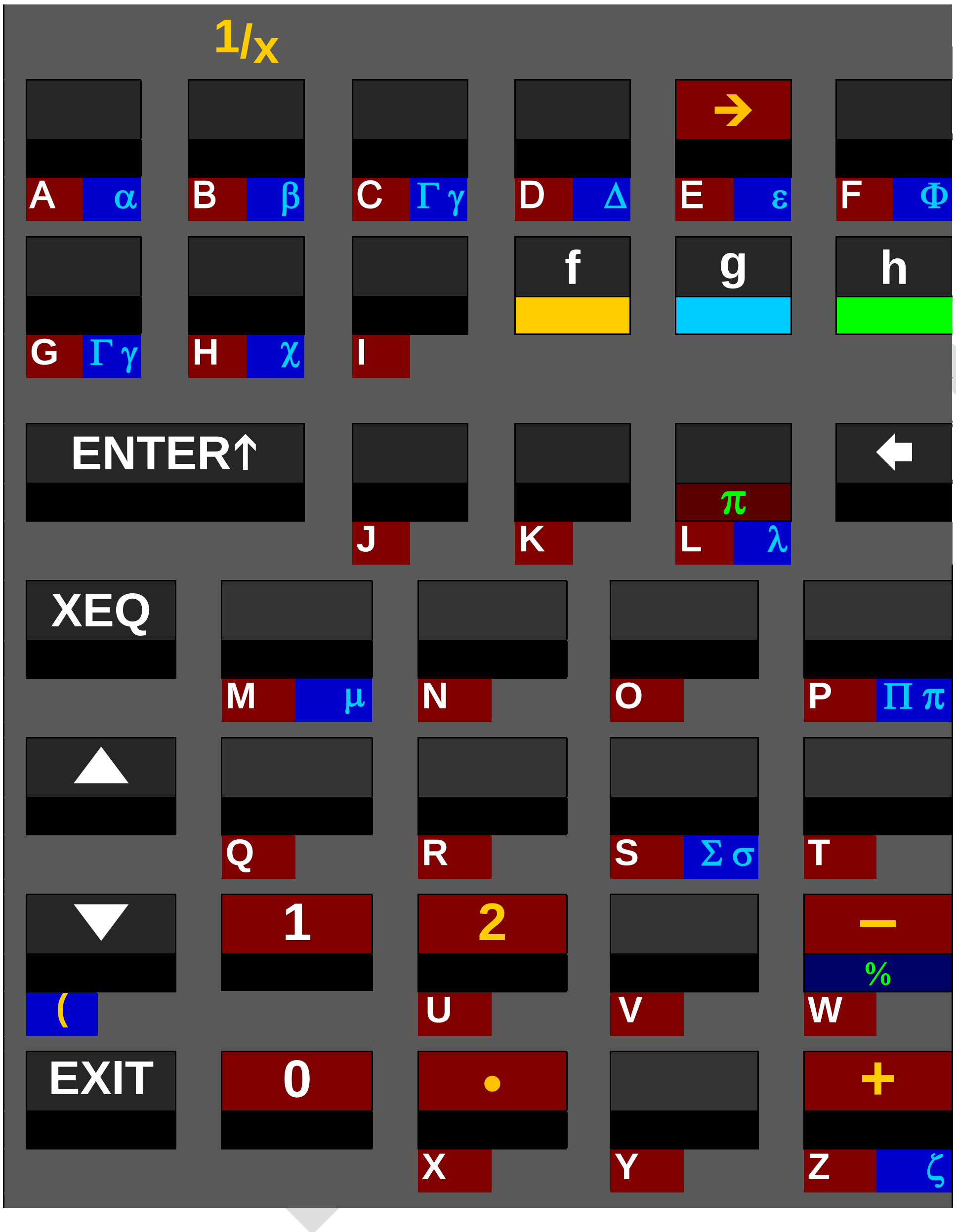
[26] The mode conditions specified will be checked top down for this command:
<u>If</u> there is a pending input, the last digit / character entered will be deleted;
<u>else</u> if alpha mode is set, the last character of ***alpha*** will be deleted;
<u>else</u> if the WP 34S is in programming mode, the current step will be deleted;
<u>else</u> CLx will be called. Period.
This method holds for all commands listed here using this symbolic.

[27] These two navigation keys will repeat with 5Hz when held down for longer than 0.5s. Out of PRG, however, SST will not repeat.

Keys to press	in modes	Remarks
EXIT	Catalog open	Leaves the catalog without executing anything.
	Input pending	Cancels the execution of pending operations, returning to the calculator status as it was before.
	Program running	Stops the running program like R/S . See below.
	PRG	Leaves programming mode like P/R . See below.
	α	Turns alpha mode off like ENTER↑ . See above.
	Else	Does nothing.
h OFF	\PRG	Turns calculator off.
ON	Calculator off	Turns calculator on.
	Else	There are several ON-key combinations available. See below for more.
h P/R	\α	Toggles programming mode for keyboard entry.
R/S	Program running	Stops the program execution immediately. “Stopped” will be shown in the upper row until the next keystroke.
	\PRG & \α	Runs the current program or resumes its execution starting with the current step.
	α	Appends an ‘Y’ to ***alpha***.
	PRG	Acts like the command STOP described above.
h SHOW	DECM & \PRG	Shows the full mantissa until the next key is pressed. See above.
	PRG	Displays a CRC checksum of program memory contents, allowing validation of program integrity.
XEQ	Catalog open	Selects the item currently displayed and exits, executing the respective command. See below.
	Else	Acts like the command XEQ described above.
f α	\α	Turns on alpha mode for keyboard entry. When entering alpha constants in programs, please note there is no concatenation character – added characters are appended to ***alpha*** always. For starting a new string, use CLα first. Alpha constants will be listed like e.g. ‘Test 1’.

CATALOGS

A catalog on your WP 34S is a collection of items, e.g. operations or characters. Opening a catalog will set alpha mode to allow for typing the first character(s) of the item wanted. A subset of the full alpha keyboard shown above is sufficient for browsing:



f B (= 1/x) allows for easy reverse conversions in CONV as described below.

f → just calls the character → while browsing a catalog.

▲ and ▼ browse the open catalog.

ENTER↑ or XEQ select the item displayed, recall or execute it, and exit the catalog.

EXIT leaves the catalog without executing anything, i.e. cancels the catalog call.

See below for some examples.

Such catalogs may be called using the keystrokes listed below:

Keys to press	in modes	Contents of said catalog
h CAT	\α	Defined alpha labels. Some special rules apply here: ▲ and ▼ browse the catalog as usual, but in the numeric line the location of the respective label is indicated (RAM, Lib for XROM, or SEG ***n*** for flash memory segment ***n***). 0 – 9 trigger a search starting in the flash segment specified (and continued in further segments as long as necessary) for the first alpha label defined. ENTER↑ goes to the alpha label as displayed, while XEQ or R/S execute it. These keystrokes will perform a label search as described *above*. Labels in XROM cannot be accessed by ENTER↑ . . goes to the first alpha label in XROM. ◄ or EXIT leave CAT returning to the state as it was before.
h CFIT	DECM	Curve fit functions.
h CONST	DECM	Constants like in HP35s. Picking a constant will recall it. See the constants listed in a *table below*.
CPX CONST	DECM	This catalog contains the same constants as in real domain. Picking one, however, does a complex recall here. So, if the stack did look like [***x***, ***y***, …] before calling CONST, it will contain [***constant***, **0**, ***x***, ***y***, …] thereafter.
h CONV	DECM	Conversions as listed in a *table below*.
f CPX	α	"Complex" letters mandatory for many languages. Case is determined by setting (see f ↑ above).
h MATRIX	DECM	Matrix operations library.
h MODE	\α	Mode setting functions.
h PROB	DECM	Extra probability distributions.
h P.FCN	\α	Extra programming and I/O functions.
f R↓	α	Subscripts.
h R↑	α	Superscripts.
h STAT	DECM	Extra statistical functions.
h TEST	\α	All tests except the two on the keyboard.
	α	Comparison symbols and brackets, except f (and g).

Keys to press	in modes	Contents of said catalog	
h X.FCN	DECM	Extra real functions.	These three catalogs are merged in mode PRG to ease programming.
	Integer	Extra integer functions.	
	α	Extra alpha functions.	
CPX X.FCN	DECM	Extra complex functions.	
h ./.	α	Punctuation marks and text symbols.	
f →	α	Arrows and mathematical symbols.	

Reopening the very last catalog called, the last command selected therein is displayed for easy repetitive use.

See the *table below about addressing cataloged items*, and the next pages for detailed item lists of the various catalogs. Within each catalog, items are sorted alphabetically (see *above* for the sorting order). You may access particular items fast and easily by typing the first characters of their names. See *below* for some examples and constraints.

A single function, e.g. CB, may be contained in more than one catalog.

The alpha catalogs are found three pages below. See also the special catalogs CONST and CONV in separate paragraphs further below.

Catalog Contents in Detail:

MODE
12h
1COMPL
24h
2COMPL
BASE
DENANY
DENFAC
DENFIX
DENMAX
DISP
D.MY
E3OFF
E3ON
FAST
FRACT
LZOFF
LZON
M.DY
RDX,
RDX.
REGS
RM
SETCHN
SETEUR
SETIND
SETUK
SETUSA
SIGNMT
SLOW
SSIZE4
SSIZE8
UNSIGN
WSIZE
Y.MD

PROB
Binom
Binom$_P$
Binom $^{-1}$
Cauch
Cauch$_P$
Cauch $^{-1}$
Expon
Expon $_P$
Expon $^{-1}$
$F_P(x)$
F(x)
$F^{-1}(p)$
Geom
Geom $_P$
Geom $^{-1}$
Lgnrm
Lgnrm $_P$
Lgnrm $^{-1}$
Logis
Logis$_P$
Logis $^{-1}$
Norml
Norml $_P$
Norml $^{-1}$
Poiss
Poiss$_P$
Poiss $^{-1}$
$t_P(x)$
t(x)
$t^{-1}(p)$
Weibl
Weibl $_P$
Weibl $^{-1}$
φ(x)
χ^2
χ^2 INV
$\chi^2{}_P$

STAT
nΣ
SEED
SERR
SERRw
SUM
sw
x̄g
x̄w
ε
ε_m
ε_p
σ
$\Sigma \ln^2 x$
$\Sigma \ln^2 y$
Σlnx
Σlnxy
Σlny
σw
Σx
Σx^2
$\Sigma x^2 y$
Σxlny
Σxy
Σy
Σy^2
Σylnx
%Σ

CFIT
BestF
COV
ExpF
LinF
LogF
L.R.
PowerF
sxy
x̂

MATRIX
DET
LINEQS
MROW+×
MROW×
MROW⇆
M+×
M^{-1}
M-ALL
M-COL
M-DIAG
M-ROW
M×
M.COPY
M.IJ
M.LU
M.REG
nCOL
nROW
TRANSP

TEST
BC?
BS?
ENTRY?
EVEN?
FC?
FC?C
FC?F
FC?S
FP?
FS?
FS?C
FS?F
FS?S
IBASE?
INTM?
INT?
KEY?
KTY?
LBL?
LEAP?
LocR?
MEM?
M.SQR?
NaN?
ODD?
PRIME?
REALM?
REGS?
RM?
SMODE?
SPEC?
SSIZE?
TOP?
WSIZE?
x < ?
x ≤ ?
x ≈ ?
x ≥ ?
x > ?
∞?

P.FCN	
BACK	RCF.RG
CF	RCF.ST
CLFLAG	RCLM
CLP	RCLS
CLSTK	RDX,
DATE	RDX.
DEC	RECV
DEL$_P$	RTN+1
DSL	R.CLR
DSZ	R.COPY
ERR	R.SORT
FF	R.SWAP
f ’(x)	SAVE
f ”(x)	SENDA
GTOα	SENDL
H.MS+	SENDP
H.MS-	SENDR
INC	SF
ISE	SKIP
ISZ	STOM
LOAD	STOS
LocR	TICKS
MSG	TIME
NOP	VWα+
PopLR	XEQα
PRCL	αGTO
PROMPT	αOFF
PSTO	αON
PUTK	αXEQ
P⇔	αVIEW
RCF	

X.FCN varies with the mode set, except in PRG. It contains in …

… alpha mode:	… decimal mode:			… integer modes:		CPX X.FCN
	AGM	LNβ	αDATE	ASR	RCF	^{C}AGM
CLALL	ANGLE	LNΓ	αDAY	BATT	RESET	^{C}CONJ
CLREG	BATT	MANT	αIP	CB	RJ	^{C}CUBE
RESET	B_n	MAX	αLENG	CLALL	RL	^{C}CUBERT
VERS	B_n*	MIN	αMONTH	CLFLAG	RLC	^{C}DROP
x → α	CEIL	MONTH	αRCL	CLREG	RR	$^{C}e^{x}$ -1
αDATE	CLALL	NAND	αRC#	CUBE	RRC	^{C}FIB
αDAY	CLREG	NEXTP	αRL	CUBERT	SB	^{C}LN1+x
αIP	CUBE	NOR	αRR	DBLR	SEED	^{C}LNβ
αLENG	CUBERT	P_n	αSL	DBL*	SIGN	^{C}LNΓ
αMONTH	DAY	RAD→	αSR	DBL/	SL	^{C}RCF
αRC#	DAYS+	RCF	αSTO	DROP	SR	^{C}SIGN
αRL	DECOMP	RESET	αTIME	FB	VERS	^{C}SINC
αRR	DEG→	ROUNDI	β	FIB	XNOR	^{C}W
αSL	DROP	SDL	Γ	GCD	XROOT	$^{C}W^{-1}$
αSR	D→J	SDR	ΔDAYS	LCM	αIP	^{C}β
αTIME	erf	SETDAT	ζ	LJ	αLENG	^{C}Γ
α → x	erfc	SETTIM	$(-1)^x$	MASKL	αRCL	$^{C}(-1)^x$
	EXPT	SIGN	%MG	MASKR	αRC#	
	e^x -1	SINC	%MRR	MAX	αRL	
	FIB	SLVQ	%T	MIN	αRR	
	FLOOR	T_n	%Σ	MIRROR	αSL	
	GCD	U_n	%+MG	NAND	αSR	
	GRAD→	VERS		nBITS	αSTO	
	H_n	W		NEXTP	$(-1)^x$	
	H_{np}	WDAY		NOR		
	Iβ	W^{-1}				
	IΓ	XNOR				
	JG1582	$^{x}\sqrt{y}$				
	JG1752	YEAR				
	J→D					
	LCM					
	L_n					
	LN1+x					
	$L_n\alpha$					

CPX					
À			à		
Á			á		
Â Ã Ā Ă			â ã ā ă		
Ä			ä (ǎ)		
Å			å		
Ć			ć		
Č			č		
Ç			ç		
È			è		
É			é		
Ê Ē Ĕ Ě			ê ē ĕ ě		
Ë			ë (ě)		
			ℏ		
Ì			ì		
Í			í		
Î Ĩ Ī Ĭ			î ĩ ī ĭ		
Ï			ï (ǐ)		
Ñ Ň			ñ ň		
Ò			ò		
Ó			ó		
Ô Õ Ō Ŏ			ô õ ō ŏ		
Ö			ö (ǒ)		
Ø			ø		
Ř			ř		
Š			š		
			ß		
Ù			ù		
Ú			ú		
Û Ũ Ū Ŭ			û ũ ū ŭ		
Ü			ü (ǔ)		
Ů			ů		
Ý			ý		
Ÿ			ÿ		
Ž			ž		

Here are the contents of the alpha catalogs making the WP 34S the most versatile global calculator known. Large font is printed in grey cells on this page. Accented letters show the same width as plain ones wherever possible.

./,
, : ; ‘ “ # * @ _ ~ ` ⊙ ⊕

TEST
< ≤ = ≈ ≥ > [ ] { }

→
→ ← ↑ ↓ ∫ ∞ ^

R↓ (subscripts)
0 1 2 A B c e k m n p u µ ∞

R↑ (superscripts)
c ° 2 x x̄ x̂ ȳ ŷ -1

The letters provided in your WP 34S allow for correct writing the languages of more than $3 \cdot 10^9$ people (still only half of mankind yet), i.e.:

Afrikaans, Català, Cebuano, Česky, Cymraeg, Deutsch, Eesti, English, Español, Euskara, Français, Gaeilge, Galego, Greek, Bahasa Indonesia, Italiano, Basa Jawa, Kiswahili, Kreyòl ayisyen, Magyar, Bahasa Melayu, Nederlands, Português, Quechua, Shqip, Slovenčina, Slovenščina, Basa Sunda, Suomeksi, Svenska, Tagalog, Winaray, Zhōngwén (with a little trick explained below), and almost Dansk and Norsk (sorry, no æ) as well as Hrvatski and Srpski (no đ). If you know further living languages covered, please tell us.

Mandarin Chinese (Zhōngwén) features four tones, usually transcribed like e.g. mā, má, mǎ, and mà. So you need different letters for ā and ǎ here, and for e, i, o, and u as well. With six pixels total character height we found no way to display these in both fonts nicely, keeping letters and accents separated for easy reading. For an unambiguous solution, we suggest using a dieresis (else not employed in Hànyǔ pīnyīn) representing the third tone here. Pinyin writers, we ask for your understanding.

Addressing Catalog Items

1	User input	CONST, CONV, MODE, PROB, P.FCN, STAT, TEST, or X.FCN	CPX, R↓, or R↑ in alpha mode	→, TEST, or ./, in alpha mode
	Dot matrix display	**Shows 1st item in selected catalog.** (e.g. BC? in P.FCN) Alpha mode is set.	(e.g. Á in CPX)	(e.g. , in ./,)
2	User input	XEQ, ▼, ▲, EXIT, or 1st character (e.g. F)	XEQ, ▼, ▲, EXIT, or character (e.g. O)	
	Dot matrix display	**Shows 1st item starting with this character *)** (e.g. FB)	**Shows 1st item starting with this letter *)** (e.g. Ó)	
3	User input	XEQ, ▼, ▲, EXIT, or 2nd character (e.g. S)		
	Dot matrix display	**Shows 1st item starting with this sequence *)** (e.g. FS?)		
4	User input	XEQ , ▼ , ▲ , or EXIT (e.g. ▼)		
	Dot matrix display	**Shows next item in this catalog** (e.g. FS?C)	(e.g. Ò)	(e.g. ?)
…		Continue browsing this way until reaching the item desired (e.g. FS?F).	(e.g. Ö).	(e.g. :).
n	User input	XEQ Calculator leaves the catalog returning to the mode set before		
	Dot matrix display	… and executes or inserts the command chosen, or recalls the constant selected. **Result**	… and appends the selected character to ***alpha***. **Contents of alpha register** (e.g. Östl. Seite:)	

*) If a character or sequence specified is not found in this catalog then the first item following alphabetically will be shown. If there is no such item, then the last item in this catalog is displayed. You may key in even more than two characters – after 3 seconds, however, or after ▼ or ▲ , the search string will be reset and you may start with a first character again.

Constants

Below you find the contents of the catalog CONST. Navigation works as in the catalogs mentioned before. Names of astronomical and mathematical constants are printed on colored background below. Values of physical constants (*incl. their relative standard deviations given in parentheses below*) are from CODATA 2010, copied in July 2011, unless stated otherwise explicitly. Green background denotes exact or almost exact values. The more the color turns to red, the less precise the respective constant is known [28].

For the units, remember Tesla with $1T = 1\frac{Wb}{m^2} = 1\frac{V \cdot s}{m^2}$, Joule with $1J = 1N \cdot m = 1\frac{kg \cdot m^2}{s^2}$

and on the other hand $1J = 1W \cdot s = 1V \cdot A \cdot s = \frac{1}{e}eV \approx 6.24 \cdot 10^6 TeV$. Thus $1\frac{J}{T} = 1A \cdot m^2$.

Name	Numeric value	Unit	Remarks
a	365.2425 *(per definition)*	d	Gregorian year
a₀	5.2917721092**E**-11 *(3.2**E**-10)*	m	Bohr radius $= \frac{\alpha}{4\pi \cdot R_\infty}$
a$_m$	384.4**E**6 *(1**E**-3)*	m	Semi-major axis of the Moon's orbit around the Earth
a⊕	1.495979**E**11 *(1**E**-6)*	m	Semi-major axis of the Earth's orbit around the sun. Within the uncertainty stated here, it equals 1 AU.
c	2.99792458**E**8 *(per definition)*	m/s	Vacuum speed of light
c$_1$	3.74177153**E**-16 *(4.4**E**-8)*	$m^2 \cdot W$	First radiation constant $= 2\pi \cdot h \cdot c^2$
c$_2$	0.014387770 *(9.1**E**-7)*	$m \cdot K$	Second radiation constant $= hc/k$
e	1.602176565**E**-19 *(2.2**E**-8)*	C	Electron charge $= \frac{2}{K_J R_K} = \Phi_0 G_0$
eE	2.718281828459045…	1	Euler's e. Please note the letter *e* represents the electron charge elsewhere in this table.
F	96485.3365 *(2.2**E**-8)*	$\frac{C}{mol}$	Faraday's constant $= e\ N_A$
Fα	2.5029078750958928...	1	Feigenbaum's α
Fδ	4.6692016091029906...	1	Feigenbaum's δ
g	9.80665 *(per definition)*	m/s^2	Standard earth acceleration

[28] The bracketed values printed here for your kind attention allow you to compute the precision of results you may obtain using these constants. The procedure to be employed is called error propagation. It is often ignored, though essential for trustworthy results – not only in science. Please turn to respective texts before you believe in 4 decimals of a calculation result based on yardstick measurements.

Name	Numeric value	Unit	Remarks
G	6.67384**E**-11 *(1.2**E**-4)*	$\frac{m^3}{kg \cdot s^2}$	Newton's gravitation constant. See **GM** below for a more precise value.
G$_o$	7.7480917346**E**-5 *(3.2**E**-10)*	$1/\Omega$	Conductance quantum $= 2e^2/h = 2/R_K$
G$_c$	0.915965594177…	1	Catalan's constant
g$_e$	2.00231930436153 *(2.6**E**-13)*	1	(Landé's) electron g-factor
GM	3.986004418**E**14 *(2.0**E**-9)*	m^3/s^2	Newton's gravitation constant times the Earth's mass with its atmosphere included (according to WGS84, see **Sa** below).
h	6.62606957**E**-34 *(4.4**E**-8)*	$J\ s$	Planck constant
$\hbar$	1.054571726**E**-34 *(4.4**E**-8)*		$= h/2\pi$
k	1.3806488**E**-23 *(9.1**E**-7)*	J/K	Boltzmann constant $= R/N_A$
Kj	4.83597870**E**14 *(2.2**E**-8)*	Hz/V	Josephson constant $= 2e/h$
l$_p$	1.616199**E**-35 *(6.0**E**-5)*	m	Planck length $= \sqrt{\hbar G/c^3} = t_p c$
m$_e$	9.10938291**E**-31 *(4.4**E**-8)*		Electron mass
M$_m$	7.349**E**22 *(5**E**-4)*		Mass of the Moon
m$_n$	1.674927351**E**-27 *(4.4**E**-8)*		Neutron mass
m$_p$	1.672621777**E**-27 *(4.4**E**-8)*	kg	Proton mass
M$_p$	2.17651**E**-8 *(6.0**E**-5)*		Planck mass $= \sqrt{\hbar c/G} \approx 22\mu g$
m$_u$	1.660538921**E**-27 *(4.4**E**-8)*		Atomic unit mass = 10^{-3} *kg / N$_A$*
m$_u$c^2	1.492 417 954**E**-10 *(4.4**E**-8)*	J	Atomic unit mass energy equivalent
m$_\mu$	1.883531475**E**-28 *(5.1**E**-8)*		Muon mass
M⊙	1.9891**E**30 *(5**E**-5)*	kg	Mass of the sun
M⊕	5.9736**E**24 *(5**E**-5)*		Mass of the Earth
N$_A$	6.02214129**E**23 *(4.4**E**-8)*	$1/mol$	Avogadro's number
NaN			"not a number"
p$_o$	101325 *(per definition)*	Pa	Standard atmospheric pressure
q$_p$	1,8755459**E**-18 *(6.0**E**-5)*	As	Planck charge $= \sqrt{4\pi\varepsilon_0\hbar c} \approx 11.7e$. This was in CODATA 2006, but in 2010 no more.

Name	Numeric value	Unit	Remarks
R	8.3144621 *(9.1**E**-7)*	$\frac{J}{mol \cdot K}$	Molar gas constant
r_e	2.8179403267**E**-15 *(9.7**E**-10)*	m	Classical electron radius $= \alpha^2 \cdot a_0$
R_K	25812.8074434 *(3.2**E**-10)*	Ω	von Klitzing constant $= h/e^2$
R_m	1.737530**E**6 *(5**E**-7)*	m	Mean radius of the Moon
R_∞	1.0973731568539**E**7 *(5.0**E**-12)*	$1/m$	Rydberg constant $= \alpha^2 m_e c / 2h$
R⊙	6.96**E**8 *(5**E**-3)*	m	Mean radius of the sun
R⊕	6.371010**E**6 *(5**E**-7)*	m	Mean radius of the Earth
Sa	6.3781370**E**6 *(per definition)*	m	Semi-major axis of the model WGS84 used to define the Earth's surface for GPS and other surveying purposes (→ http://earth-info.nga.mil/GandG/publications/tr8350.2/tr8350_2.html)
Sb	6.3567523142**E**6 *(1.6**E**-11)*	m	Semi-minor axis of WGS84
Se^2	6.69437999014**E**-3 *(1.5**E**-12)*	1	First eccentricity squared of WGS84
Se'^2	6.73949674228**E**-3 *(1.5**E**-12)*	1	Second eccentricity squared of WGS84 (it is really called e'^2 in this article, I apologize)
Sf^{-1}	298.257223563 *(per definition)*	1	Flattening parameter of WGS84
T_o	273.15 *(per definition)*	K	= 0°C, standard temperature
t_p	5.39106**E**-44 *(6.0**E**-5)*	s	Planck time $= \sqrt{\hbar G / c^5} = \frac{l_p}{c}$
T_p	1.416833**E**32 *(6.0**E**-5)*	K	Planck temperature $= \frac{c^2}{k}\sqrt{\frac{\hbar c}{G}} = \frac{M_p c^2}{k} = \frac{E_p}{k}$
V_m	0.022413968 *(9.1**E**-7)*	m^3/mol	Molar volume of an ideal gas at standard conditions $= RT_0/p_0$
Z_o	376.730313461…	Ω	Charact. impedance of vacuum $= \sqrt{\frac{\mu_0}{\varepsilon_0}} = \mu_0 c$
α	7.2973525698**E**-3 *(3.2**E**-10)*	1	Fine-structure constant $= \frac{e^2}{4\pi\varepsilon_0 \hbar c} \approx \frac{1}{137}$
γEM	0.57721566490153286…	1	Euler-Mascheroni constant
γ_p	2.675222005**E**8 *(2.4**E**-8)*	$\frac{1}{s \cdot T}$	Proton gyromagnetic ratio $= 2\mu_P/\hbar$

Name	Numeric value	Unit	Remarks
ε_0	8.854187817…**E**-12	$\frac{A \cdot s}{V \cdot m}$ or F/m	Electric constant, vacuum permittivity $= \frac{1}{\mu_0 c^2}$
λ_c	2.4263102389**E**-12 *(6.5**E**-10)*	m	Compton wavelength of the electron $= h/m_e c$
λ_{cn}	1.3195909068**E**-15 *(8.2**E**-10)*		Compton wavelength of the neutron $= h/m_n c$
λ_{cp}	1.32140985623**E**-15 *(7.1**E**-10)*		Compton wavelength of the proton $= h/m_p c$
μ_0	1.2566370614…**E**-6	$\frac{V \cdot s}{A \cdot m}$	Magnetic constant, also known as vacuum permeability $= 4\pi \cdot 10^{-7} \frac{V \cdot s}{A \cdot m}$ (per definition)
μ_B	9.27400968**E**-24 *(2.2**E**-8)*	J/T or $A \cdot m^2$	Bohr's magneton $= e\hbar/2m_e$
μ_e	-9.28476430**E**-24 *(2.2**E**-8)*		Electron magnetic moment
μ_n	-9.6623647**E**-27 *(2.4**E**-7)*		Neutron magnetic moment
μ_p	1.410606743**E**-26 *(2.4**E**-8)*		Proton magnetic moment
μ_u	5.05078353**E**-27 *(2.2**E**-8)*		Nuclear magneton $= e\hbar/2m_p$
μ_μ	-4.49044807**E**-26 *(3.4**E**-8)*		Muon magnetic moment
π	3.141592653589793…	1	
σ_B	5.670373**E**-8 *(3.6**E**-6)*	$\frac{W}{m^2 K^4}$	Stefan Boltzmann constant $= \frac{2\pi^5 k^4}{15 h^3 c^2}$
Φ	1.618033988749894…	1	Golden ratio $= \frac{1+\sqrt{5}}{2}$
Φ_0	2.067833758**E**-15 *(2.2**E**-8)*	$V\,s$	Magnetic flux quantum $= h/2e = 1/K_J$
ω	7.292115**E**-5 *(2**E**-8)*	rad/s	Angular velocity of the Earth according to WGS84 (see **Sa** above)
$-\infty$		1	Negative and positive infinity (may the Lord of Mathematics forgive us calling these two 'constants')
∞			

Unit Conversions

Find below the contents of the catalog CONV [29]. Navigation works as in the other catalogs. There is one specialty, however: f B (i.e. 1/x) will execute the inverse of the conversion displayed and leave CONV.

Example: Assume the display set to FIX 3. Then keying in

4 h CONV A	will display	acres→ha and 1.619 below telling you 4 acres equal 1.619 hectares.
Now press f B	and you will get	9.884 instead, being the amount of acres equaling 4 hectares.
Press h CONV again and you will see		acres→ha and 4.000 below confirming what was just said.
Leave the catalog via EXIT		and the display will return to 9.884.

The constant $\mathbf{T_o}$ may be useful for conversions of temperatures, too; it is found in the *catalog CONST* and is not repeated here since being only added or subtracted. The conversion factors or divisors listed below for your information are user transparent in executing a conversion – those printed on light green background in this table apply exactly.

Conversion		Remarks	Class
°C→°F	* 1.8 + 32		Temperature
°F→°C	- 32) / 1.8		Temperature
°→G	/ 0.9	Converts to 'grads' or 'gon'	Angle
°→rad	* π / 180	Equals D→R	Angle
acres→ha	* 0.4046873	1 ha = 10^4 m²	Area
ar.→dB	$20\lg\left(a_1/a_2\right)$	Amplitude ratio	Ratio
atm→Pa	* 1.01325**E**5		Pressure
AU→km	* 1.495979**E**8	Astronomic units	Length
bar→Pa	* 1**E**5		Pressure
Btu→J	* 1055.056	British thermal units	Energy
cal→J	* 4.1868		Energy
cft→l	* 28.31685	Cubic feet	Volume

[29] For most readers, many of the units appearing in CONV may look obsolete at least. They die hard, however, in some corners of this world. All these corners have in common is English being spoken there. For symmetry reasons, we may also add some traditional Indian and Chinese units. Anyway, this catalog provides the means to convert local to common units.

Conversion		Remarks	Class
cm→inches	/ 2.54		Length
dB→ar.	$10^{R_{dB}/20}$	Amplitude ratio	Ratio
dB→pr.	$10^{R_{dB}/10}$	Power ratio	Ratio
fathom→m	* 1.8288		Length
feet→m	* 0.3048		Length
flozUK→*ml*	* 28.41306	1 *l* = $^{1}/_{1000}$ m^3	Volume
flozUS→*ml*	* 29.57353		
galUK→ *l*	* 4.54609		
galUS→ *l*	* 3.785418		
G→°	* 0.9	Grads or gon	Angle
g→oz	/ 28.34952		Mass
G→rad	* π / 200		Angle
g→tr.oz	/ 31.10348		Mass
ha→acres	/ 0.4046873	1 ha = 10000 m²	Area
HP_e→W	* 746	Electric horse power	Power
hpUK→W	* 745.6999	British horse power	Power
inches→cm	* 2.54		Length
inHg→Pa	* 3386.389		Pressure
J→Btu	/ 1055.056		Energy
J→cal	/ 4.1868		Energy
J→kWh	/ 3.6**E**6		Energy
kg→lb	/ 0.4535924		Mass
kg→stones	/ 6.35029318		Mass
km→AU	/ 1.495979**E**8	Astronomic units	Length
km→*l.y.*	/ 9.460730**E**12	Light years	Length
km→miles	/ 1.609344		Length
km→nmi	/ 1.852	Nautical miles	Length
km→pc	/ 3.085678**E**16	Parsec	Length

Conversion		Remarks	Class
kWh→J	* 3.6E6		Energy
lbf→N	* 4.448222		Force
lb→kg	* 0.4535924		Mass
l.y.→km	* 9.460730**E**12	Light years	Length
l →cft	/ 28.31685	1 l = $^{1}/_{1000}$ m^3	Volume
l →galUK	/ 4.54609		
l →galUS	/ 3.785418		
miles→km	* 1.609344		Length
ml→flozUK	/ 28.41306	1 ml = 1 cm^3	Volume
ml→flozUS	/ 29.57353		
mmHg→Pa	* 133.3224	1 torr = 1 mm Hg	Pressure
m→fathom	/ 1.8288		Length
m→feet	/ 0.3048		Length
m→yards	/ 0.9144		Length
nmi→km	* 1.852	Nautical miles	Length
N→lbf	/ 4.448222		Force
oz→g	* 28.34952	Ounces	Mass
Pa→atm	/ 1.01325**E**5	1 Pa = 1 N/m^2	Pressure
Pa→bar	/ 1**E**5		Pressure
Pa→inHg	/ 3386.389		Pressure
Pa→mmHg	/ 133.3224		Pressure
Pa→psi	/ 6894.757		Pressure
Pa→torr	/ 133.3224		Pressure
pc→km	* 3.085678**E**16	Parsec	Length
pr.→dB	$10\lg\left(P_1/P_2\right)$	Power ratio	Ratio
psi→Pa	* 6894.757	Pounds per square inch	Pressure
PS(hp)→W	* 735.4988	Horse power	Power
rad→°	* 180 / π	Equals R→D	Angle

Conversion		Remarks	Class
rad→G	* 200 / π		Angle
stones→kg	* 6.35029318		Mass
s.tons→t	* 0.9071847	Short tons	Mass
tons→t	* 1.016047	Imperial tons	Mass
torr→Pa	* 133.3224	1 torr = 1 mm Hg	Pressure
tr.oz→g	* 31.10348	Troy ounces	Mass
t→s.tons	/ 0.9071847	1 t = 1000 kg	Mass
t→tons	/ 1.016047		
W→HP_e	/ 746		Power
W→hpUK	/ 745.6999		Power
W→PS(hp)	* 735.4988		Power
yards→m	* 0.9144		Length

You may, of course, combine conversions as you like. For example, filling your tires with a maximum pressure of 30 psi the following will help you at a gas station in Europe and beyond:

3 0 h CONV P S XEQ
h CONV P ▼ XEQ resulting in 2.1 bar.

Now you can set the filler and will not blow your tires.

In cases of emergency of a particular kind, remember Becquerel equals Hertz, Gray is the unit for deposited or absorbed energy ($1Gy = 1J/kg$), and Sievert (Sv) is Gray times a radiation dependant dose conversion factor for the damage caused in human bodies.

In this area also some outdated units may be found in older literature: Pour les amis de Mme. Curie, $1Ci = 3.7 \cdot 10^{10} Bq = 3.7 \cdot 10^{10} decays/s$. And for those admiring the very first Nobel laureate in physics, Mr. Röntgen, for finding the x-rays (ruining his hands in these experiments), the charge generated by radiation in matter was measured by the unit $1R = 2.58 \cdot 10^{-4} \, {}^{As}\!/\!{}_{kg}$. A few decades ago, Rem (i.e. Röntgen equivalent men) was measuring what Sievert does today.

Predefined Global Alpha Labels

There are a few labels employed and provided for particular tasks already. You find them listed in CAT when the respective routines are loaded in XROM. Thus they will not take any steps from user program memory.

WHO	Displays credits to those brave men who did the work.
δx	Provides the step size for differentiation. See f '(x) and f "(x) in the [*Index of Operations*](#) for more information.

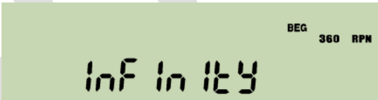
More routines are found at *http://wp34s.svn.sourceforge.net/viewvc/wp34s/library/* as text files with extension *.wp34s* by convention. This includes, for example, a suite of basic 3D vector operations, a TVM application, and more. You may open these files using e.g. Notepad, and download them following your needs. README_ASM explains the loading procedure.

MESSAGES

There are some commands generating messages, also in the dot matrix section of the display. Four of them, DAY, DAYS+, STATUS, and VERS, were introduced above in the [*paragraph about display*](#) already. Others are PROMPT, αVIEW and many more alpha commands, and the test commands as mentioned [*above*](#).

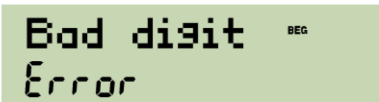
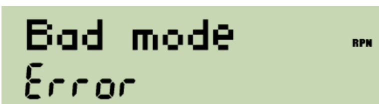
Also two constants will return a special display when called: **NaN** and ∞ will show

or



, respectively.

Furthermore, there are a number of error messages. Depending on error conditions, the following messages will be displayed in the mode(s) listed:

Message	Error Code	Mode(s)	Explanation and Examples
	2	DECM	Invalid date format or incorrect date or time in input, e.g. month >12, day >31 etc.
	9	Integer	Invalid digit in integer input, e.g. 2 in binary, 9 in octal, or +/- in unsigned mode.
	13	All	Caused by calling an operation in a mode where it is not defined, e.g. SIN in hexadecimal.

Message	Error Code	Mode(s)	Explanation and Examples
Domain Error	1	\α	An argument exceeds the domain of the mathematical function called. May be caused by roots or logs of negative numbers (if not preceded by CPX), by 0 / 0, LN(0), $\Gamma(0)$, TAN(90°) and equivalents, ATANH(x) for $\lvert\mathrm{Re}(x)\rvert \ge 1$, ACOSH(x) for $\mathrm{Re}(x) < 1$, etc.
Illegal OPErAtion	7	All	Self-explanatory.
Invalid dAtA	18	All	Set when there is a checksum error either in flash or as part of a serial download. It is also set if a flash segment is otherwise unusable.
Invalid PArAMEtEr	16	\α	Similar to error 1 but a parameter specified in **J** or **K** is out of supported range for the function called. May appear e.g. if LgNrm is called with $j < 0$.
I/O Error	17	\α	Please see Appendix A.
Matrix dIMEnSIon	21	DECM	• A matrix isn't square when it should be. • Matrix sizes aren't miscible.
No such LAbEL	6	All	Attempt to address an undefined label.
No write In FLASH	19	All	Attempt to delete program lines while inside a flash segment..
Out of range Error	8	All	• A number exceeds the valid range. Caused e.g. by specifying decimals >11, word size >64, negative flag numbers, integers $\ge 2^{64}$, hours or degrees >9000, invalid times, denominators ≥9999 etc. • A register address exceeds the valid range. May also happen in indirect addressing. • An R-operation (e.g. R.COPY) attempts exceeding valid register numbers (0 .. 99). • A matrix descriptor would go beyond the registers available or a row or column index is too large.

Message	Error Code	Mode(s)	Explanation and Examples
RTN Stack FuLL	11	PRG	No more space in return stack (see above).
Singular Error	22	DECM	• Attempt to use a LU decomposed matrix for solving a system of equations. • Attempt to invert a matrix when it isn't of full rank.
Solve FAILEd	20	DECM	The solver did not converge.
Stack CLASH	12	All	STOS or RCLS attempt using registers that would overlap the stack. Will happen with e.g. SSIZE = 8 and STOS 94.
Too few dAtA PointS	15	DECM	A statistical calculation was started based on too few data points, e.g. regression or standard deviation for < 2 points.
Too long Error	10	All	Keyboard input is too long for the buffer (should never happen, but who knows).
undefined OP-COdE	3	All	An instruction with an undefined op-code occurred (should never happen, but who knows).
Word size too SMALL	14	Integer, \PRG	Stack or register content is too big for the word size set.
+∞ Error	4	\α, \PRG	• Division of a number > 0 (or < 0) by zero. • Divergent sum or product or integral. • Positive (or negative) overflow in DECM (see above).
−∞ Error	5		

Error messages are temporary. Any key pressed will erase the error message displayed and execute with the stack contents present. Thus, an easy return to the display shown before the error occurred is pressing a prefix twice.

PROGRAMMED INPUT AND OUTPUT

A number of commands may be employed for controlling I/O of programs. In the index above, their behavior is described if they are entered from the keyboard. Executed by a program, however, this will differ in a characteristic way.

With a program running, the display will be updated at certain instances only instead of after each operation. So where a command in manual mode shows an information until the next key is pressed, it will show it until the next display update in automatic mode. Such an update will occur with PROMPT, PSE, STOP, VIEW, VWα+, and αVIEW only. This allows for the following operations (please note parameters are omitted here):

- Output of messages or other information for a defined time interval using the following code segment
  …
  VIEW
  PSE
  …
  (or simply PSE alone) for plain numeric calculated output or
  …
  αVIEW (or even VWα+)
  PSE
  …
  for complex alphanumeric information you composed in ***alpha***.

- Asking ("prompting") for numeric input employing
  …
  αVIEW (or VWα+)
  STOP
  …
  or simply PROMPT, the latter being identical to VWα+ X plus STOP.
  Whatever number you key in will be in **X** when you continue the program by pressing R/S . If you want it elsewhere, take care of it.

- Prompting for alphanumeric input by
  …
  αON
  PROMPT
  αOFF
  …
  Whatever you key in will be appended to ***alpha*** here. Again, the program will continue when you pressed R/S .

Please see the index for more information about these commands and their parameters.

INTERACTIVE PROGRAMMING

This chapter deals with writing programs that interact with the user. Topics covered are the display of messages, getting input from the user, hot keys and truly interactive "real time" programs.

Interrupting a Program for Display of Information

When a program is started, the display contents are replaced by the "Running Program" message. To display a number while a program is executing, use VIEW in programming and specify a register to display. Here, **X** is a valid parameter so you can present the standard top stack level contents to the user. The command formats the number to the present settings and updates the LCD to display it. This causes a small overhead so expect that your program slows down a bit with each update. This is especially true if the displays follow each other in a tight loop because the flicker avoidance logic needs to wait for a complete display refresh cycle before the next update is allowed.

Another way to show what would normally appear on the display without a program running is to use the PSE instruction specifying the time in 10ths of seconds to suspend execution. A time of zero will have the same effect as a VIEW **X** instruction. PSE following VIEW ***s*** works as expected: it will display the contents of address ***s***. The display will then stay unchanged until the next VIEW or PSE instruction is executed, not only for the time specified with PSE. The next PSE or STOP will switch back to the normal display of ***x*** . VIEW ***s*** followed by STOP will display the contents of address ***s*** until the user presses R/S .

To make things clearer: VIEW immediately displays the register when encountered in program execution. When followed by PSE or STOP, the display persists. Only the next PSE or STOP (or keyboard entry after the program has halted) will revert to the normal ***x*** display. To make sure that STOP or PSE always display a specific information it is best to directly precede it by the respective VIEW instruction. There is no way to get the "Running Program" message back once it has been replaced by a programmed display.

Generally speaking, a message is a string of characters that is shown in the upper region of the display. The program interface to this area is via the alpha register. You need to switch to alpha mode to access most of the commands that deal with this register. The annunciator INPUT lights if alpha mode is active. The X.FCN catalogue changes in alpha mode to contain alpha commands. Displaying a message will normally start with a CL[alpha] instruction because most commands append their output to what is already stored. To save space, characters in program mode may be entered in groups of three by typing α while already in alpha mode. This saves one program step per three characters but does not allow all special symbols to be entered because the catalogues are not available in this mode. Single characters and grouped characters can be freely mixed. The register is 31 characters wide. The display capacity is considerably smaller and depends on the width of each symbol. The display switches to a smaller font if necessary. The contents can be scrolled in interactive alpha mode with the up and down arrow keys (as described above).

If you just want to display a text message and no number with it, use αVIEW. To get to this command you must be out of alpha mode and open the P.FCN catalogue. g A brings you to the alpha commands. The αVIEW display starts at the first character of the string. The numeric portion of the LCD is replaced by three dashes. You can of course display a message together with a chosen register. Go to alpha mode and press

VIEW. This will produce the αVW+ nn command. It is meant to display alpha together with ('+') numeric data coming from any register. As with VIEW, X is allowed here. The above comments regarding PSE or STOP following any of these commands are valid here, too.

Another way to display the alpha register is to switch to alpha mode with αON. The main difference is that you are presented the tail of the string instead of its head. Also, a PSE is necessary to update the actual display which αON alone does not do. If followed by a STOP, alpha mode stays on causing user input to go to the upper display! αOFF returns everything to normal.

Temporary Displays

Whenever the display does not show the actual contents of the X register in the current mode, this is considered a temporary display. To distinguish this from the normal display, the RPN annunciator is off during temporary displays and on otherwise. The following displays are considered temporary:

1. Any errors,
2. αVIEW,
3. αVW+ ***nn***,
4. VIEW ***nn*** where ***nn*** is not X,
5. VIEW X if encountered in a program because X may have changed before the stop,
6. H.MS display,
7. Temporary display in another base (not programmable).

Press EXIT or ⬅ to get back to the normal display.

Data Input

The easiest way of getting user input, apart from expecting everything on the stack, is just stopping the program with STOP, letting the user input a number and let him press R/S to continue execution. Without any clue what the program is asking for, this is only suitable for very simple programs. The least you want to do is present a message to the user what he is supposed to enter when the program stops. This can be done with any of the [alpha]VIEW commands followed by STOP. There is a shorthand especially made for this: PROMPT. It is a combination of [alpha]VW+ X and STOP. It displays the alpha register together with the current X register and halts program execution. This is good for entering a lengthy list of parameters in a given order without much programming.

Hotkeys

A more versatile way of doing things is using the dedicated keys A to D in the top row. If the user presses one of these keys the program executes the next subroutine or program with a label of the same name. If you have more than one program using labels A to D in RAM or in a flash region, it's necessary to move the program counter (PC) to the top of the program and stop there. A typical program structure might be the following:

```
LBL 'MYP'
CL[alpha]
[alpha]'Hel'
[alpha]'lo!'
LBL 00
PROMPT
BACK 01
LBL A
ENTRY?
SKIP 01
XEQ 01
STO 01
GTO 00
LBL B
...
```

This sets up a message and stops. R/S does nothing, it simply returns to the prompt. If the user enters a number and hits A, the program starts with the ENTRY? test which is true if the user has entered fresh data. The input will be stored in register 01 and the program jumps back to the prompt. If the user has not entered any information after the last prompt, subroutine 01 will be called to compute a new value which is then stored and displayed. This is the way the TVM application is implemented.

Keyboard Codes

Sometimes, the hot keys A to D aren't enough. But there are ways to extend the number of directly addressable subroutines by a simple trick: shorthand addressing of numeric labels. To make this possible, each key is identified by a row and a column, each starting with one.

A	B	C	D	->	CPX
11	12	13	14	*15*	*16*
STO	RCL	Rv	f	g	h
21	*22*	*23*	24	25	26
ENTER^		x<>y	+/-	EEX	<-
31		*32*	*33*	*34*	35
XEQ	7	8	9	/	
41	42	43	44	*45*	
^	4	5	6	x	
51	52	53	54	*55*	
v	1	2	3	-	
61	62	63	64	*65*	
EXIT	0	.	R/S	+	
71	72	73	*74*	*75*	

Whenever you are asked for the entry of a two-digit label, any of the keys marked in *italic* in the above picture can be used as direct input. The label will be replaced by the row/column code of the respective key. Some keys are not available this way because they have a predefined meaning in this context. They can still be used for a short address by preceding the key with the f prefix. Only the f prefix itself cannot be used for shorthand addressing. If you want to associate a program with the key [STO], just put the label 21 in front of the routine and it can be conveniently called with [XEQ] [STO] by the user.

Direct Keyboard Access

The same codes are returned by the KEY? command which allows true "real time" response to user input from the keyboard. KEY? takes a register argument (X is allowed but does not lift the stack) and stores the key most recently pressed during program execution in the specified register. R/S and EXIT cannot be queried, they stop program execution immediately. The keyboard is active during execution but it is of course desirable to show a message and suspend the program with the PSE command while waiting for user input. PSE is interrupted by a key press, so you can simply use a PSE 99 statement in a loop to wait for input. KEY? acts as a conditional at the same time so a typical user input loop will look like this:

```
LBL 'USR'
CLα
α 'KEY'
α ?
LBL 00
αVIEW
PSE 99
KEY? 00
GTO 00
LBL?->00
XEQ->00
GTO 00
```

This code fragment prompts for a key and stores it in register 00. The line directly after KEY? is executed when no key was pressed. The statement KEY? is only executed every 9.9 seconds if the user does not press a key. If he does, the PSE is immediately terminated, KEY? is executed, finds the key code and stores it in register 00. The LBL→00 instruction checks if a label corresponding to the key code has been defined and executes it if found. Instead of the dumb waiting loop, the program can do some computations and update the display before the next call to PSE and KEY? – think of a lunar lander game.

To be even more versatile, the instruction KTY? ***nn*** is designed to return the key type of a row / column code in register ***nn***: 0 to 9 for the respective digits, 10 for the other numeric keys (. , +/- and EEX), 11 for any of the three shift keys and 12 for the rest. An invalid code in the target register throws an "Invalid Range Error".

If you decide not to handle the key in the program you may feed it back to the main processing loop of the calculator with the PUTK ***nn*** command. What happens is that the program halts and the key is treated as if pressed after the stop. This is especially useful if you want to allow numeric input while waiting for some special keys like the arrows. This allows writing of a vector or matrix editor in user code. After execution of the PUTK command the user is responsible for letting the program continue its work by pressing [R/S] or a hot key.

APPENDIX A: SUPPORT FOR FLASHING, SERIAL I/O ETC.

How to Flash Your HP 20b or 30b

You may do the flashing yourself. Then you need your calculator, a special connecting cable, and specific software on your PC or Mac. A PC featuring an hardware serial port and running Windows XP is beneficial. **Please read this paragraph completely before actually starting the procedure.**

- You will get the necessary software – the SAM-BA In-system Programmer – here for free:

  http://www.atmel.com/dyn/products/tools_card.asp?tool_id=3883

  Install it as explained by Atmel.

- You will get the cable from Gene Wright.

- The specific file you will need to transmit to your calculator to make it your WP 34S is called `calc.bin` and is included in the zipped package you can download from here:

  http://sourceforge.net/projects/wp34s/files/

  Alternatively, you may download `calc.bin` alone from

  http://wp34s.svn.sourceforge.net/viewvc/wp34s/trunk/realbuild/

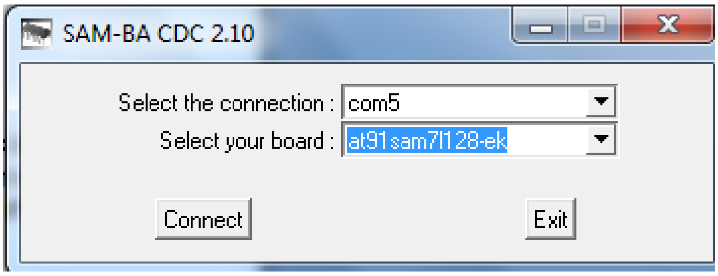
Now, having got these three (SAM-BA, the cable, and `calc.bin`), please turn to the file *http://dl.dropbox.com/u/10022608/Flashing%20a%2020b%20Calculator.pdf* (edited by Tim Wessmann and Gene Wright). Read it thoroughly for information about connecting and flashing.

> **ATTENTION:** If your PC does not feature an hardware serial interface, you will need an USB-to-serial converter to connect the special cable to your PC. Following our experience, converters containing FTDI chips will work – others may not.
>
> On other operating systems than XP flashing may work or not (definitively not on Windows 2000 or earlier). Please check.
>
> On Windows 7 load MS Windows Virtual PC and Windows XP Mode, then work therein.

Then proceed as described in *Flashing a 20b Unit* in said file, steps 1 to 3 only.

> **ATTENTION:** Flashing your *HP 20b* or *30b* will erase the HP firmware in step 3, meaning your business calculator will be gone then. The firmware will be replaced with the WP 34S file completely! After this flash is finished, you will have a *WP 34S RPN Scientific* – i.e. your calculator will react as documented in this very manual.
>
> This also means your device will not do anything useful for you between step 3 and 13. It may even look dead – it is not, be assured, at least it will just be eating your batteries (see below)! If you (have to) interrupt the flashing process at any time in this interval for any reason whatsoever, don't worry: simply start again. You may, however, not get any feedback displayed in step 3 anymore. That does not matter, just stick to the procedure.
>
> As long as the cable is connected to your calculator, it will draw a considerable current from the calculator batteries. If you happen to hang anywhere in the flashing process, also the processor is left running at full speed. So chances are high your coin cells will be drained while you are trying to find out what is going wrong. Thus it is wise to disconnect the cable from your calculator when you will not need the cable for the next couple of minutes. For repeated flashing, an external 3V DC supply may pay very fast. Take care to connect + to the outer and – to the inner contact. The following will work with a good 3V supply only.

Having completed step 3 of said file, call your SAM-BA for step 4. It may take a long time to start up (some sixty seconds), so be patient. When it launches (step 5), a window pops up:



Choose the correct connection (take the port you put your cable in – it may differ from what is printed here). Select the board built in your calculator (i.e. AT91SAM7L128-EK as shown). Press [Connect] then. This was step 6.

In step 7, put in the address of `calc.bin` on your PC. Then continue according to steps 8 to 13. Not reaching step 7 may be due to low supply voltage on your calculator (see above).

Overlays – Where to Get and How to Make

After flashing successfully, a keyboard overlay is very helpful for further work since most labels deviate from the ones used on said business calculators. You may get adhesive overlays from Eric Rechlin. Preliminary paper overlays are most easily made of a file provided here:

> *http://wp34s.svn.sourceforge.net/viewvc/wp34s/artwork/wp34s_overlay.png*
>
> Set the overall width of this picture to 68mm and print it. Cut it out, span it over your WP 34S using some transparent adhesive tape, and you are done.

Commands for Handling Flash Memory on Your WP 34S

Flash memory is very useful for backups as explained *above*. Alternatively to the commands SAVE and LOAD contained in X.FCN (see the *index of operations*), you may use another approach. Hold down **ON** (i.e. **EXIT**) and press one of the following keys:

STO for backup:	Creates a copy of the RAM in flash memory like SAVE does.
RCL for restore:	Restores the most recent backup like LOAD does.
S (i.e. **6**) for SAM-BA:	Clears the GPNVM1 bit and turns the calculator off. **ATTENTION:** You can now only boot in SAM-BA mode! Without the SAM-BA software and the cable mentioned above, you will be lost!

These ON key combinations have to be pressed twice in a row without releasing the ON key to be executed.

We recommend doing a SAVE or **ON** + **STO** before flashing a new release! After flashing, your backup will still be available – if you used **ON** + **S** to get into SAM-BA boot mode and didn't accidently press the ERASE button on the cable.

Further flash memory operations are PRCL, PSTO, P⇆, RCF, RCF.RG, and RCF.ST. See the *index*.

Mapping of Memory Regions to Emulator State Files

Region	Start address in flash	State file	Remarks
Unnamed	`0x11FC00`	`wp34s-R.dat`	Backup of 112 registers, flags and state.
0	`0x11F800`	`wp34s-0.dat`	Backup of program memory (506 steps).
1	`0x11F400`	`wp34s-1.dat`	Space for generic user programs. Each region contains 506 steps again. The files `wp34s-`***n***`.dat` are written whenever a respective flash command is executed. You will find some sample files at Sourceforge.
2	`0x11F000`	`wp34s-2.dat`	
3	`0x11EC00`	`wp34s-3.dat`	
4	`0x11E800`	`wp34s-4.dat`	
5	`0x11E400`	`wp34s-5.dat`	
6	`0x11E000`	`wp34s-6.dat`	
7	`0x11DC00`	`wp34s-7.dat`	
8	`0x11D800`	`wp34s-8.dat`	
9	`0x11D400`	`wp34s-9.dat`	
RAM	n/a	`wp34s.dat`	Backup of the emulator RAM area (registers, state, and programs) – this file is written only when exiting the emulator.

All files are only read into memory at emulator startup.

Data Transfer Between Your WP 34S and Your PC (SAM-BA)

This method is superseded by the one using serial I/O commands – see next paragraph. It is still interesting enough to leave it here as a reference.

The entire RAM is saved to address `0x11F800` (relative address `0x1F800`) by SAVE or its equivalent ON + STO . This content can be copied to your PC or loaded from it if the special interface cable mentioned above is connected. Then, the transfer is performed as follows:

1. From calculator to PC:
   a. Press ON + STO,
      then ON + D (see below),
      then ON + S .
   b. Press ON once again and start SAM-BA on the PC. Both devices should connect.
   c. Set the start address to `0x11F800` and the size to `0x800`.
   d. Enter a file name of your choice in the receive field. You can now receive the file with SAM-BA.
   e. Move it into your emulator directory (where `wp34sgui.exe` is stored) under the name `wp34s.dat` .
   f. The emulator should accept the file. Your registers and programs will then be in place.

g. To get your calculator back in business, start the "Boot from flash" script in SAM-BA – the same procedure you should know from flashing the firmware.

h. Reset and press ON to power up. Restore with ON + RCL. If RAM is lost (most probably due to an accidental press of the ERASE button on the cable), the most recent backup (i.e. the one of step a. here) will be automatically restored.

2. From PC to calculator:

   a. Execute steps 1.a + b.

   b. Set the start address to `0x11F800`.

   c. Point SAM-BA to your `wp34s.dat` file from the emulator.

   d. You can now send the short file with SAM-BA.

   e. Execute steps 1.g + h.

The program regions accessible with the commands PSTO, PRCL and P⇆ are stored at addresses mentioned in the table above and have a length of `0x400` (1 kB) each. The emulator creates files `wp34s-`***n***`.dat`, with ***n*** being the region number. You can handle these files the same way as the complete state file from the emulator. The regions have identical formatting and can be swapped by copying their data to the 'wrong' place. The register and state portion of the backup area at `0x11FC00` is formatted differently.

If you want to get your emulator data from your PC into your calculator all in once, do the following in Windows:

```
copy /b calc.bin+wp34s-9.dat+wp34s-8.dat+ … +wp34s-2.dat
        +wp34s-1.dat+wp34s.dat calc-full.bin
```

As an alternative, the following will copy the backup data instead of the RAM state file:

```
copy /b calc.bin+wp34s-9.dat+ …
        +wp34s-1.dat+wp34s-0.dat+wp34s-R.dat calc-full.bin
```

The resulting file can be transferred into flash and all data will be readily available.

Data Transfer Between Your WP 34S and Your PC (Serial I/O)

You will need the special interface cable mentioned above once again, or a *modified 20b or 30b* as described elsewhere. Said special cable draws current from the batteries of your calculator; it shall thus be disconnected from your WP 34S as soon as not needed anymore.

Communication is between your WP 34S and another WP 34S. The Windows emulator counts as a valid partner so you can exchange data between your WP 34S and the PC. Since PCs tend to have more than one port you have to tell the emulator which one to use. Create a text file `wp34s.ini` in the directory where the state files `wp34s.dat` reside and put the name of the port as the only line in this file, e.g. `COM5:` – the very same port SAM-BA uses to access your WP 34S for flashing.

The following commands allow for sending programs, registers or all RAM. They are found in the P.FCN catalog.

On the receiving device, start the command RECV. It will display `Wait...`.

On the sender you have four choices:

1. SENDP will send the user program space. After successful termination, the receiver will display `Program`.

2. SENDR will send the registers 00 to 99. The receiver will display `Register` after successful termination.
3. SENDA will send the complete 2 KB of non-volatile memory. The receiver will display `All RAM` after successful termination.
4. SENDL ***n*** will send a library region directly. It will arrive in RAM and may be stored using PSTO.

The commands for sending and receiving feature a fixed timeout of some 10 seconds for setting up the connection. After an interval of inactivity of said length, `I/O Error` is displayed indicating no communication has occurred. If `I/O Error` appears in the middle of a transmission try again.

> On a device without the crystal installed, you may get said error because of the baud rate setting may be a bit too far off. To determine the speed, use the loop
>
> ```
> CLx
> INC X
> BACK 01
> ```
>
> and let it run for 30 seconds. The expected result at nominal speed is around 191000. The I/O commands accept a correction factor in percent in **X**. Try with 95 if your device is a bit too slow or 105 if it is a bit too fast. Values between 80 and 120 are accepted – all other are ignored. On the emulator or with the crystal installed, ***x*** is ignored.

The little "=" annunciator is lit while the serial port is in use. **EXIT** can be used to abort the communication.

More Keyboard Commands Employing ON

ON + **+** or **−** : Adjust display contrast.

ON + **C** : Tells the system a quartz **c**rystal is installed for the real time clock. The quartz is inevitable prerequisite for the clock being useful in medium to long range (see TICKS). Its installation is a hardware modification described elsewhere.

> **ATTENTION:** If this command is entered though the hardware does not contain said modification, the system will hang and can only be brought back to live with a reset or a battery pull!

ON + **D** : Enters **d**ebugging mode (use at your own risk).

ON + **.** : Toggles the radix mark as **./,** does.

<u>**APPENDIX B: MEMORY MANAGEMENT**</u>

This chapter discusses how the two kilobytes of non volatile RAM available are divided in program area, local and global data. There are five distinct sections:

1. Program Memory,
2. Subroutine return stack,
3. Registers used for cumulative statistics (optional)
4. Global registers, i.e. general purpose registers and stack,
5. Status and configuration data.

These sections are ordered as above. This chapter deals with information about the variable boundaries between them.

A complete copy of the nonvolatile RAM can be written to flash region 0 using SAVE. See *Appendix A* for more information about the handling of flash memory.

The Memory Sections in Detail

The block reserved for **program data** is 1kB. This is exactly the size of a flash region. Each step takes two bytes usually, the fields holding the current size and the checksum are two bytes wide each. This translates to a maximum of 510 standard program steps. The multi byte labels and multi character alpha strings take two such steps each.

The program area is followed by the **subroutine return stack**. Unused program steps can be taken over by return addresses and local data. There is no command to set the size. The boundary between the return stack and the program data is determined by the number of steps currently stored. When new program steps are entered, the subroutine return stack is reset to make room. There are situations where this will not return program memory to the full size: The summation registers may have taken their share (see below).

Global registers are placed near the end of available memory. The numbered registers **R00** to **R99** precede the stack and special registers **X, Y, Z, T, A, B, C, D, L, I, J**, and **K** as shown *above*. This totals to 112 registers – i.e. 896 bytes, since each register occupies eight bytes (or four steps). For programs not needing all the numbered registers the amount can be reduced by specifying the highest numbered register using the command REGS (see above).

When REGS is executed is the beginning of the register area is moved. When the new upper limit is lower than it was, the move goes towards higher addresses. Increasing the number of registers will move the boundary towards lower addresses. The memory contents are moved, too, preserving the register data for the remaining registers. Any registers added are zeroed. The lettered registers stay in their place.

> For example, start with the default 100 registers allocated. After a while, call REGS 85. Thereafter, **R85** is followed directly by **X**, and ***r86*** to ***r99*** are lost. The physical address of **R00** is where **R14** was before. The indirect indices change so **R00** is always addressed by 0 in the index register, regardless of memory allocation. In this example, index values 86 to 99 become invalid and will throw a range error.

The space gained is added to the return stack section. It may be necessary to reduce the number of global registers before using Σ+ the first time because the summation registers reside outside the numbered register set (see below). Since the return stack and the summation data are tied to the boundary just moved, they are copied into their new location together with the register data. This makes it possible to execute REGS in the middle of a subroutine without disrupting the program.

Summation Registers

The registers needed for cumulative statistics are no longer held in user register space but are allocated separately. This offers the opportunity to deal with higher internal precision and avoids clobbering their contents manually. The only way to set these registers is with Σ+ and Σ– . They can be recalled by their respective dedicated commands only, and cannot be accessed by STO or RCL.

The first invocation of Σ+ allocates the required space just below **R00** and above the subroutine return stack, pushing the latter down in memory. It may be necessary to make room first (see below) – 140 bytes are required for the 14 summation registers (4 bytes for Σn, 4 × 16 bytes for Σx^2, Σy^2, Σxy, and $\Sigma x^2 y$, and 9 × 8 bytes for the remaining sums). When the number of data points reaches zero, either by Σ–, by CLΣ, CLALL, or RESET, the block is deallocated again. All pointers are automatically adjusted so that this allocation or deallocation does not disrupt a running program.

When memory allocation fails, an error is thrown. Statistical functions will throw an error if not enough data points have been entered. Recall commands like e.g. Σn return zero if no data is allocated.

Local Registers and Flags, and How to Make Room

Being able to store data locally without worrying about overwriting the user's resources adds greatly to the flexibility of a program. This is accomplished by using the subroutine return stack to store not only return addresses but also local registers and flags. Before you can use this feature, you may have to make room.

The total size available for the return stack section is given by the following equation:

number of levels = free program memory in steps
+ fixed minimum return stack section (24 levels)
+ memory freed by REGS if applicable (i.e. 4 levels per register)
– memory used by summation data (i.e. either 0 or 70 levels).

With 100 general purpose registers and the program area filled completely, you will get just 24 return levels. In this configuration it is impossible to allocate the summation registers.

You have two options for increasing the space available:

1. Reduce the number of fixed registers. Each register is worth four additional levels.
2. Move programs to a flash region and clear the respective steps in RAM. Four cleared steps allow for one additional register.

Which solution suites you best depends on the application. You may of course do both. Use MEM? to query the free space in steps. Use LOCR to reserve local registers and flags. Further commands involved are LOCR?, POPLR and RTN – see the *index*.

Addressing Local Data

Local registers and flags are addressed with a dot before the number, from .00 to .15. The highest possible direct address is .15 (due to a technical limit explained below). Higher numbered registers can be addressed indirectly only. The maximum value for ***nn*** is 99, giving 100 local registers. Local flags are always allocated from .00 to .15, regardless of the argument given to LOCR.

Indirect addresses go from 112 (a magic number) to 211 (magically a palindrome, in fact just 112 + 100 – 1). The directly addressable registers and the flags range from 112 to 127 (127 is the highest argument that can be stored in an op-code, hence the limit). An easier to remember alternate addressing scheme uses negative indices but this way .00 cannot be accessed and this will not work for any unsigned integer mode. It's just an alternative for those who don't like the magic value 112. Flags and registers are addressed with identical index values.

This scheme allows for addressing a global register using a local index register (like STO→.00 with ***r.00*** < 112), a local register using a global index (STO→00 with ***r00*** ≥ 112), or a local register using a local index (STO→.00 with ***r.00*** ≥ 112).

Allocation and Visibility

After executing LOCR, a frame is allocated on the return stack which contains:

1. a marker,
2. a flag word (2 bytes), and
3. the register data (4 steps per register).

A pointer in memory is initialized to point to this frame. If the pointer is zero, no local registers exist. The number of registers (actually the size of the frame in steps) is encoded in the marker. The flag word is sometimes allocated at the end of the frame instead of its start to ensure a 32 bit alignment of the register data. This is handled transparently. Newly allocated registers are cleared to zero.

Executing LOCR again in the same subroutine level will adjust the number of registers. This requires data copying since the registers are allocated from low to high addresses and the stack grows in the opposite direction. With the number of registers increasing new ones are zeroed.

LOCR? will give you the number of local registers allocated in the routine you are in.

XEQ in a program does not do anything special, it just pushes the return address before it branches to the target. This means a subroutine not executing a LOCR command itself will have access to the caller's local registers. Once LOCR is executed by the subroutine called, the caller's local registers are hidden from view.

RTN or POPLR in a program check if the current stack pointer points to a local frame. If true the stack pointer is moved above the frame where the return address is found and the rest of the stack is searched upwards for another local frame. If one is found its pointer is stored, else the pointer to the active local frame is cleared. RTN will pop and branch to the return address while POPLR will just continue execution. As a result, the current local frame is dropped and the next higher frame reactivated.

Manually executing RTN, starting a new program with XEQ, or program editing will clear the return stack and remove any local data by clearing the pointer. Any contents in any local register or flag are lost then!

Recursive Programming

Using local registers allow for creating a subroutine that calls itself recursively, each invocation dealing with its local data only. Of course the RPN stack is global so you should be careful not to corrupt it.

Here is a recursive implementation of the factorial (not setting LastX correctly):

```
LBL 'FAC'
IP
x>1?
SKIP 02
1
RTN
LocR 00
STO .00
DEC X
XEQ 'FAC'
RCL× .00
RTN
```

APPENDIX C: MORE ROUTINES AND COMMANDS

Library Routines

TVM and WHO live in the library file `wp34s-1.dat`, located in the library directory. Here is how to install these two routines in the emulator and on the calculator.

1. Copy `wp34s-1.dat` into the emulator directory.
2. Start the emulator and the calculator with the serial cable still connected. Make sure a file `wp34s.ini` exists in the emulator directory naming the COM port in use.
3. Make sure you have a backup of your programs on the calculator and on the emulator.
4. Use PRCL 1 on the emulator to copy the library into user program RAM.
5. Use RECV on the calculator and SENDP on the emulator. This will transfer the program memory of the emulator to the calculator.
6. On the calculator, use PSTO to save the library.
7. Restore your backups.

Alternatively use SAM-BA to transfer the image directly to a RAM region as described elsewhere.

APPENDIX D: RELEASE NOTES

	Date	Release notes
1	9.12.08	Start
1.1	15.12.08	Added the table of indicators; added NAND, NOR, XNOR, RCLWS, STOWS, //, N, SERR, SIGMA, < and >; deleted HR, INPUT, 2 flag commands, and 2 conversions; extended explanations for addressing and COMPLEX & …; put XOR on the keyboard; corrected errors.
1.2	4.1.09	Added ASRN, CBC?, CBS?, CCB, SCB, FLOAT, MIRROR, SLN, SRN, >BIN, >DEC, >HEX, >OCT, BETA, D>R, DATE, DDAYS, D.MY, M.DY, Y.MD, CEIL, FLOOR, DSZ, ISZ, D>R, R>D, EMGAM, GSB, LNBETA, LNGAMMA, MAX, MIN, NOP, REAL, RJ, W and WINV, ZETA, %+ and %-; renamed the top left keys B, C, and D, and bottom left EXIT.
1.3	17.1.09	Added AIP, ALENG, ARCL, AROT, ASHF, ASTO, ATOX, XTOA, AVIEW, CLA, PROMPT (all taken from 42S), CAPP, FC?C, FS?C, SGMNT, and the …# commands; renamed NBITS to BITS and STOWS to WSIZE; specified the bit commands closer; deleted the 4 carry bit operations.
1.4	10.2.09	Added CONST and a table of constants provided, D>J and J>D, LEAP?, %T, RCL and STO ▲ and ▼, and 2 forgotten statistics registers; deleted CHS, EMGAM, GSB, REAL and ZETA; purged and renamed the bit operations; renamed many commands.
1.5	5.3.09	Added RNDINT, CONV and its table, a memory table, the description of XEQ B, C, D to the operation index, and *a* and g_e to the table of constants; put CLSTK on a key, moved CLΣ and FILL, changed the % and log labels on the keyboard, put CLALL in X.FCN; checked and cleaned alpha mode keyboard and added a temporary alpha keyboard; rearranged the alphabet to put Greek after Latin, symbols after Greek consistently; separated the input and non-programmable commands; cleaned the addressing tables.
1.6	12.8.09	Added BASE, DAYS+, DROP, DROPY, E3OFF, E3ON, FC?F, FC?S, FIB, FS?F, FS?S, GCD, LCM, SETDAT, SETTIM, SET24, SINC, TIME, VERS, αDAY, αMONTH, αRC#; %Σ, as well as F-, t-, and χ^2-distributions and their inverses; reassigned DATE, modified DENMAX, FLOAT, αROT, and αSHIFT; deleted BASE arithmetic, BIN, DEC, HEX, and OCT; updated the alpha keyboards; added flags in the memory table; included indirect addressing for comparisons; added a paragraph about the display; updated the table of indicators; corrected errors.
1.7	9.9.09	Added P.FCN and STAT catalogs, 4 more conversions, 3 more flags, Greek character access, CLFLAG, DECOMP, DENANY, DENFAC, DENFIX, Iβ, IΓ, αDATE, αRL, αRR, αSL, αSR, αTIME, 12h, 24h, fraction mode limits, normal distribution and its inverse for arbitrary μ and σ, and Boolean operations working within FLOAT; deleted αROT, αSHIFT, the timer, and forced radians after inverse hyperbolics; renamed WINV to W^{-1}, and beta and gamma commands to Greek; added tables of catalog contents; modified label addressing; relabeled PRGM to P/R and PAUSE to PSE; swapped SHOW and PSE as well as Δ% and % on the keyboard; relabeled Q; corrected CEIL and FLOOR; updated X.FCN and alpha commands; updated the virtual alpha keyboard.
1.8	29.10.09	Added R-CLR, R-COPY, R-SORT, R-SWAP, RCLM, STOM, alpha catalogs, 1 more constant and some more conversions, a table of error messages, as well as the binomial, Poisson, geometric, Weibull and exponential distributions and their inverses; renamed some commands; put √‾ instead of π on hotkey D.
1.9	14.12.09	Added two complex comparisons; swapped and changed labels in the top three rows of keys, dropped CLST; completed function descriptions in the index.
1.10	19.1.10	Added IMPFRC, PROFRC, ^{C}ENTER, αBEG, αEND, and an addressing table for items in catalogs; updated temporary alpha mode, display and indicators, RCLM and STOM, alpha-commands and the message table; renamed the exponential distribution; wrote the introduction.
1.11	21.9.10	Changed keyboard layout to bring Π and Σ to the front, relabeled binary log, swapped the locations of π, CLPR, and STATUS, as well as SF and FS?; created a menu TEST for the comparisons removed and the other programmable tests from P.FCN; added %MG, %+MG, %MRR, RESET, SSIZE4, SSIZE8, SSIZE?, ^{C}DROP, ^{C}FILL, ^{C}R↓, ^{C}R↑, registers J and K, a table of contents and tables for stack mechanics and addressing in complex operations; updated memory and real number addressing tables, DECOMP, αOFF, αON, Π, and Σ; renamed ROUNDI, WSIZE?, β(x,y), Γ(x) and the constant p_0 ; deleted DROPY (use x⇔y, DROP instead), αAPP, αBEG, αEND, and the “too long error” message; deleted Josephson and von Klitzing constants (they are just the inverses of other constants included in CONST already); brought more symbols on the alpha keyboard.
1.12	22.12.10	Modified keyboard layout; added catalogs MODE and PROB; changed mode word, catalog contents and handling (XEQ instead of ENTER), as well as some non-programmable info commands; expanded IMPFRC and PROFRC; added a paragraph about the fonts provided and explained alpha catalogs in detail; added PRIME? and some conversions; deleted FRACT, OFF and ON.
1.13	3.2.11	Modified keyboard layout; modified αTIME, radix setting, H.MS+ and H.MS-; added EVEN?, FP?, INT?, LZOFF, LZON, ODD?, RCLS, STOS, returned FRACT; added and renamed some conversions; updated the paragraph about display; added appendices A and B; baptized the device WP 34S.

1.14	18.3.11	Started the Windows emulator. Added DEC and INC, renamed FLOAT to DECM; redefined αTIME and H.MS mode; updated appendix A; documented the annunciators BEG and = as well as underflows and overflows in H.MS; corrected some errors showing up with the emulator.
1.15	21.3.11	Modified FIX, removed ALL from MODE, updated CONV.
1.16	27.3.11	Added LBL?, f'(x), and f"(x); modified PSE; upgraded catalog searching.
1.17	9.5.11	Modified keyboard layout for adding a fourth hotkey; added AGM, BATT, B_n, B_n*, Cauch, Lgnrm, Logis and their inverses, all the pdf, COV, CUBE, CUBERT, DEG→, ENGOVR, ENTRY?, erfc, GRAD→, GTO . *hotkey*, KEY?, RAD→, SCIOVR, SERRw, SLVQ, sw, sxy, TICKS, TVM, xg, ε, ε_m, ε_p, ζ, σw, $(-1)^x$, the polynomials, four angular conversions, four Planck constants, the regional settings, global alpha labels, and three messages; renamed most cdf; changed →DEG, →RAD, →GRAD to leaving angular mode as set; altered PSE for early termination by keystroke; made D.MY default instead of Y.MD; moved degrees to radians conversions to CONV; removed ^{C}CLx, H.MS mode, %+ and %-; corrected errors.
1.18	5.6.11	Expanded program memory; modified label addressing (A ≠ 'A') and fraction mode limits, changed ANGLE to work in real and complex domains, renamed MOD to RMDR, changed the keyboard layout; put BACK, ERR, SKIP, and SPEC? to the main index; added CAT and the I/O commands for flash memory, expanded R-COPY; corrected x→α.
2.0	21.7.11	Entered beta test phase. Added DAY, MONTH, YEAR, FAST, SLOW, S.L, S.R, VWα+, flag A, ON + and –, some constants, and a paragraph about I/O; renamed old DAY to WDAY, RRCL to RCFRG, SRCL to RCFST; added an inverse conversion shortcut, stones⇆kg, and changed Pa⇆mbar to Pa⇆bar; modified the VIEW commands, ALL, DISP, MODE, RCLM, STOM, and X.FCN; repaired hyperlinks; corrected some errors; included flash.txt; updated the first chapters, explained stack mechanics in more detail.
2.1	3.10.11	Added serial I/O commands, DEL_P, DSL, EXPT, IBASE?, INTM?, ISE, KTY?, MANT, NEXTP, PUTK, REALM?, RM, RM?, SMODE?, TOP?, $\sqrt[x]{y}$, signed tests for zero, some constants, and the paragraph about interactive programming; updated the values in CONST to CODATA 2010, also updated SLVQ, SHOW, Σ, Π, and the paragraphs about statistics, predefined alpha labels and memory; corrected some errors; deleted complex ANGLE, →BIN, →DEC, →HEX, and →OCT; redistributed the contents of X.FCN and P.FCN; renamed S.L and S.R to SDL and SDR; put '?' on the alpha keyboard and moved £ to P to make room for π; expanded Appendix A; reorganized the structure of the document; added first aid to the front page; rewrote the keyboard chapter.
2.2	1.11.11	Added MSG, y⇔, z⇔, and matrix operations, a paragraph about them and two new error messages for them, plus a footnote for DEL_P; updated the introduction to statistics.
3.0	17.11.11	Added LocR, LocR?, MEM?, PopLR, REGS, REGS?, as well as CFIT and MATRIX catalogs; removed LNβ, LNΓ, β, and Γ from STAT; changed keyboard layout to bring MATRIX to the front and to swap OFF and SHOW; removed x⇔a from the key plate; slightly modified the virtual alpha keyboard; redistributed commands in the catalogs; updated the sections about memory and messages, added one about local registers and another one about memory management, deleted the one about internal commands. **WARNING: This is just a working document to support discussion of new features.**