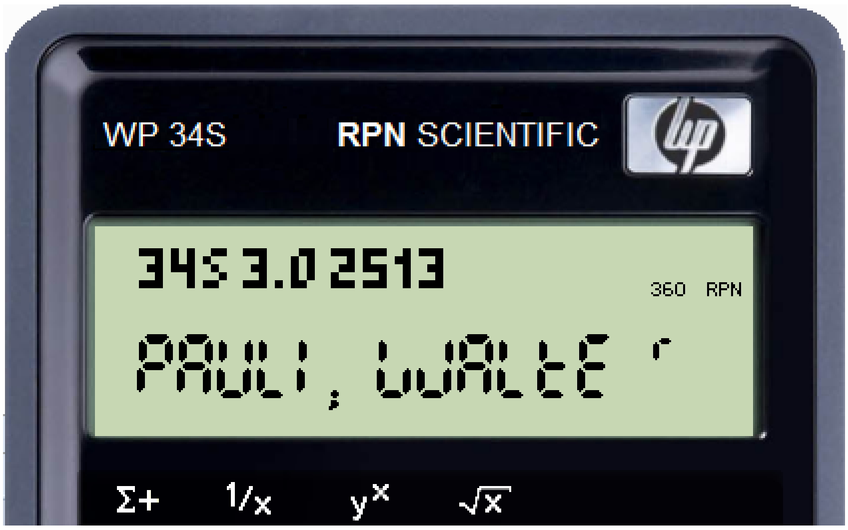
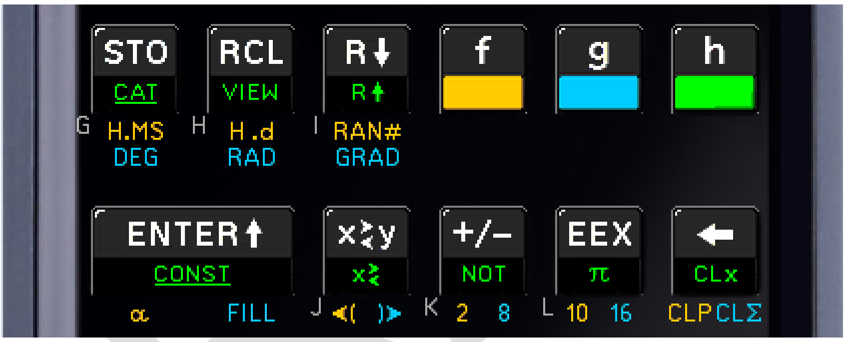


3.0 Owner's Manual



First aid if 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 default 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 for 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 a *HP-20b Business Consultant* or a *HP-30b Business Professional*, so you benefit from their excellent processor speed. And with a *HP-30b* you get the famous rotate-and-click keys in addition, giving you the tactile feedback appreciated in vintage *Hewlett-Packard* calculators for decades.

On the other hand, the firmware and user interface of your *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 <u>scientific</u> calculator like you have never had before** – keystroke programmable and comfortably fitting in your shirt pocket.

The function set of your *WP 34S* 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 like featured by the *HP-21S*, and added **many more useful functions for mathematics, statistics, physics, engineering, programming etc.** like

- Euler's Beta and Riemann's Zeta functions, Bernoulli and Fibonacci numbers, Lambert's W, the error function as well as Chebyshev's, Hermite's, Laguerre's and Legendre's orthogonal polynomials, and 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,
- integer computing in arbitrary bases from binary to hexadecimal,

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

- financial operations like mean rate of return and margin calculations,
- 84 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 from mathematics, astronomy, and surveying,
- complete Greek and extended Latin letter sets covering the languages of almost half of this planet (upper and lower case in two font sizes each).

The *WP 34S* is the first RPN calculator overcoming the limits of a four-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 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 up to 112 global general purpose registers, 112 global user flags, a 31 byte alpha register for message generation, up to 925 program steps in *RAM*, several thousand steps in flash, 16 local flags and up to 144 local registers allowing for recursive programming, and 4 programmable hotkeys for your favorite functions or routines. Memory layout is user-settable to a large extent. And you may save your work in battery-fail-safe on-board backup memory, and communicate serially with a PC.

The *WP 34S* is the result of a collaboration of two individuals, an Australian and a German, since 2008. 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, we discussed our project in 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 – even a symbolic preprocessor was added with v2.1. Marcus did a great job presenting v2.2 at the HHC 2011 in San Diego.

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.

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, titles or abbreviations* 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 quick and easy handling of repeated or iterative computations. Further documentation, also about the other calculators mentioned in this manual, 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 presenting you all the new features. It contains the necessary information including some formulas and technical explanations but is not intended to replace textbooks about mathematics, statistics, physics, engineering, programming, or the like.

Your *WP 34S* is designed to help you in calculations and computations. It is, however, just a tool – though a very powerful one – it cannot think for you nor can it check the sense of the problem you apply it. Gather information, think before keying in and check your results: these tasks will remain yours always.

The following text starts presenting you the user interface as it will be active in various modes, so you know where to find what you are looking for. It continues demonstrating some basic methods, the calculator memory and addressing items therein, as well as the display and indicators giving you feedback what is going on. Then the major part of this booklet is taken by an index of all operations featured and how you access them, as well as lists of catalog contents including the constants and conversions featured. This manual closes with some special topics, e.g. a list of messages your *WP 34S* will return if abnormal conditions prevent it from executing your command as expected.

THE USER INTERFACE

Let us investigate your *WP 34S*: Please take off the battery cover, locate the little hole between the batteries labeled RESET, and use a paper clip to reset. This will erase all user contents and give you a fresh start. Close the cover again and press ON (the bottom left key) to turn your calculator on. You will get what you see below:

Keyboard Basics

As usual, white labels denote the default *primary functions* of the respective keys.

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

To access a white label, just press the corresponding key (thus it is called the *primary* function). For a golden, blue, or green label, press the *prefix* f, g, or h, respectively, then the corresponding key. Let us take 5, for example:

- 5 will enter the digit 5,
- f + 5 will calculate the arithmetic mean values of the data accumulated in the statistic registers via $\bar{x}$,
- g + 5 will return the standard deviations for the same data via s ,
- h + 5 will open a *catalog* of extra statistical functions via STAT . Generally, labels printed underlined point to catalogs.
- The grey letter R will become relevant in *alpha mode*, i.e. for input of text.

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

Time for a little example. Turn your *WP 34S* on again if necessary – it may have gone off automatically in between. Anyway it will still show

.

Generally, 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 80 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. As soon as you press →P, your *WP 34S* does the necessary calculation of $\sqrt{40^2 + 30^2}$ automatically. Of course it will calculate as well whatever other distances apply for you. 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, pressing →P calls the function →POL, converting rectangular to polar coordinates. Most labels printed on your *WP 34S* simply call operations carrying the same name as the respective label – there is only a limited number of cases like →P. Let us introduce them, starting top left on the keyboard:

- A, B, C, and D are named *hotkeys*, since they immediately call the user programs carrying these labels if defined. If the respective labels are not defined (yet), these keys act as Σ+, 1/x, y^x, or $\sqrt{x}$, respectively.
- HYP is the prefix for hyperbolic functions SINH, COSH, and TANH, as HYP^{-1} is for their inverses ASINH, ACOSH, and ATANH. In analogy, SIN^{-1} stands for ASIN, etc.

[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.

- ➡ 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 that arrow. Additionally, ➡ trailed by 2, 8, or 16 will show x converted to an integer number of the respective base until the next keystroke. And furthermore, ➡ 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 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, with H.d standing for decimal hours and also for floating point numbers in general (see DECM).
- α enters *alpha mode*, while 2, 8, 10, or 16 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, and ⬆ upper and lower case in alpha mode.
- |x| calls ABS.

These are all the special labels featured by your *WP 34S*. You will find these as well as each and every other command provided, the keys to access it, and the necessary individual explanation in the *index of operations* below for your reference.

In four decades of pocket calculators, a wealth of nice to sophisticated application examples were created and described by different authors – more and better than we can ever create ourselves. It is not our intention to copy these old examples. Instead, we recommend the DVD mentioned *above* once again: 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.

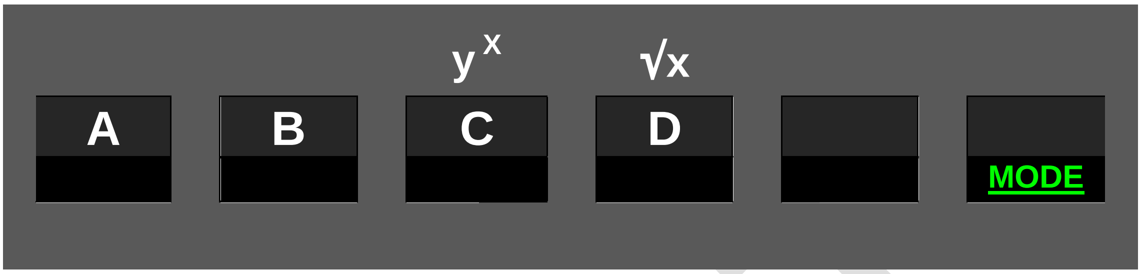
Let us return to our introductory example for two remarks:

1. There is no need to enter any units. The example will work with e.g. meters as well.
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.

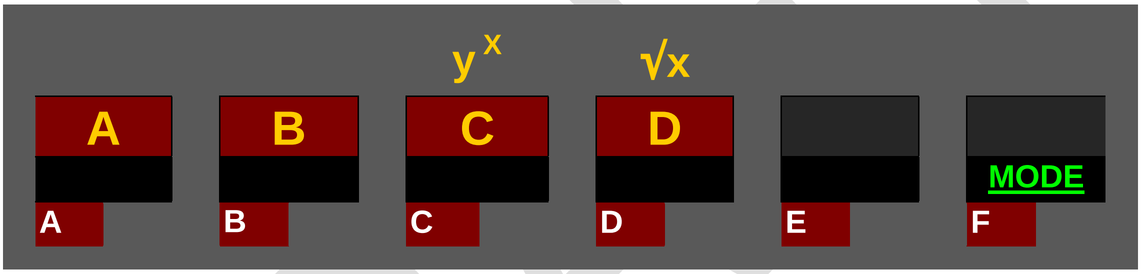
Your *WP 34S* features more modes – we want to briefly introduce some of them to you (you will find a complete list of all modes provided in a separate chapter further below).

Integer Modes and the Top Row of Keys

These 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 integer computing. In these modes, operations like SIN make no 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, on the other hand, primary functions of these top keys will be reassigned automatically, becoming direct numeric input:



The dark red background is used to indicate changed key functionality in this figure – and this holds for the remaining text as well. White print still denotes primary functions (e.g. the top right key will enter the digit F here directly). Wherever default primary functions are not primary anymore after reassignment, prefix f will access them (e.g. f + D will call √x here). To ease life, pressing any key (or its combination with a prefix) will display its present assignment in current mode for a short time in the top line for checking; it will turn to NULL after 0.5 seconds.

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.

Keyboard Reassignment in Temporary Alpha Mode

This mode is entered during input processing in comparisons and in memory addressing, e.g. during storing, regardless of the mode set before. Examples are shown *below*. See the respective virtual keyboard here:

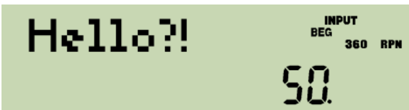
Note all keys are primary in this mode – no shift keys are used.

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

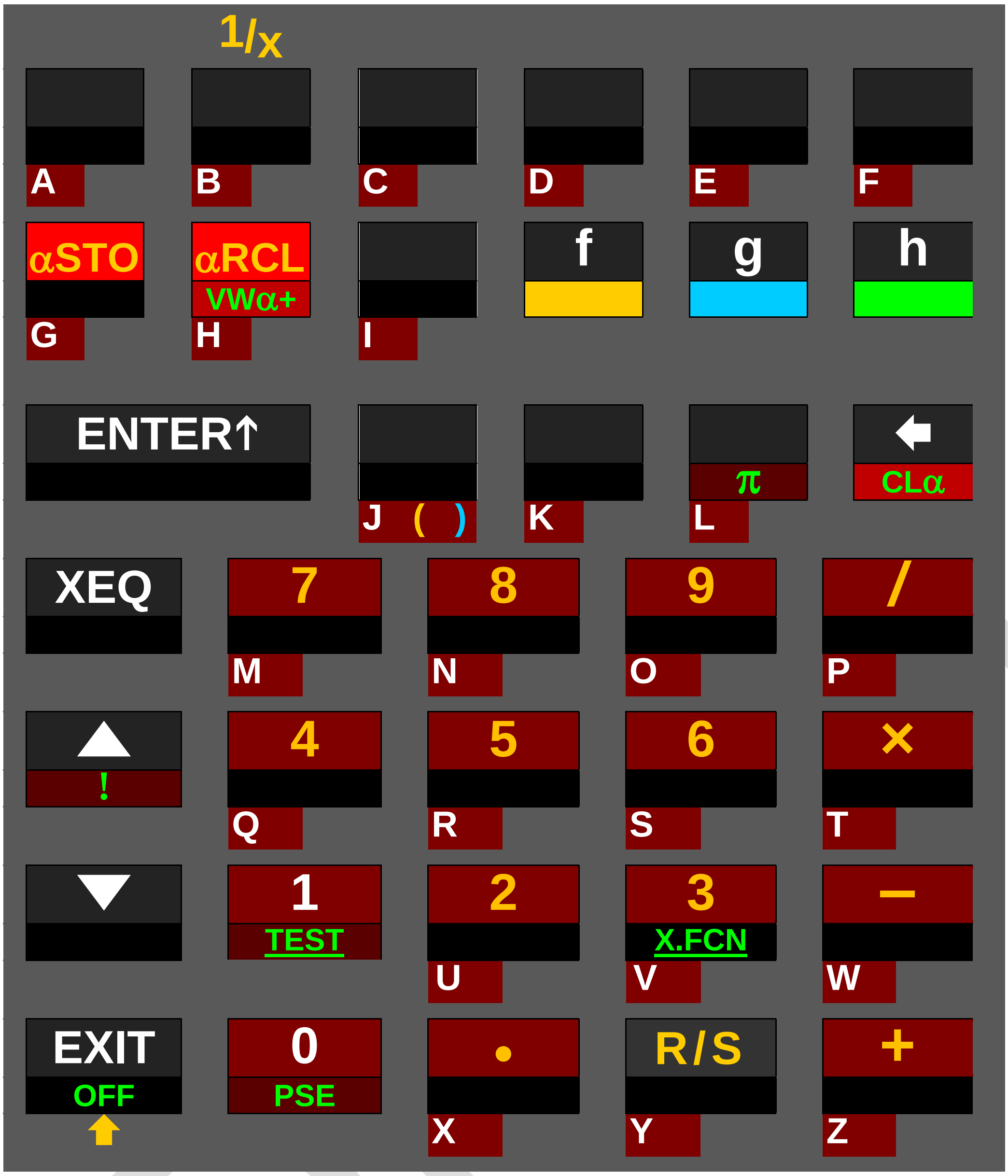
Temporary alpha mode is closed (returning to the mode set before) when sufficient characters are put in for the respective command. Pending input may be canceled and temporary alpha mode left early by EXIT.

The Virtual Keyboard in Full Alpha Mode

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, most mathematical operations are neither needed nor applicable. So the keyboard is reassigned 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.

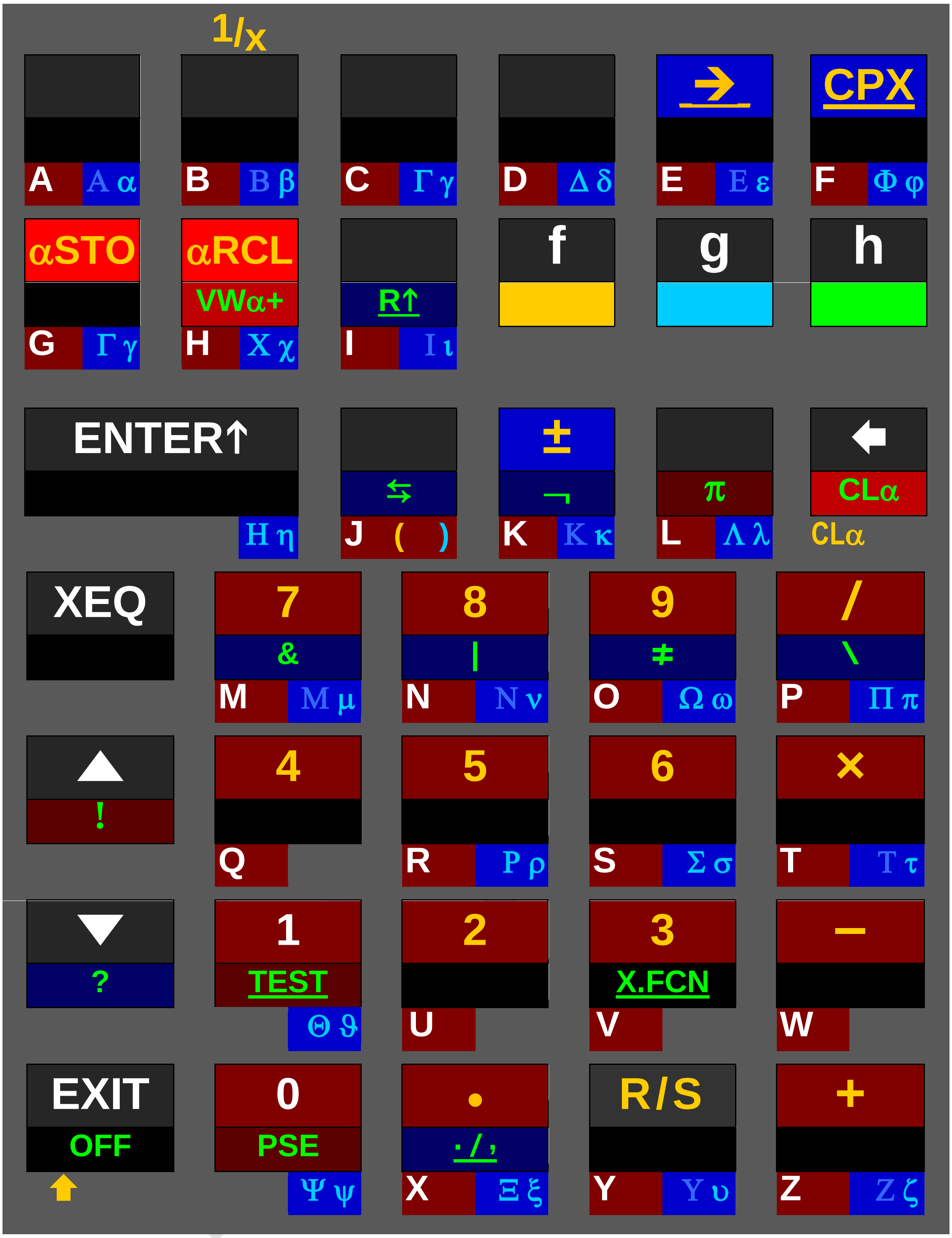
Primary function of most keys is appending the letter printed bottom left of this key – grey on the key plate. PSE appends a space. When ***alpha*** exceeds 31 characters, the leftmost character(s) are discarded. Alpha mode starts with capital letters, and ⇑ toggles upper and lower case. As in integer modes, f will access default primary functions wherever necessary [3].

Looking at the standard labels on the keyboard, we can safely offer you even more: All labels printed on dark blue background overleaf append characters to ***alpha*** as well. They are related to the labels printed on your *WP 34S* keyboard at these locations, but deviate from them. Prefix g leads to homonymic Greek letters where applicable [4]. And h allows accessing logic symbols via the Boolean operations.

---

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

[4] "Homonymic" according to ancient Greek pronunciation. And we assigned **Gamma** also to **C** due to the alphabet, and **Chi** to **H** since this letter comes next in pronunciation. Three Greek letters require special handling: **Psi** is accessed via g 0 (below PSE), **Theta** via g 1 (below TEST and following 'T'),



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

Nevertheless we will not forget your *WP 34S* is a calculator.

---

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.

CALCULATING

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. Note integer numbers like 8, 9, 10, or -1 are just a subset of real numbers.

Basic Calculations in Real Domain

Many real number functions provided operate on one number only. 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 value shown in the display, by the function result **f(*x*)** .

Some of the most popular mathematical functions, however, operate on two numbers. Think of + and –, for example. Assume having an account of 1,234 $ and taking 56.7 $ away from it. What will remain? One easy way to solve such a task works as follows:

On a piece of paper		On your *WP 34S*		
Write down the 1st number:	*1234*	Key in the 1st number:	[1] [2] [3] [4]	1234
Go to next line, write down the 2nd number:	*56.7*	Terminate 1st input, key in the 2nd number:	[ENTER↑] [5] [6] [.] [7]	56.7
Subtract:	*1177.3*	Subtract:	[−]	1177.3

That’s the essence of RPN. Enter the necessary operands, then enter and execute the operation at once.

As the paper holds your operands before you calculate, a place holding your operands on your *WP 34S* is required: the *stack* does that. It will also take care of intermediate results, if applicable, as your paper may do. Think of the stack like a pile of registers: bottom up, they are named **X, Y, Z, T** (for tradition), optionally followed by **A, B, C**, and **D**.

New input is always loaded in **X**, and only ***x*** is displayed in any case at any time. [ENTER↑] copies ***x*** to ***y*** (‘pushes ***x*** on the stack’), so **X** can take another input then. Having completed that input, [−] subtracts ***x*** from ***y***, putting its result **f(*x, y*)** in **X** for display. This holds for almost all two-number real functions. See next paragraph for more.

Stack Mechanics

For the first time ever in a calculator, your *WP 34S* offers a choice of 4 or 8 stack levels. Thus, what will happen with the stack content depends on the particular operation executed, its domain and the stack size chosen. Real functions in a 4-level stack work as known from vintage RPN calculators for decades. In the larger stack of your *WP 34S*, everything works alike – just with more levels for intermediate results. See overleaf for details of the stack mechanics:

	Level	Assumed stack contents at the beginning:	Stack contents after executing … … the stack register operations ENTER	FILL	DROP	x⇆y	R↓	R↑	LASTx	… 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*

Please note the stack contents will drop when a two-number function is executed. The content of the top stack level is repeated (since there is nothing available above for dropping) then. You may employ this top level repetition for some nice tricks.

Using the stack, RPN also makes parentheses obsolete. There is no operator precedence. Here is an example:

$$\frac{1+\left|(4.32-5.67\times 0.89)^4+(1.23-9.88)^3\right|^{0.35}}{\left\{sin\left[\pi\left(\frac{7}{4}-\frac{5}{6}\right)\right]+1.78\times(6.54+5.79)^{(3/7)}\right\}^2-3.5}$$

5.67 ENTER↑ .89 × 4.32 x⇆y − 4 y^x 1.23 ENTER↑ 9.88 − x^3 + |x| .35 y^x 7 ENTER↑ 4 ÷ 5 ENTER↑ 6 ÷ − π × SIN 6.54 ENTER↑ 5.79 + 3 ENTER↑ 7 ÷ y^x 1.78 × + x^2 3.5 − ÷

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

There are also a few three-number real functions included in your *WP 34S* – e.g. Iβ, →DATE and %MRR – replacing x by the result **f(x, y, z)** . Then t drops in $\mathbf{Y}$ and so on, and the content of the top level is repeated twice.

Some real functions (e.g. DECOMP, DATE→) operate on one number but return two or three. 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

Some Special Functions: Statistical Distributions, Probabilities etc.

You will find a lot of statistics built 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 distributions in 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.

[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“.

- 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*). This naming convention holds for **Binom**ial, **Cauch**y, **Expon**ential, **F**isher, **Geom**etrical, **L**o**gN**o**rm**al, **Logis**tic, **Norm**al, **Poiss**on, Studen**t**, and **Weib**u**ll** distributions. Chisquare and Standard Normal distributions are named differently. Please see the *index* and the *catalog PROB*.

- 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.
  - Do you want an upper and 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.

There is a wealth of commands for sample and population statistics in one and two dimensions featured as well. Please see the *index* and the *catalogs STAT and SUMS*.

MEMORY AND ADDRESSING

	Stack registers		General purpose registers	Program steps	User flags
	D *	J ***	R00	000	00
	C *	K ***	R01	001	01
	B *		R02	002	02
Mode	A *		…	…	…
	T				
Alpha (31 B)	Z		…	…	…
	Y	I **	R95	923	99
Display	X	L	R96	924	X = 100
			R97	925	Y = 101
			R98		Z = 102
			R99		T = 103

General purpose registers		User flags
	X = R100	A '='
	Y = R101	B *Big, overfl.*
R.00 = R112	Z = R102	C *Carry*
R.01 = R113	T = R103	D *Danger*
…	A = R104	L = 108
	B = R105	I = 109
…	C = R106	J = 110
R.15 = R127	D = R107	K = 111
R128	L = R108	.00 = 112
…	I = R109	…
	J = R110	
…	K = R111	…
R254		.14 = 126
R255		.15 = 127

This is the **address space** of your *WP 34S*. Depending on the way you configure its memory, a subset of all these addresses will be accessible.

Offering two stack sizes, either **T** or **D** will be the top stack level. **A - D** will be allocated for the stack if required. **I** will be loaded in complex calculations (see *below*). **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 general purpose registers.

Direct numeric addressing works up to **R99**. For *indirect addressing*, the lettered registers and flags carry the fixed addresses 100 … 111 as shown.

Some user flags have special use or effects. Flag A lights the big '=' symbol in display. Flags B and C will be set by the system in integer modes like 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 this flag.

Addresses ≥112 may be used for *local* registers and flags (see *below*).

Note you will not get 926 program steps and 256 registers and 127 flags all together at the same time – see *Appendix B* for memory management.

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? _**	**OP?→ _**	**OP *x*** e.g. SF K	**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.* 0 0 … 9 9	Look right for more about indirect addressing.	Sets flag 111.	*Stack level etc.* X, Y, Z, ... , K	*Register number* 0 0 … 9 9
	Dot matrix display			**OP? *nn*** e.g. x≠? 23			**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, while SCI ENTER↑ calls SCIOVR. See the index of operations.

[7] You may skip this keystroke for numbers >19 or local registers. The latter start with a . – see the chapter about Programming above and Appendix B below.

[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 … 255 (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 are 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.

Advanced Calculations: Real Matrices

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

Else 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***| .

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:

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

Depending on the current contents of these registers, the actual matrix may look like this:

$$(M) = \begin{pmatrix} 2.3 & 0 & 7.1 \\ 0.4 & 8.5 & 6.9 \end{pmatrix}$$, for example.

If ***cc*** is omitted in a descriptor, it is set to ***rr*** so a square matrix is assumed. E.g. a descriptor 13.04 belongs to a 4×4 matrix with its elements stored in **R13** through **R28**. 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` .

Please see the *index* and the *catalog MATRIX* for all commands featured.

More Advanced Calculations: Complex Domain

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.

Please turn to the stack diagrams printed on next page for further details.

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 e.g. ^{C}1/x, ^{C}ABS, ^{C}FP, ^{C}IP, ^{C}RND, ^{C}x!, $^{C}x^2$, $^{C}x^3$, $^{C}\sqrt{x}$, $^{C3}\sqrt{x}$, ^{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 $\mathbf{f}(x, y)$ as shown above. 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 on next page. 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.

See the *index* for all commands supported in complex domain. Most of them are contained in the *complex X.FCN catalog*.

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.

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? _**	**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 ^{C}LASTx – the imaginary part is recalled from register **I** to **Y**.	*Stack level or named register* X, Y, ... , K	*Register number* 0 0 … 9 9
	Dot matrix display			**OP? *nn*** e.g. `x≠? 26`			**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.

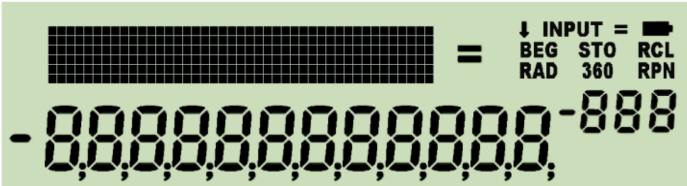
[12] You may skip this keystroke for numbers >19 or local registers. The latter start with a . – see the chapter about Programming above and Appendix B below.

[13] Exceptions: ^{C}RCL Z, ^{C}RCL + Z, ^{C}STO Z, and ^{C}STO + 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.

DISPLAY AND MODES

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 priorities are as follows:

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

Modes and Annunciators

The *annunciators* or specific characters in the LCD indicate 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 2nd . 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.

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

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. 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 preferences may be set at once using shortcuts:

Command	Radix mark	Three digit separators	Time	Date	JG [15]	Remarks
SETCHN	Point	Off	24h	Y.MD	1949	Would require separators every four digits.
SETEUR	Comma	On	24h	D.MY	1582	Applies also for South America.
SETIND	Point	Off	24h	D.MY	1752	Would require separators every two digits over 10^5.
SETJPN	Point	On	24h	Y.MD	1873	
SETUK	Point	On	12h	D.MY	1752	
SETUSA	Point	On	12h	M.DY	1752	

Please note the peoples 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.

Also 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:

[15] This column states the year the Gregorian Calendar was introduced in the particular region, typically replacing the Julian Calendar (in East Asia, national calendars were replaced in the respective years). The WP 34S supports both 1582 and 1752. See the index of operations.

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

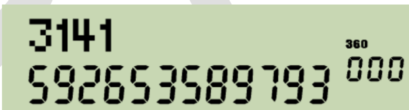
Please see the *index of operations* for the commands printed on white background, and the *catalog of unit conversions* for those printed on yellow.

Command and Mode Specific Output

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

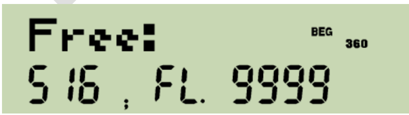
1. **VERS** generates a display similar to the one shown on the title page of this manual. This temporary message will vanish with the next key pressed. ◄ will just clear the message, any other key will be executed.

2. **SHOW** displays the full mantissa of x, i.e. all 16 digits present internally, and the exponent. E.g. π SHOW returns



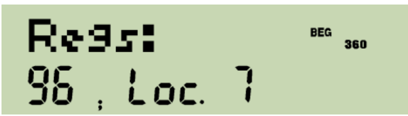
as a temporary message.

3. **STATUS** shows the amount of free memory in RAM and flash first, e.g.:



.

Press ▼ and read the number of global numbered registers allocated, and the amount of local registers (note these are displays of the emulator):



,

After another ▼ you will see the status of 30 user flags, 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, 23, and X 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 in the same format, and finally the top 12 flags as shown here:

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. Pressing a letter like [D] will display the top 12 flags. The numeric exponent always indicates the status of the four hotkeys top left on the keyboard – if all four labels are defined in programs then **ALL** will be shown there.

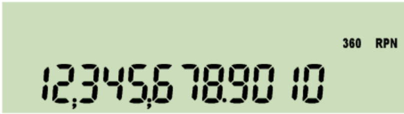
The status will be displayed until [EXIT] is pressed.

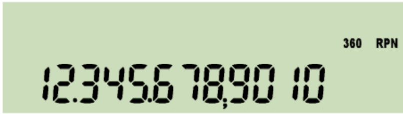
4. During **command input**, the dot matrix displays the command chosen until input is completed, i.e. until all required trailing parameters are entered. The prefixes [f], [g], and [h] are shown until they are resolved. If you pressed any of [f], [g], or [h] erroneously, recovery is as easy as follows:
   - [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*.

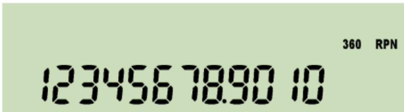
5. In **programming mode**, the numeric display indicates the program step (000 – 925) in the mantissa and the number of free steps in the exponent, while the dot matrix shows the command contained in the respective step, e.g.:

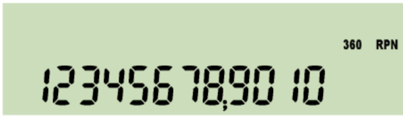
6. For **floating point decimal numbers**, the mantissa will be displayed adjusted to the right, the exponent to the left. Within the mantissa, either points or commas may be selected as radix marks[16], and additional marks may be chosen to separate thousands. Assume the display set to FIX 4, then 12.345678901 millions may look like:



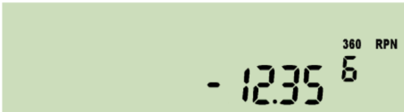
or


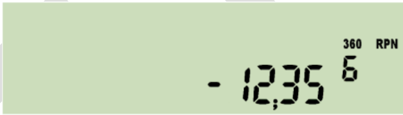
with thousands separators on, and without them like:



or


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



or


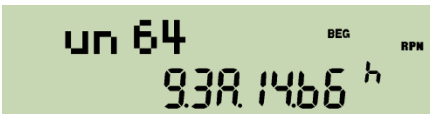
If the last operation executed was a complex one, a capital **C** is displayed top left in the dot matrix pointing to the fact that you find the result of this function in **X** and **Y**.

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*).

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, with or without separators:

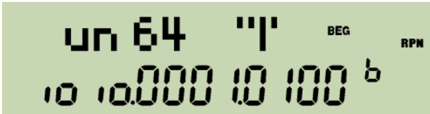


or


[16] 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.

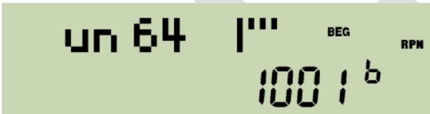
After changing to binary mode, this number will need 28 digits, being 1001001110100001010010110110. The 12 least significant digits will be displayed initially together with an indication that there are four display windows in total with the rightmost shown:

Note the least significant byte is emphasized. Now press ◀ and you will get the next byte (note there is an overlap with the first display):



Press ◀ again:

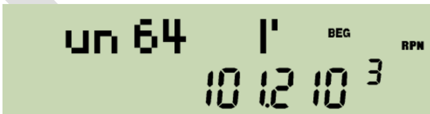
And finally:



If leading zeros were turned on, there will be eight display windows (corresponding to eight bytes) in this case, with the four “most significant” containing only zeros.

Going through the number in steps of eight digits is a specialty of binary integer mode. In any other base the full display width is used as step size. See e.g. the least significant part of the same number in base 3:

and its most significant part:

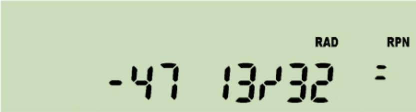


.

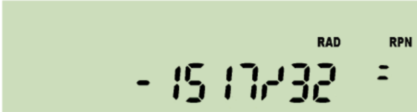
Please note numeric input is limited to 12 digits in any integer base.

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:

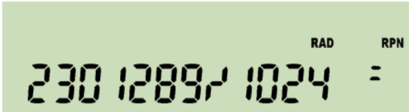


or after d/c :



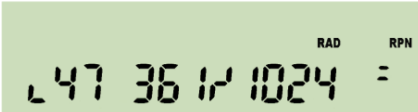
[17].

Squaring this improper fraction results in



.

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



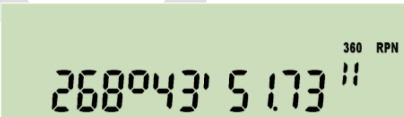
with a little hook left of the first digit shown. This indicates the leading number is displayed incompletely – there are at least two digits preceding 47 but no more display space. Press SHOW to unveil the integer part of this proper fraction is 2247.

Input in fraction mode is straightforward and logically coherent:

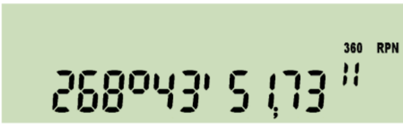
Key in:	and get in proper fraction mode:	
1 2 . 3 . 4 ENTER↑	12 3/4	
1 . 2 ENTER↑	1 1/5	
. 1 . 2 ENTER↑	1/2	
. 1 2 ENTER↑	3/25	(= 0.12)
1 . . 2 ENTER↑	1 0/1	(= 1 $^0/_2$!)

For comparison, note *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 9,000. Output may look like this:



or



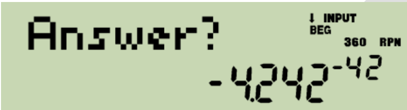
depending on the radix setting. For decimal times less than 5ms or 0.005 angular seconds but greater than zero, an “u” for underflow will be lit in the exponent section. For times or angles exceeding the upper limit, an “o” will be shown there signaling an overflow, and the value is displayed modulo 9,000.

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

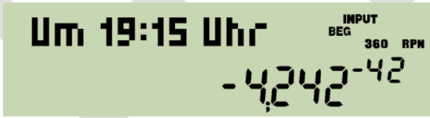
10. Output of the function **WDAY** will look as follows for an input of 1.13201 in M.DY mode (equivalent to inputs of 13.01201 in D.MY or 2010.0113 in Y.MD):

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.:

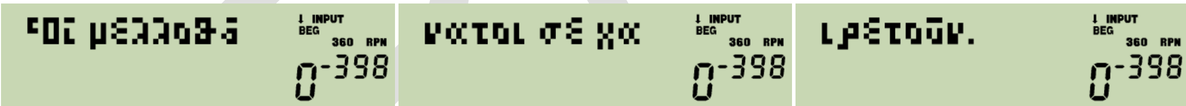


Different information may be appended to ***alpha***. See the commands starting with “α” in the index of operations below. E.g. αTIME allows creating texts like

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*** takes up to 31 characters. And your *WP 34S* features a rich set of special letters and further characters. So you may easily store a message like



▲ and ▼ will browse such long messages in steps of 6 characters. ▲ will stop with the very first characters shown, ▼ stops showing the right end completely, i.e.

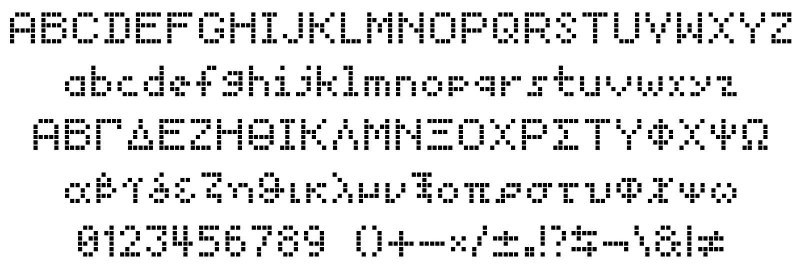

in this very special case.

Continue with reading the next paragraph for more information about the alpha capabilities of your WP 34S.

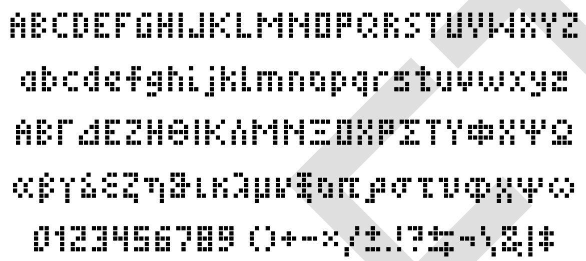
Character Set and Fonts

You may have noticed already your *WP 34S* features a large and a small alphanumeric font for display. Both are based on Luiz Viera’s (Brazil) fonts as distributed in 2004. Some letters were added and some modified for better legibility, since the dot matrix of your *WP 34S* is only 6 pixels high.

Below you see all characters directly evocable through the keyboard in alpha mode as explained *above*:



As soon as a message exceeds the display using the large font, your *WP 34S* will take the small font automatically to show as much as possible within the 43 pixels the hardware features:



Many more characters of both fonts live in the alpha catalogs. You find them *below*.

PROGRAMMING

Your *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* first for an introduction into keystroke-programming, then continue reading here.

The basic building blocks within program memory are routines (or programs). Typically, a routine starts with a LBL statement and ends with RTN or END. In between, you may store any sequence of instructions (commands, operations, statements) for repeated use. Choose any operation featured – only a few commands are not programmable. The statements in your routine 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 price: you shall take care that your routines 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 routine, and documenting the purposes or contents of them for later reference.

Labels

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 overleaf. Furthermore, different programs may be separated by END statements. Think of the beginning and the end of program memory containing implicit END statements.

Addressing Labels

1 User input	A, B, C, or D	XEQ, GTO, LBL, LBL?, SLV, ∫, π, Σ, αGTO or αXEQ				
Dot matrix display	**XEQ *label*** e.g. XEQ C	**OP _** e.g. GTO _				
2 User input	Calls the function labeled **C**.	A, B, C, or D	ENTER↑ sets alpha mode.	→ [18] opens indirect addressing and sets temporary alpha mode.		*2-digit numeric (local) label* 0 0 … 9 9
Dot matrix display		**OP *label*** e.g. Σ B	**OP '_**	**OP →_**		**OP *nn*** e.g. LBL 07
3 User input		Sums up the function given in a routine labeled **B**.	***Alphanumeric (global) label*** (1 to 3 characters [19])	*Stack level or named register* X, Y, Z, ... , K	*Register number* 0 0 … 9 9 [20]	
Dot matrix display			**OP '*label*'** e.g. SLV'F1µ'	**OP → *x*** e.g. ∫→T	**OP → *nn*** e.g. XEQ→44	
			Solves the function given in the routine labeled **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**.	

See next page for the way your *WP 34S* searches labels, and look up GTO in the [index of operations] for special cases applying to this command exclusively.

---

[18] Works with all these operations except LBL .

[19] The 3rd character terminates entry and closes alpha mode – 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.

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

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 or a hotkey, it will be searched forward from the current position of the program pointer. When an END statement is reached without finding ***xy***, the quest will continue right after previous END. This is as known from *HP-41C*.
2. Else, i.e. if ***xy*** is an alpha label of up to three characters of arbitrary case (automatically enclosed in ‘ like **‘Ab1’**), searching will start at program step 000 and cover the entire memory in the order RAM, FM, and XROM, independent of the position of the program pointer.

Local Data

If – after some time – you have a number of different routines stored, keeping track of their memory requests may become a challenge. Most of modern 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. On the *WP 34S*, registers are for data storage – so we offer you *local registers* allocated to your routine alone.

Example: Let us assume you write a routine labeled ‘P1’:

1. You just enter the command LocR 5 in your routine specifying you want five local registers,
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 some local registers. They will then carry local register numbers **.**00 etc. again, but the local register **.**00 of *P2* will be different from the 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 routine 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, 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.

Tests

Like the vintage keystroke-programmable calculators before, your *WP 34S* features a set of tests. The respective command names feature a trailing ‘?’ typically. Generally, tests will work as in *HP-42S*: they will return ‘true’ or ‘false’ in the dot matrix if called from the keyboard; if called in a program, they will execute the next program step only if the test is true, else skip that step.

As mentioned above, programs typically end with RTN or END. In running programs, both statements work very similar and show only subtle differences:

- An RTN statement immediately after a test returning “false” will be skipped – an END will not.
- SKIP and BACK may jump over RTN – they cannot pass an END.

See the *index of operations* below for more information. The majority of tests is contained in the *catalog TEST*.

Programmed Input and Output

A number of commands may be employed for controlling I/O of programs. In the index *below*, 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):

- Showing 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 as soon as you pressed R/S .

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

Flash Memory (*FM*) and XROM

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. Its first section is the backup region (2kB), holding the image of the entire program memory, registers and calculator state as soon as you completed a SAVE. The remaining part holds programs only (several kB depending on configuration). Alphanumeric labels (see below) in *FM* can be called via XEQ like in *RAM*. This allows creating program libraries in *FM*. Use CAT to see the labels defined already.

FM 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 *FM* 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 routines 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*.

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 keyboard, the other commands may be picked from *catalogs*. The command names will show up identically in catalogs and program listings unless specified otherwise explicitly. 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 "G" at the end of 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. Please refer to the manuals of the vintage calculators mentioned for additional information about traditional commands.

Functions available on your *WP 34S* for the first time ever on an RPN calculator got their remarks printed on yellow background. Operations carrying a familiar name but deviating in their functionality from said calculators are marked light red.

Parameters will be taken from the lowest stack level(s) unless mentioned explicitly in the 2^{nd} 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** as 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 "¬" for "not". So e.g. **2^X** works in all modes but alpha. All operations may also be entered in programming mode 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*. Whenever a complex operation is executed, a capital C in the dot matrix will remind you to look at y as well.
10^x	f 10^x	DECM	
12h	h MODE …	¬α	Sets 12h time display mode: then e.g. 1:23 will become 1:23 AM, 23:45 will become 11:45 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.
	f 1/x	In CONV	Inverts the current conversion (see *below*).
24h	h MODE …	¬α	Sets 24h time display mode: then e.g. 1:23 AM will become 1:23, and 11:45 PM will become 23:45. This makes a difference in αTIME only.
2COMPL	h MODE …	¬α	Sets 2's complement mode like in *HP-16C*.
2^x	f 2^x	¬α	
$\sqrt[3]{x}$	h X.FCN …	¬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 .

Name	Keys to press	in modes	Remarks
ALL	h ALL ***n***	¬α	ALL 00 works almost 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 and ENGOVR). The same will happen if $x < 10^{-n}$ and more than 12 digits are required to show x completely. **Example:** Input: / Display: 700 / 700 ALL 03 / 700. 1/x / 0.00 142857 143 10 / / 1.42857 142857 $^{-4}$
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*** (≤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.
BACK	h P.FCN BACK ***n***	PRG	Jumps ***n*** program steps backwards (0 ≤ ***n*** ≤ 255). So e.g. BACK 001 goes to the previous step. If BACK attempts to cross an END statement, an error is thrown. Reaching step 000 stops program execution. Compare SKIP.

Name	Keys to press	in modes	Remarks
BASE	h MODE BASE *n*	¬α	Sets the base for integer calculations, with $2 \le n \le 16$. 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:** Stack contents are converted when switching from integer to DECM, and are truncated vice versa. Other registers stay as they are. The results may be surprising (see below).
BASE10	f 10		
BASE16	g 16		
BASE2	f 2		
BASE8	g 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 MODE …	DECM	Selects the best curve fit model, maximizing the correlation like BEST does in *HP-42S*.
Binom Binom$_P$ Binom $^{-1}$	h PROB …	DECM	Binomial distribution with the number of successes g in $\mathbf{X}$, the probability of a success p_0 in $\mathbf{J}$ and the sample size n in $\mathbf{K}$. Binom$_P$ [21] 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 $\mathbf{X}$. Binom^{-1} returns m for given probabilities F_B in $\mathbf{X}$ and p in $\mathbf{J}$ with sample size n in $\mathbf{K}$.
B_n	h X.FCN …	DECM	Returns the Bernoulli number for an integer $n > 0$ given in $\mathbf{X}$: $B_n = (-1)^{n+1} n \cdot \zeta(1-n)$. See below for ζ. This lives in XROM, may be less accurate.
B_n*	h X.FCN …	DECM	Returns the Bernoulli number according to its old definition for integer $n > 0$ given in $\mathbf{X}$: $B_n^* = \frac{2 \cdot (2n)!}{(2\pi)^{2n}} \cdot \zeta(2n)$. See below for ζ. This lives in XROM, may be less accurate.
BS?	h TEST BS? *n*	Integer	Tests the specified bit in x .

[21] 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
Cauch Cauch$_P$ Cauch $^{-1}$	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** : Cauch$_P$ returns $f_{Ca}(x)=\frac{1}{\pi\gamma}\cdot\frac{1}{1+\left(\frac{x-x_0}{\gamma}\right)^2}$, 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	g CF *n*	¬α	Clears the flag specified.
CLALL	h P.FCN …	¬α	Clears all registers and programs if confirmed.
CLFLAG	h P.FCN …	¬α	Clears all user flags.
CLP	f CLP	All	Clears the current program, i.e. the one the program pointer is in.
CLPALL	h P.FCN …	All	Clears all programs if confirmed.
CLREG	h P.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	¬α	Clears **X** only, disabling stack lift as usual.
CLα	h CLx h P.FCN …	α All	Clears the alpha register like CLA in *HP-42S*.
CLΣ	g CLΣ	DECM	Clears the summation registers and releases the memory allocated for them.

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 STAT …	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.
DATE	h X.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).
DATE→	h X.FCN …	DECM	Assumes x containing a date in the format selected and pushes its three components on the stack.
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 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 word length commands for remainder, multiplication and division like in *HP-16C*.
DEC	h P.FCN DEC *r*	¬α	Decrements *r* by one. Equivalent to 1 STO– *r*, but without modifying the stack.

Name	Keys to press	in modes	Remarks
DECM	f H.d	¬α	Sets default decimal mode for calculations.
DECOMP	h X.FCN …	DECM	Decomposes ***x*** (after converting it into an improper fraction, if applicable), resulting in a stack [***denominator(x), numerator(x), y,*** …] . Reversible by division. **Example:** If **X** contains 2.25 then DECOMP will result in ***x*** = 4 and ***y*** = 9, pushing previous content of **Y** to **Z** etc.
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 in fraction display – any denominator up to the value set by DENMAX may appear. **Example:** If DENMAX = 5 then DENANY allows denominators are 2, 3, 4, and 5.
DENFAC	h MODE …	¬α	Sets "factors of the maximum denominator", i.e. all integer factors of DENMAX may appear. **Example:** If DENMAX = 12 then DENFAC allows denominators 2, 3, 4, 6, and 12.
DENFIX	h MODE …	¬α	Sets fixed denominator format, i.e. the one and only denominator allowed is the value set by DENMAX.
DENMAX	h MODE …	¬α	Works like /c in *HP-35S*, but the maximum denominator settable is 9,999. It will be set to this value if ***x*** < 1 or ***x*** > 9,999 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 itself 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.

Name	Keys to press	in modes	Remarks
DSE	f DSE *r*	PRG	Given `cccccc.fffii` in ***r***, DSE decrements ***r*** by `ii`, skipping next program line if then `ccccccc` ≤ `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.
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 1, and skips if $\lvert r \rvert < 1$ 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	See the *catalog of conversions* for conversions from degrees to radians.
END	h P.FCN …	PRG	Last command in a routine and terminal for searching local labels as described above. Works like RTN in all other aspects.
E3OFF E3ON	h MODE …	¬α	Toggle the thousands separators for DECM (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 $erfc(x) = 1 - erf(x)$

Name	Keys to press	in modes	Remarks
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 MODE …	DECM	Selects the exponential curve fit model $y = a_0 e^{a_1 x}$.
Expon Expon$_P$ Expon $^{-1}$	h PROB …	DECM	Exponential distribution with the rate λ in $\mathbf{J}$: Expon$_P$ [22] 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 $\mathbf{X}$ and rate λ in $\mathbf{J}$.
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 …	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 startup default and is kept for fresh batteries. Compare SLOW.
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.
FLASH?	h TEST …	All	Returns the number of free words in flash memory.
FLOOR	h X.FCN …	DECM	Returns the largest integer $\leq x$.

[22] 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
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?	h TEST FS? ***n*** etc.	¬α	Tests if the flag specified is set. Clears, flips, or sets this flag after testing, if applicable.
FS?C			
FS?F			
FS?S			
$F_P(x)$	h PROB …	DECM	Fisher's 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)			
$F^{-1}(p)$			
f'(x)	h P.FCN f'(x) ***label***	DECM	Returns the first derivative of the function ***f(x)*** at position x. The function must be specified in a routine starting with LBL ***label***. The return stack will have y, z, and t cleared and the position x in **L**. This 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 used.
f"(x)	h P.FCN f"(x) ***label***	DECM	Works like f'(x) but returns the second derivative of ***f(x)***.
GCD	h X.FCN …	¬α	Returns the Greatest Common Divisor of x and y .
g_d	h X.FCN …	DECM	Returns the Gudermannian function or its inverse $g_d(x)=\int_0^x \frac{d\xi}{\cosh\xi}$ or $g_d^{-1}(x)=\int_0^x \frac{d\xi}{\cos\xi}$, respectively.
g_d^{-1}			
Geom	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 p_0 for a success in each such experiment must be specified in **J**. Geom $^{-1}$ returns the number of failures f before 1st success for given probabilities F_{Ge} in **X** and p_0 in **J**.
Geom_P			
Geom $^{-1}$			

Name	Keys to press	in modes	Remarks
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	… to step ***nnn*** .
	h GTO . ▲	¬α	… directly after previous END.
	h GTO . ▼	¬α	… directly after next END.
	h GTO . .	¬α	… 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.
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 <u>above</u>. Will return to the previous decimal display with the next keystroke thereafter.
H.MS+ / **H.MS–**	h X.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).

Name	Keys to press	in modes	Remarks
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*** .
iRCL	h X.FCN iRCL ***s***	¬α	Assumes the source ***s*** contains integer data and recalls them as such. See *below*.
ISE	h P.FCN ISE ***r***	PRG	Works like ISG but skips if `ccccccc` ≥ `fff` .
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 *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 $\Gamma(x)$ see below.

Name	Keys to press	in modes	Remarks
JG1582 JG1752	h MODE …	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 no 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 stored in address ***a*** . Key codes reflect the rows and columns on the keyboard starting top left – so e.g. A corresponds to 11, CPX to 16, STO to 21, and + to 75.
KTP?	h TEST KTP? ***a***	All	Assumes a key code in address ***a*** . Checks this code and returns the key type: • 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 LBL for more.
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 LgNrm_P	h PROB …	DECM	Lognormal distribution with $\mu = \ln \bar{x}_g$ specified in **J** and $\sigma = \ln \varepsilon$ in **K**. See x̄g and ε below. LgNrm_P returns $f_{Ln}(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{(\ln x - \mu)^2}{2\sigma^2}}$, LgNrm returns $F_{Ln}(x) = \Phi\left(\frac{\ln x - \mu}{\sigma}\right)$ with $\Phi(z)$ denoting the standard normal *cdf*.

Name	Keys to press	in modes	Remarks
LgNrm $^{-1}$	h PROB …	DECM	Returns ***x*** for a given probability F_{Ln} in **X**, ***μ*** in **J**, and ***σ*** in **K**.
LINEQS	h X.FCN …	¬α	Takes a base register in **X**, a vector *descriptor* in **Y**, and a descriptor of a square matrix in **Z**. Solves the system of linear equations $(Z)\cdot\vec{x}=\vec{y}$ and returns the filled in vector descriptor in **X**.
LinF	h MODE …	DECM	Selects the linear curve fit model $y=a_0+a_1x$.
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*.
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	Restores the entire backup from flash. Compare SAVE. See *Appendix A* for more.
LOADP	h P.FCN …	¬α	Loads the complete program memory from the backup and appends it to the programs already in RAM. This will only work if there is enough space.
LOADR	h P.FCN …	¬α	Recovers numbered general purpose registers from the backup (see SAVE and *above*). The number of registers copied is the minimum of the registers held in flash and RAM at execution time.

Name	Keys to press	in modes	Remarks
LOADST	h P.FCN …	¬α	Recovers the system state from the backup. See *Appendix B* for more.
LOADΣ	h P.FCN …	¬α	Recovers the summation registers from the backup. Throws an error if there are none. See *Appendix B* for more.
LocR	h P.FCN LocR ***n***	PRG	Allocates ***n*** local registers (up to 144) 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 MODE …	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**.
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 STAT …	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.

Name	Keys to press	in modes	Remarks
MASKL MASKR	h X.FCN MASKL *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 multiplies 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 multiplies 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*.
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 in internal 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.

Name	Keys to press	in modes	Remarks
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.
M×	h MATRIX …	DECM	Takes two matrix *descriptors* in **Y** and **Z** and the integer part of x as the base address of the result. Returns $(Z)\cdot(Y)=(X)$. All calculations are done in internal high precision. The fractional part of x is updated to match the resulting matrix – no overlap checking is performed.
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 formatted 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*.

Name	Keys to press	in modes	Remarks
M.LU	h MATRIX …	DECM	Takes a *descriptor* of a square matrix in **X**. Transforms the matrix into its LU decomposition. 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*.
nCOL	h MATRIX …	DECM	Takes a matrix *descriptor* in **X**, saves it in **L**, and returns the number of columns in this matrix.
NEIGHB	h X.FCN …	DECM	Returns the nearest machine-representable number to x in the direction toward y in the mode set. For $x<y$ (or $x>y$), this is the machine successor (or predecessor) of x , for $x=y$ it is y. You may find NEIGHB useful investigating numeric stability. See NEIGHBOR in the HP-71 Math Pac.
		Integer	Returns $x+1$ for $x<y$, y for $x=y$, or $x-1$ for $x>y$.
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$_P$ [23] 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 Φ.

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

Name	Keys to press	in modes	Remarks
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**[24].
NOT	h NOT	Integer	Inverts bit-wise as on *HP-16C*.
		DECM	Returns 1 for 0, and 0 for any other input.
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 SUMS …	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.
P_n	h X.FCN …	DECM	Legendre's polynomials: $P_n(x) = \frac{1}{2^n n!} \cdot \frac{d^n}{dx^n}\left[\left(x^2-1\right)^n\right]$ with ***n*** in **Y**, solving the differential equation $\frac{d}{dx}\left[\left(1-x^2\right) \cdot \frac{d}{dx} f(x)\right] + n(n+1) f(x) = 0$.
Poiss	h PROB …	DECM	Poisson distribution with the number of successes ***g*** in **X**, the gross error probability p_0 in **J**, and the sample size ***n*** in **K**.
Poiss_P			Poiss_P[25] returns $P_P(g;\lambda) = \frac{\lambda^g}{g!} e^{-\lambda}$ with $\lambda = n \cdot p_0$. Poiss returns $F_P(m;\lambda) = \sum_{g=0}^{m} P_P(g;\lambda)$, with the maximum number of successes ***m*** in **X**.

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

[25] 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
Poiss $^{-1}$	h PROB …	DECM	Returns ***m*** for given probabilities ***F***$_P$ in **X** with ***p***$_0$ in **J** and sample size ***n*** in **K**.
Poisλ Poisλ$_P$ Poisλ $^{-1}$	h PROB …	DECM	Poisson distribution, like Poiss… above with ***g*** in **X**, but the Poisson parameter ***λ*** in **J**. Poisλ $^{-1}$ returns ***m*** for a given probability ***F***$_P$ in **X** and ***λ*** in **J**.
PopLR	h P.FCN …	PRG	Pops the local registers without returning. See LocR and RTN.
PowerF	h MODE …	DECM	Selects the power curve fit model $y = a_0 x^{a_1}$.
PRCL	h P.FCN …	¬α	Copies the current program (from flash or RAM) and appends it to RAM where it can be edited then (see *above*). Allows duplicating programs in RAM. Will only work with enough space at destination.
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 $^2/_3$.
PROMPT	h P.FCN …	PRG	Displays ***alpha*** and stops program execution (equaling αVIEW followed by STOP). See *above* for more.
PSE	h PSE ***n***	PRG	Refreshes the display and pauses program execution for ***n*** ticks, with $0 \le n \le 99$. The pause will be terminated early as soon as a key is pressed.
PSTO	h P.FCN …	¬α	Copies the current program from RAM and appends it to flash library. The program must feature at least one LBL statement with an alphanumeric label (preferably at its beginning). If a program with the same label exists in the library already it will be deleted first. Alphanumeric labels present in flash may be browsed by CAT (see *below*) and called by XEQ.

Name	Keys to press	in modes	Remarks
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.
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.
RCL	RCL ***s***	¬α	See the *addressing table above* for ᶜRCL.
RCLM	RCL MODE ***s***	¬α	Recalls mode settings stored via STOM as described *above*. Also contained in MODE.
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.
RCL+	RCL + ***s***	¬α	Recalls the content of the source ***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, ᶜRCL–12 subtracts ***r12*** from ***x*** and ***r13*** from ***y***. See the *addressing table above* for ᶜRCL.
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.
RDX, / **RDX.**	h ./,	DECM	Toggles the radix mark.
RDX.	h MODE RDX.	¬α	Sets the decimal mark to a point.
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 SEND… and *Appendix A* for more.

Name	Keys to press	in modes	Remarks
REGS	h MODE REGS ***n***	¬α	Specifies the number of global general purpose registers accessible. With REGS 100 you get the default state (**R00 – R99**), REGS 0 leaves not even a single such register for use.
REGS?	h TEST …	¬α	Returns the number of global general purpose registers allocated (0 … 100).
RESET	h X.FCN …	All	Executes CLALL and resets all modes to start-up default, i.e. 24h, 2COMPL, ALL 00, DBLOFF, DEG, DENANY, DENMAX 9999, DECM, LinF, PROFRC, RDX**.**, REGS 100, 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***		
RM	h MODE …	¬α	Sets floating point rounding mode. This 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.

Name	Keys to press	in modes	Remarks
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***		
RSD	h X.FCN RSD ***d***	DECM	Rounds x to ***d*** significant digits, taking the RM setting into account.
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 last XEQ instruction executed. If there was 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 <u>second</u> line following the most recent XEQ instruction encountered. Halts if there is none.
R-CLR	h P.FCN …	DECM	Interprets x in the form `sss.nn`. Clears ***nn*** registers starting with number ***sss*** . **Example:** For x = 34.56, R-CLR will clear **R34** through **R89**.
R-COPY	h P.FCN …	DECM	Interprets x in the form `sss.nnddd`. Takes ***nn*** registers starting with number ***sss*** and copies their contents to ***ddd*** etc. **Example:** For x = 7.03045678, ***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 |***sss***| there.
R-SORT	h P.FCN …	DECM	Interprets x in the form `sss.nn`. Sorts the contents of ***nn*** registers starting with number ***sss*** . **Example:** 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	See the *catalog of conversions* 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↓		

Name	Keys to press	in modes	Remarks
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. Recall your backup by LOAD. See *Appendix A* for more. **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*** decimal positions, equivalent to multiplying x by 10^n .
SDR	h X.FCN SDR ***n***	DECM	Shifts digits right by ***n*** decimal positions, equivalent to dividing x through 10^n .
SEED	h STAT …	DECM	Stores a seed for random number generation.
SENDA SENDP SENDR SENDΣ	h P.FCN …	¬α	SENDA sends all RAM data, SENDP the program memory, SENDR the global general purpose registers, and SENDΣ the summation registers, respectively, via serial I/O to the device connected. See RECV and *Appendix A* for more.
SEPOFF	h MODE …	¬α	Toggle the separators for integer modes. Points or commas will be displayed every … … four digits in bases 2 and 4, … two digits in base 16, … three digits in all other bases.
SEPOFF / SEPON	h ./,	Integer	
SEPON	h MODE …	¬α	
SERR	h STAT …	DECM	Works like s but pushes the standard errors $s/\sqrt{n}$ on the stack (i.e. the standard deviations of x̄ and ȳ).
$SERR_w$	h STAT …	DECM	Works like sw but returns the standard error $s/\sqrt{\sum y_i}$ (i.e. the standard deviation of x̄w).
SETCHN	h MODE …	¬α	Sets some regional preferences (see *above*).

Name	Keys to press	in modes	Remarks
SETDAT	h MODE …	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 SETJPN	h MODE …	¬α	Set some regional preferences (see above).
SETTIM	h MODE …	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	f 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**.
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 (0 ≤ ***n*** ≤ 255). So e.g. SKIP 02 skips over the next two steps, going e.g. from step 123 to step 126. If SKIP attempts to cross an END statement, an error is thrown. Compare BACK.
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. Compare FAST.

Name	Keys to press	in modes	Remarks
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. Please refer to the *HP-15C Owner's Handbook*, Section 13 and Appendix D, for more information about automatic root finding and some caveats.
SLVQ	h X.FCN …	DECM	Solves the quadratic equation $ax^2 + bx + c = 0$, with its real parameters 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.
SR	h X.FCN SR ***n***	Integer	Works like ***n*** consecutive SRs on *HP-16C*. SR 0 executes as NOP.
sRCL	h X.FCN sRCL ***s***	¬α	Assumes the source ***s*** contains single precision real data and recalls them as such. See *below*.
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.

Name	Keys to press	in modes	Remarks
STATUS	h STATUS	¬PRG	Shows – among other data – the status of all user flags, similar to STATUS on *HP-16C*. See *above* for more.
STO	STO ***d***	¬α	See the *addressing table above* for [C]STO.
STOM	STO MODE ***s***	¬α	Stores mode settings for later use as described *above*. Take RCLM to recall them. STOM is also contained in MODE.
STOP	R/S	PRG	Stops program execution. May be used to wait for an input, for example.
STOPW	h X.FCN …	¬PRG	Stopwatch application. Works only with the quartz installed (or on the emulator).
STOS	h P.FCN STOS ***d***	¬α	Stores all stack levels in a set of 4 or 8 registers, starting at destination ***d***. See RCLS.
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.
s_w	h STAT …	DECM	Returns the standard deviation for weighted data with the weights entered in y via Σ+ : $$s_w = +\sqrt{\frac{\sum y_i \cdot \sum \left(y_i \cdot x_i^2\right) - \left[\sum \left(y_i \cdot x_i\right)\right]^2}{\left(\sum y_i\right)^2 - \sum y_i^2}} \quad .$$
sxy	h STAT …	DECM	Returns the sample covariance for two data sets. It depends on the fit model selected. For LinF, it returns $$s_{xy} = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{n \cdot (n-1)} \quad .$$ See COV for the population covariance.
TAN	f TAN	DECM	Returns the tangent of the angle in $\mathbf{X}$.
TANH	f HYP TAN	DECM	Returns the hyperbolic tangent of x.

Name	Keys to press	in modes	Remarks
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 an inevitable prerequisite for the clock being useful in medium to long range.
TIME	h X.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 $T_n(\boldsymbol{x})$ with $\boldsymbol{n}$ in **Y**, solving the differential equation $(1-x^2)y''-x\cdot y'+n^2y=0$.
TOP?	h TEST …	PRG	Executes the next step only if TOP? is called in a program that isn't a subroutine, i.e. if the program running flag is set and the subroutine return stack pointer is clear.
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*.
t⇆	h P.FCN t⇆ ***r***	¬α	Swaps the contents of **T** and ***r***, in analogy to ***x***⇆.
ULP	h X.FCN …	¬α	Returns 1 times the smallest power of ten which can be added to ***x*** or subtracted from ***x*** to actually change the value of ***x*** in the machine in the mode set. Thus, in integer mode, 1 is returned.
U_n	h X.FCN …	DECM	Chebychev's polynomials of second kind $U_n(\boldsymbol{x})$ with $\boldsymbol{n}$ in **Y**, solving the differential equation $(1-x^2)y''-3x\cdot y'+n(n+2)y=0$.
UNSIGN	h MODE …	¬α	Sets unsigned mode for integers.
VERS	h X.FCN …	¬PRG	Shows your firmware version and build number.
VIEW	h VIEW ***s***	¬α	Displays the content of address ***s*** until the next key is pressed. See *above* for more.

Name	Keys to press	in modes	Remarks
VWα+	h VIEW *s*	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 *above* for more.
WHO	h X.FCN …	¬α	Displays credits to the brave men who made the project work.
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$_p$ W$_m$	h X.FCN …	DECM	W$_p$ returns the principal branch of Lambert's W for given $x \geq -1/e$. W$_m$ returns the negative branch.
W $^{-1}$	h X.FCN …	DECM	Returns ***x*** for given W$_p$ (≥ -1). See there.
Weibl Weibl$_P$ Weibl $^{-1}$	h PROB …	DECM	Weibull distribution with the shape parameter ***b*** in **J** and the characteristic lifetime ***T*** in **K**: Weibl$_P$ [26] returns $f_W(t) = \frac{b}{T}\left(\frac{t}{T}\right)^{b-1} e^{-\left(\frac{t}{T}\right)^b}$, 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 *n*	¬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	
x^3	h X.FCN …	¬α	

[26] 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
XEQ	[XEQ] ***label***	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.
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	Pushes $\bar{y} = \frac{1}{n}\sum y$ 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 arithmetic mean $\sum xy \big/ \sum y$ for weighted data with the weights entered in ***y*** via Σ+. See also s_w and $SERR_w$.
x̂	[h] [STAT] …	DECM	Returns a forecast ***x*** for a given ***y*** (in **X**) acc. to the fit model chosen. See L.R. for more.
$\sqrt[x]{y}$	[h] [X.FCN] …	¬α	
x !	[h] [!]	DECM	Returns the factorial for integer input. Generally, returns *Γ(x + 1)* .
x → α	[h] [X.FCN] ...	All	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***. See above for ^{C}***x⇆***.

Name	Keys to press	in modes	Remarks
x*⇆ *Y	x⇆y	¬α	Swaps *x* and *y*, performing Re⇆Im if a complex operation was executed immediately before. See above for $^{C}x⇆y$.
x < ?	h TEST x< ? *a*	¬α	Compare *x* with ***a***. E.g. h TEST x< ? K will compare *x* with ***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⇆	h P.FCN y⇆ *r*	¬α	Swaps the contents of **Y** and *r*, in analogy to *x*⇆.
z⇆	h P.FCN z⇆ *r*	¬α	Swaps the contents of **Z** and *r*, in analogy to *x*⇆.
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	Interprets ***rnn*** 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.

Name	Keys to press	in modes	Remarks
αIP	h X.FCN ...	All	Appends the integer part of ***x*** to ***alpha***, similar to AIP in *HP-42S*.
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	Interprets the content of the source ***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.

Name	Keys to press	in modes	Remarks
αVIEW	h VIEW α h P.FCN ... h X.FCN ...	All	Displays ***alpha*** in the top line and - - - in the bottom line until the next key is pressed. See above for more.
αXEQ	h P.FCN αXEQ ***nn***	¬α	Interprets ***rnn*** 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 ...	All	Returns the character code of the leftmost character in ***alpha*** and deletes this character, like ATOX in *HP-42S*.
β	h X.FCN ...	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 ...	DECM	Returns $\Gamma(x)$. Additionally, h ! calls $\Gamma(x+1)$.
ΔDAYS	h X.FCN …	DECM	Assumes $\mathbf{X}$ and $\mathbf{Y}$ 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 …	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 multiplicative.
εm	h STAT …	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 …	DECM	Works like ε but with a denominator ***n*** instead of ***n – 1***, returning the scattering factors of the populations.

Name	Keys to press	in modes	Remarks
ζ	h X.FCN …	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.
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.
$\Sigma\ln^2x$	h SUMS …	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.
$\Sigma\ln^2y$			
Σlnx			
Σlnxy			
Σlny			
Σxlny			
Σylnx			

Name	Keys to press	in modes	Remarks
σw	h STAT ...	DECM	Works like sw but returns the standard deviation of the population instead. $\sigma_w = +\sqrt{\frac{\sum y_i (x_i - \bar{x}_w)^2}{\sum y_i}}$
Σx	h SUMS …	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			
$\Sigma x^2 y$			
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 …	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 …	¬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*** · ***x*** .
/	/	¬α	Returns ***y*** / ***x*** .
+/–	+/–	¬α	Unary minus like CHS in *HP-35*.

Name	Keys to press	in modes	Remarks
→DATE	h X.FCN …	DECM	Assumes the three components of a date on the stack in proper order for the date format selected and converts them to a single date in ***x***. Thus inverts DATE→.
→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.
→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***) .
⇆	h P.FCN ⇆ _ _ _ _	¬α	Shuffles the lowest four stack contents. Examples: ⇆XXYZ works like ENTER↑, ⇆YZTX works like R↓, but also ⇆ZZZX is possible. Play! But remember it only affects the bottom four stack levels.
%	f %	DECM	Returns $\frac{x \cdot y}{100}$, leaving **Y** unchanged.
%MG	h X.FCN …	DECM	Returns the margin [27] $100 \cdot \frac{x - y}{x}$ in % for a price ***x*** and cost ***y*** , like %MU-Price in *HP-17B*.

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

Name	Keys to press	in modes	Remarks
%MRR	h X.FCN ...	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 ...	DECM	Returns $100 \cdot \frac{x}{y}$, interpreted as % of total.
%Σ	h X.FCN ... h STAT ...	DECM	Returns $100 \cdot \frac{x}{\sum x}$.
%+MG	h X.FCN ...	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 ∫f_y^x ***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*. Please refer to the *HP-15C Owner's Handbook*, Section 14 and Appendix E, for more information about automatic integration and some caveats.
∞?	h TEST ...	¬α	Tests x for infinity.
||	g ||	DECM	Returns $\left(\frac{1}{x} + \frac{1}{y} \right)^{-1}$, being very useful in electrical engineering especially.
#	h CONST # ***n***	¬α	Inserts a short integer value ***n*** in a single program step (0 ≤ ***n*** ≤ 255), thus saving up to two steps and an ENTER.

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 the number ranges.
	0, 1, f 2, … , f 9	α	Appends the respective digit to ***alpha***.
A … F	A … F (grey print)	11, 12, 13, 14, 15, h	Numeric input for digits >10. See *above* for more information.
A … Z	A … Z (grey print)	in addressing	Register input. See the *tables* above for the letters applicable.
		α	Appends the respective Latin letter to ***alpha***. Use f ↑ to toggle cases.
E	EEX	DECM & ¬FRACT	Works like E in the Pioneers.
i	CPX .	DECM & ¬FRACT	Enters complex number ***i***, i.e. x=0 and y=1.
A … Ω	g A … g O (grey print)	α	Appends the respective Greek letter to ***alpha***. f ↑ will toggle cases. See *above* for more.
(	f ◄(	α	Appends the respective symbol to ***alpha***.
)	g)►	α	Appends the respective symbol to ***alpha***.
+	f +	α	Appends the respective symbol to ***alpha***.
–	f −	α	Appends the respective symbol to ***alpha***.
x	f ×	α	Appends the respective symbol to ***alpha***.

Character	Keys to press	in modes	Remarks
/	Second .	DECM	A persistent 2nd . 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 2nd dot while editing the input line.
		FRC	First . is interpreted as a space, 2nd 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 ▼		
⇆	h x⇆		
≠	h XOR		
&	h AND		
\	h /		
|	h OR		

Non-programmable Control, Clearing and Information Commands

Keys to press	in modes	Remarks
⬅ [28]	Input pending	Deletes the last digit or character put in.
	α	Deletes the rightmost character in ***alpha***.
	PRG	Deletes current program step.
	Else	Acts like CLx.
▲ / ▼ [29]	Status display 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 & ¬PRG	Shifts the display window to the left / right like in *HP-16C*. Helpful while working with small bases.
	DECM & ¬PRG	Act like SHOW described below.
f ⇧	α	Toggles upper and lower case (indicated by ⬇).
	¬α	Enters a memory browser.
ENTER↑	Catalog open	Selects the current item like XEQ below.
	α	Leaves alpha mode.
	Else	Acts like the command ENTER described above.

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

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

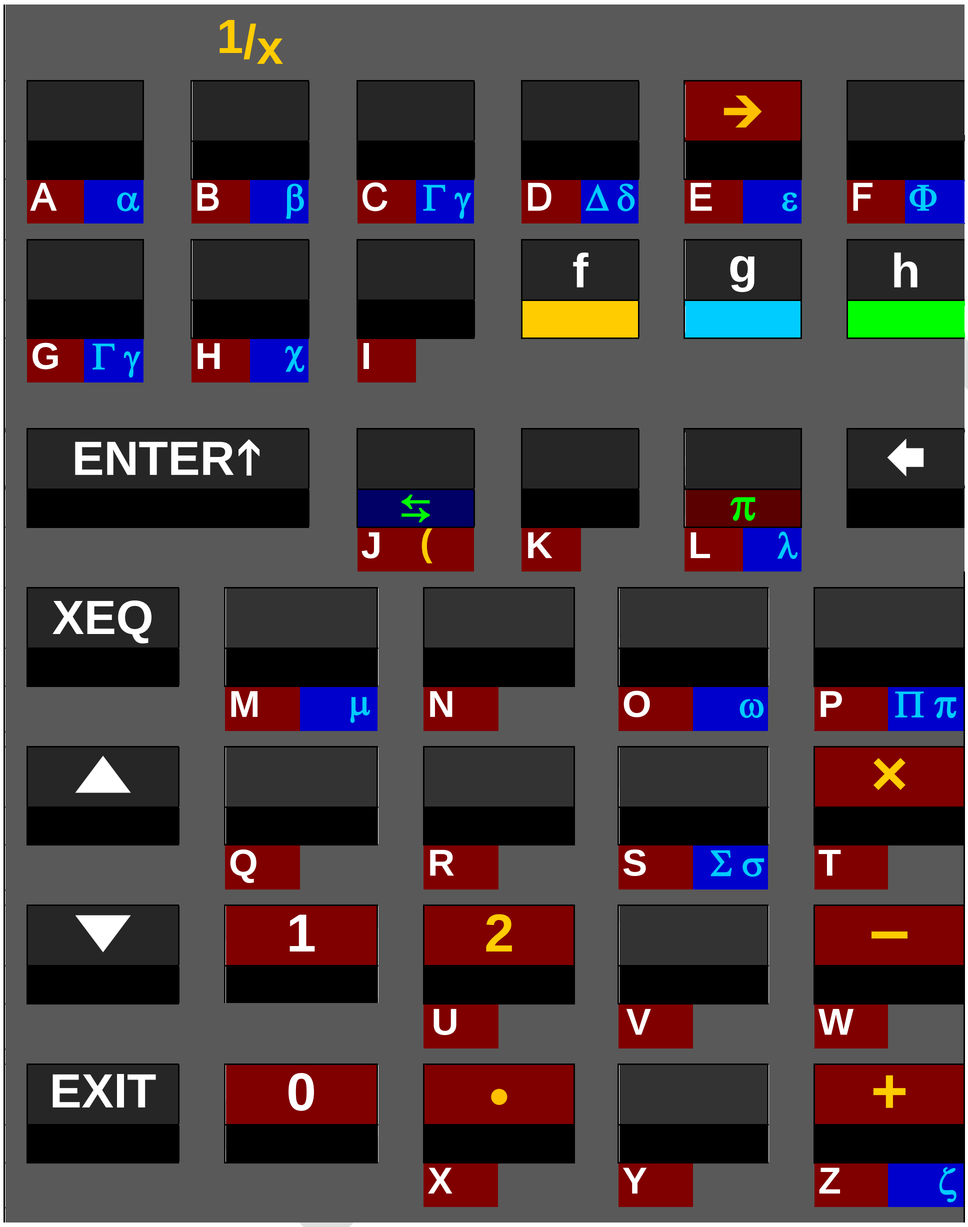
Keys to press	in modes	Remarks
EXIT	Catalog open	Leaves the catalog without executing anything.
	Input pending	Cancels the execution of pending operations, returning to the calculator status as it was before.
	Program running	Stops the running program like R/S . See below.
	PRG	Leaves programming mode like P/R . See below.
	α	Leaves alpha mode.
	Else	Does nothing.
h OFF	¬PRG	Turns your *WP 34S* off.
	PRG	Enters the command OFF as described above.
ON	Calculator off	Turns your *WP 34S* on.
	Else	There are several ON-key combinations available. Most important: ON + + or − adjusts display contrast. Further combinations are found in the appendices.
h P/R	¬α	Toggles programming mode (PRG).
R/S	Asking for confirmation (Sure?)	Confirms – with Y for 'yes'. Any other input will be interpreted as 'no'.
	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.
g 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 2	DECM	Shows x as integer to base 2. Returns to the base set with the next keystroke.

<table>
<tr><th>Keys to press</th><th>in modes</th><th>Remarks</th></tr>
<tr><td>→ 8</td><td rowspan="2">DECM</td><td rowspan="2">Shows x as integer to base 8 or 16, respectively. Returns to the base set with the next keystroke. Prefix g may be omitted here.</td></tr>
<tr><td>→ 16</td></tr>
<tr><td rowspan="3">f α</td><td>¬PRG, ¬α</td><td>Turns on alpha mode for keyboard entry.</td></tr>
<tr><td>PRG & ¬α</td><td>Turns on alpha mode for direct entry of alpha constants in programming. Your WP 34S will display α' in the top line. Now enter the characters you want – they will be appended to alpha. For starting a fresh new string, use CLα first. Alpha constants will be listed in groups of three as long as applicable.

Example: Entering
f α T f ↑ e s t h PSE 1 h ▼ f α
will result in three program steps stored:
α'Tes'
α't 1'
α ?
and Test 1? appended to alpha in program execution.</td></tr>
<tr><td>α</td><td>Leaves alpha mode.</td></tr>
</table>

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 reverting conversions in CONV easily as described below.

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

▲ and ▼ will 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. it cancels the catalog call.

See below for some examples.

In addition, you may switch catalogs easily by just calling a new one accessible in current mode directly from the catalog you are browsing – no need for EXITing this first.

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 only here: ▲ and ▼ browse the catalog as usual, but in the numeric line the location of the respective label is indicated (rAM, Lib for flash memory). 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 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 (see *below*). Case may be toggled here (see f ↑ *above*).
h MATRIX	DECM	Matrix operations library.
h MODE	¬α	Mode setting functions.
h PROB	DECM	Probability distributions beyond standard normal.
h P.FCN	¬α	Extra programming and I/O functions.
h R↑	α	Superscripts and subscripts (see *below*).
h STAT	DECM	Extra statistical functions.
h SUMS	DECM	Read access to all statistical sums.
h TEST	¬α	All tests except the two on the keyboard.
	α	Comparison symbols and brackets, except f (and g) (see *below*).

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 (see below).	
f →	α	Arrows and mathematical symbols (see below).	

Reopening the very last catalog called, the last command selected therein is displayed for easy repetitive use. A single function may be contained in more than one catalog.

See 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.

Catalog Contents in Detail

MODE

12h
1COMPL
24h
2COMPL
BASE
BestF
DENANY
DENFAC
DENFIX
DENMAX
DISP
D.MY
E3OFF
E3ON
ExpF
FAST
FRACT
JG1582
JG1752
LinF
LogF
LZOFF
LZON
M.DY
PowerF
RCLM
RDX,
RDX.
REGS
RM
SEPOFF
SEPON
SETCHN
SETDAT
SETEUR
SETIND
SETJPN
SETTIM
SETUK
SETUSA
SIGNMT
SLOW
SSIZE4
SSIZE8
STOM
UNSIGN
WSIZE
Y.MD

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

PROB

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

STAT

COV
L.R.
SEED
SERR
$SERR_w$
SUM
s_w
sxy
$\bar{x}g$
$\bar{x}_w$
$\hat{x}$
ε
ε_m
ε_p
σ
σ_w
%Σ

SUMS

nΣ
Σln^2x
Σln^2y
Σlnx
Σlnxy
Σlny
Σx
Σx^2
Σx^2y
Σxlny
Σxy
Σy
Σy^2
Σylnx

TEST

BC?
BS?
CNVG?
DBL?
ENTRY?
EVEN?
FC?
FC?C
FC?F
FC?S
FLASH?
FP?
FS?
FS?C
FS?F
FS?S
IBASE?
INTM?
INT?
KEY?
KTP?
LBL?
LEAP?
LocR?
MEM?
M.SQR?
NaN?
ODD?
PRIME?
REALM?
REGS?
RM?
SMODE?
SPEC?
SSIZE?
TOP?
WSIZE?
x < ?
x ≤ ?
x = +0?
x = -0?
x ≈ ?
x ≥ ?
x > ?
∞?

P.FCN

BACK
CLALL
CLFLAG
CLPALL
CLREG
CLSTK
CLα
DEC
DROP
DSL
DSZ
END
ERR
FF
f ’(x)
f ”(x)
GTOα
INC
ISE
ISZ
LOAD
LOADP
LOADR
LOADST
LOADΣ
LocR
MSG
NOP
PopLR
PRCL
PROMPT
PSTO
PUTK
RCLS
RECV
RTN+1
R-CLR
R-COPY
R-SORT
R-SWAP
SAVE
SENDA
SENDP
SENDR
SENDΣ
SKIP
STOS
TICKS
t⇆
VWα+
XEQα
y⇆
z⇆
αGTO
αOFF
αON
αVIEW
αXEQ
⇆

X.FCN varies with the mode set, except in programming [30]. It contains in …

… alpha mode:
RESET
VERS
x→α
αDATE
αDAY
αIP
αLENG
αMONTH
αRC#
αRL
αRR
αSL
αSR
αTIME
α→x

… decimal mode:		
$^3\sqrt{x}$	LCM	W_m
AGM	L_n	W_p
ANGLE	LN1+x	W^{-1}
BATT	$L_n\alpha$	XNOR
B_n	LNβ	x^3
B_n*	LNΓ	x→α
CEIL	MANT	$^x\sqrt{y}$
DATE	MAX	YEAR
DATE→	MIN	αDATE
DAY	MONTH	αDAY
DAYS+	NAND	αIP
DECOMP	NEIGHB	αLENG
DEG→	NEXTP	αMONTH
DROP	NOR	αRCL
D→J	P_n	αRC#
erf	RAD→	αRL
erfc	RESET	αRR
EXPT	ROUNDI	αSL
e^x -1	RSD	αSR
FIB	SDL	αSTO
FLOOR	SDR	αTIME
GCD	SIGN	α→x
g_d	SINC	β
g_d^{-1}	SLVQ	Γ
GRAD→	sRCL	ΔDAYS
H_n	STOPW	ζ
H_{np}	TIME	$(-1)^x$
H.MS+	T_n	→DATE
H.MS-	ULP	%MG
iRCL	U_n	%MRR
Iβ	VERS	%T
IΓ	WDAY	%Σ
J→D	WHO	%+MG

… integer modes:	
$^3\sqrt{x}$	RR
ASR	RRC
BATT	SB
CB	SEED
DBLR	SIGN
DBL*	SL
DBL/	SR
DROP	sRCL
FB	ULP
FIB	VERS
GCD	WHO
LCM	x^3
LJ	XNOR
MASKL	x→α
MASKR	$^x\sqrt{y}$
MAX	αIP
MIN	αLENG
MIRROR	αRCL
NAND	αRC#
nBITS	αRL
NEIGHB	αRR
NEXTP	αSL
NOR	αSR
RESET	αSTO
RJ	α→x
RL	$(-1)^x$
RLC	

CPX X.FCN
$^{C3}\sqrt{x}$
^{C}AGM
^{C}CONJ
^{C}DROP
$^Ce^x$ -1
^{C}FIB
^Cg_d
$^Cg_d^{-1}$
^{C}LN1+x
^{C}LNβ
^{C}LNΓ
^{C}SIGN
^{C}SINC
^CW_p
$^CW^{-1}$
$^Cx^3$
$^{C\,x}\sqrt{y}$
$^C\beta$
$^C\Gamma$
$^C(-1)^x$

[30] In programming mode, these three contents will be merged.

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

Here are the contents of the alpha catalogs making the *WP 34S* the most versatile global calculator known. Small font is printed on grey background on this page. The catalog CPX is listed left. Accented letters are as wide as plain ones wherever possible.

./,
, : ; ‘ “ * @ _ ~ % $ € £ ¥ ⊙ ⊕ #

TEST
< ≤ = ≈ ≥ > [ ] { }

→
→ ← ↑ ↓ √ ∫ ∞ ^ ↕

R↑
$_0$ $^{\circ}$ $_1$ $_2$ ^{2} ^{3} $_A$ $_B$ $_c$ ^{c} $_d$ $_e$ $_k$ $_m$ $_n$ $_p$ $_u$ $_w$ ^{x} $_\mu$ $^{-1}$ $_\infty$

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

Afrikaans, Català, Cebuano, Česky, Cymraeg, Dansk, Deutsch, Eesti, English, Español, Euskara, Français, Gaeilge, Galego, Ελληνικά, Hrvatski, Bahasa Indonesia, Italiano, Basa Jawa, Kiswahili, Kreyòl ayisyen, Magyar, Bahasa Melayu, Nederlands, Norsk, Português, Quechua, Shqip, Slovenčina, Slovenščina, Srpski, Basa Sunda, Suomeksi, Svenska, Tagalog, Winaray, and Zhōngwén (with a little trick explained below). 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 we need different letters for ā and ă 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.

Accessing Catalog Items the Fast Way

Each and every catalog may be browsed just by using the cursors as explained above. You may reach your target significantly faster, however, taking advantage of the alphabetical method demonstrated in the left columns of the table below:

1	User input	CONST, CONV, MATRIX, MODE, PROB, P.FCN, STAT, TEST, SUMS, or X.FCN	CPX or R↑ in alpha mode	→, TEST, or ./, in alpha mode
	Dot matrix display	**Shows 1st item in selected catalog.** (e.g. BC? in TEST)	**Shows 1st item in selected catalog.** (e.g. ä in CPX)	**Shows 1st item in selected catalog.** (e.g. ; in ./,)
2	User input	1st character of command desired (e.g. F)	Desired basic letter (e.g. U)	↓
	Dot matrix display	**Shows 1st item starting with this character *)** (e.g. FC?)	**Shows 1st item starting with this letter *)** (e.g. ü)	↓
3	User input	2nd character (e.g. S)	↓	↓
	Dot matrix display	**Shows 1st item starting with this sequence *)** (e.g. FS?)	↓	↓
…		Continue browsing with ▼ until reaching the item desired		
		(e.g. FS?C).	(e.g. Ü).	(e.g. €).
n	User input	XEQ or ENTER↑		
		Your *WP 34S* leaves the catalog returning to the mode set before … … and executes or inserts the command chosen, or recalls the constant selected.	… and appends the selected character to ***alpha***.	
	Dot matrix display	**Result** (e.g. True)	**Contents of alpha register** (e.g. 3 Rüben ä 0,25€)	

*) If a character or sequence specified is not found in this catalog then the first item following alphabetically will be shown – see the sorting sequence above. 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 (CONST)

Your *WP 34S* contains a rich set of constants. Navigation works as explained above. 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, even by the national standards institutes and the international scientific community [31].

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 $1J = 1W \cdot s = 1V \cdot A \cdot s$. Thus $1\frac{J}{T} = 1A \cdot m^2$.

Employ the constants stored for further useful equivalences, like expressing Joules in Electron-Volts $1A \cdot s \cdot V = \frac{1}{e} eV \approx 6.24 \cdot 10^{18} eV$, or calculating the wavelength from the frequency of electromagnetic radiation via $\frac{f}{c} = \lambda$, or whatever else crosses your mind.

	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
a⊕	1.495979**E**11 *(1**E**-6)*	m	Semi-major axis of the Earth's orbit. 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

[31] 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.

	Numeric value	Unit	Remarks
Fδ	4.6692016091029906...	1	Feigenbaum's δ
g	9.80665 *(per definition)*	m/s^2	Standard earth acceleration
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)*	kg	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)*		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_μ	1.883531475**E**-28 *(5.1**E**-8)*	kg	Muon mass
M☉	1.9891**E**30 *(5**E**-5)*		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"

	Numeric value	Unit	Remarks
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.
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 (please look at the site 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 $=\sqrt{\frac{\hbar G}{c^5}}=\frac{l_p}{c}$
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 $=\frac{e^2}{4\pi\varepsilon_0\hbar c}\approx\frac{1}{137}$

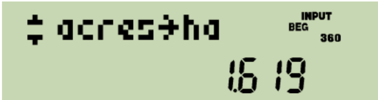
	Numeric value	Unit	Remarks
γEM	0.57721566490153286…	1	Euler-Mascheroni constant
γ_p	2.675222005**E**8 *(2.4**E**-8)*	$\frac{1}{s \cdot T}$	Proton gyromagnetic ratio $= 2\mu_P / \hbar$
ε_0	8.854187817…**E**-12	$\frac{A \cdot s}{V \cdot m}$ or F / m	Electric constant, vacuum permittivity $= \frac{1}{\mu_0 c^2}$
λ_c	2.4263102389**E**-12 *(6.5**E**-10)*	m	Compton wavelengths of the electron $= h / m_e c$, the neutron, and the proton, respectively.
λ_{cn}	1.3195909068**E**-15 *(8.2**E**-10)*		
λ_{cp}	1.32140985623**E**-15 *(7.1**E**-10)*		
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)
− ∞		1	Negative and positive infinity (may the Lord of Mathematics forgive us calling these 'constants')
∞			
#		1	See the very last command in the index *above*.

Unit Conversions (CONV)

This catalog mainly provides the means to convert local to common units [32]. 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

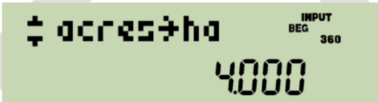


telling you 4 acres equal 1.619 hectares.

Now press f B and you will get

being the amount of acres equaling 4 hectares.

Press h CONV again and you see



confirming what was just said.

Leave the catalog via EXIT and the display will return to 9.884.

The constant **T₀** 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

[32] 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.

Conversion		Remarks	Class
Btu→J	* 1055.056	British thermal units	Energy
cal→J	* 4.1868		Energy
cft→*l*	* 28.31685	Cubic feet	Volume
cm→inches	/ 2.54		Length
cwt→kg	* 50.80235	(Long) hundredweight = 112 lbs	Mass
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	$1\ l = {}^{1}/_{1000}\ \mathrm{m}^3$	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→cwt	/ 50.80235	(Long) hundredweight = 112 lbs	Mass
kg→lb	/ 0.4535924		Mass
kg→stones	/ 6.35029318		Mass
kg→s.cwt	/ 45.35924	Short hundredweight = 100 lbs	Mass

Conversion		Remarks	Class
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
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
m*l*→flozUK	/ 28.41306	1 *ml* = 1 cm^{3}	Volume
m*l*→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

Conversion		Remarks	Class
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
rad→G	* 200 / π		Angle
stones→kg	* 6.35029318		Mass
s.cwt→kg	* 45.35924	Short hundredweight = 100 lbs	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 may be labels employed and provided for particular tasks already. You find them listed in CAT when the respective routines are loaded in flash memory. Thus they will not take any steps from user program memory in RAM.

Such 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.

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*** 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 CLα 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. A few special characters are not allowed in the last position for technical reasons. Single characters and grouped characters can be freely mixed. The alpha 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 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 catalog. 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 VWα+ command. It is meant to display alpha together with ('+') numeric data coming from any register. As with VIEW, register **X** is a legal input 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. VWα+ ***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 VWα+ 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'
CLα
α 'Hel'
α 'lo!'
PROMPT
BACK 001
LBL A
ENTRY?
SKIP 001
XEQ 01
STO 01
RTN
LBL B
...
END
```

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 four hotkeys might not suffice. There is, however, an easy way to extend the number of directly callable subroutines: shorthand addressing of numeric labels using keyboard codes as defined at right. Each key gets a code simply given by its row and column on the keyboard.

Whenever you are asked for the entry of a two-digit label, any of the keys highlighted green in this picture may be used for direct input. The label will then be replaced by the row/column code of the respective key. Keys not available this way (since they have another fixed meaning in this context) may still be used for a short address by pressing f before. Only f itself cannot be used for shorthand addressing.

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

If, for example, you want to associate a program with the key STO, just put the label 21 at the beginning of the routine, then it can be called via XEQ STO by the user conveniently.

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 **R00**. 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 pause is terminated immediately, KEY? is executed, finds the key code and stores it **R00**. LBL→00 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 e.g. a lunar landing game.

To be even more versatile, the instruction KTP? ***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 [.], [+/-], and [EEX]; 11 for [f], [g], and [h]; 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 your program you may feed it back to the main processing loop of the calculator with the PUTK ***nn*** command. It will cause the program to halt, and the key will be handled 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 a vector or matrix editor in user code. After execution of the PUTK command you are responsible for letting the program continue its work by pressing [R/S] or a hotkey.

APPENDIX A: SETUP AND COMMUNICATION

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/

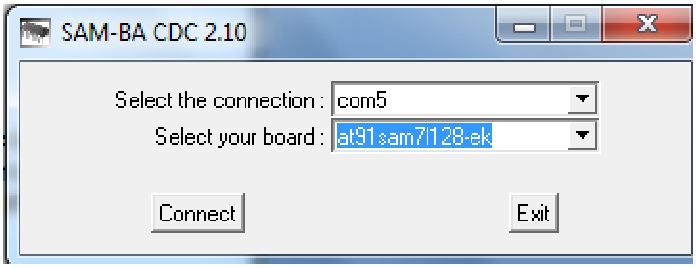
Now, having got these three (SAM-BA, the cable, and `calc.bin`), please turn to the file *http://dl.dropbox.com/u/10022608/Flashing%20a%2020b%20Calculator.pdf* (edited by Tim Wessmann and Gene Wright). Read it thoroughly for information about connecting and flashing.

> **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 Them and How to Make Them Yourself

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 fine adhesive vinyl overlays from Eric Rechlin.

If those are not available for any reason whatsoever, preliminary paper overlays are most easily made of a file contained in *http://wp34s.svn.sourceforge.net/viewvc/wp34s/artwork/* carrying the word “overlay” in its name. There are various ways to use it. E.g. on a PC with MS Word installed, open a new Word file, <insert> <graphics> <from file>, and select the desired file. Then format the picture setting its width to 68mm, and print that page. Cut it out, span it over your *WP 34S* using some transparent adhesive tape, and you are done.

You may – if you know how to handle a sharp pointed knife carefully – also cut along the thin white lines on the left, top, and right side of each key. Thereafter, attach the base paper to the key plate and each key will peek through its little door. When you stick the end of each such door or flap to the respective key then, your paper cover will come pretty close to a professional overlay already.

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.
6 (i.e. **S**) for SAM-BA:	Clears the GPNVM1 bit and turns the calculator off. Will work in debug mode only (see *below*) to prevent accidental access to this possibly dangerous feature. **ATTENTION:** You can now only boot in SAM-BA boot mode! Without the SAM-BA software and the cable mentioned above, you are lost!

These key combinations have to be pressed twice in a row without releasing ON 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 didn't accidently press the ERASE button on the cable but used ON + 6 instead to get into SAM-BA boot mode.

Further flash memory operations are LOADR, LOADST, LOADΣ, PRCL, and PSTO. See the *index*.

Mapping of Memory Regions to Emulator State Files

Region	State file	Remarks
Backup	`wp34s-backup.dat`	Is created by SAVE.
Lib	`wp34s-lib.dat`	Is written whenever a flash command is executed.
RAM	`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. It needs Atmel's original SAM_BA.exe program. For regular flashing of the firmware, MySamba from the flashtools directory on SF is recommended.

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 might know from flashing the firmware with this tool..
   h. Reset and press ON to power up. The RAM is automatically restored from the backup.
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.

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.

Both the emulator and the calculator can talk to each other with the same cable used for programming. In the emulator directory a text file wp34s.ini must be placed that contains the name of the port such as COM2:

The new Qt based emulators for Windows and MacOS contain a setup option for the serial interface. They will eventually replace the current Windows emulator completely. With a proper cable it is even possible to transfer data between two calculators with the same set of commands.

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 current program. After successful termination, the receiver will display `Program` .
2. SENDR will send the global numbered registers. The receiver will display `Register` after successful termination.
3. SENDA will send the complete two kilobytes of non-volatile RAM. The receiver will display `All RAM` after successful termination.

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 001
> ```
>
> and let it run for 30 seconds. The expected result at nominal speed is around 191,000. 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.

APPENDIX B: MEMORY MANAGEMENT

This chapter discusses how the available memory is divided in program area, local and global data. The two kilobytes (or 1024 *words*) of non-volatile RAM are divided in four distinct sectors:

1. Status and configuration data
2. Global registers, i.e. general purpose registers and stack
3. Registers used for cumulative statistics (optional)
4. Subroutine return stack (*SRR*) and program memory.

These sectors are ordered top down. This chapter covers the variable boundaries between them.

A complete copy of the nonvolatile RAM can be written to flash memory using SAVE (or ON+STO). See *Appendix A* for more information about the handling of flash memory.

Status and Configuration Data

This sector of 88 bytes is fixed at the very top of available memory and is completely user transparent. Thus it is not covered here besides saying it contains 42 bytes of status and modes data, the alpha register, and 14 bytes holding the 112 global user flags.

Global Registers

Global registers are placed near the end of available memory. In startup default memory layout, 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 global registers, which is the maximum available. Their number can be reduced down to 12 using REGS (see *above*). REGS? will return an integer between 0 and 100 corresponding to the number of global numbered registers currently allocated.

REGS controls the lower boundary of the global register sector (abbreviated *LBG* in the following). Reducing the number of registers will pull up the *LBG* to higher absolute addresses; increasing their number will push it down. The memory contents are moved accordingly, thus preserving the data in the surviving registers. Contents of deallocated registers are lost, newly added registers are cleared. The lettered registers do not move.

> **Example:** Please see the global register sector at startup default in the left three columns of the following memory table. The next two sets of two columns show what happens after subsequent execution of REGS 96 and REGS 98. The registers are loaded with arbitrary values here to allow tracing them easily. The *LBG* is indicated by a red line.

Absolute address	Default startup memory allocation		After executing REGS 96		Then after executing REGS 98	
	Contents	Relative register address	Contents	Relative register address	Contents	Relative register address
X+11	*k = 40.7*	R111 = K	*40.7*	K = R111	*40.7*	K = R111
…	…	…	…	…	…	…
X+2	*z = 4.5*	R102 = Z	*4.5*	Z = R102	*4.5*	Z = R102
X+1	*y = -33.8*	R101 = Y	*-33.8*	Y = R101	*-33.8*	Y = R101
X	*x = 123.0*	R100 = X	*123.0*	X = R100	*123.0*	X = R100
X-1	*r99 = -13.6*	R99	*23.1*	R95	*0.0*	R97
X-2	*r98 = 67.9*	R98	*6.4*	R94	*0.0*	R96
X-3	*r97 = -45.2*	R97	4.8	R93	*23.1*	R95
X-4	*r96 = 9.7*	R96	…	R92	*6.4*	R94
X-5	*r95 = 23.1*	R95	…	R91	*4.8*	R93
X-6	*r94 = 6.4*	R94	…	R90	…	R92
X-7	*r93 = 4.8*	R93	…	R89	…	R91
…	…	…	…	…	…	…
X-94	…	R06	*5.7*	R02	…	R04
X-95	…	R05	*-2.4*	R01	*81.3*	R03
X-96	…	R04	*1.1*	R00	*5.7*	R02
X-97	*r03 = 81.3*	R03			*-2.4*	R01
X-98	*r02 = 5.7*	R02			*1.1*	R00
X-99	*r01 = -2.4*	R01				
X-100	*r00 = 1.1*	R00				
…						

Please note the absolute addresses of **R00** up to **R*n-1*** change after REGS ***n*** whenever ***n*** is changed, while their contents are copied.

In indirect addressing, zero in the index register points to **R00** always. Index values exceeding the maximum set by REGS will throw an "out of range" error, unless they fall between 100 and 111 – where the lettered registers live.

The two sectors following in lower memory (summation registers and *SRR*) are tied to the *LBG* – they will be copied whenever it moves. This allows to execute REGS in the middle of a subroutine without disrupting the program.

Summation Registers

The memory needed for cumulative statistics is allocated separately – these data are no longer held in global registers. This allows for higher internal precision and prevents destroying these data manually by chance. The only way to update statistical data is via Σ+ and Σ– . The accumulated data are evaluated and recalled by dedicated commands, they are not accessible by STO or RCL.

The first invocation of Σ+ allocates 70 words for the 14 summation registers [33]. They are inserted just between *LBG* and *SRR*, pushing the latter down in memory. Depending on the competing requirements for program and data space, it may be necessary to make room first (see *below*).

When the number of statistical data points reaches zero, either by Σ–, CLΣ, CLALL, or RESET, the memory allocated for the summation registers is released again. All pointers are automatically adjusted so this allocation or deallocation will not disrupt a running program. Recall commands like e.g. Σn will return zero if no data are allocated, other statistical operations will throw an error if not enough data are present.

> **ATTENTION:** The summation data will be cleared automatically when a long program is loaded (from flash or via the serial interface) and after that load the registers would no longer fit in memory. You can avoid this by reducing the amount of numbered registers using REGS before the load attempt. This will (hopefully) move the summation data out of the way.

SRR and Program Memory

Both share the remaining space at lowest memory addresses.

The ***SRR*** is used for return addresses and local data (registers and flags). Its upper boundary is given by the *LBG* or the lowest summation register if applicable. There is no command to set the size of the *SRR* – it fills all the space down to the top program step currently stored. When new program steps are entered, the *SRR* is reset, not only to make room but because any stored address may become invalid by changing the program.

Local data are pushed on the *SRR*. Thus global data cannot be overwritten by them, enhancing the flexibility of programs significantly. LocR ***n*** allocates ***n*** local registers and a fixed amount of 16 local flags. Executing LocR, a frame is pushed on the *SRR* containing a marker, a flag word, and the register data. The marker contains the frame size in words, depending on the precision mode set (see *below*). A pointer to this frame in memory is initialized. If the pointer is zero, no local registers exist. Newly allocated registers are cleared.

Calling LocR again in the same subroutine will adjust the number of local registers. This requires data copying since these registers are allocated from low to high addresses and the *SRR* grows in the opposite direction. LocR? will return the number of local registers currently allocated in the routine you are in.

See *below* for addressing local data, and for an example of recursive programming. The *SRR* must be large enough to hold these data, however, so you may have to make room first – see next paragraph.

Below of the SRR, **program memory** holds the program steps stored. A typical program step is just one word. Multi byte labels and multi character alpha strings take two words each. The total size of program memory is dependent on the number of registers allocated (see next paragraph).

---

[33] Herein, 2 words are employed for Σn, 4 × 8 words for Σx^2, Σy^2, Σxy, and $\Sigma x^2 y$, and 9 × 4 words for the other sums. If memory allocation for these 70 words fails, an error will be thrown.

Making Room for Your Needs

The 12 special registers are always present. The *SRR* has a minimum size of six words or levels. Everything else is user distributable within the 980 words left for sections 2 to 4, so:

980 = ***r*** + ***s*** + ***p*** with

r = number of words allocated for global registers. These are 4 per standard register. There are at least 12 and max. 112 of them. So ***r*** varies between 48 and 896 (this maximum is explained *below*). Startup default is 448.

s = words allocated for summation registers (70 if applicable, startup default is 0).

p = number of words available for program steps and *SRR*. One step is taken by the inevitable final END statement already, 6 words are the minimum size of the *SRR*. So STATUS will show you a maximum of 931 free words in RAM, meaning up to 925 free program steps. Startup default is 532 steps. Subroutine nesting and employing local registers make the *SRR* grow and thus reduce the number of program steps available.

If, for instance, you need to do statistics and also use 20 global numbered registers, there will be space for 775 program steps maximum.

You have several options for increasing the free space where you need it:

1. Reduce the number of global numbered registers allocated. One register less allows for four additional program steps typically.
2. Move programs to flash memory and clear the respective steps in RAM. Four cleared steps allow for one additional register typically.
3. Deallocate the summation registers when you do not need them. The space may be distributed to up to 70 additional program steps, up to 17 additional registers, or a mix.

Which solution serves you best depends on the application. You may of course combine different options. Use STATUS to monitor the free space and the amount of global numbered registers allocated.

Addressing and Accessing Local Data

Global data take relative addresses from 0 to 111 as described *above*. So, relative addresses of local data begin with 112 and may go up to 255 if 144 local registers are allocated. The first 16 local registers and all local flags may be also directly addressed using a dot heading the number – the arguments go from **.**00 to **.**15, corresponding to relative addresses from 112 to 127 [34]. Any registers beyond are only indirectly addressable. This scheme allows for indirectly addressing

- a global register via a global index register (e.g. STO→23 with ***r23*** < 112),
- a global register via a local index register (e.g. STO→**.**15 with ***r.15*** < 112),
- a local register via a global index register (e.g. STO→47 with ***r47*** ≥ 112), and
- a local register via a local index register (e.g. STO→**.**06 with ***r.06*** ≥ 112).

Subroutine calls: XEQ – executed in a program – just pushes the return address on the *SRR* before it branches to the target. The subroutine called will keep having access to the caller's local data as long as it does not execute LocR itself. As soon as it does, the pointer to the local data is newly set, and the subroutine called cannot access the caller's local data anymore.

[34] Only arguments up to 127 are storable in an op-code, hence the limit.

RTN or PopLR – executed in a program – check if the current *SRR* pointer points to a local frame (as explained *above*). If true then the pointer is moved above that frame, and the *SRR* is searched from this point upwards for another local frame. If such a frame is found then its pointer is stored, else the pointer to the active local frame is cleared. RTN will branch to the return address found while PopLR will just continue execution. So the – until then – current local frame is dropped and the next higher (older) frame reactivated if existent.

Manually executing RTN, starting a new program with XEQ, or program editing will clear the *SRR* and remove all local data by clearing the pointer. Thus, all contents of local registers and flags are lost then!

Recursive Programming

Using local registers allows for creating a subroutine that calls itself recursively. Each invocation deals with its local data only. Of course the RPN stack is global so 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 001
STO .00
DEC X
XEQ 'FAC'
RCL× .00
RTN
```

Switching between Real and Integer

Your *WP 34S* starts in standard real mode (DECM) when you get it new. You may use it for integer computations as well, as shown above many times. Going from DECM to any integer mode, the values on stack will be truncated to integers. Going from integer mode to DECM, the integer stack contents will be converted to decimal. All other memory contents will stay as they were!

See the fate of some register contents undergoing subsequent mode switches in the following examples. The registers **J**, **K**, **R00**, and **R01** will be checked by recalling their contents to **X**:

	X	**Y**	**J**	**K**	**R00**	**R01**
Contents at start e.g.	1.1	20.2	300.3	4,000.4	50,000.5	600,000.6
Then after WSIZE 32, BASE 10	1 d	20 d	3,075 d	40,964 d	512,005 d	6,291,462 d
Recall the registers by sRCL			300 d	4,000 d	50,000 d	600,000 d
567 STO J, -9 STO 00	-9 d	567 d	567 d	4,000 d	-9 d	600,000 d
DECM	-9.0	567.0	5.7 E-396	4,000.4	3.0 E-389	600,000.6
Recall the registers by iRCL			567.0	40,964.0	-9.0	6,291,462.0

APPENDIX C: MESSAGES AND ERROR CODES

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 message when called:

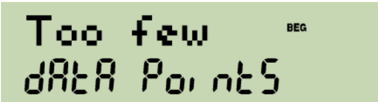
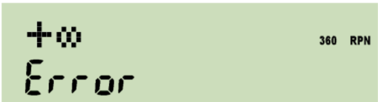
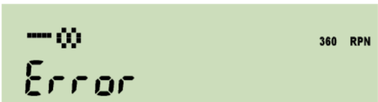
NaN will show `not nuMEric` and ∞ will display `InFInItY`.

Furthermore, there are a number of error messages. Depending on error conditions, the following messages will be displayed in the mode(s) listed [35]:

Message	Error Code	Mode(s)	Explanation and Examples
Bad time or dAtE	2	DECM	Invalid date format or incorrect date or time in input, e.g. month >12, day >31 etc.
Bad digit Error	9	Integer	Invalid digit in integer input, e.g. 2 in binary, 9 in octal, or +/- in unsigned mode (should never happen, but who knows).
Bad mode Error	13	All	Caused by calling an operation in a mode where it is not defined, e.g. SIN in hexadecimal.
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), Γ(0), TAN(90°) and equivalents, ATANH(x) for $\lvert \mathrm{Re}(x) \rvert \geq 1$, ACOSH(x) for $\mathrm{Re}(x) < 1$, etc.
Flash is FuLL	23	All	No more space in flash memory.
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.

35 Each of these messages may also be displayed using the command ERR in all modes.

Message	Error Code	Mode(s)	Explanation and Examples
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 MISMAtCH	21	DECM	• A matrix isn't square when it should be. • Matrix sizes aren't miscible.
No root Found	20	DECM	The solver did not converge.
No such LAbEL	6	All	Attempt to address an undefined label.
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 $\geq 2^{64}$, hours or degrees >9000, invalid times, denominators ≥9999 etc. • A register address exceeds the valid range. May also happen in indirect addressing or calling nonexistent locals. • 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.
RAM is FuLL	11	All	No more space in RAM (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.
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.

Message	Error Code	Mode(s)	Explanation and Examples
	15	DECM	A statistical calculation was started based on too few data points, e.g. regression or standard deviation for < 2 points.
	10	All	Keyboard input is too long for the buffer (should never happen, but who knows).
	3	All	An instruction with an undefined op-code occurred (should never happen, but who knows).
	14	Integer, ¬PRG	Stack or register content is too big for the word size set.
	19	All	Attempt to delete or edit program lines in flash memory.
	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*).
	5		

Error messages are temporary. ⌫ will erase the message and allow continuation. Any other key pressed will erase it as well and execute with the stack contents present. Thus, an easy and safe return to the display shown before the error occurred is pressing an arbitrary prefix twice.

APPENDIX D: INTERNAL COMMANDS

The following operations are for advanced users. They are not documented above to avoid overloading. They ease some internal jobs, mainly in programming, but require special care and/or a deeper understanding of the respective 'mechanics' of the *WP 34S*. Use them at your own risk! They are collected in the internal catalog (CPX h MODE). The table below follows the rules applying for the master index above.

Name	Keys to press	in modes	Remarks
BSRB	… BSRB ***n***	PRG	BSRB (BSRF) calls a subroutine starting ***n*** program steps backwards (forwards) with $0 \le n \le 255$. It pushes the program counter on the return stack and executes BACK ***n*** (SKIP ***n***) then. The subroutine called might not start with a label. So, these two commands are most useful if you are short on local labels, but they carry the same risks as BACK and SKIP when editing programs.
BSRF	… BSRF ***n***		
CNVG?	… CNVG? ***c***	Integer	Tests for $x = y$.
		DECM	Checks for convergence by comparing x and y as determined by the lowest five bits of ***c***. The very lowest two bits set the tolerance limit: 0 = 1E-14, 1 = 1E-24, 2 = 1E-32, 3 = Choose the best for the modes set, resulting in taking 0 for single precision and 2 for double precision. The next two bits determine the comparison mode: 0 = compare the real numbers x and y relatively, 1 = compare them absolutely, 2 = check the absolute difference between the complex values $x + i\,y$ and $z + i\,t$, 3 = works as 0 so far. The top bit tells how special numbers are handled: 0 = NaN and infinities are considered converged, 1 = they are not considered converged.
DBLOFF DBLON	…	¬α	Toggles double precision mode. Setting becomes effective in DECM only. See next page.
DBL?	…	¬α	Checks if double precision mode is turned on.
dRCL	… dRCL ***s***	¬α	Assumes the source ***s*** contains double precision real data and recalls them as such. See next page.

[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).

Switching between Single Precision, Double Precision and Integer

Your *WP 34S* starts in standard (i.e. single precision or *SP*) real mode per default. Switching between *SP* and integer modes was discussed *above* already. You may also use your *WP 34S* in double precision (*DP*) mode for (hopefully) more precise decimal calculations.

Each *DP* register will contain 16 bytes instead of eight, allowing for 34 digits instead of 16. [SHOW] (and [◀]) display the 16 most significant digits only in this mode, while [▶] shows the 18 trailing digits. Please note matrix commands will not work in *DP*.

The following figure illustrates what happens in memory in transitions between *SP* and *DP* modes, assuming startup in *SP* mode with REGS 16:

	Startup memory allocation		After executing DBLON		Then after executing DBLOFF	
Absolute address	Contents	Relative register address	Contents	Relative register address	Contents	Relative register address
X+11	*k = 1.40E-397*	**R111 = K**	*1.40E-397*	**K = R111**	*1.40E-397*	**K = R111**
X+10	…	…			…	…
…	…	…	…	…	…	…
X+1	*y = -33.8*	**R101 = Y**	…	**C = R106**	*-33.8*	**Y = R101**
X	*x = 123.0*	**R100 = X**			*123.0*	**X = R100**
X-1	*r15 = -43.6*	**R15**	…	**B**	*0.0*	**R15**
X-2	*r14 = 167.9*	**R14**			*0.0*	**R14**
X-3	…	**R13**	…	**A**	*0.0*	**R13**
X-4	…	**R12**			*0.0*	**R12**
X-5	…	**R11**	…	**T**	*0.0*	**R11**
X-6	…	**R10**			*0.0*	**R10**
X-7	…	**R09**	…	**Z**	*0.0*	**R09**
X-8	…	**R08**			*0.0*	**R08**
X-9	…	**R07**	*-33.8*	**Y = R101**	*0.0*	**R07**
X-10	…	**R06**			*0.0*	**R06**
X-11	…	**R05**	*123.0*	**X = R100**	*0.0*	**R05**
X-12	*r04 = -12.9*	**R04**			*0.0*	**R04**
X-13	*r03 = -1234.89*	**R03**	*-1.95E184*	**R01**	*-1234.89*	**R03**
X-14	*r02 = 5.43E-396*	**R02**			*5.43E-396*	**R02**
X-15	*r01 = 6.6*	**R01**	*1.03E182*	**R00**	*6.6*	**R01**
X-16	*r00 = 0.54*	**R00**			*0.54*	**R00**
X-17						

Going from *SP* to *DP* mode, the contents of the twelve special registers **X … K** are copied, cutting 48 bytes into the former *SP* numbered register sector. So the top twelve *SP* numbered registers will be lost in such a transition. All other memory contents stay where and as they were – just each relative *DP* register address covers what were two *SP* registers before. – Starting with the default memory configuration and executing DBLON then will leave you with 44 *DP* registers. Executing REGS with an argument >44 in *DP* is legal, but the sector of global numbered registers will cut into the former program sector then.

Returning from *DP* to *SP*, the lettered registers are copied again. And everything else stays where and as it was, if you used ≤44 *DP* registers – just each relative *SP* register address points to only one half of a former *DP* register; and the memory released by the shrinking special registers allows adding (or returning) twelve numbered registers on top, each containing zero now. With >44 *DP* registers, the correspondence becomes more complicated - the number of global registers will not, however, exceed 112.

The space allocated for summation registers will not change in such transitions.

For the following table, assume startup in BASE 10, WSIZE 32, REGS 16. Now see the contents of **J**, **K**, and the lowest numbered registers, checked by recalling them to **X**:

	J	**K**	**R00**	**R01**	**R02**	**R03**
Starts with e.g.	3,504 d	14 d	54 d	66 d	543 d	126,441 d
DECM	34.32	1.40E-397	0.54	6.6	5.43E-396	-1234.89
DBLON	34.32	1.40E-397	1.03E182	-1.95E184	n/a	n/a
Recall by sRCL	34.32	1.40E-397	0.54	6.6	5.43E-396	-1234.89
… and by iRCL	3,504.00	14.00	54.00	66.00	543.00	126,441.00
DBLOFF	34.32	1.40E-397	0.54	6.6	5.43E-396	-1234.89
Recall by dRCL	Out of range	Out of range	1.03E182	-1.95E184	n/a	n/a
DBLON	34.32	1.40E-397	1.03E182	-1.95E184	n/a	n/a
RCL J, STO J, RCL K, STO K, then recall by sRCL	1.40E-397	1.28E-23	0.54	6.6	5.43E-396	-1234.89
… and by iRCL	14.00	0.00	54.00	66.00	543.00	126,441.00
DBLOFF	34.32	1.40E-397	0.54	6.6	5.43E-396	-1234.89

Please note iRCL and sRCL will keep working as explained *above*.

APPENDIX E: LIBRARY ROUTINES

TVM lives in the library file `wp34s-1.dat`, located in the library directory. Here is how to install this routine 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 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 memory region as described *above*.

APPENDIX F: 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?, $^x\sqrt{y}$, 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. With build 1990, this version is available as the last one working with the old overlays.
3.0	27.2.12	Added CLPALL, iRCL, sRCL, END, FLASH?, g_d, g_d^{-1}, GTO.▲ and ▼, LOAD…, LocR, LocR?, MEM?, NEIGHB, PopLR, REGS, REGS?, RSD, SEPOFF, SEPON, SETJPN, t⇆, ULP, ⇆, #, as well as SUMS and MATRIX catalogs and four conversions; renamed KTY? to KTP?, CUBE to x^3 and CUBERT to $^3\sqrt{x}$; split Lambert's W into W_p and W_m; returned →BIN, →HEX, and →OCT; deleted DEL_P, removed LNβ, LNΓ, β, and Γ from STAT; changed keyboard layout to bring MATRIX, CLP, SF, and CF to the front and to swap OFF and SHOW, removed x⇔α from the key plate; modified the virtual alpha keyboard, some characters and the respective catalogs; redistributed commands in the catalogs; updated and rearranged the introductory chapters and those about memory, display, programming, messages, and internal commands; added sections about local data and memory management.