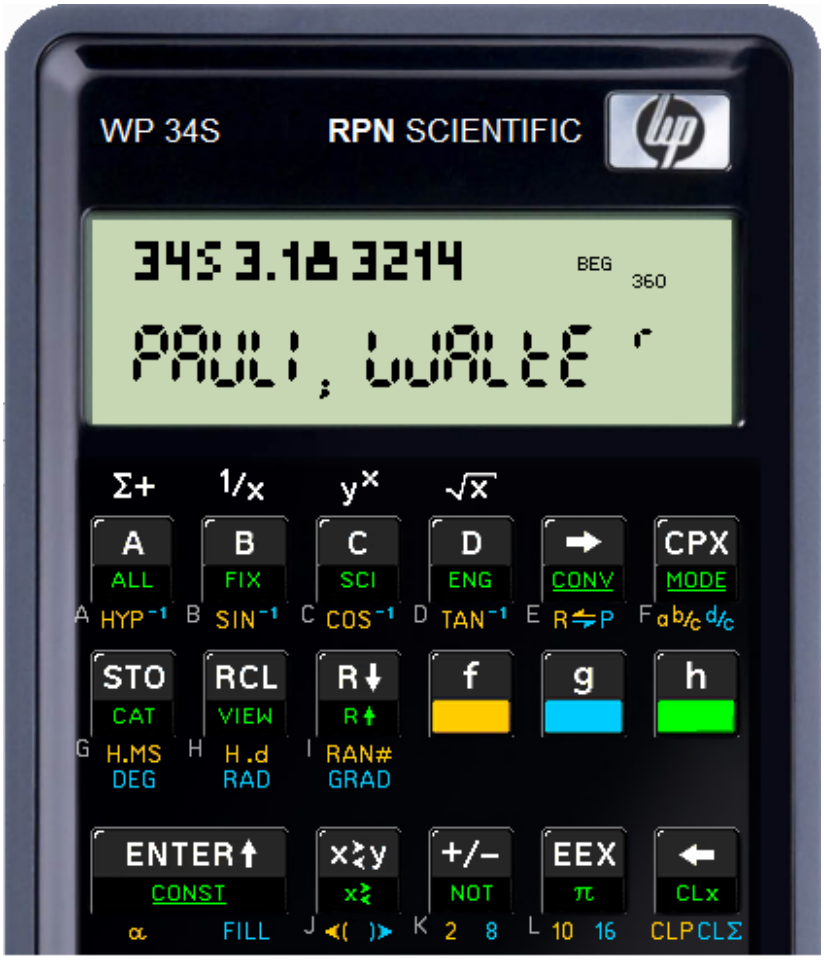
WP 34S OWNER'S MANUAL



This manual documents *WP 34S*. *WP 34S* is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

WP 34S is distributed in the hope that it will be useful, but without any warranty; without even the implied warranty of merchantability or fitness for a particular purpose. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with *WP 34S*. If not, please see *http://www.gnu.org/licenses/* .

This manual contains valuable information. It was designed and written in the hope that it will be read thoroughly by you. For really quick and impatient users, there is even a stand-alone *Troubleshooting Guide* included.

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

This manual may change without notice if we, the developers, modify *WP 34S*. We reserve the right to do so at any time we think it appropriate. We recommend you watch *http://sourceforge.net/projects/wp34s/develop* to stay informed.

WP 34S includes a full size emulator, so you may test *WP 34S* on your computer before you buy any calculator hardware. And you may test your programs as well before downloading them to your calculator.

TABLE OF CONTENTS

WP 34S would not have reached its present state without our love for *Classics, Woodstocks, Spices, Nuts, Voyagers*, and *Pioneers*. Thus we want to quote what was printed in *HP* pocket calculator manuals until 1980, so it will not fade:

"The success and prosperity of our company will be assured only if we offer our customers superior products that fill real needs and provide lasting value, and that are supported by a wide variety of useful services, both before and after sales."

Statement of Corporate Objectives.
Hewlett-Packard

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 instructions how to convert it into a full-fledged *WP 34S* yourself. On the other hand, if you do not want to bother with cables on your desk connecting it to your computer, with flashing the calculator firmware and attaching a sticky overlay, you can purchase an *HP-30b*-based *WP 34S* readily on the internet; see e.g. *http://commerce.hpcalc.org/34s.php* or

http://www.thecalculatorstore.com/epages/eb9376.sf/en_GB/?ObjectPath=/Shops/eb9376/Products/%22WP34s%20Pack%22 .

The first way (doing it yourself) will just cost you time, the second will cost you some money. If you choose buying your *WP 34S* at the addresses mentioned, we (the developers) will get a very modest fraction of the price to support our otherwise unpaid efforts on the *WP 34S* project.. Either way will work – it is your choice.

Furthermore, there are two optional hardware modifications requiring some soldering, cutting, and drilling a little hole in the plastic case of your business calculator. It is no sorcery, but check where you get this done unless you want to do it yourself.

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 an *HP-20b Business Consultant* or an *HP-30b Business Professional*, so you benefit from their excellent processor speed. And with an *HP-30b* you also get the famous rotate-and-click keys, 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 scientific calculator like you have never had before** – fully 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 [1]. We expanded this set, incorporating the functions of the renowned computer scientist's *HP-16C*, the fraction mode of the *HP-32SII*, and probability distributions like those of the *HP-21S.* And we added **many more useful functions for mathematics, statistics, physics, engineering, programming, I/O, etc.** like

- Euler's Beta and Riemann's Zeta functions, Bernoulli and Fibonacci numbers, Lambert's W, the error function, and Chebyshev's, Hermite's, Laguerre's, and Legendre's orthogonal polynomials (no more need for carrying heavy printed tables),
- many statistical distributions and their inverses: Poisson, Binomial, Geometric, Cauchy-Lorentz, Exponential, Logistic, Weibull, Lognormal, and Gaussian,
- programmable sums and products, first and second derivatives,
- testing for primality,
- integer computing in fifteen bases from binary to hexadecimal,
- financial operations like mean rate of return and margin calculations,
- extended date and time operations and a stopwatch [2] based on a real-time clock,
- 84 conversions, mainly from old *Imperial* to universal *SI* units and vice versa,
- 50 fundamental physical constants as precise as known today by national standards institutes like *NIST* or *PTB*, plus a selection of important constants from mathematics, astronomy, and surveying,
- bidirectional serial communication with your computer, prints on an *HP 82240A/B* [3],
- battery-fail-safe on-board backup memory,

[1] Still true in 2012 although it was introduced in 1988. – The matrix menu of the *HP-42S* cannot be supported by *WP 34S* for hardware reasons. Your *WP 34S* features a set of basic matrix commands and several library routines instead.

[2] The stopwatch requires that a quartz crystal and two capacitors are added to the electronics.

[3] Printing requires an IR-diode and a resistor added.

- Greek and an extended Latin letter set covering the languages of almost half of this planet (upper and lower case in two font sizes each) plus mathematical symbols.

***WP 34S* is the first *RPN* calculator overcoming the limits of a four-level stack** – forget worries about stack overflow in calculations. *WP 34S* 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 in 4D, or whatever application you have in mind. You will find a full set of commands for stack handling and navigation in either size.

Furthermore, your *WP 34S* features up to 108 global general purpose registers, 112 global user flags, up to 928 program steps in *RAM*, several thousand steps in flash memory, a 30 byte alpha register for message generation, 16 local flags and up to 144 local registers allowing for recursive programming, and 4 user-programmable hotkeys for your favorite functions. Memory layout is user-settable to a large extent.

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 other names for this). From its beginning, we discussed our project in the *Museum of HP Calculators*, and we thank all the international forum members there who taught us a lot and gave us their ideas and support in several stages of our project. Special thanks go to *Marcus von Cube* (Germany) joining us in bringing *WP 34S* to life, starting with an emulator for v1.14, allowing widespread use and convenient testing. With v1.17, the software began running on an *HP-20b*. A very useful assembler / disassembler is supplied by *Neil Hamilton* (Canada) since v1.18 – even a symbolic preprocessor was added with v2.1. For v3.0, *Pascal Méheut* (France) contributed a versatile flashing tool for various operating systems. With v3.1, output on an *HP82240B* printer became possible thanks to gracious support by *Christoph Gießelink* (Germany), *Ciaran Brady* (UK) wrote a *Beginner's Guide* for our calculator, and *Andrew Davie* and *Peter Murphy* (both USA) did carefully proofread this manual here. Thanks to all of them!

We baptized our baby *WP 34S* in honor of the *HP-34C* of 1979, one of the most powerful compact LED pocket calculators. *WP 34S* is our humble approach – within the constraints of *HP's* hardware – to a future *43S* we can only dream of becoming the successor of the *HP-42S*. May our project help to convince those that have access to more resources than we do that catering to the market for serious scientific instruments is well worthwhile.

We have carefully checked everything we could think of to the best of our ability, so our hope is that *WP 34S* is free of severe bugs. We cannot warrant this, however. In any case, we promise to continue improving *WP 34S* whenever it turns out to be 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 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 or **bold** printing. *Specific terms, titles, trademarks, names* or *abbreviations* are printed in italics, *hyperlinks* in blue underlined italics. Bold italic letters like ***n*** are used for variables; constant sample values (e.g. of labels or displayed characters) use bold normal letters. Calculator COMMANDS are generally called by their names, printed in capitals in running text.

- This CPX font is taken for explicit references to calculator keys. Alphanumeric and numeric displays (like Hello! and 1234) are quoted using the respective calculator fonts where applicable and beneficial.

- Courier is employed for `file names` and numeric `formats`.

- Lower case italic Times New Roman is for *units*.

- Register **ADDRESSES** are printed using bold Times New Roman capitals, while lower case bold italics of this font are employed for variable ***register contents***. So e.g. ***y*** lives in stack register **Y**, ***r45*** in general purpose register **R45**, and ***alpha*** in the alpha register. Overall stack contents are generally quoted in the order [***x***, ***y***, ***z***, …] .

All this holds unless stated otherwise.

Finally: **WARNING** indicates the risk of severe errors. There are only four warnings contained in this manual. Locking up your calculator is the worst that can happen to it, as far as we know.

GETTING STARTED

If you know how to use a good old *Hewlett-Packard RPN* scientific calculator, you can start using your *WP 34S* almost right away. Use this manual to get information on some basic design concepts that put your *WP 34S* ahead of previous *RPN* calculators. Continue using this manual for reference.

> On the other hand, if this is your first *RPN* scientific calculator, or the first you use for a long time, we recommend you get an *HP-42S Owner's Manual*. It is available at low cost on a *DVD* distributed by the *Museum of HP Calculators*.
>
> Read Part 1 of said manual as a starter. It includes an excellent introduction to *RPN,* a very effective and coherent method for calculations. *RPN* is the reason why your *WP 34S* works without an [=] key. Once you got used to *RPN* you may never choose a calculator featuring [=] again.
>
> Part 2 of the *HP-42S Owner's Manual* will help you programming your calculator for quick and easy handling of lengthy, repeated or iterative computations. Further documentation, including complete information about the other vintage calculators and the famous *PPC ROM* mentioned in this manual, is readily accessible on said *DVD*, too.
>
> Alternatively to the *HP-42S Owner's Manual*, you can download a dedicated *WP 34S Beginners Guide*, recently written by one of our users:
> *http://wp34s.svn.sourceforge.net/viewvc/wp34s/doc/WP_34S_Beg_Guide.pdf* .

Most traditional commands on your *WP 34S* will work as they did on the *HP-42S*. This little manual is meant as a supplement presenting all the new features. It contains the necessary information about them, including equations and technical explanations; it is not intended to replace textbooks on mathematics, statistics, physics, engineering, programming, or the like, nor is it a hypothetical *Beginner's Guide to RPN Computing*.

Your *WP 34S* is designed to help you in calculations and computations. It is, however, just a tool – although a very powerful one – it cannot think for you nor can it check the sensibility of a problem you apply it to. Do not blame us nor your *WP 34S* for errors you have made. Gather information, think before and while keying in, and check your results: these will remain your responsibilities always.

The following text starts by presenting you the user interface, so you learn where you will find what you are looking for. It continues by demonstrating some basic methods, the calculator memory and addressing items therein, and the display and indicators that give you feedback about what is going on. Then the major part of this manual consists of an index of all available operations and how to access them, and of lists of all catalog contents. This manual closes with appendices covering special topics, e.g. a list of error messages your *WP 34S* will return if abnormal conditions prevent it from executing your command as expected.

Keyboard Basics

Start exploring your *WP 34S*: Press its bottom left key to turn it on – notice that **ON** is printed below that key. If you turn on your *WP 34S* the very first time, you will get what you see displayed below. To turn it off again, press the green key **h** (notice a little ʰ showing up top left in display), then **EXIT** (which has **OFF** printed on its lower part). Since your *WP 34S* has *Continuous Memory*, turning it off does not affect the information it contains. To conserve battery energy, your *WP 34S* shuts down some five minutes after you stop using it – when you turn it on again, you can resume your work right where you left off.

To adjust display contrast, hold down **ON** while you press **+** or **−** – like on an *HP-42S*.

The most striking difference to the *HP-42S* is the colorful keyboard of your *WP 34S.* You get five functions per key on average. White print on key top is for the *primary* function of the key. For *secondary* functions, green labels are put on the slanted lower faces of 34 keys, golden and blue labels are printed below them on the *key plate.* Grey letters are bottom left of 26 keys.

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

Take the key **5** for **example**. Pressing

- **5** will enter the digit 5 in display,
- **f** + **5** will calculate the arithmetic mean values of data accumulated in the statistic registers via **x̄**,
- **g** + **5** will compute the standard deviations for the same data via **s**,
- **h** + **5** will open a *catalog* (i.e. a set) of extra statistical functions via **STAT**. All 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. The active prefix is indicated by f, g, or h top left in the display for visual feedback. You may hold down f, g, or h if you want to call several functions in sequence showing the same color.

Time for a little problem solving **example**. Turn your *WP 34S* on again if necessary (it may have shut down automatically in the meantime). Anyway it will still show its last display

[4].

Now let us assume you want to fence a little rectangular patch of land, 40 *yards* long and 30 *yards* wide [5]. You have already set the first corner post (A), and also the second (B) in a distance of 40 *yards* from A. Where do you place the third post (C) to be sure that the fence will form a proper rectangle? Simply key in

4 0	
ENTER↑	 (this key separates the two numbers in input here; note 40 is shifted to the right and a radix mark is added [6])
3 0	
→P	 (→P is reached by pressing g, then →)

So, just take 80 *yards* of rope, nail its one end on post A and its other end on B, fetch the loose loop and walk 30 *yards* away. When both sections 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: whatever other distances may apply in your case. As soon as you press →P, your *WP 34S* does the necessary calculation of the diagonal automatically for you. You just provide the land, posts, rope, hammer and nails. And it will be up to you to set the posts!

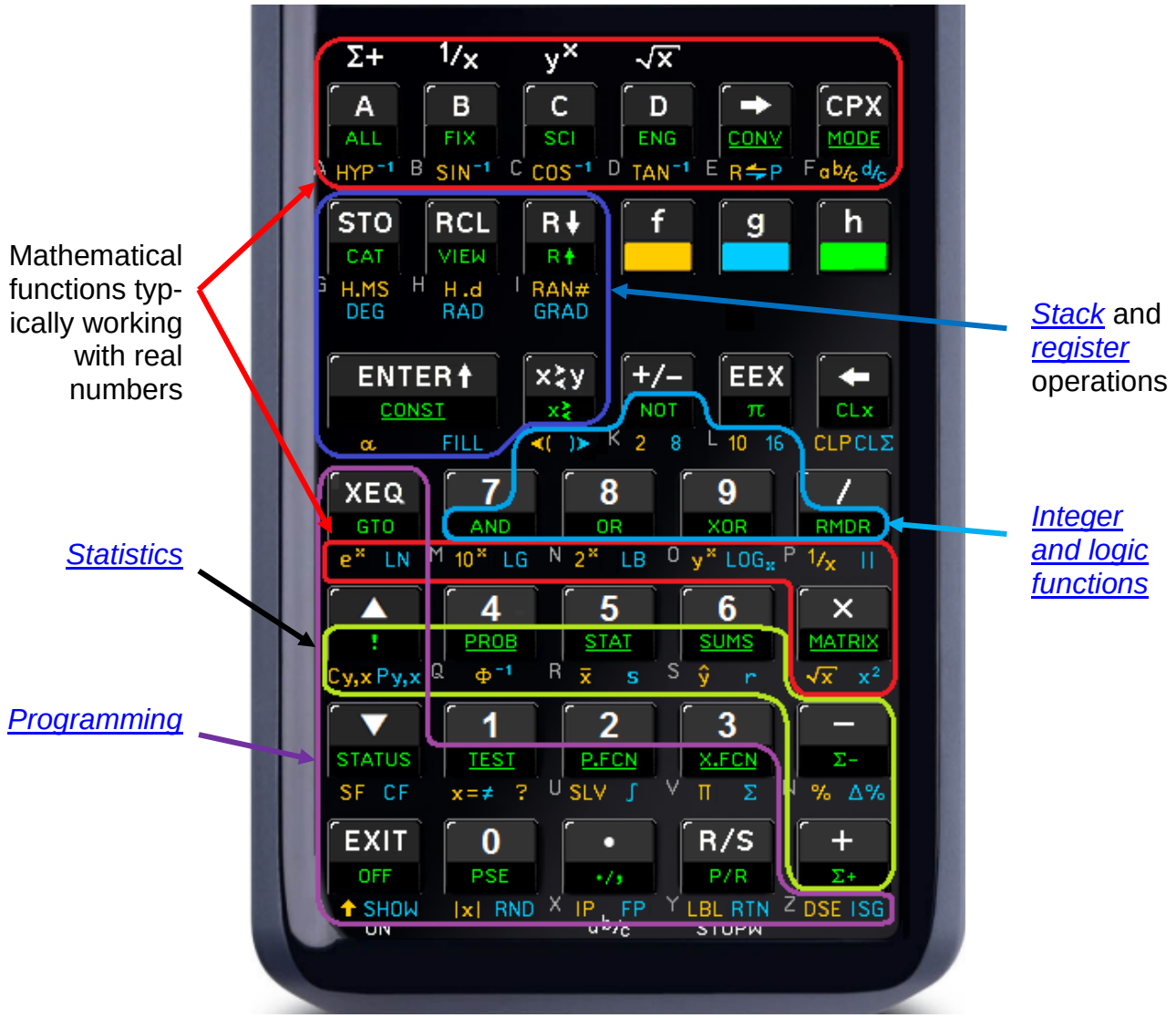
---

[4] If your *WP 34S* fails to show this display for any reason whatsoever (e.g. because you played around with it a bit), you will get it by sliding the battery cover open, locating the little hole below the label RESET and actuating the button behind it using a suitable pin. Then close the cover again.

[5] 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. Despite this fact, we use (old British) Imperial units here so our US-American readers can follow. But the example will work with *meters* as well.

[6] This indicates input being finished for this number. – Generally, we shall quote only numeric displays in the following, using the proper font. And we will refer to keyboard labels in this text using dark print on white like e.g. EEX or π, omitting the prefix h for the latter since redundant.
Also starting here, points will be used as radix marks, although significantly less visible than commas, unless specified otherwise explicitly. By experience, “comma people“ seem to be more capable of reading radix points and interpreting them correctly than vice versa.

As you have found [→P] next to other labels showing an arrow as well, the labels on your *WP 34S* are generally grouped according to their purposes. Besides the keys for numeric input (the ten digits, [.], [+/−], [EEX], and [←]), the three prefixes explained so far ([f], [g], and [h]), and the four elementary arithmetic operations ([+], [−], [×], and [/]), there are five somewhat larger groups of labels – see a general map of function grouping here:



Most of the 168 labels printed on your *WP 34S* point to operations carrying simply the same name. [ALL], for instance, calls the function ALL and [FIX] calls FIX. In the example above, however, pressing [→P] called the function →POL – there are only ten cases like this needing a little extra translation. Let us introduce them to you, starting top left on the keyboard:

1. [A], [B], [C], and [D] are named *hotkeys*, since they will directly call the user programs carrying these labels. If the respective labels are not defined (yet), these keys act as [Σ+], [1/x], [y^x], or [√x], respectively, as is printed above them.

2. HYP is the prefix for hyperbolic functions SINH, COSH, and TANH, as HYP^{-1} is for their inverses ASINH, ACOSH, and ATANH. Similarly, SIN^{-1} stands for ASIN, etc.

3. → is a prefix for direct conversion of the value currently displayed (i.e. ***x***). It may be trailed by H.MS, H.d, DEG, RAD, or GRAD (the respective function names then read like →H.MS).
→ trailed by 2, 8, or 16 will display ***x*** converted to an integer of the respective base until the next keystroke.
→ is also used for accessing registers indirectly (see below).
R← converts polar to rectangular coordinates in a plane (see →REC), →P converts vice versa. So the pair R←→P covers the two classic coordinate transformations.

4. CPX is mainly for calling complex operations (see *below* for more). a b/c and d/c enter the fraction mode for proper or improper fractions, respectively (see PROFRC and IMPFRC).

5. H.MS and H.d represent the classic two time modes, where H.d stands for decimal hours and also for decimal floating point numbers in general (see DECM).

6. α enters *alpha mode*, while 2, 8, 10, or 16 enter *integer modes* for calculating with binary, octal, decimal, or hexadecimal numbers (see next pages).

7. LG returns the logarithm for base 10, LB does the same for base 2.

8. ! calls ***x!*** in all numeric modes and inserts an exclamation mark in alpha mode. Cy,x and Py,x work like on an *HP-15C* (calling COMB and PERM here, respectively).

9. |x| calls ABS, and RND works like on an *HP-15C* (calling ROUND on your *WP 34S*).

10. There are three toggles: ./, for radix marks, P/R for programming (like on the *HP-15C*), and ↑ for upper and lower case in alpha mode – else the latter will send ***x*** to the printer.

These are all the special labels featured. You will find a complete list of each and every command provided, the keystrokes calling it, and the necessary individual explanation in the *Index of Operations (IOP)* below for your reference.

Let us return to our introductory example [7] for three remarks:

1. Note any numeric input will just fill the display and is interpreted when completed, not earlier.
2. There is no need to enter any units in your calculations. Just stay with a consistent set of units and you will get meaningful results within this set [8]. If you want to convert results from one unit to another, see the catalog CONV described further below.
3. Although we entered integer numbers only for both sides of our little ground, your *WP 34S* calculated in default floating point mode. This allows for decimal fractions of e.g. *yards* in input and output as well. Another mode lets you enter proper fractions like e.g. 6 ¼ where you need them. Your *WP 34S* features more modes – we will introduce them to you *later*). Before, let us show you some more typical calculations – starting with some ways to put numbers in your *WP 34S*.

Entering Numbers

… is as easy as typing. For 12.34, for example, press [1][2][.][3][4] . Any digit mistyped may be deleted immediately by [←] and you can enter it correctly thereafter.

For negative numbers like –5.6 , press [5][+/−][.][6] or [5][.][+/−][6] or [5][.][6][+/−] – the [+/−] changes the sign of the number put in.

For putting in really big stuff like the age of the universe as we know it, do it this way: [1][3][.][8][EEX][9] resulting in 13.8 with nine digits trailing the point, i.e. 13 800 000 000 years. Really tiny

[7] Generally, we assume you have graduated from US High School at minimum, passed Abitur, Matura, or an equivalent graduation. So we will not explain basic mathematical rules and concepts here.
And in four decades of scientific pocket calculators, a wealth of funny to sophisticated applications has been created and described by different authors – more and better than we can ever invent ourselves. It is not our intention to copy them. Instead, we recommend the DVD mentioned *above* once again: it contains nearly all the user guides, handbooks, and manuals published for vintage *Hewlett-Packard* calculators beginning with their very first, the *HP-9100A* of 1968. Be assured that almost every calculation described there for any scientific calculator can be done significantly faster on your *WP 34S* – and often even in a more elegant way.

[8] The big advantage of *SI* is that it is the largest consistent set available.

numbers like the diameter of an atom (0.000 000 000 1*m* – ten zeroes heading the 1) are entered in analogy: [EEX] [+/-] [1] [0] , corresponding to $1 \cdot 10^{-10}\, m$.

Calculating in Real Domain

Most of the commands your *WP 34S* features are mathematical operations or functions taking and returning real numbers like 1 or -2.34 or π or 5.6E-7. Note that integer numbers like 8, 9, 10, or -1 are just a subset of real numbers.

Many real number functions provided operate on one number only. For **example**,

enter [.] [4] [9] 0.49

and press [√x] . You will get 0.7 since $0.7^2 = 0.49$.

That is easy, isn't it? Generally, such functions replace ***x*** (the value displayed) by the function result **f(*x*)** . The vast majority of calculators works this way, so this is no real surprise.

Some of the most popular mathematical functions, however, operate on two numbers instead. Think of + and –, for example.

> **Example**: Assume having an account of 1,234 US$ and taking 56.7 US$ 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
Start a new line and write down the 2nd number:	56.7	Separate 1st from 2nd:	[ENTER↑]	
		Key in the 2nd number:	[5] [6] [.] [7]	56.7
Subtract:	1177.3	Subtract:	[−]	1177.3

This is the essence of *RPN*:

Provide the necessary operands, then execute the requested operation.

And a major advantage of *RPN* compared to other entry systems for calculators is that it sticks to this basic rule. Always [9].

[9] Some people claim this being true for *RPL* only. *RPL* is a language developed from *RPN* in the 1980's. Maybe they are even right. In my opinion, however, *RPL* strains the underlying *postfix* principle beyond the pain barrier, exceeding the limit where it becomes annoying for the human brain. Not for everybody, of course, but also for many scientists and engineers. Thus we decided to stick to *RPN* on the *WP 34S*.

As the paper holds your operands before you calculate manually, 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.

Elementary Stack Mechanics

Think of the stack like a pile of registers: bottom up, they are traditionally named **X, Y, Z, T**, optionally followed by **A, B, C**, and **D** on your *WP 34S*. New input is always loaded in **X**, and only ***x*** is displayed on your *WP 34S*.

	D
	C
	B
	A
	T
	Z
	Y
Display	**X**

[ENTER↑] separates two input numbers by closing the number ***x*** and copying it into **Y** [10], so **X** can take another input then without losing information.

Having completed that second input in the example above, [−] subtracts ***x*** from ***y*** and puts the result **f(*x*, *y*)** = ***y*** − ***x*** into **X** for display. This method applies for most two-number real functions.

A large fraction of mathematics is covered by two-number functions in the way we have just seen. Let us take the next step: a chain calculation, for **example**

$$\frac{(12.3-45.6)\cdot(78.9+1.2)}{(3.4-5.6)^7}\,.$$

Look at this as a combination of six two-number functions: three additions, a product, an exponentiation, and a division. That is exactly how it is solved on your *WP 34S*.

Let us start:

T				
Z				
Y		12.3	12.3	
X	12.3	12.3	45.6	-33.3
Input	[1][2][.][3]	[ENTER↑]	[4][5][.][6]	[−]

You will have recognized that the first parenthesis was solved exactly as shown above. Now proceed to the second:

[10] This is the classic way ENTER worked from the *HP-35* of 1972 until the *HP-42S* ceased in 1995. It is often said ENTER 'pushes ***x*** on the stack'. In doing so, the higher stack contents are lifted out of the way before. So ***z*** goes into **T** and ***y*** into **Z** before ***x*** goes into **Y**. See page 17 for detailed stack pictures. The *HP-30b* employs a different ENTER – the *WP 34S* sticks to classic *RPN*, however.

T					
Z		-33.3	-33.3		
Y	A -33.3	78.9	78.9	-33.3	
X	78.9	78.9	1.2	80.1	-2667.33
Input	78.9	ENTER↑	1.2	+	×

Note the result of the first parenthesis was lifted automatically (A) to **Y** to avoid overwriting when the next number was keyed in step 1 of this row. This is called *automatic stack lift* and is standard in *RPN* calculators [11].

And after having solved the second parenthesis in step 4 of row 2, we had the results of both upper parentheses on the stack – so everything was ready for multiplication to complete the numerator.

Now we will simply go on and start calculating the denominator:

T						
Z		-2667.33	-2667.33		A -2667.33	
Y	A -2667.33	3.4	3.4	-2667.33	A -2.2	-2667.33
X	3.4	3.4	5.6	-2.2	7	-24943...
Input	3.4	ENTER↑	5.6	−	7 [12]	y^x

Last job remaining is the final division of numerator by denominator. Both are on the stack in the right order. Just press / and see the result: 106934534648 .

As you have observed several times now, the contents of the stack registers drop when a two-number function is executed. The top stack level content is repeated then (since there is nothing available above for dropping). You may employ this top level repetition for some nice tricks. See the following compound interest calculation, for **example**:

Assume the bank pays you 3.25% p.a. on an amount of 15,000 US$, what would be your account status after 2, 3, 5, and 8 years? You are interested in currency values only, so you set the display format to FIX 2 for this, causing the output being shown rounded to next cent (internally, the numbers are kept with far higher precision).

T		1.0325	1.0325	1.0325	1.0325	1.0325	1.0325
Z		1.0325	1.0325	1.0325	1.0325	1.0325	1.0325
Y		1.0325	1.0325	1.0325	1.0325	1.0325	1.0325
X	1.0325	1.03	15 003	1599.084	1651.055	1760.117	1937.366
Input	1.0325	FILL	15 EEX 3	× ×	×	× ×	× × ×

You see the interest rate is kept as a constant on the stack, so the accumulated capital value computation becomes a simple series of × strokes. Each such

[11] In fact it is worth mentioning when automatic stack lift is disabled since this is under fixed conditions only and occurs far rarer.

[12] In the following, we shall turn to using plain text for numeric input for space reasons, unless mentioned otherwise.

multiplication consumes ***x*** and ***y*** for the new result put in **X**, followed by ***z*** dropping into **Y**, and ***t*** copied into **Z**.

Debt calculations are significantly more complicated – so avoid debts whenever possible! In the long run, it is better for you and the economy. Nevertheless, you can cope with those calculations using your WP 34S as well (see further below).

In addition to the stack operations [ENTER↑], [x⇄y], [R↓], LASTx, CLSTK, [R↑] and [x⇄] – known for decades and found within the blue frame on the keyboard – your *WP 34S* features [FILL], DROP, RCLS, STOS, SSIZE4, SSIZE8, SSIZE?, Y⇆, Z⇆ and T⇆. See the *IOP* below for more information about all these commands.

For the first time ever in a calculator, your *WP 34S* offers you a choice of four or eight stack levels. Thus, the fate of stack contents depends on the particular operation executed, its domain and the stack size chosen. Real functions in a four-level stack work as known for decades. In the larger stack of your *WP 34S* everything works alike – just with more levels for intermediate results. Turn overleaf for details.

	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 = 44.4*	*33.3*	*11.1*	*44.4*	*44.4*	*11.1*	*33.3*	*33.3*	*44.4*	*44.4*
	Z	*z = 33.3*	*22.2*	*11.1*	*44.4*	*33.3*	*44.4*	*22.2*	*22.2*	*33.3*	*44.4*
	Y	*y = 22.2*	*11.1*	*11.1*	*33.3*	*11.1*	*33.3*	*11.1*	*11.1*	*22.2*	*33.3*
	X	*x = 11.1*	*11.1*	*11.1*	*22.2*	*22.2*	*22.2*	*44.4*	*last x*	*123.21*	*2*
With **8** stack levels	**D**	*d = 88.8*	*77.7*	*11.1*	*88.8*	*88.8*	*11.1*	*77.7*	*77.7*	*88.8*	*88.8*
	C	*c = 77.7*	*66.6*	*11.1*	*88.8*	*77.7*	*88.8*	*66.6*	*66.6*	*77.7*	*88.8*
	B	*b = 66.6*	*55.5*	*11.1*	*77.7*	*66.6*	*77.7*	*55.5*	*55.5*	*66.6*	*77.7*
	A	*a = 55.5*	*44.4*	*11.1*	*66.6*	*55.5*	*66.6*	*44.4*	*44.4*	*55.5*	*66.6*
	T	*t = 44.4*	*33.3*	*11.1*	*55.5*	*44.4*	*55.5*	*33.3*	*33.3*	*44.4*	*55.5*
	Z	*z = 33.3*	*22.2*	*11.1*	*44.4*	*33.3*	*44.4*	*22.2*	*22.2*	*33.3*	*44.4*
	Y	*y = 22.2*	*11.1*	*11.1*	*33.3*	*11.1*	*33.3*	*11.1*	*11.1*	*22.2*	*33.3*
	X	*x = 11.1*	*11.1*	*11.1*	*22.2*	*22.2*	*22.2*	*88.8*	*last x*	*123.21*	*2*

Using the stack, *RPN* makes all parentheses like {, [, (,),], or } completely unnecessary in calculations. There is no operator precedence. Here is another **example** showing a slightly more complicated formula and the keystrokes used for solving it:

$$\sqrt{\frac{1+\left|\left(\frac{30}{7}-7.6\times 0.8\right)^{4}-\left(\sqrt{5.1}-\frac{6}{5}\right)^{2}\right|^{0.3}}{\left\{sin\left[\pi\left(\frac{7}{4}-\frac{5}{6}\right)\right]+1.7\times(6.5+5.9)^{3/7}\right\}^{2}-3.5}}$$

7 ENTER 4 / 5 ENTER 6 / − π × SIN	$sin\left[\pi\left({}^{7}/_{4}-{}^{5}/_{6}\right)\right]$
6.5 ENTER 5.9 + 3 ENTER 7 / y^x 1.7 × +	$\{sin\,[\,\ldots)^{3/7}\}$
x^2 3.5 −	complete denominator
7.6 +/− ENTER .8 × 30 ENTER 7 / + 4 y^x	$(30/7-7.6\times 0.8)^4$
6 ENTER 5 / 5.1 √x − x^2	$\left(\sqrt{5.1}-6/5\right)^2$
− |x| .3 y^x 1 +	complete numerator
x≷y / √x	complete result (0.37)

Even the solution of this formula requires only four stack levels as indicated by the colors above. Note there are no *pending operations* – each operation is executed individually, one at a time, allowing perfect control of each and every intermediate result.

Calculating such a formula from inside out stays a wise strategy. If you had started with the numerator of that sample formula straight ahead, you would need five levels for its complete solution. With eight levels, however, you will be on the safe side even with the most advanced equations you compute in your life as a scientist or engineer. **Let your *WP 34S* do the arithmetic while you do the mathematics!**

Error recovery even in long calculations is very easy since supported by LASTx in *RPN*: your *WP 34S* loads ***x*** into the special register **L** (for Last ***x***) automatically every time just before a function is executed.

If you have erroneously called a wrong one-number function, just invert it. Generally, the inverses are placed next to the original operations on the keyboard. If e.g. – instead of pressing SIN above – you hit COS, just call COS^{-1} to undo that error, restoring the stack exactly as it was before, then continue calculating with SIN as you would have done without that mistake.

The procedure for two-number functions requires three steps most times:

Example: Assume – while you were watching an attractive fellow student or collaborator – you pressed × accidentally instead of / in the second last step of the example on page 17. Murphy's law! Do you have to start the calculation all over now? No, that error is easily undone by the following three steps:

RCL L recalling the complete numerator, the last content of **X** before the error,

/ undoing the erroneous operation by executing its inverse,

RCL L regaining the stack exactly as it was before the mistake.

Now simply continue calculating / $\sqrt{x}$ (as you would have done without that mistake) and you will get the correct complete result [13].

There are also a few three-number real functions featured by your *WP 34S* (e.g. →DATE and %MRR) replacing ***x*** by the result **f(*x*, *y*, *z*)** . Then ***t*** drops into **Y** and so on, and the content of the top stack level is repeated twice.

Some real functions (e.g. DECOMP or 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 numbers will be pushed on the stack, taking one level per real number.

---

[13] This works for /, + and − as well. Advanced functions may require more effort for error recovery:
E.g. an erroneous y^x needs RCL L ENTER↑ R↓ 1/x y^x R↑ to restore the stack as it was before (except its top level). An erroneous LOG_x requires RCL L ENTER↑ R↓ x≷y y^x R↑ instead.
More complex operations like Π, COMB, PERM, etc. are easier recalculated than inverted.
In recovering from such errors, you will lose the previous content of the top stack register at least – another reason for choosing an 8-level stack so that you do not have to worry about such losses except in extremely rare special cases.

Calling Commands

Your *WP 34S* features more than 600 different commands. 168 labels are printed on the keyboard. So how do you learn about the other commands? And when you know their names, how can you call them?

The answer to the first question is most easy: read! The *IOP* below contains everything.

The answer to the second question is less obvious, but easy as well: the 'hidden commands' are stored in catalogs. Remember labels underlined point to such collections of commands. You will find such labels on your *WP 34S* on the slanted front of some keys. For example, h + 3 points to X.FCN, the largest catalog provided.

> **Example**: Assume you want to call DECOMP – it was mentioned in previous paragraph. You went back to the *Table of Contents* above, looked for the *Index of Operations*, then jumped directly to the letter **D** therein and browsed to get the necessary information where DECOMP lives and what it does. OK, enter 0.375 and let DECOMP work – calling it is even easier than looking it up:
>
> You have read DECOMP is stored in X.FCN.
>
> So just key in … and your WP 34S displays …

X.FCN	↕ ³√x	being the first command stored in X.FCN.
D	↕ DATE	for obvious reasons.
▼	↕ DATE→	
▼	↕ DAY	
▼	↕ DAYS+	
▼	↕ DECOMP	voilà! Now execute it …
XEQ	y/x = 8.	
x⇄y	3.	since 0.375 = 3 / 8 . Expected that. But …
0.46875	0.46875	what is this?
X.FCN	↕ DECOMP	oh, good, that function is memorized !
XEQ	y/x = 32.	
x⇄y	15.	meaning 0.46875 = 15 / 32 .

As ▼ browses forwards, ▲ does backwards. There is a more elegant method for calling cataloged commands described further below, but this here will do for the time being.

Some Special Real Functions: Statistical Distributions, Probabilities etc.

Besides the basic functions [%] and [Δ%], you will find a lot of statistical commands embedded in your *WP 34S*, going far beyond the Gaussian distribution. They are all concentrated in the light green frame mentioned above. Many preprogrammed operations are implemented here for the first time ever in an *RPN* calculator – we packed-in everything we always had missed.

The labels below [5] and following to the right cover sample statistics. The five labels at left deal with general probability.

[!], [Cy,x], and [Py,x] stand for ***x!***, COMB, and PERM as introduced above.

The shifted functions of [4] cover statistical distributions. All these functions have a few features in common:

- Discrete statistical distributions (e.g. Poisson, Binomial) are confined to integers. Whenever your *WP 34S* sums up a probability mass function (*pmf* [14]) $p(n)$ to get a cumulated distribution function (*cdf*) $F(m)$ it starts at $n = 0$. Thus,

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

- Whenever your *WP 34S* integrates a function, it starts 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

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

[14] 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* gives 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* gives 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.

BEWARE: This is a very rudimentary sketch of this topic only – turn to good 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".

- 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. Thus, your *WP 34S* also computes Q for each distribution, independent of P.

- On your *WP 34S*, with an arbitrary *cdf* named ***XYZ*** you will find the name

  ***XYZ*$_u$** for its *error probability* (also known as upper tail probability), if applicable,

  ***XYZ*$^{-1}$** for the inverse of the *cdf* (the so-called *quantile function* or *qf*), and

  ***XYZ*$_P$** for the *pdf* or *pmf*.

  This naming convention holds for **Binom**ial, **Cauch**y (a.k.a. Lorentz, Breit-Wigner), **Expon**ential, Fisher's **F**, **Geom**etrical, **LogNorm**al, **Logis**tic, **Norm**al, **Poiss**on, Student's **t**, and **Weibu**ll distributions. Chisquare and Standard Normal (Gaussian) distributions are named differently. See the *catalog PROB* and the respective entries in the *IOP*.

There is also a wealth of commands for sample and population statistics featured, applicable in one or two dimensions. All these are located to the right of key 4 on the keyboard. After clearing the summation registers by CLΣ, use Σ+ (or the shortcut Σ+ top left on the keyboard) to accumulate your experimental data – typically counted or measured values – as on the *HP-42S* etc. (weighed data require the weight in **Y**, pairs of data or coordinates of data points must be provided in **X** and **Y**). Σ− is provided for easy data correction.

Regarding the analysis functions featured, you find the arithmetic mean $\bar{x}$ and the standard deviation s handy on the keyboard, as well as a forecasting function $\hat{y}$ and the correlation coefficient r (see CORR) for four different regression models (linear, exponential, logarithmic, and power – see the commands LINF, EXPF, LOGF, POWERF). Many more functions (like standard errors, covariances, means and standard deviations for weighted data, standard deviations for populations, curve fitting parameters, geometric means and scattering factors) are in *STAT*, and all the accumulated data are in *SUMS* – just look them up in these catalogs and then check the respective entries in the *IOP*.

To get an idea of the possibilities provided and some constraints inherent to statistics, see the two sample applications shown here:

> **Application 1**: Assume you drew a *sample* of nominally equal specimens and want to calculate the confidence limits for the 'true value'. That can be done easily based upon analysis of that sample, provided you know the applicable distribution. Besides specifying a suitable confidence level (e.g. 95%), you must know your objective:
>
> - Do you want to know the upper limit, below which the 'true value' will lie with a probability of 95%? Then take the applicable sample mean value and 0.95 as arguments of the *qf* to get said limit – but remember there is an inevitable chance of $100\% - 95\% = 5\%$ that the 'true value' will be greater than that calculated limit. And 5% is no less than one in twenty!

- Do you want both upper and lower limits confining the 'true value'? Then there will be an inevitable chance of 5% / 2 = 2.5% that said value will be less than the lower limit and an equal chance that it will be greater than the upper limit. So take 0.025 and 0.975 as the arguments in two subsequent calculations using the *qf* to get both limits below and above the sample result.

These chances [15] are an inevitable consequence of the fact that you know something about a *sample* only (being a limited number of specimens drawn from a population), but want or have to tell something about said total population. If you cannot live with these chances, do not blame statistics but collect more (or more precise) data instead.

Application 2: Assume you have taken a sample out of a process at day 1, then changed the process parameters, waited for stabilization, and now have taken another sample of same size at day 2. Being serious, you have thoroughly measured and recorded the critical value (e.g. a characteristic dimension) for each specimen investigated at both days. Now: do the results of both samples show any *significant difference*? The following simple three-step test is well established. It may easily save yourself some unwanted embarrassments in your next presentation or after your next publication [16] :

1. Let your *WP 34S* compute $\bar{x}$ and the *standard error* s_E for both samples, then their *normalized distance* $d = \dfrac{|\bar{x}_1 - \bar{x}_2|}{\sqrt{s_{E1}^2 + s_{E2}^2}}$. Assume you are working with four stack levels still, this calculation could look like the following:

STAT S ▼ ÷ SERR	
XEQ	returns both standard errors in **X** and **Y**.
x^2 x≷y x^2 + $\sqrt{x}$	this is the complete denominator now.
$\bar{x}$ − |x| x≷y / STO D	and this is d.

   Also provide the *degrees of freedom* for the next two steps:

SUMS ÷ nΣ	
XEQ	recall the number of specimens measured.
1 −	calculate the *degrees of freedom*
STO J	and store them where they belong (learn more about the registers of your *WP 34S* in *next section*).

---

[15] These chances are also called '*probabilities of a type I error*' or '*probabilities of an error of the first kind*'. By the way, '*confidence limit*' translates to German "*Grenze des Vertrauensbereichs*", '*confidence level*' to „*Vertrauensniveau*", and '*type I error*' to "*Fehler 1. Art*".

[16] This test goes back to *DGQ* (*Deutsche Gesellschaft für Qualität*). It assumes your data are drawn from a Gaussian process, which is frequently the case in real life (but needs to be checked). Note the term '*significant*' is well defined in statistics – this definition may deviate from common language.
Generally, standard confidence limits and levels, also those defined for indicating *significant differences*, may depend on the country or industry you are working in. Be sure to check the applicable valid standards.

2. Let your *WP 34S* calculate the critical limit t_{cr} of *Student's t* for f *degrees of freedom* and a probability of 97.5% now:

.975	0.975	
PROB U ▲	÷ t-1(P)	as mentioned above, the requested *qf* lives in catalog PROB.
XEQ		executes this function to get t_{cr}.

If $d < t_{cr}$ then the test indicates the difference between both samples being due to random deviations only. Congratulations – you have got a robust process regarding the parameters you changed.

3. Let your *WP 34S* compute a new critical limit t_{cs} for f and 99.5%:

.995	0.995	
PROB	÷ t-1(P)	note this function shows up immediately when opening the catalog again.
XEQ		get t_{cs}.

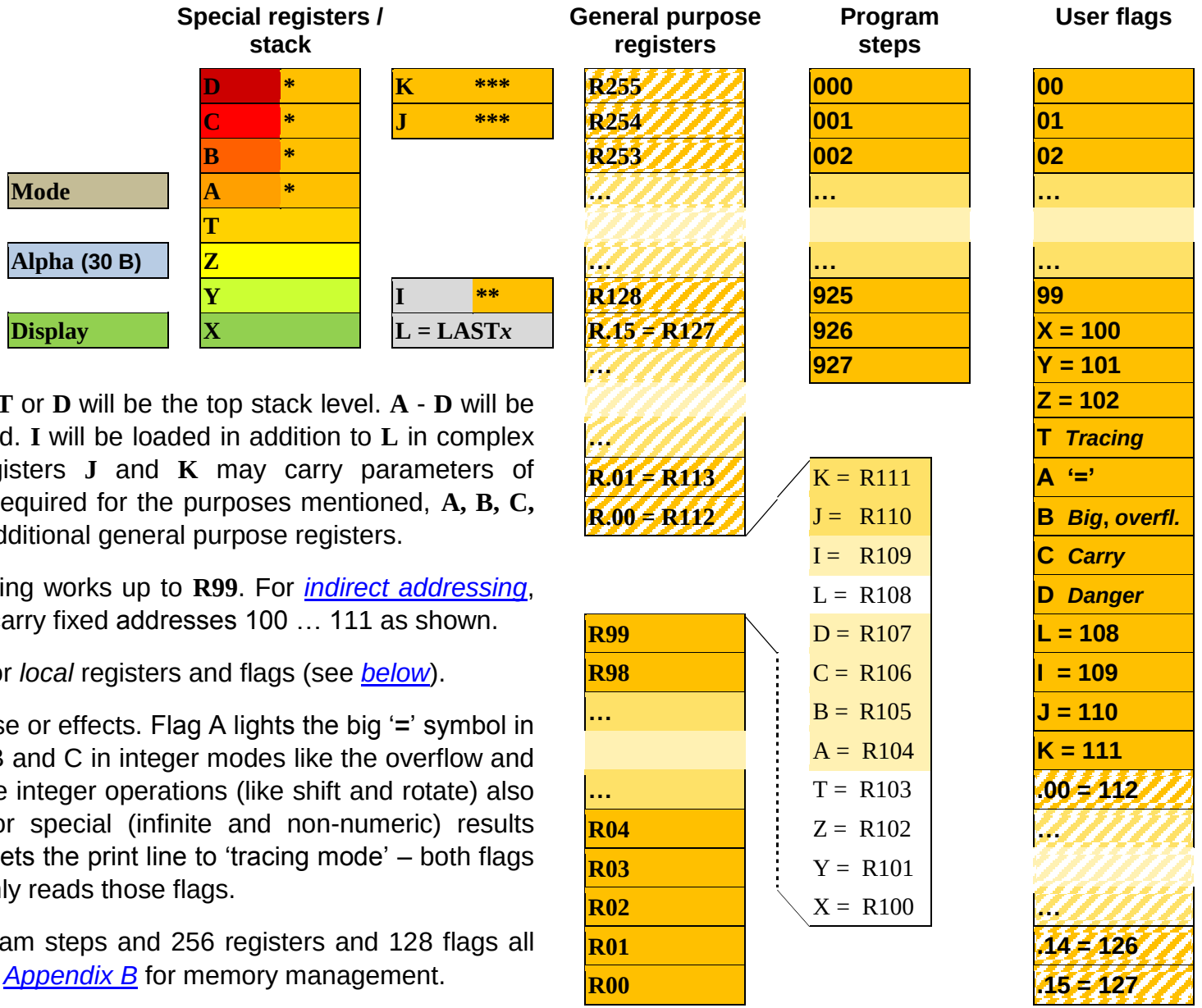
If $d \geq t_{cs}$ then the test indicates a *significant difference* between both samples. Congratulations – your parameter change caused an effect.

For $t_{cr} \leq d < t_{cs}$, however, you cannot decide based on the information provided – your samples may contain too little data or your measurements were not precise enough – so you better stay silent or mumble something like "investigation in progress".

We strongly recommend you turn to a good statistics textbook for more information about statistical methods, the terminology used, and the mathematical models provided.

MEMORY AND ADDRESSING

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 addition to **L** in complex calculations (see *below*). Registers **J** and **K** may carry parameters of statistical distributions. Unless required for the purposes mentioned, **A, B, C, D, I, J,** and **K** are available as additional general purpose registers.

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

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

Some user flags have special use or effects. Flag A lights the big '**=**' symbol in display. The system sets flags B and C 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 allows for special (infinite and non-numeric) results without getting an error, flag T sets the print line to 'tracing mode' – both flags are user-settable, the system only reads those flags.

Note you will not get 928 program steps and 256 registers and 128 flags all together at the same time – see *Appendix B* for memory management.

Keyboard Reassignment in Transient Alpha Mode (α_T)

This mode is entered during input processing in memory addressing, e.g. during storing and recalling, in comparisons, and also by two browsers, regardless of the mode set before. Examples are shown *below*. In the following, the keyboard will be reassigned to work as shown here:

This kind of picture is called a *virtual keyboard* since it deviates from the physical (or real) one of your *WP 34S*. In such a picture, dark red background is used to highlight changed key functionality. White print denotes primary functions also on virtual keyboards, like the top left key entering the letter A in α_T mode directly. On the other hand, what is printed white on your physical *WP 34S* is called a default primary function.

Note all keys are primary in α_T mode – no shift keys needed. This allows for fast and easy input of a limited character set. So you can reach all register addresses available with a minimum of keystrokes.

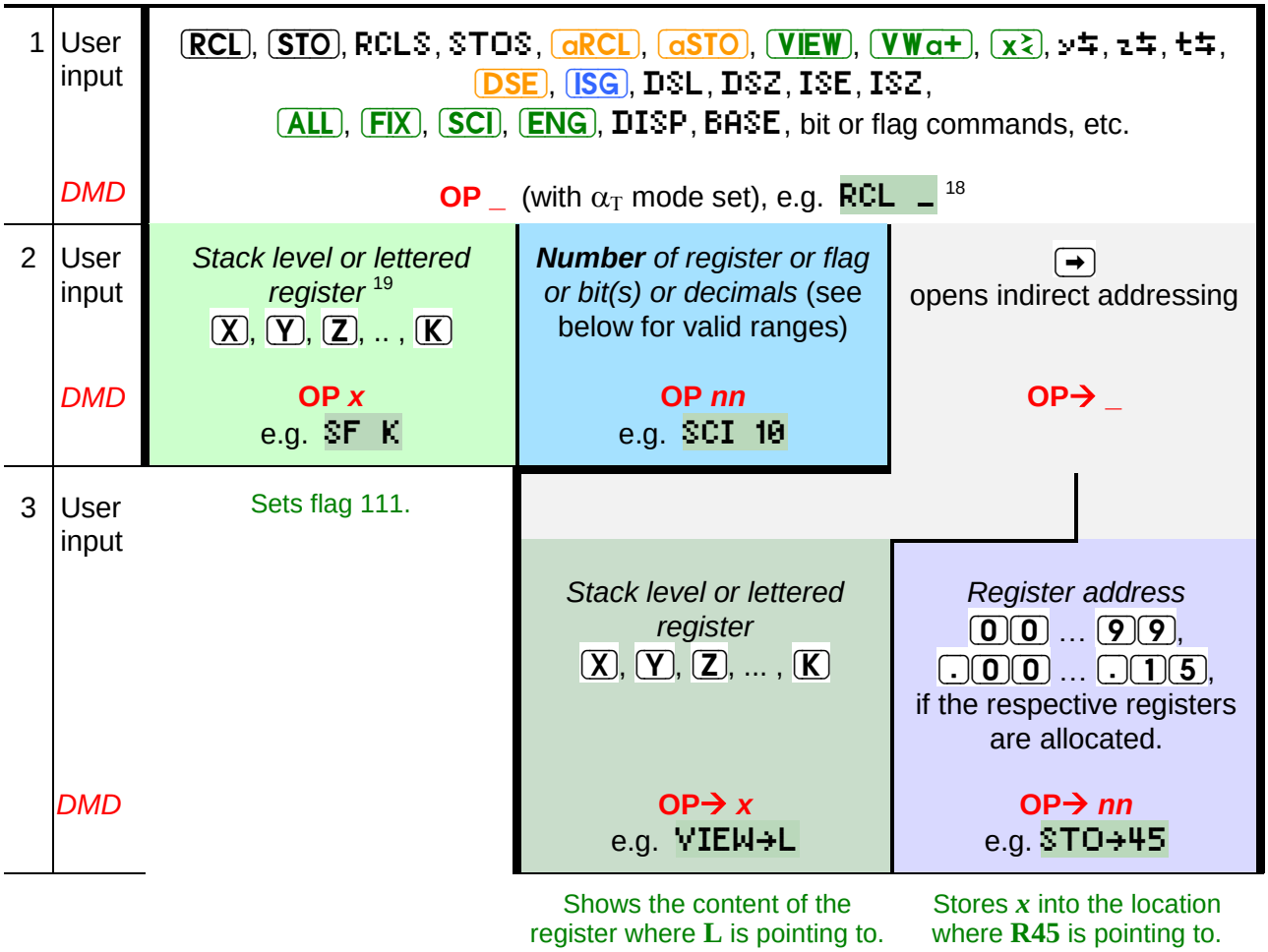
Special rules apply for **T**, **X**, and **Z** here – see *below*

α_T mode will be terminated (returning to the mode set before) as soon as sufficient characters are put in for the respective step. You may delete pending input character by character using ← or just abort the pending command by EXIT – the latter will leave α_T mode immediately.

Addressing Real Numbers

1	User input Dot matrix display	[x= ?], [x≠ ?], x< ?, x≤ ?, x≥ ?, x> ?, or x≷ ? **OP _ ?** (with α_T mode set), e.g. x≷_?			
2	User input *DMD*	[0] or [1] **OP *n* ?** e.g. x≤0?	*Stack level or lettered register* [Y], [Z], ..., [K] **OP? *x*** e.g. x≷? Y	[ENTER↑] [17] leaves α_T mode **OP? _**	[→] opens indirect addressing **OP?→ _**
3	User input *DMD*	Compares ***x*** with the real number **0**.	Compares ***x*** with the number on stack level **Y**.	*Register address* [0][0] … [9][9], [.][0][0] … [.][1][5], if the respective registers are allocated. **OP? *nn*** e.g. x≠? 23 Compares ***x*** with the number stored in **R23**.	See next page for more about indirect addressing.

[17] You may skip this keystroke for register addresses >19 or local registers. The latter start their address with a [.] – see the section about programming and *Appendix B* below.



Type	Number range[20] (some more registers and flags carry letters)	
Registers	0 … 99 for direct addressing of global numbered registers .0 … .15 for direct addressing of local registers 0 … 255 for indirect addressing (≤111 without local registers)	} upper limits depend on allocation
Flags	0 … 99 for direct addressing of global numbered flags .0 … .15 for direct addressing of local flags if allocated 0 … 127 for indirect addressing (≤111 without local flags)	
Decimals	0 … 11	
Integer bases	2 … 16	
Bits	0 … 63, word sizes up to 64 *bits*	

[18] For RCL and STO, any of +, −, ×, /, ▲, or ▼ may precede step 2, except in RCL MODE and STO MODE.
Note ENG ENTER↑ calls ENGOVR and SCI ENTER↑ calls SCIOVR. See the index of operations.

[19] Exceptions: ALL, FIX, SCI, ENG, DISP, and BASE accept lettered registers in indirect addressing only. Else, specifying register X as well as RCL T, STO T, RCL× T, STO× T, RCL Z, STO Z, RCL+ Z and STO+ Z require an ENTER↑ heading the letter, e.g. STO + ENTER↑ Z for the latter.

[20] For short numbers, you may key in e.g. 5 ENTER↑ instead of 0 5.

Advanced Calculations: Real Matrices and Vectors

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

If you know of matrices, however, 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 in the form of keystroke programs. I.e. they represent the basic linear algebra subprograms of the *WP 34S* matrix support. On the other hand, your *WP 34S* also provides functions for computing determinants or 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 rows and

cc the number of columns it features. Thus this 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 they 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 you 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 defaults to ***rr*** so a square matrix is assumed. For **example**, 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 for such a task.

See the *IOP* and the *catalog MATRIX* for all commands featured.

> **ATTENTION:** Your *WP 34S* cannot know whether a particular real number is a matrix descriptor or a plain number. It is your task to take care of that.

A *vector* may be regarded as a special case of a matrix featuring either one row or one column only. Thus, a vector descriptor looks like `bb.01cc` or `bb.rr01`. Library routines are readily provided for 3D vector calculus.

If you just want to do vector operations in 2D, there are simple alternatives (known for long from earlier calculators) to full-fledged descriptor controlled computations: enter the Cartesian components of each vector in **X** and **Y** (e.g. by converting its polar

components into Cartesian ones by [→R], if necessary) and choose one of the following alternative opportunities:

1. use Σ+ or Σ– and recall the result via SUM, or
2. calculate in *complex domain* (see next paragraph). Therein, vector multiplication is possible, too, using the commands ^{C}DOT or ^{C}CROSS.

Turn to a good textbook covering *linear algebra* for more information.

More Advanced Calculations: Complex Domain

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

If you know of complex numbers, however, 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 in your *WP 34S*).

All functions operating on complex numbers require them in Cartesian coordinates exclusively on your *WP 34S*. Each such number takes two adjacent registers[21]: the lower one for its real part and the higher one for its imaginary part. You may use [→P] to convert $x_c = x + i \cdot y$ to its polar equivalent $x_c = r \cdot e^{i\varphi}$ any time you like – just remember reversing this by [R←] before starting a calculation..

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

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

Where **one-number** real functions replace x by the result **f**(x), one-argument complex functions replace x by the real part and y by the imaginary part of the complex result ^{C}**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, ^{C}**+/–**, ^{C}Γ, the logarithmic and exponential functions with bases 10, 2, and e, as well as hyperbolic and trigonometric functions and their inverses.

Two-number real functions replace x by the result **f**(x, y) as shown above. In complete analogy, two-argument complex functions replace x by the real part and y by the imaginary part of the complex result ^{C}**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 the basic arithmetic operations in complex domain as well as ^{C}LOG$_X$, ^{C}y^x, ^{C}β(x,y), and ^{C}||. Turn to the stack diagrams on the next page for further details.

---

[21] The *HP-42S* supported a special data type for complex numbers. This is not viable using the hardware of the *HP-30b*, however. See *Appendix F* for the reasons.

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

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	$\mathbf{Im}(y_c) = \mathbf{Im}(t_c)$	$\mathbf{Im}(x_c)$		$y_c = t_c$	$\mathbf{Im}(x_c)$	x_c		x_c	$y_c = t_c$	$y_c = t_c$
	Z	$\mathbf{Re}(y_c) = \mathbf{Re}(t_c)$	$\mathbf{Re}(x_c)$			$\mathbf{Re}(x_c)$					
	Y	$\mathbf{Im}(x_c)$	$\mathbf{Im}(x_c)$		y_c	$\mathbf{Im}(y_c)$	y_c		*last* x_c	$\mathbf{Im}((x_c)^2)$	$\mathbf{Im}(y_c / x_c)$
	X	$\mathbf{Re}(x_c)$	$\mathbf{Re}(x_c)$			$\mathbf{Re}(y_c)$				$\mathbf{Re}((x_c)^2)$	$\mathbf{Re}(y_c / x_c)$
With **8** stack levels	D	$\mathbf{Im}(t_c)$	z_c	x_c	t_c	t_c	x_c	z_c	z_c	t_c	t_c
	C	$\mathbf{Re}(t_c)$									
	B	$\mathbf{Im}(z_c)$	y_c	x_c	t_c	z_c	t_c	y_c	y_c	z_c	t_c
	A	$\mathbf{Re}(z_c)$									
	T	$\mathbf{Im}(y_c)$	x_c	x_c	z_c	x_c	z_c	x_c	x_c	y_c	z_c
	Z	$\mathbf{Re}(y_c)$									
	Y	$\mathbf{Im}(x_c)$	x_c	x_c	y_c	y_c	y_c	t_c	*last* x_c	$(x_c)^2$	y_c / x_c
	X	$\mathbf{Re}(x_c)$									

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

Note you can use complex domain for 2D vector algebra as well. The functions ^{C}ABS, ^{C}+, ^{C}–, ^{C}CROSS, and ^{C}DOT wait for you.

After pressing CPX, your *WP 34S* allows for the complex operations shown on this virtual keyboard (but STOPW represents an application in real domain, see *below*).

Note that a second CPX, a ⬅, or EXIT directly after CPX will just cancel the complex prefix, so you return to the default keyboard assignments.

Constants in complex domain and such calculations occupy two registers like all other complex numbers!

Complex constants are simply entered as ***imaginary_part*** ENTER↑ ***real_part*** .

Pure real constants, identified by a zero imaginary part, are easily put in for complex calculations as follows:

0 ENTER↑ ***real_constant*** , or alternatively

CPX ***n*** for integers $0 < n \le 9$ only.

In programming, CPX h CONST # ***n*** with $0 \le n \le 256$ will save steps.

Example: The real number π may be loaded via 0 ENTER↑ π or CPX π – both will result in $y = 0$ and $x = \pi$.

Pure imaginary constants, on the other hand, having a zero real part, are entered thus:

imaginary_constant ENTER↑ 0 .

Example: The complex unit i may be loaded via 1 ENTER↑ 0 or CPX . – both will result in $y = 1$ and $x = 0$.

Compare the stack mechanics shown above.

Addressing Complex Numbers

1	User input *DMD*	CPX x=? or x≠? **OP _** (with α_T mode set) e.g. ˢx=_?			
2	User input *DMD*	0 or 1 **OP *n* ?** e.g. ˢx=0?	*Stack level or lettered register* Z, A, C, L, or J **OP? *x*** e.g. ˢx≠? Z	ENTER↑ [22] leaves α_T mode **OP? _**	→ opens indirect addressing. **OP?→_**
3	User input *DMD*	Compares $x + i\,y$ with the real number **0**.	Compares $x + i\,y$ with $z + i\,t$.	*Register address* 00 … 98, .00 … .14, if the respective registers are allocated **OP? *nn*** e.g. ˢx≠? 26	See next page for more about indirect addressing.
				Compares $x + i\,y$ with $r26 + i\,r27$.	

[22] You may skip this keystroke for register numbers >19 or local registers. The latter start with a . – see the section about programming and *Appendix B* below.

1	User input *DMD*	[CPX] [RCL], [STO], or [x≷] **OP _** (with α_T mode set) e.g. ᶠRCL _ [23]		
2	User input *DMD*	*Stack level or lettered register* [Z] [24], [A], [C], [L], or [J] **OP *x*** e.g. ᶠRCL L	*Register address* [0][0] .. [9][8], [.][0][0] … [.][1][4], if the respective registers are allocated. **OP *nn*** e.g. ᶠSTO 18	[➡] opens indirect addressing, generally working as *in real domain*. **OP→_**
3	User input *DMD*	This is ᶜLASTx – the real part is recalled from register **L** to **X**, the imaginary part from **I** to **Y**.	*Stack level or lettered register* [X], [Y], ... , [K] **OP→ *x*** e.g. ᶠx⇆→Z	*Register address* [0][0] … [9][9], [.][0][0] … [.][1][5], if the respective registers are allocated. **OP→ *nn*** e.g. ᶠSTO→45
			Swaps ***x*** with the content of the register where **Z** is pointing to, and ***y*** with the content of the next one.	Stores $x + i\,y$ into 2 consecutive registers, starting with the one where ***r45*** is pointing to.

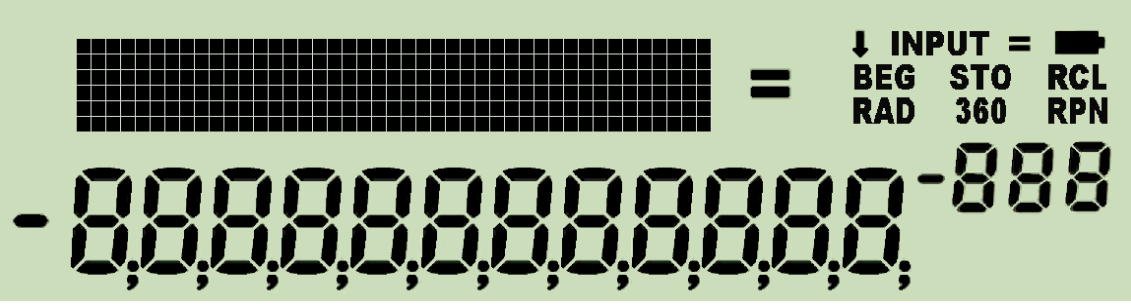
ATTENTION: A complex operation will always affect a pair of registers: the one specified and the one following this. To avoid ambiguity we strongly recommend storing complex numbers with their real parts at even register addresses always.

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

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

DISPLAY AND MODES

The display of your *WP 34S* shows 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 on the top right side (except the big '=') are called *annunciators*, and are for indicating modes.



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

The dot matrix section above is used for

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

   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, h, and CPX are shown until they are resolved. → goes with g by default. If you pressed any such prefix erroneously, recovery is as easy as follows:

   - f f = g g = h h = CPX CPX = → → = NOP
   - CPX → = → CPX = NOP
   - 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 edit or cancel such pending operations by ← or EXIT as described *below*.

If two or more requests compete 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 either in the dot matrix or in the exponent section indicate most modes and system states:

Indicator	Set by	Cleared by	Explanation, remarks	Sets mode
=	SF A	CF A	Flag A may be used e.g. to extend ***alpha*** by this character (see *below*).	
↓	↑	↑	Lower case letters will be entered in alpha mode (see *below*).	
INPUT	α, αON	ENTER↑, αOFF, EXIT	Alpha mode (see *below*)	α
=	*look at right*		Steady while serial I/O is in progress (see *Appendix A*). Flashing when timer is running (except in *stopwatch*).	
▬	battery low	battery voltage > 2.5 *V*	Your *WP 34S* will shut off when voltage drops below 2.1*V*.	
BEG	*look at right*		Program pointer at step 000	
STO	P/R	P/R, EXIT	Programming mode	PR
RCL	*look at right*		Flashes while a program is running.	
RAD	RAD	DEG, GRAD	Angular mode (see next page)	
360	DEG	GRAD, RAD		
RPN	almost every command	a *temporary message*	See *below* for handling of such messages in general.	
b_ _	2	any other BASE setting a b/c, d/c and FRACT will set fraction mode (i.e. FRC). ALL, FIX, SCI, ENG, H.MS, →H.MS, H.d, and TIME will set default floating point decimal mode (i.e. DECM).	Binary integer mode	2
3_ _	BASE 3		Respective integer modes (see *below* for all of these modes)	3
…	…			…
7_ _	BASE 7			7
o_ _	8		Octal integer mode	8
9_ _	BASE 9		Integer mode of base 9	9
d_ _	10		Decimal integer mode	10
-1_ _	BASE 11		Respective integer modes (see *below* for all of these modes)	11
…	…			…
-5_ _	BASE 15			15
h_ _	16		Hexadecimal integer mode	16

Indicator	Set by	Cleared by	Explanation, remarks	Sets mode
_c_	carry, SF C	CF C, carry clear	Indicate the respective bits set in integer modes (see *below*).	
__o	overflow, SF B	CF B		
			The indicators following below are all lit in the dot matrix exclusively!	
▫	CPX	see *above*	Transient signal of prefix pending	
C	complex result	else	Indicates a complex result returned by the last operation (see *above*).	
D	DBLON	DBLOFF	See *Appendix H*.	
f	f	see *above*	Transient signal of prefix pending	
g	g			
g →	→			
G	GRAD	DEG, RAD	Angular mode (see next page)	
h	h	see *above*	Transient signal of prefix pending	
M.DY	M.DY, SETUS	any other date or region setting	Date modes (see next page)	M.DY
Y.MD	Y.MD, SETJPN, SETCHN			Y.MD
÷	entering a catalog or browser	leaving it	See *below* for more about catalogs and browsers.	α

Defaults D.MY and DECM are not indicated. Radix marks and separators are seen in the numeric output immediately, time modes (12h / 24h) in the time string. The numeric format of fraction mode is unambiguous as well. Check the examples shown *below*.

Some mode and display settings may be saved and recovered collectively by STOM and RCLM. These are stack depth, display contrast, and complete decimal display settings, trig mode, choices for date and time display, parameters of integer and fraction mode, curve fitting model, rounding mode, and precision selected. STOM stores this information in the register you specify. RCLM recalls such a register content and sets the calculator modes accordingly.

> **ATTENTION:** Ensure that you actually recall mode data – else your *WP 34S* may be driven into very strange settings and it may cost you considerable effort to recover from that unless you find your previous modes stored elsewhere! See *Appendix H*.

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

Commands Returning Specific Displays

Some common commands use the display in a special way. The respective operations are listed below. Three of them (and more mentioned further below) present *temporary messages* as defined here:

> Whenever anything different from the actual contents of **X** in current mode is displayed or any additional information is shown in the dot matrix, these extra data are considered being a *temporary message*. This is further indicated by the annunciator **RPN** turned off as mentioned *above*.
>
> If such extra data are displayed outside of a catalog or browser (see a separate section about these *below*), they will vanish with the next keystroke. Pressing EXIT or ← will just clear the *temporary message* returning to the normal display, any other key will be executed in addition.

Now here are the common commands delivering special information:

1. STATUS returns very useful information about current memory allocation and the space available. It continues showing the user flags set. See *below* for a detailed description.

2. **VERS** generates a *temporary message* similar to the one shown on page 1, so you know which version and build of the firmware is running on your *WP 34S*. If the quartz crystal and capacitors are built-in and the timer firmware is installed, '**T**' will trail the version number like here:

   If the IR-diode is built-in as well and the corresponding firmware is installed then a printer character will appear like on page 1. If just the basic firmware is installed instead, the build number will follow the version immediately. See *Appendix A* for more.

3. **ERR** and **MSG** display a *temporary message* like the corresponding error message. See *Appendix C* for more.

4. A few far-reaching commands (like CLALL, for example) will ask you for confirmation before executing. The question Sure? must then be answered by Y or N, i.e. pressing R/S or 8, respectively. Also EXIT or ← will be interpreted as N, any other input will be ignored.

Further specialties are more specific to particular modes of your *WP 34S* and are thus covered in the sections following.

FLOATING POINT MODES

Floating point modes cover the 'usual' numbers you calculate with: decimal real or complex numbers, fractions, measured values, times and dates. Some information applies exclusively to DECM (the startup default mode of the WP 34S) and its sub-modes.

Setting the Scene for Different Regions of this Planet

Set display preferences according to your region's practices at once using single dedicated commands:

Command	Radix mark[25]	Time	Date[26]	JG [27]	Three digit separators	Remarks
SETCHN	RDX**.**	24h	Y.MD	1949	E3OFF	Would require separators every four digits.
SETEUR	RDX**,**	24h	D.MY	1582	E3ON	Applies to South America, too. Also applies to Russia, Vietnam, Indonesia, and South Africa, but with deviating JG's.
SETIND	RDX**.**	24h	D.MY	1752	E3OFF	Would require separators every two digits over 10^5. Applies also to Pakistan and Sri Lanka.
SETJPN	RDX**.**	24h	Y.MD	1873	E3ON	
SETUK	RDX**.**	12h	D.MY	1752	E3ON	Applies also to Australia and New Zealand. 24h is taking over in the UK.
SETUSA	RDX**.**	12h	M.DY	1752	E3ON	

[25] See *http://upload.wikimedia.org/wikipedia/commons/a/a8/DecimalSeparator.svg* for a world map of radix mark use. Looks like an even score in this matter. Thus, ISO 31-0 allows either a decimal point or a comma as radix mark, and requires a narrow blank as separator of digit groups to avoid misunderstandings.

[26] See *http://upload.wikimedia.org/wikipedia/commons/0/05/Date_format_by_country.svg* for a world map of date formats used. The standard ISO 8601:2004 states 24h for times, Y.MD for dates.

[27] 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). Your *WP 34S* supports both 1582 and 1752. See the index of operations.

Displaying Floating Point Numbers

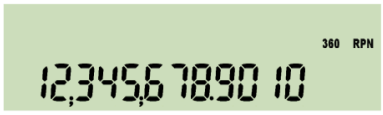
1. For **floating point decimal numbers**, startup default display format is ALL 00, SCIOVR, i.e. all digits will be displayed as long as they fit the display; it will switch to SCIentific (i.e. mantissa plus exponent) notation otherwise. There are two more numeric display formats, FIX and ENG. Their effects can be most easily demonstrated and distinguished using an **example**:

Input \ Format	ALL 00	FIX 4	SCI 4	ENG 4
107.12345678	107.12345678	107.1235	1.0712 2	107.12 0
1/x 2 ×	1.867004725311 -2	0.0187	1.8670 -2	18.670 -3

Within FIX, the radix mark will always stay at the FIXed position defined. The radix mark floats in the other notations, where e.g. 1.0712 2 represents $1.0712 \cdot 10^{2}$, while 18.670 -3 V would mean $18.67 \cdot 10^{-3}\ V = 18.67\ mV$. Within ENG, the exponent will always be a multiple of three – thus it is called the ENGineers notation.

As soon as a number is completely entered, 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, and additional marks may be chosen to separate thousands.

Assume the display set to FIX 4 again. Key in 12345678 . 901 ENTER↑, and you get



or


with E3ON [28].

Without these separators (i.e. E3OFF), the same number will look like this:

or


With ENG 3 and after +/-, you will get

or


When the last operation executed has returned a *complex* result, C is lit in the top left of the *LCD* pointing to the fact that you will find the result of this function in **X** and **Y**.

Floating point decimal numbers within $10^{-383} \le |x| < 10^{+385}$ may easily be entered directly. Within this range, your *WP 34S* calculates with 16 digits [29]. Values less than 10^{-398} are set to zero. For results $|x| \ge 10^{+385}$, errors 4 or 5 will appear (see *below*).

[28] These separators may also be beneficial in fraction mode described below.

2. ◀ and ▶ in default DECM show the full mantissa of x, i.e. all digits present internally (think a radix mark after the first digit), and the exponent like in SCIentific display format. All this is shown in one *temporary message*.

   For **example**, π 3 y^x ◀ returns 

.

3. **Fraction mode** works similarly to the one in the *HP-35S* or the *HP-32SII*. In particular, DENMAX sets the maximum allowable denominator (see the *IOP*). 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 and overflows will be displayed in the format set before fraction mode was entered.

   Example: Enter the following: and you will see:

   0 MODE D ▼▼▼ 

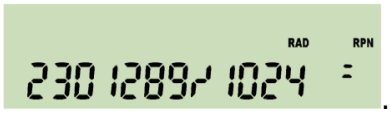
[30]

   XEQ

   47 . 40625 +/− a b/c 

   d/c 

[31].

   x^2 

.

   Now, enter a b/c for converting this into a *proper* fraction [32]. You 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 ◀ or ▶ (see the previous point) to unveil the integer part of this proper fraction as 2247.

---

[29] Even smaller numbers may be entered using a decimal mantissa, but you will lose one digit per factor of ten. The same happens if you divide 10^{-383} by 10 several times. At 10^{-398}, only one digit will be left. Divide it by 1.999 999 999 99 and the result will remain 10^{-398} in default rounding mode (and in RM 1, 2, 3, and 5). Divide it by 2 instead and the result will become zero. See RM and Appendix B for the reasons.

[30] This is picking a function from a catalog again.

[31] Note pure integers like 123 will be displayed as 123 0/1 or 123/1 in fraction mode, respectively, to indicate this mode.

[32] 'Proper fractions' cover "echte Brüche" (like ¾) and "gemischte Brüche" (like 2 ½) in German.

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 1/2	(decimal input)
. 1 . 2 ENTER↑	1/2	
. 1 2 ENTER↑	3/25	(decimal input)
1 . . 2 ENTER↑	1 0/1	(= 1 $^{0}/_{2}$! [33])

4. There are three **angular modes** featured: DEG, RAD, and GRAD [34]. And *degrees* (DEG) may be displayed in decimal numbers as well as in *hours*, *minutes*, *seconds* and hundredth of *seconds* (H.MS, see next point). Conversions are provided for going from one to the other:

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→	—

See the *IOP* for the commands printed on white background, and the *catalog of unit conversions* for those printed on light grey.

5. **For fitting** measured and accumulated data points with a regression curve, four mathematical models are provided as in the *HP-42S*. See the commands EXPF, LINF, LOGF, POWERF in the *IOP* below. The command BESTF will set your *WP 34S* to select the model resulting in the greatest absolute correlation coefficient (see CORR).

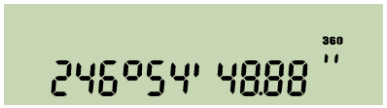
[33] For comparison, note the *HP-32SII* reads the last input here as ½ – which is, however, not consistent with its other input interpretations in fraction mode.

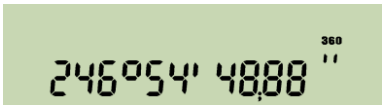
[34] This is confusing in German: DEGrees on your *WP 34S* mean “Grad”, while GRAD means “Neugrad”.

As shown in this **example**

the fit model applied is displayed temporarily after each command related to fitting (i.e. after CORR, COV, L.R., s_{XY}, x̂, ŷ). Like with all other auto-functionality, you should know what you are doing here.

6. In **H.MS display mode**, decimal numbers are converted and displayed in a format `hhhh°mm'ss.dd"` with the number of *hours* or *degrees* limited to 9,000. This *temporary message* may look like



for RDX**.** or 

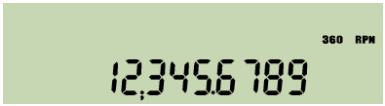
for RDX**,** .

For decimal times less than 5*ms* or 0.005 *angular seconds* but greater than zero, an `u` for underflow will be lit in the exponent section:

.

Note there are no leading zeroes in the *hours*, *minutes,* and *seconds* sections.

For times or angles exceeding 9,000, an `o` is shown in the exponent section signaling an overflow, and the value is displayed modulo 9,000.

For **example**: 

will become after [H.MS] 

until the next key is pressed.

7. **WDAY** returns a display looking like the following for an input of 13.01201 in default D.MY mode (equivalent to inputs of 2010.0113 in Y.MD or 1.13201 in M.DY):

.

Expect similar displays after DAYS+ [35].

[35] 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. We count on your understanding.

SOMETHING DIFFERENT: INTEGER MODES

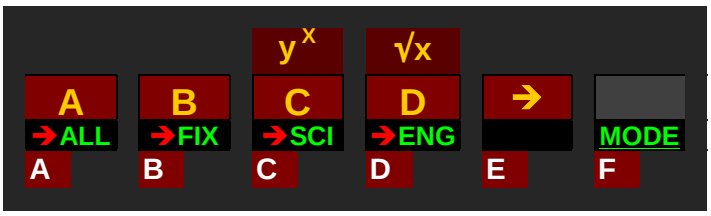
These modes are meant to deal with integers only – in input, output, and calculations. This is useful e.g. for computer logic and system programming – typical applications of an *HP-16C*. Your *WP 34S* contains all the functions of the *HP-16C* and more, and it allows for integer computing in fifteen bases from binary to hexadecimal (see overleaf).

The Virtual Keyboard in Integer Modes

In integer modes, functions like SIN make no sense for obvious reasons. Thus, for integer bases ≤ 10, the virtual keyboard of your *WP 34S* will look as shown left (where labels headed by a red arrow will leave integer modes when called, typically returning to default floating point decimal mode).

For base 16, on the other hand, primary functions of the top six keys will be reassigned automatically, becoming direct numeric input. So this row will then look virtually as shown below.

Wherever a default primary function is not primary anymore after reassignment, prefix f will allow for accessing it (e.g. f D will call √x here [36]). To ease operation, pressing any key (or a sequence of prefix(es) and a key) will display its present assignment in the top line for checking. If the last key is held down for > 0.5 seconds, the display will fall back to NULL and no operation will be executed.



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 from the keyboard – like e.g. 4 in binary – will be blocked.

[36] In such cases, operations printed golden on the key plate cannot be called anymore. This means for the key D, for example, we cannot access TAN in hexadecimal mode – no loss here, of course. Reassignments are generally chosen this way. – Note f D may call a program if defined.

Displaying Integers

In integer modes, the mantissa section of numeric display shows the integer in **X**. The exponent section is not needed for numeric display here but may transmit other information. Sign and first digit of the exponent indicate the base set:

Base	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Exponent starts with	b	3	4	5	6	7	o	9	d	-1	-2	-3	-4	-5	h

Carry and overflow – if set – show up as a `c` in the second or an `o` in the third digit of the exponent, respectively. See the table *above*. They behave and are treated like in the *HP-16C*.

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. Startup default is **2c**. These modes control the handling of negative numbers and are understood most easily with a little **example**:

Set your *WP 34S* to WSIZE 12, LZON. This setting allows seeing all 12 *bits* in one calculator display easily. Enter 147. Then turn to 1COMPL, BASE 2. You will see:

and – after +/− –


.

Forget the `1` top right for the moment – it will be explained later here. Note the low byte of our number is displayed larger than its top four bits for easy reading. Obviously +/− in **1c** inverts every bit, being equivalent to NOT here.

Return to the original number via +/− now, choose 2COMPL and you will get:

and – after +/− –


.

Note the negative number equals the inverse plus one in **2c**.

Now return again to the original number via +/−, choose SIGNMT and you will see:

and – after +/− –


.

Negating a number will just flip the top bit in **sm**.

Finally, return to the original number via +/−, choose UNSIGN and you will get:

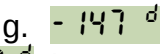
and – after +/− –


.

Note the second number looks like in **2c**, but in addition an overflow is set here.

This needs explanation, since changing signs has no meaning in **un** per definition [37], where the most significant bit adds magnitude, not sign, so the largest value represented by a 12-bit word is 4095 instead of 2047. Thus, +/− should be illegal here and result in no operation. Nevertheless, +/− was allowed and implemented in the *HP-16C*, so we follow this implementation for sake of backward compatibility.

Thus, pressing +/− will not suffice anymore for returning to the original number here; you must also clear the overflow flag by CF B explicitly (see *above*).

As you have seen, positive numbers stay unchanged in all those modes. Negative numbers, on the other hand, are represented in different ways. Therefore, taking a negative number in one mode and switching to another one will lead to different interpretations. The fixed bit pattern representing e.g.  in default **2c** will be displayed as in **1c**, in **sm**, and in **un**.

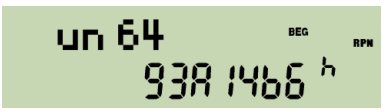
Keeping the mode (e.g. **2c** again) and changing the bases will produce different views of the constant bit pattern as well. You will notice that the displays for bases 4, 8, and 16 will look similar to those shown above, presenting all twelve bits to you, while in the other bases a signed mantissa will be displayed instead.

Compare for **example** the outputs

 for base 4 and

 for base 5 [38].

Let us look to bigger words now: For **example**, turn to UNSIGN, LZOFF, WSIZE 64, BASE 16, and enter 93A14B6. Then your *WP 34S* will display:

 with or 

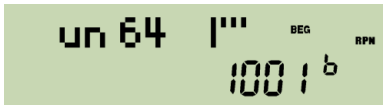
without separators selected (see SEPON and SEPOFF). This number will need 28 digits in binary representation, being 1001.0011.1010.0001.0100.1011.0110. Now choose SEPON, BASE 2, and the twelve least significant digits will be displayed initially. They show up together with an indication  that there are four display windows in total with the rightmost shown:

---

[37] This is clearly stated also in the *HP-16C Computer Scientist Owner's Handbook* of April 1982 on page 30. Unfortunately, however, they did not stick to this.

[38] This takes into account that bases 2, 4, 8, and 16 are most convenient for bit and byte manipulations and further close-to-hardware applications. On the other hand, the bases in between will probably gain most interest in dealing with different number representations and calculating therein, where base 10 is the common reference standard.

The least significant byte is emphasized as you know it from the example *above*. Press ◀ and you will get the more significant bytes (note the constant 4-bit overlap with the previous display here):



The last display shows the four most significant bits of this binary number as the indication `1'''` confirms.

If leading zeros were turned on (see LZON), there would be eight display windows (corresponding to eight bytes) here, with the four 'most significant' bytes containing only zeros.

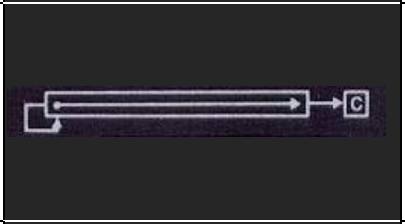
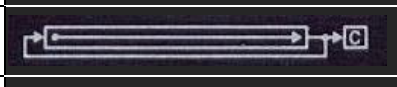
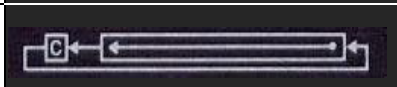
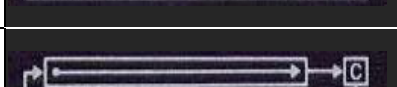
Note numeric input is limited to 12 digits in all integer bases.

Browsing a large integer in steps of eight digits is a specialty of binary mode. In any other base the step size is the full display width, i.e. twelve digits without any overlap. See, for **example**, the most and least significant parts of the same number in base 3:

Bitwise Integer Operations

Your *WP 34S* carries all the operations you may know from the vintage *HP-16C* plus more. For seven functions, you will find the schematic pictures in the table below as they are printed on the backplate of the *HP-16C*. The 'C' in a box stands for the carry bit there.

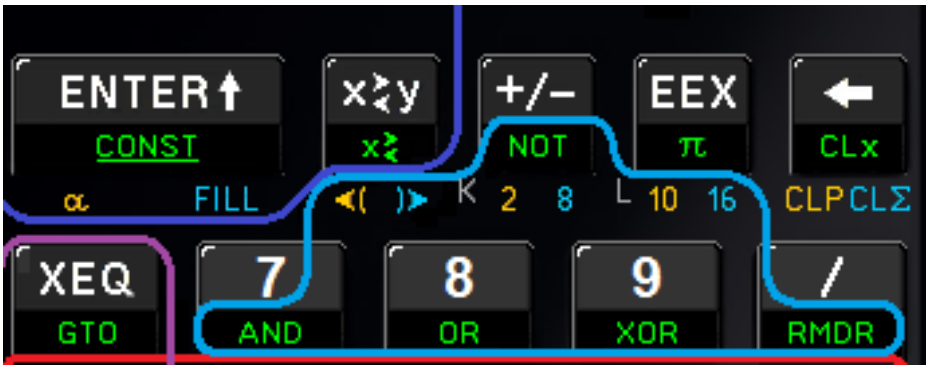
The following examples deal with 8-bit words showing leading zeros for easy reading. Input is `11101011 b` always. For further details about the respective operations, turn to the *IOP* below.

	Schematic picture	Example	Output
Shift Left		SL 2	10101100 bc
Shift Right		SR 3	00011101 b
Arithmetic Shift Right		ASR 3	in 2COMP and 1COMP: 11111101 b in UNSIGN: 00011101 b in SIGNMT: 10001101 b
Rotate Left		RL 2	10101111 bc
Rotate Right		RR 3	01111101 b
Rotate Left through Carry		RLC 2	10101101 bc
Rotate Right through Carry		RRC 3	11011101 b

The outputs in mode 1COMP are identical to those for 2COMP in these examples. Note the picture for ASR correctly describes this operation for 1's and 2's complement modes only. In all modes of the *HP-16C*, however, ASR 3 equals a signed division by 2^3, hence the different results for the latter two modes shown above. The other bitwise operations are insensitive to complement mode setting.

Bits are counted from right to left, starting with number 1 for the least significant bit. This is important for specifying bit numbers for the operations BC?, BS?, CB, FB, and SB.

Let us show you the bitwise two-number functions provided as well. Boole's operators AND, OR, and XOR are found in the light blue frame mentioned above:



Again, we will use 8-bit words for the following **examples**:

Common input		
	Y	01101011 b
	X	10111001 b
Operation		**Output**
AND		00101001 b
NAND		11010110 b
OR		11111011 b
NOR		00000100 b
XOR		11010010 b
XNOR		00101101 b

See the *IOP* for those and further commands working on bit level in integer modes (NOT, LJ and RJ, MASKL and MASKR, MIRROR, RAN#, and nBITS). Unless on the keyboard, the commands mentioned so far are found in the catalog *X.FCN* in integer modes. And there are also BS? and BC? in *TEST*.

Integer Arithmetic

Of the four basic arithmetic operations (+, -, ×, and /), the first three work in integer modes as they do in DECM, but with up to 64 digits precision in binary mode. Divisions, however, are handled differently in integer modes since the result cannot feature a fractional part here.

Generally, $\frac{a}{b} = (a \operatorname{div} b) + \frac{1}{b} \cdot \operatorname{rmdr}(a;b)$ applies, with the horizontal bar denoting real division, div representing integer division, and rmdr standing for the remainder of the latter. While remainders for positive parameters are simply found, remainders for negative dividends or divisors lead to confusion sometimes. The formula above, however, is easily employed for calculating such remainders (also for real numbers – see the first example):

Examples: $\frac{25}{7} = 3 + \frac{1}{7} \cdot 4$ (and for a real case: $\frac{25}{7{,}5} = 3 + \frac{1}{7{,}5} \cdot 2{,}5$)

$$\frac{-25}{7} = -3 + \frac{1}{7} \cdot (-4) \quad \Rightarrow \quad \operatorname{rmdr}(-25;7) = -4$$

$$\frac{25}{-7} = -3 + \frac{1}{-7} \cdot 4 \quad \Rightarrow \quad \operatorname{rmdr}(25;-7) = 4$$

$$\frac{-25}{-7} = 3 + \frac{1}{-7} \cdot (-4) \quad \Rightarrow \quad \operatorname{rmdr}(-25;-7) = -4$$

In general: $a - b \cdot (a \operatorname{div} b) =: \operatorname{rmdr}(a;b)$.

Unfortunately, there is a second function doing almost the same: it is called mod. With the same pairs of numbers as above, it returns:

$mod(25;7) = 4$,

$mod(-25;7) = 3$,

$mod(25;-7) = -3$,

$mod(-25;-7) = -4$.

So mod returns the same as rmdr if both parameters have equal signs only. The general formula for mod is a bit more sophisticated than the one above:

$a - b \cdot \mathrm{floor}\left(\frac{a}{b}\right) =: \mathrm{mod}(a;b)$ with e.g. $\mathrm{floor}\left(\frac{25}{7}\right) = 3$ and $\mathrm{floor}\left(-\frac{25}{7}\right) = -4$.

By the way, this formula applies to real numbers as well. So it may be used straightforwardly for calculating e.g. $\mathrm{mod}(25.3;-7.5) = 25.3 - (-7.5) \cdot (-4) = -4.7$.

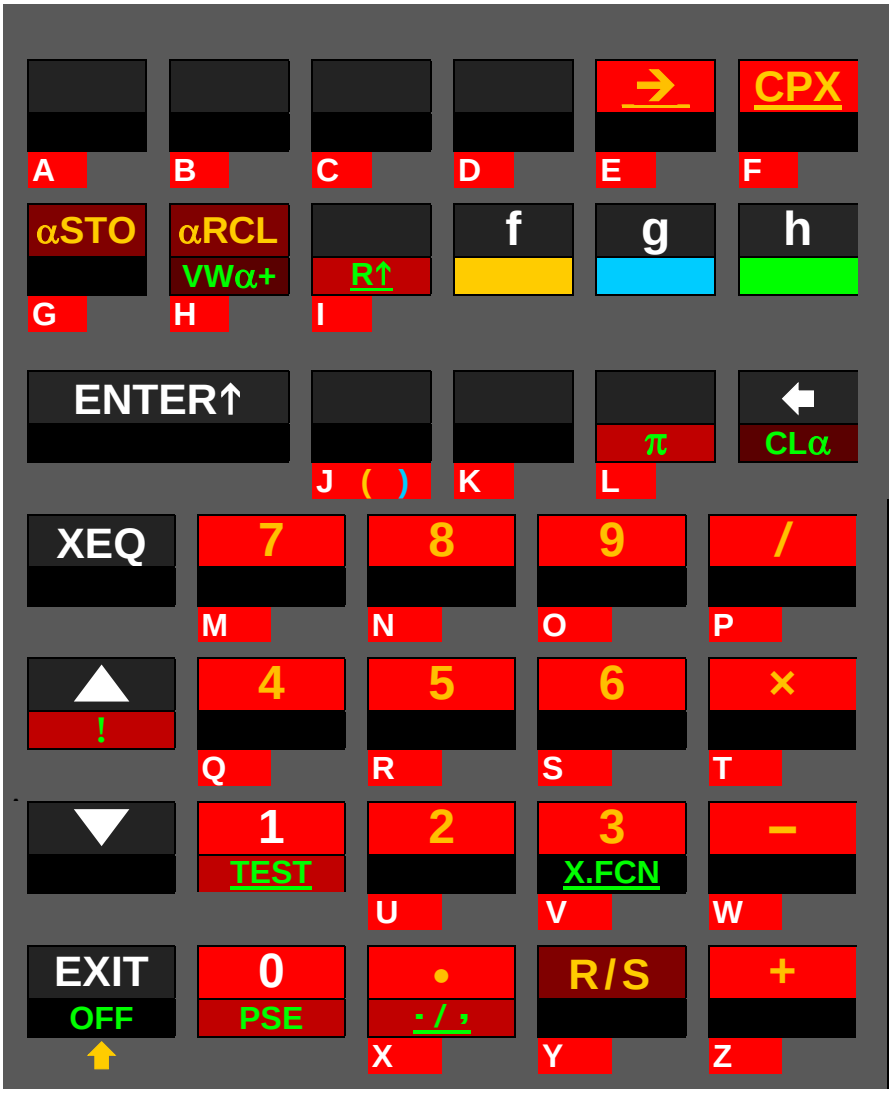
These three functions are called RMDR, MOD and FLOOR in your *WP 34S* for obvious reasons.

Furthermore, the exponential and logarithmic operations, x^2 and $\sqrt{x}$, x^3 and $\sqrt[3]{x}$, COMB and PERM work in integer modes, too. Beyond these keyboard functions and those operations mentioned further above, there are more in the *catalog X.FCN* like MOD, ×MOD, and ^MOD. See the *IOP* for further information about them.

SOMETHING COMPLETELY DIFFERENT: FULL ALPHA MODE

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

The Virtual Keyboard in Full Alpha Mode



In alpha mode, most mathematical operations are neither necessary nor applicable. So the keyboard is reassigned automatically when you enter alpha mode.

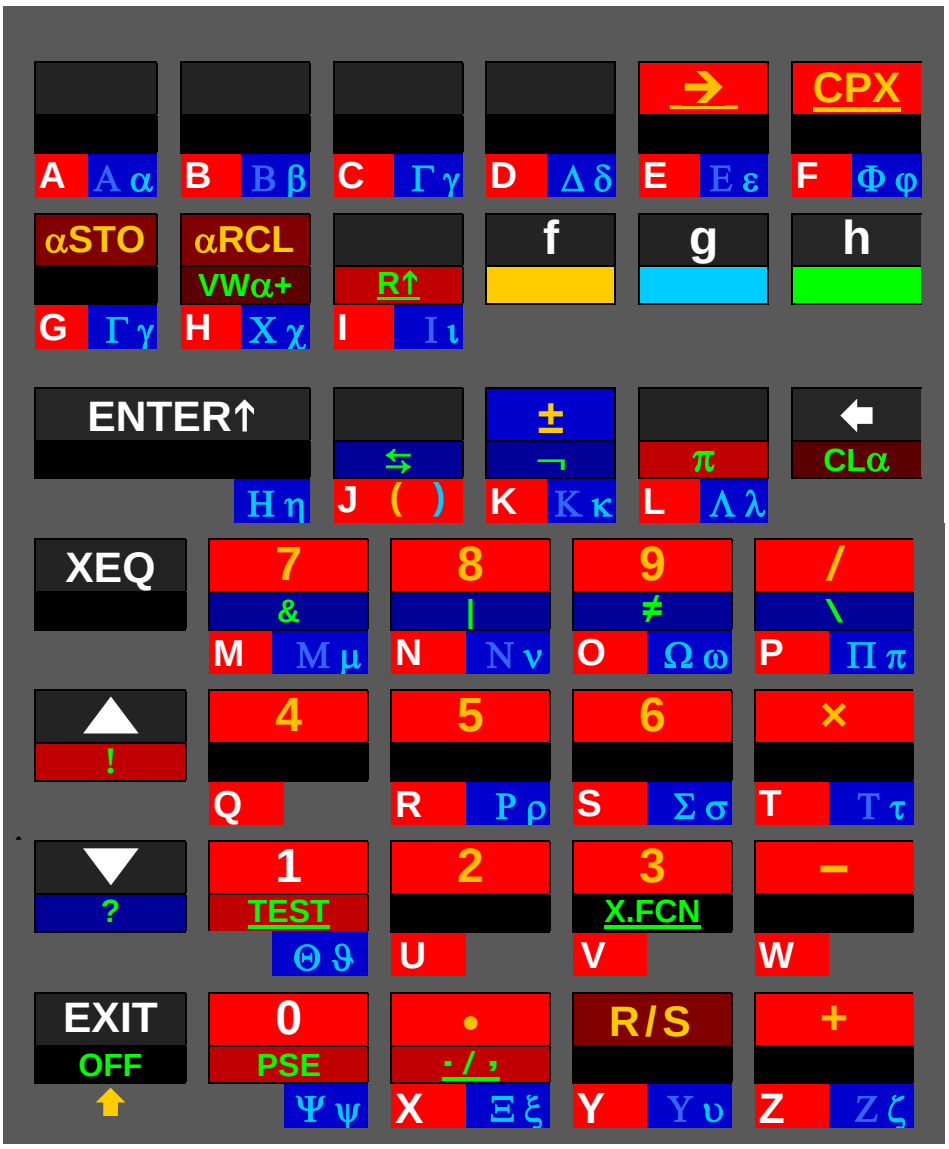
All labels printed on red background here append corresponding characters to ***alpha*** directly or via alpha catalogs. Note four new catalogs become active in this mode. See the keys [➡] and [CPX], and the labels [R↑] and [.,/] on your *WP 34S*.

Those labels printed on darker red background changed their functionality in other ways. See the keys [STO], [RCL], and [R/S], as well as the labels [VIEW] and [CLx].

Within alpha mode, primary function of most keys becomes appending the letter printed bottom left of this key – grey on the key plate – to ***alpha***. [PSE] appends a space. When ***alpha*** exceeds 30 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 [39].

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

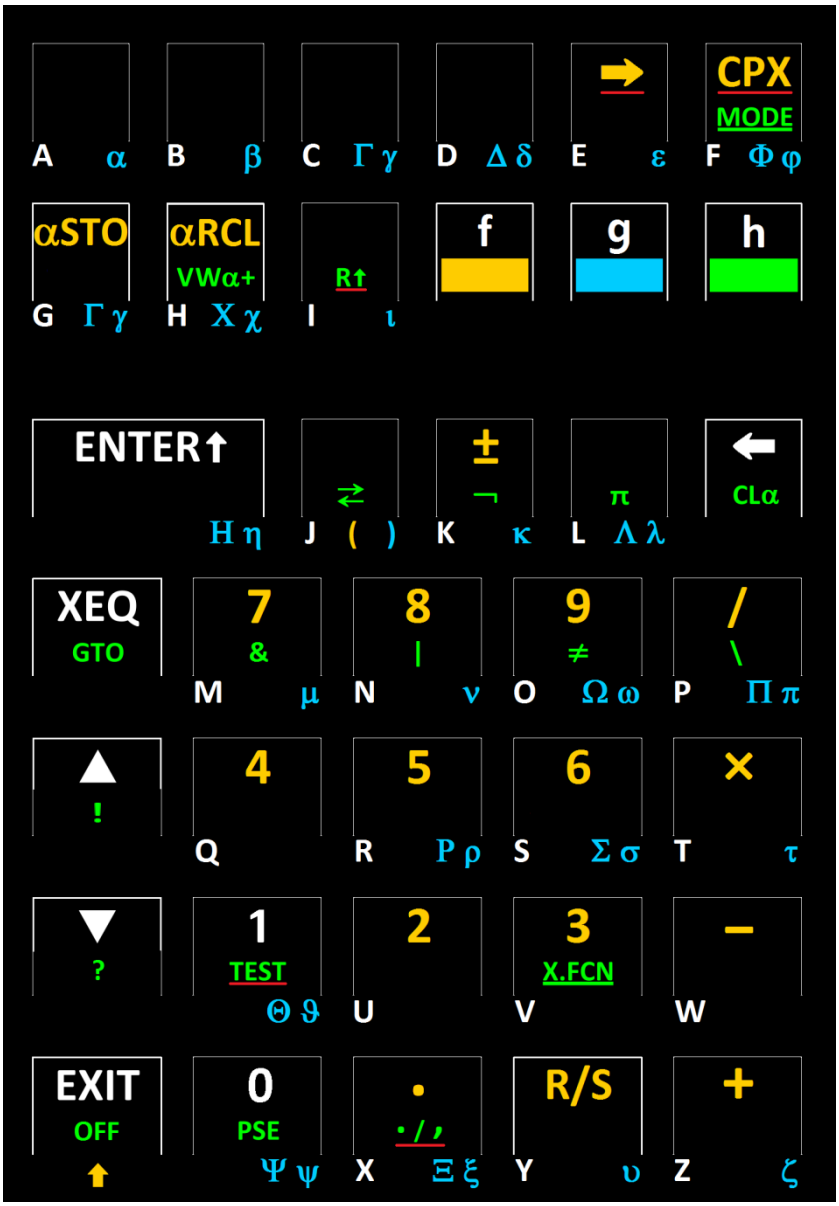
Looking at the standard labels on the keyboard, we can safely offer you even more in this mode: All labels printed on dark blue background in the virtual keyboard below append characters to ***alpha*** as well. They are related to the labels printed on your *WP 34S* keyboard at these locations, but deviate from them. Prefix g leads to homophonic Greek letters where applicable [40]. And h allows also accessing logic symbols via the Boolean operations.



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

For a clearer picture of this virtual keyboard – with the extra colors and redundancies removed – turn overleaf.

[40] "Homophonic" 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**), and **Eta** via g ENTER↑. **Omicron** is not featured because it looks 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. Professors of Greek, we count on your understanding.



The labels underlined in red call alpha catalogs. Labels underlined in green and labels shown between heavier white key contours do not append characters to ***alpha*** directly – all other labels do.

A copy of this virtual keyboard suitable for attachment to the back of your WP 34S is available – see *Appendix A*.

Character Set and Fonts

Your *WP 34S* features a large and a small alphanumeric font for display. Both are based on fonts by Luiz Viera (Brazil) as distributed in 2004. Some letters were added and some modified for better legibility, also given that the dot matrix of your display is only six pixels high.

See here all characters directly evocable through the virtual alpha keyboard above:

ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
ΑΒΓΔΕΖΗΘΙΚΛΜΝΞΟΠΡΣΤΥΦΧΨΩ
αβγδεζηθικλμνξοπρστυφχψω
0123456789 ()+−×/±.!?≒¬\&|≠

As soon as a string exceeds the visible display using this large font, your *WP 34S* will take the following small font automatically to show as much as possible:

ABCDEFGHIJKLMNOPQRSTUVWXYZ

abcdefghijklmnopqrstuvwxyz

ΑΒΓΔΕΖΗΘΙΚΛΜΝΞΟΠΡΣΤΥΦΧΨΩ

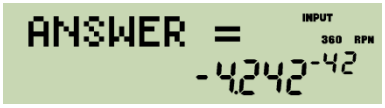
αβγδεζηθικλμνξοπρστυφχψω

0123456789 ()+−×/±.!?←¬\&|#

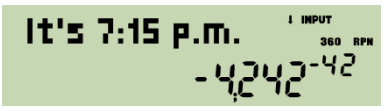
Many more characters of both fonts live in the alpha catalogs. You will find them *below*.

Displaying the Alpha Register

As soon as you enter alpha mode and as long as you stay therein, the contents of the alpha register (abbreviated by ***alpha***) are displayed in the dot matrix, showing its right end (i.e. the last characters it is containing), while the numeric section keeps the result of the last numeric operation. The display may look like:



Different information may be appended to ***alpha***. See the commands starting with 'α' in the *IOP* below. For **example**, αTIME allows creating texts like

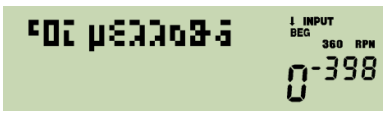


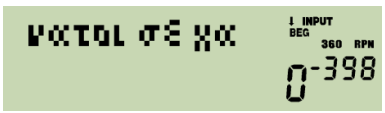
or


depending on time mode setting (12h / 24h).

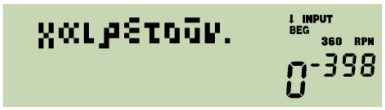
And αDATE will append – depending on date format setting – either `2012-04-16` or `16.04.2012` or `04/16/2012` to ***alpha***.

Note ***alpha*** can take up to 30 characters. And your *WP 34S* features a rich set of special letters and further characters. So you may easily store a Greek message like this, for **example**:





▲ 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.

Having left alpha mode, you can still display ***alpha***: use VIEWα or VWα+ for this – it will show you the left end (i.e. the first characters the alpha register contains).

Nevertheless, do not forget that your *WP 34S* is mainly designed as a calculator.

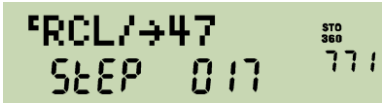
PROGRAMMING

Your *WP 34S* is a *keystroke-programmable* calculator. If this statement makes you smile with delight, this section is for you. Else please turn to the *HP-42S Owner's Manual* first for an introduction into keystroke-programming for solving repetitive tasks.

The basic building blocks within program memory are routines (also known as programs). Such routines may contain subroutines, but that is not mandatory. Typically, a routine or subroutine 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 global register provided – there are (almost) no limits. You are the sole and undisputed master of the memory!

This freedom, however, has a price: take care that each routine does not interfere with any other in its quest for data storage space. It is good practice to note the global registers a particular routine uses, and to document their purposes and contents for later reference.

In programming mode (i.e. while editing routines), the numeric display will indicate the current program step (000 – 927) in the mantissa and the number of free steps in the exponent, while the dot matrix will show the command contained in the current step, e.g.:



There is no routine-specific step counting like in the *HP-42S*.

The commands particularly required for programming are located within the violet frame on the keyboard as mentioned above. Here you find, starting top left, commands for executing (XEQ) or going to a specific program (GTO), keys for browsing program steps (▲ and ▼), flag commands (SF, CF, and STATUS), catalogs of *TESTs*, programming functions (*P.FCN*), and extra functions (*X.FCN*), commands for SoLVing, integrating, products, and sums, a key for EXITing programming mode, PSE for pausing

program execution, some commands for extracting parts (|x|, IP, and FP), a key for Running / Stopping programs or switching between programming and run mode (P/R), commands for labeling a routine (LBL) or returning from a routine (RTN), and for loop control (DSE and ISG). See the *IOP* for more about these commands.

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 program 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.

See the next page for addressing labels.

Searching labels, however, obeys the rule below. When a command like XEQ ***lbl*** is encountered, with ***lbl*** representing a label of one, two or three characters (like **A**, **BC**, **12**, **Tst**, **Pg3**, **x1µ**, etc.), your *WP 34S* will look for this label using the following method:

1. If ***lbl*** 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 ***lbl***, the quest will continue right after previous END (so the search will stay in the current routine). This is the search procedure for local labels. It is as known from the *HP-41C*.

2. If, however, ***lbl*** 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. This is the search procedure for global labels.

Tests

Like keystroke-programmable calculators before, your *WP 34S* features a set of tests. Their command names have a trailing ‘?’. Generally, tests will work as in the *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. So the general rule reads ‘skip if false’ (exception: KEY?).

As mentioned above, programs typically end with RTN or END. In running programs, both statements work very similar and show only subtle differences: a RTN statement immediately after a test returning **false** will be skipped – an END will not.

See the *IOP* below for more information. All tests are contained in the *catalog TEST*.

Note that there are also commands featuring a trailing ‘?’ but returning numbers (e.g. BASE?) or codes (e.g. KTP?) instead of **true** or **false** only – you will find these commands in the *catalog P.FCN*.

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.	→[41] opens indirect addressing and sets α_T mode.	*2-digit numeric (local) label* 00 … 99
	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 [42])	*Stack level or lettered register* X, Y, Z, ... , K	*Register number* 00 … 99, .00 … .15, if applicable [43]
	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** (keyed in as explained in footer).	Integrates the function whose label is on stack level **T**.	Executes the routine whose label is in **R44**.

Look up *GTO* in the *IOP* for special cases applying to this command exclusively.

---

[41] Works with all these operations except LBL .

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

[43] Some registers may be allocated to special applications. Check the memory table above.

Local Data

After some time with your *WP 34S* you will have a number of routines stored, so keeping track of their resource requirements may become a challenge. Most 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. On your *WP 34S*, registers are for data storage – so we offer you *local registers* allocated to your routines exclusively.

> **Example:** Let us assume you write a routine labeled **P1** and need five registers for your computations therein.
>
> Then all you have to do is just enter the command LOCR 5 in **P1** specifying you want five local registers. Thereafter, you can access these registers by using local numbers **.**00 … **.**04 throughout **P1**.
>
> Now, if you call another routine **P2** from **P1**, also **P2** may contain a step LOCR requesting local registers. These will also carry local register numbers **.**00 etc., but the local register **.**00 of **P2** will be different from the local register **.**00 of **P1**, so no interference will occur. As soon as the return statement is executed, the local registers of the corresponding routine are released and given back to free memory.

In addition, you get sixteen local flags as soon as you request at least one local register.

Local data holding allows for recursive programs, since every time such a routine is called again it will allocate a new set of local registers and flags being different from the ones it got before.

See the commands LOCR, LOCR?, MEM?, and POPLR in the *IOP* and in *Appendix B* below for more information, also about the limitations applying to local data.

Programmed Input and Output, User Interaction and Dialogues

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

When a program is started, the prior display contents are replaced by the ‘Running Program’ message and will be updated at certain events only – not after each operation. So where in manual mode a command shows an information immediately after execution, in automatic mode only PROMPT, PSE, STOP, VIEW, VIEWα, or VWα+ will trigger a display update, and the display will hold until the next such command is encountered. ERR 0 or MSG 0 are the only ways to get ‘Running Program’ back once this message has been replaced by a programmed display. See the following **examples** – parameters are omitted here:

- Take VIEW, VIEWα, or VWα+ for plain display updates. X is a valid parameter for VIEW and VWα+. Note frequent updates will slow down program execution, since the anti-flicker logic waits for a complete display refresh cycle before allowing the next update.

- Use one of the following four code segments for displaying messages or other information for a defined minimum time interval, given by PSE:

PSE	VIEW PSE	VIEWα PSE	VWα+ PSE
for plain numeric output		for complex alphanumeric messages	

- Ask ('prompt') for numeric input employing one of these four:

STOP	VIEWα STOP	VWα+ STOP	PROMPT	combining VWα+ X and STOP in one command

  Whatever 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.

- Prompt for alphanumeric input by the following steps:

αON	sets alpha mode and prepares for showing the final part of ***alpha***.
PROMPT	displays this part and waits for user input, terminated by R/S .
αOFF	returns to the numeric mode previously set.

  Whatever you key in will be appended to ***alpha*** here. The program will continue as soon as you press R/S .

See the *IOP* for more information about these commands and their parameters.

If you press – instead of or after keying in alphanumeric data – one of the hotkeys A to D in input, the program will call the next routine beginning with a label carrying this name.

The following **example** shows how a typical program structure might look:

```
001  LBL 'MYP'
002   CLα
003   α 'Hel'        Sets up a message …
004   α 'lo!'
005  LBL 00
006   PROMPT         … and stops waiting for user input.
007   GTO 00         R/S does nothing, it simply returns to the prompt.
008  LBL A           Called if user input after step 006 was terminated by A.
009   ENTRY?         Any new numeric data entered by the user?
010   GTO 11         Then go to step 012 where local label 11 lives.
011   XEQ 01         Else call subroutine 01 for computing a new number instead.
012  LBL 11
013   STO 01         Store the new number (input or computed).
014   RTN            Return to prompting via step 007.
```

```
015  LBL 01
...
...    RTN
...    LBL B
...
...    RTN
...
...    END
```

Compute the new number for missing numeric user input.

Called if user input after step 006 was terminated by B.

This is the way the TVM application is implemented.

If there is more than one program using labels A to D in *RAM* or *FM*, you must move the program counter (PC) into the desired program and stop there – provided programs are separated by END (else see *above* to learn which label will be found).

Keyboard Codes and Direct Keyboard Access

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.

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	

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.

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

The same keyboard codes are returned by the KEY? command, which allows '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 register specified. R/S and EXIT cannot be queried; they stop program execution immediately. The keyboard is active during program execution – but it is desirable to display a message and suspend the program by PSE while waiting for user input. Since PSE will be terminated by a key press, simply use PSE 99 in a loop to wait for input. Since KEY? acts as a test as well, a typical user input loop may well look like this **example**:

```
001  LBL 'USR'
002   CLα
003   α 'KEY'      Sets up a message …
004   α ?
005  LBL 00
006   VIEWα        … and displays it.
007   PSE 99       Waits 9.9 s for user input unless a key is pressed.
008   KEY? 00      Test for user input and put the key code in R00.
009   GTO 00       If there was none then go back to step 005.
010   LBL?→00      If a label corresponding to the key code has been defined …
011   XEQ→00       … then call it,
012   GTO 00       … else return to step 005.
```

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, KTP? ***nn*** is designed to return the type of the key pressed if its row / column code is given in register ***nn***. It will return:

- 0 to 9 for the respective digit keys;
- 10 for [.], [+/-], and [EEX];
- 11 for [f], [g], and [h];
- and 12 for the other keys.

An invalid code in the input register will throw 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.

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. The first section of *FM* is a 2*kB* backup region, holding the image of the entire user program memory, registers, and calculator states as soon as you completed a SAVE. The remaining part of *FM* (several *kilobytes* depending on setup) will hold programs only. Alphanumeric labels (see *above*) in *FM* can be called via XEQ like in *RAM*. This allows creating program libraries in *FM*. Use CAT to see the global labels already defined – labels in *FM* are tagged with Lib there.

FM is ideal for backups or other long-living data, but should not be used for repeated transient storage like in programmed loops [44]. Conversely, registers and standard user program memory residing in *RAM* are designed for data changing frequently but will not

[44] *FM* may not survive more than some 10,000 flashes. Thus, we made commands writing to *FM* (like SAVE or PSTO) non-programmable.

hold data with the batteries removed. So both kinds of memory have advantages and disadvantages you shall take into account for optimum benefit and longevity of 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. These, though written in user code, are read-only and thus can be called and executed but not edited. For you, it makes no difference whether a pre-programmed routine executes in *ROM* or *XROM*.

INDEX OF OPERATIONS (*IOP*)

All commands available (more than 600) are found below with their *names* and their necessary *keystrokes*. 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, ( ) + – × / ± , . ! ? : ; ‘ “ * @ _ ~ → ← ↑ ↓ ↔
< ≤ = ≈ ≠ ≥ > % $ € £ ¥ √ ∫ ∞ & \ ^ | [ ] { } 🖶 #

Superscripts and subscripts are handled like normal characters in sorting. The 🖶 near the end of the sorting order list above is the printer symbol heading all print commands.

Generally, functions and keystroke-programming will work as on the *HP-42S*, bit and integer functions as on the *HP-16C*, unless specified otherwise. Please refer to the manuals of the vintage calculators mentioned below for additional information about traditional commands.

A ^{C} heads the names of complex operations (see *above*). [CPX] is a legal prefix for all functions whose *names are printed in italics* in this list. Whenever a complex result may be returned, a capital C in the dot matrix will remind you to look at ***y*** as well.

Some 300 functions are available on your *WP 34S* for the first time ever on an *RPN* calculator. They got their remarks printed on yellow background. Operations carrying a familiar name but deviating in their functionality from previous *RPN* calculators are marked light red.

The vast majority of remarks for the respective operation start with a number:

(0) represents functions without effects on the stack,

(1) and (2) are for real or complex one- or two-number functions as defined above, and

(3) is for real three-number functions;

(-1) and (-2) stand for functions pushing numbers on the stack thus lifting it by 1 or 2 levels, respectively.

Operations disabling stack lift got a special remark.

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

In the following, each function is listed stating the mode(s) it will work in, abbreviated by their names. In this column, ‘integer’ stands for an arbitrary integer mode, ‘&’ for a Boolean AND, a comma for an OR, and ‘¬’ for NOT. So e.g. $\mathbf{2^X}$ works in all modes but alpha, even

in complex domain. All operations may also be entered in programming mode unless stated otherwise explicitly. Many functions contained in P.FCN make sense in programs only.

Name	Keys to press	in modes	Remarks (see *above* for general information)
10^x	f 10^x	¬α	(1)
12h	h MODE 12h	All	(0) Sets 12h time display mode: e.g. 1:23 will become 1:23 AM, 23:45 will become 11:45 PM. This will make a difference in αTIME only.
1COMPL	h MODE 1COMPL	All	(0) Sets 1's complement mode for integers.
1/x	f 1/x	DECM	(1)
	B	DECM	(1) Shortcut working if label B is not defined.
24h	h MODE 24h	All	(0) Sets 24h time display mode. Compare 12h.
2COMPL	h MODE 2COMPL	All	(0) Sets 2's complement mode for integers.
2^x	f 2^x	¬α	(1)
$\sqrt[3]{x}$	h X.FCN ∛x	¬α	(1)
ABS	f |x|	¬α	(1) Returns the absolute value.
	CPX f |x|	DECM	(1) Returns $r = \sqrt{x^2 + y^2}$ in **X** and clears **Y**.
ACOS	g COS^-1	DECM	(1) Returns $arccos(x)$.
ACOSH	g HYP^-1 COS	DECM	(1) Returns $arcosh(x)$. Note there is no need for pressing f here.
AGM	h X.FCN AGM	DECM	(2) Returns the arithmetic-geometric mean. See *http://en.wikipedia.org/wiki/Arithmetic-geometric_mean* .

Name	Keys to press	in modes	Remarks (see *above* for general information)
ALL	h ALL ***n***	¬α	(0) Sets the numeric display format to show all decimals whenever possible. ALL 0 works almost like ALL in the *HP-42S*. For $x \geq 10^{13}$, 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 / → 1.428571428571⁻⁴
AND	h AND	Integer	(2) Works bitwise as in the *HP-16C*. See *above*.
		DECM	(2) Works like AND in the *HP-28S*, i.e. x and y are interpreted before executing this operation. Zero is 'false', any other real number is 'true'.
ANGLE	h X.FCN ANGLE	DECM	(2) Returns $arctan(y/x)$.
ASIN	g SIN⁻¹	DECM	(1) Returns $arcsin(x)$.
ASINH	g HYP⁻¹ SIN	DECM	(1) Returns $arsinh(x)$. Note there is no need for pressing f here.
ASR	h X.FCN ASR ***n***	Integer	(1) Works like ***n*** (≤ 63) consecutive ASR commands in the *HP-16C*, corresponding to a division by 2^n . See *above* for details. ASR 0 executes as NOP, but loads **L**.
ATAN	g TAN⁻¹	DECM	(1) Returns $arctan(x)$.
ATANH	g HYP⁻¹ TAN	DECM	(1) Returns $artanh(x)$. Note there is no need for pressing f here.
BACK	h P.FCN BACK ***n***	¬α	(0) Jumps ***n*** steps backwards (0 ≤ ***n*** ≤ 255). E.g. BACK 1 goes to the previous program step. If BACK attempts to cross an END, an error is thrown. Reaching step 000 stops program execution. Compare SKIP. **ATTENTION**: If you edit a section of your routine that is crossed by one or more BACK, SKIP, or CASE jumps, this may well result in a need to manually maintain all those statements individually.

Name	Keys to press	in modes	Remarks (see *above* for general information)
BASE	h MODE BASE *n*	All	(0) Sets the base ***n*** for integer calculations, with $2 \le n \le 16$. Popular bases are directly accessible on the keyboard. See *above* for more information about integer modes. Furthermore, BASE 0 sets DECM, and BASE 1 calls FRACT. See there. **ATTENTION:** Stack contents are converted when switching from an integer mode to DECM, and are truncated vice versa. Other register contents stay as they are (see *below*). – BASE 10 is not DECM.
BASE 10	f 10	¬α	
BASE 16	g 16		
BASE 2	f 2		
BASE 8	g 8		
BASE?	h P.FCN BASE?	Integer	(-1) Returns the integer base set.
		DECM	(-1) Returns the integer base set before DECM.
BATT	h X.FCN BATT	DECM	(0) Measures the battery voltage in the range between $1.9V$ and $3.4V$ and returns this value.
		Integer	(0) Works like in DECM but returns the value in units of $100mV$.
BC?	h TEST BC? *n*	Integer	(0) Tests the specified bit in ***x***.
BestF	h MODE BestF	All	(0) Selects the best curve fit model, maximizing the correlation like BEST does in the *HP-42S*.
Binom	h PROB Binom	DECM	(1) Binomial distribution with the number of successes ***g*** in **X**, the probability of a success p_0 in **J** and the sample size ***n*** in **K**. Binom$_P$ [45] returns $p_B(g;n;p_0)=\binom{n}{g}\cdot p_0^g\cdot(1-p_0)^{n-g}$. Binom returns $F_B(m;n;p_0)=\sum_{g=0}^{m} p_B(g;n;p_0)$ with the maximum number of successes ***m*** in **X**. Binom^{-1} returns ***m*** for a given probability F_B in **X**, p_0 in **J** and ***n*** in **K**.
Binom$_P$	h PROB Binom$_P$		
Binom$_u$	h PROB Binom$_u$		
Binom $^{-1}$	h PROB Binom^{-1}		
B_n	h X.FCN B_n	DECM	(1) B_n returns the Bernoulli number for an integer $n > 0$ given in **X**. B_n* works with the old definition instead. See *below* for details.
B_n*	h X.FCN B_n*		
BS?	h TEST BS? *n*	Integer	(0) Tests the specified bit in ***x*** .

[45] 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 (see *above* for general information)

CASE	h P.FCN CASE ***s***	¬α	(0) Works like SKIP below but takes the number of steps to skip from a valid source register ***s***.
			Example: Assume the following program section:
			`...`
			`100 CASE 12`
			`101 GTO 01`
			`102 GTO 02`
			`103 GTO 07`
			`104 GTO 05`
			`105 LBL 01`
			`...`
			`132 LBL 02`
			`...`
			`153 LBL 05`
			`...`
			`234 LBL 07`
			`...`
			In program execution, ***r12*** will be checked in step 100: if ***r12*** ≤ 1 then the program will proceed to step 101 and continue with a jump to step 105, for ***r12*** = 2 the program will go to step 102, etc., resulting in a nice controlled dispatcher for 1 ≤ ***r12*** ≤ 4 .
			ATTENTION 1: CASE might surprise you for ***r12*** > 4 in the example above. Take care of the input you provide in ***s*** !
			ATTENTION 2: If you edit a section of your routine that is crossed by one or more BACK, SKIP, or CASE jumps, this may well result in a need to manually maintain all those statements individually.

	h CAT	¬α	Browser. See *below*.
Cauch	h PROB Cauch	DECM	(1) 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**. See *below* for details. Cauch^{-1} returns ***x*** for a given probability F_{Ca} in **X**, x_0 in **J** and γ in **K**.
Cauch$_P$	h PROB Cauch$_P$		
Cauch$_u$	h PROB Cauch$_u$		
Cauch $^{-1}$	h PROB Cauch^{-1}		
CB	h X.FCN CB ***n***	Integer	(1) Clears the specified bit in ***x*** .
CEIL	h X.FCN CEIL	¬α	(1) Returns the smallest integer ≥ ***x*** .
CF	g CF ***n***	¬α	(0) Clears the flag specified.
CFALL	h P.FCN CFALL	¬α	(0) Clears all user flags.

Name	Keys to press	in modes	Remarks (see *above* for general information)
CLALL	h P.FCN CLALL	¬PR, ¬α	(0) Clears all registers, flags, and programs in *RAM* if confirmed. Compare RESET.
CLP	f CLP	All	(0) Clears the current program, i.e. the one the program pointer is in.
CLPALL	h P.FCN CLPALL	¬PR, ¬α	(0) Clears all programs in *RAM* if confirmed.
CLREGS	h P.FCN CLREGS	¬α	(0) Clears all global and local general purpose registers allocated (see REGS and LocR). The stack contents as well as those of **L** and **I** are kept.
CLSTK	0 g FILL	¬α	Clears all stack registers currently allocated (i.e. **X** through **T** or **X** through **D**, respectively). All other register contents are kept.
	h P.FCN CLSTK	¬α	
CLx	h CLx	¬α	Clears register **X** only, disabling stack lift as usual.
CLα	h CLx	α	(0) Clears the alpha register like CLA in the *HP-42S*.
	h P.FCN CLα	¬α	
CLΣ	g CLΣ	DECM	(0) Clears the summation registers and releases the memory allocated for them.
CNST	h P.FCN CNST ***n***	¬α	(-1) Allows indirect addressing of the contents of CONST.
ᶜCNST	CPX h X.FCN ᶜCNST ***n***	¬α	(-2) Works like CNST but does a complex recall. See *above*.
CNVG?	h TEST CNVG? ***d***	DECM	(0) Checks for convergence by comparing ***x*** and ***y*** as determined by the lowest five bits of ***d***. a) The very lowest two bits set the tolerance limit: 0 = 10^{-14}, 1 = 10^{-24}, 2 = 10^{-32}, 3 = Choose the best for the mode set, resulting in taking 0 for single precision and 2 for double precision (see *below*). b) The next two bits determine the comparison mode using the tolerance limit set: 0 = compare the real numbers ***x*** and ***y*** relatively, 1 = compare them absolutely, 2 = check the absolute difference between the complex values $x + iy$ and $z + it$, 3 = works as 0 so far.

Name	Keys to press	in modes	Remarks (see *above* for general information)
CNVG? (contd.)			c) The top bit tells how special numbers are handled: 0 = NaN and infinities are considered converged, 1 = they are not considered converged. Now, $\boldsymbol{d} = \mathrm{a} + 4\mathrm{b} + 16\mathrm{c}$.
		Integer	(0) Tests for $\boldsymbol{x} = \boldsymbol{y}$.
COMB	f Cy,x	¬α	(2) Returns the number of possible sets of $\boldsymbol{y}$ items taken $\boldsymbol{x}$ at a time. No item occurs more than once in a set, and different orders of the same $\boldsymbol{x}$ items are not counted separately. Formula: $C_{y,x} = \binom{y}{x} = \frac{y!}{x!\cdot(y-x)!}$. Compare PERM.
ᶜCONJ	CPX h X.FCN ᶜCONJ	DECM	(1) Flips the sign of $\boldsymbol{y}$, thus returning the complex conjugate of $\boldsymbol{x_c}$.
	h CONST	¬α	Catalog. See *below*.
	h CONV	DECM	Catalog. See *below*.
CORR	g r	DECM	(-1) Returns the correlation coefficient for the current statistical data and curve fitting model.
COS	f COS	DECM	(1) Returns the cosine of the angle in $\mathbf{X}$.
COSH	f HYP COS	DECM	(1) Returns the hyperbolic cosine of $\boldsymbol{x}$.
COV	h STAT COV	DECM	(-1) Returns the population covariance for two data sets. It depends on the fit model selected. See s_{XY} for the sample covariance and *below* for more.
ᶜCROSS	CPX h X.FCN ᶜCROSS	DECM	(2) Interprets $\boldsymbol{x}$ and $\boldsymbol{y}$ as Cartesian components of a first vector, and $\boldsymbol{z}$ and $\boldsymbol{t}$ as those of a second one, and returns $[\boldsymbol{x}\cdot\boldsymbol{t} - \boldsymbol{y}\cdot\boldsymbol{z}$, 0, …] , dropping two stack levels.

Name	Keys to press	in modes	Remarks (see *above* for general information)
DATE	h X.FCN DATE	DECM	(-1) Recalls the date from the real time clock into the numeric section in the format selected. See D.MY, M.DY, and Y.MD. In addition, DATE shows the day of week in the dot matrix. The function DATE of the *HP-12C* corresponds to DAYS+ in your *WP 34S* (see below).
DATE→	h X.FCN DATE→	DECM	(-2) Assumes ***x*** containing a date in the format selected and pushes its three components on the stack.
DAY	h X.FCN DAY	DECM	(1) Assumes ***x*** containing a date in the format selected and extracts the day.
DAYS+	h X.FCN DAYS+	DECM	(2) Adds ***x*** days on a date in **Y** in the format selected and displays the resulting date including the day of week in the same format as WDAY does. Works like DATE in the *HP-12C.*
DBLOFF	h MODE DBLOFF	All	(0) Toggles double precision mode. Setting becomes effective in DECM only and is indicated by D in the dot matrix. See *Appendix H*.
DBLON	h MODE DBLON		
DBL?	h TEST DBL?	¬α	(0) Checks if double precision mode is turned on.
DBLR	h X.FCN DBLR	Integer	(2) Double word length commands for remainder, multiplication and division like in the *HP-16C*. If the division remainder is ≠ 0 , the carry flag is set. DBL× and DBL / clear the overflow flag.
DBL×	h X.FCN DBL×		
DBL /	h X.FCN DBL/		
DEC	h P.FCN DEC ***r***	¬α	(0) Decrements the register ***r*** by 1.
DECM	f H.d	¬α	(0) Sets default decimal floating point mode.
DECOMP	h X.FCN DECOMP	DECM	(-1) Decomposes ***x*** (after converting it into an improper fraction, if applicable), returning y/x = in top row and a stack [***denominator(x), numerator(x),*** …] . Reversible by division. **Example:** If **X** contains 2.25 then DECOMP will return ***x*** = 4 and ***y*** = 9 , pushing previous content of **Y** to **Z** etc.
DEG	g DEG	DECM	(0) Sets angular mode to *degrees*.
DEG→	h X.FCN DEG→	DECM	(1) Takes ***x*** as *degrees* and converts them to the angular mode currently set.

Name	Keys to press	in modes	Remarks (see *above* for general information)
DENANY	h MODE DENANY	All	(0) Sets default fraction format like in the *HP-35S*, allowing maximum precision in fraction display with the startup default – any denominator up to the value set by DENMAX may appear. **Example:** If DENMAX = 5 then DENANY allows denominators 1, 2, 3, 4, and 5.
DENFAC	h MODE DENFAC	All	(0) Sets 'factors of the maximum denominator', i.e. all integer factors of DENMAX may appear. **Example:** If DENMAX = 60 then DENFAC will allow denominators 1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 30, and 60. Now you know why 60 was a holy number in ancient Babylon.
DENFIX	h MODE DENFIX	All	(0) Sets fixed denominator format, i.e. the one and only denominator allowed is the value set by DENMAX.
DENMAX	h MODE DENMAX	All	(0) Works like /c in the *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 DENMAX execution time. For $x = 1$ the current setting is recalled.
DET	h MATRIX DET	DECM	(1) 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*	All	(0) 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).
^{C}DOT	CPX h X.FCN ᶜDOT	DECM	(2) Interprets x and y as Cartesian components of a first vector, and z and t as those of a second one, and returns $[x \cdot z + y \cdot t, 0, \ldots]$, dropping two stack levels.
dRCL	h P.FCN dRCL *s*	¬α	(-1) Assumes the source ***s*** contains double precision real data and recalls them as such. See *Appendix H*.
DROP	h X.FCN DROP	¬α	Drops x. See *above* for details.
^{C}DROP	CPX h X.FCN ᶜDROP	DECM	Drops x_c. See *above* for details.

Name	Keys to press	in modes	Remarks (see *above* for general information)
DSE	f DSE ***r***	¬α	(0) Given `ccccc.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 `ccccc` > 0 .
DSL	h P.FCN DSL ***r***	¬α	(0) Works like DSE but skips if `ccccccc` < `fff` .
DSZ	h P.FCN DSZ ***r***	¬α	(0) Decrements ***r*** by 1, and skips the next step if $\lvert r\rvert < 1$ thereafter. Known from the *HP-16C*.
D.MY	h MODE D.MY	All	(0) Sets the format for date display.
D→J	h X.FCN D→J	DECM	(1) Takes ***x*** as a date in the format set and converts it to a Julian day number according to JG…
D→R		DECM	(1) See the *catalog of conversions* for conversions from *degrees* to *radians*.
E3OFF	h MODE E3OFF	All	(0) Toggle the thousands separators for DECM (points or commas depending on radix setting).
E3ON	h MODE E3ON		
END	h P.FCN END	¬α	(0) Last command in a routine and terminal for searching local labels as described *above*. Works like RTN in all other aspects.
ENG	h ENG ***n***	¬α	(0) Sets engineering display format.
ENGOVR	h ENG ENTER↑	¬α	(0) Defines that numbers exceeding the range displayable in ALL or FIX will be shown in engineering format. See SCIOVR.
ENTER↑	ENTER↑	¬α	(-1) Pushes ***x*** on the stack, disabling stack lift as usual. See *above* for details.
ENTRY?	h TEST ENTRY?	¬α	(0) 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	h X.FCN erf	DECM	(1) Returns the error function or its complement: $erf(x)=\frac{2}{\sqrt{\pi}}\int_0^x e^{-\tau^2}d\tau$ and $erfc(x)=1-erf(x)$.
erfc	h X.FCN erfc		

Name	Keys to press	in modes	Remarks (see *above* for general information)
ERR	h P.FCN ERR ***n***	¬α	(0) Raises the error specified. The consequences are the same as if the respective error really occurred, so e.g. a running program will be stopped. See *below* for the respective error codes. Compare MSG.
EVEN?	h TEST EVEN?	¬α	(0) Checks if x is integer and even.
e^x	f e^x	¬α	(1)
	EXIT	All	See *below*.
ExpF	h MODE ExpF	All	(0) Selects the exponential curve fit model $y = a_0 e^{a_1 x}$.
Expon	h PROB Expon	DECM	(1) Exponential distribution with the rate λ in **J** : Expon$_P$ [46] returns $f_{Ex}(x) = \lambda \cdot e^{-\lambda x}$, Expon returns $F_{Ex}(x) = 1 - e^{-\lambda x}$. Expon^{-1} returns the survival time t_s for a given probability F_{Ex} in **X** and λ in **J**.
Expon$_P$	h PROB Expon$_P$		
Expon$_u$	h PROB Expon$_u$		
Expon $^{-1}$	h PROB Expon^{-1}		
EXPT	h X.FCN EXPT	DECM	(1) Returns the exponent h of the number displayed $x = m \cdot 10^h$. Compare MANT.
e^x -1	h X.FCN e^x–1	DECM	(1) For $x \approx 0$, returns a more accurate result for the fractional part than e^x does.
FAST	h MODE FAST	All	(0) Sets the processor speed to 'fast'. This is start-up default and is kept for fresh batteries. Compare SLOW.
FB	h X.FCN FB ***n***	Integer	(1) Inverts ('flips') the specified bit in x .
FC?	h TEST FC? ***n***	¬α	(0) Tests if the flag specified is clear.
FC?C	h TEST FC?C ***n*** etc.	¬α	(0) Tests if the flag specified is clear. Clears, flips, or sets this flag after testing, respectively.
FC?F			
FC?S			
FF	h P.FCN FF ***n***	¬α	(0) Flips the flag specified.

[46] The pdf corresponds to EXPONDIST(*x; λ;* **0**) and the cdf to EXPONDIST(*x; λ;* **1**) in MS Excel.

Name	Keys to press	in modes	Remarks (see *above* for general information)
FIB	h X.FCN FIB	Integer	(1) Returns the Fibonacci number f_n with $n = x$. These numbers are defined as $f_0 = 0$, $f_1 = 1$, and $f_n = f_{n-1} + f_{n-2}$ for $n \geq 2$.
		DECM	(1) Returns the extended Fibonacci number. See *below* for details.
FILL	g FILL	¬α	(0) Copies x to all stack levels. See details *above*.
FIX	h FIX ***n***	¬α	(0) Sets fixed point display format.
FLASH?	h P.FCN FLASH?	¬α	(-1) Returns the number of free words in flash memory.
FLOOR	h X.FCN FLOOR	¬α	(1) Returns the largest integer $\leq x$.
FP	g FP	¬α	(1) Returns the fractional part of x.
FP?	h TEST FP?	¬α	(0) Tests x for having a nonzero fractional part.
FRACT	h MODE FRACT	All	(0) Sets fraction mode like in the *HP-35S*, but keeps the display format as set by PROFRC or IMPFRC.
FS?	h TEST FS? ***n***	¬α	(0) Tests if the flag specified is set.
FS?C FS?F FS?S	h TEST FS?C ***n*** etc.	¬α	(0) Tests if the flag specified is set. Clears, flips, or sets this flag after testing, respectively.
$F_P(x)$	h PROB $F_P(x)$	DECM	(1) Fisher's F-distribution. F(x) equals 1 – Q(F), $F_u(x)$ equals Q(F) and $F^{-1}(p)$ equals F_p in the *HP-21S*.. The degrees of freedom are specified in **J** and **K**. The F-distribution is heavily used in *ANOVA*.
$F_u(x)$	h PROB $F_u(x)$		
F(x)	h PROB F(x)		
$F^{-1}(p)$	h PROB $F^{-1}(p)$		

Name	Keys to press	in modes	Remarks (see *above* for general information)
f'(x)	h P.FCN f'(x) ***label***	DECM	Returns the first derivative of the function ***f(x)*** at position ***x***. This ***f(x)*** must be specified in a routine starting with LBL ***label*** . On return, **Y**, **Z**, and **T** will be cleared and the position ***x*** will be in **L**. **ATTENTION**: f'(x) will look for a user routine labeled 'δx', returning a fixed step size ***dx*** in **X**. If that routine is not defined, ***dx*** = *0.1* is set for default (arbitrary, but a choice had to be made). – Then, f'(x) will evaluate ***f(x)*** at ten points equally spaced in the interval $x \pm 5\,dx$. If you expect any irregularities within this interval, change ***dx*** to exclude them.
f"(x)	h P.FCN f"(x) ***label***	DECM	Works like f'(x) but returns the second derivative.
GCD	h X.FCN GCD	¬α	(2) Returns the Greatest Common Divisor of ***x*** and ***y*** [47]. This will always be positive.
gCLR	h P.FCN gCLR ***n***	¬α	(0) Clears the pixel at position ***x***, ***y*** in the graphic data set starting at register address ***n***. Valid ranges are $0 \le x \le w-1$ and $0 \le y \le h-1$. Pixel 0, 0 is top left. See gDIM for more.
g_d	h X.FCN g▪	DECM	(1) Returns the Gudermann function or its inverse. See *below* for details.
g_d^{-1}	h X.FCN g▪⁻¹		
gDIM	h P.FCN gDIM ***n***	¬α	(0) Initiates a set of registers starting at address number ***n***, allowing for graphic data featuring ***x*** (≤ 166) pixel columns and ***y*** pixel rows. For $x \le 0$, the width ***w*** will be set to 166. For $y \le 0$, the height ***h*** will be set to 8. The first two bytes are reserved to take ***w*** and $\check{h} = floor\left(\frac{h+7}{8}\right)$. The number of registers needed for the set is $floor\left(\frac{w \cdot \check{h}+9}{8}\right)$ in startup standard mode. E.g. 21 registers are required for maximum width and standard height. The command can be exactly emulated in integer mode by storing $256 \cdot \check{h} + w$ in the first register and clearing the rest. See FLOOR and 🖶PLOT.
gDIM?	h P.FCN gDIM? ***n***	¬α	(-2) Assumes a graphic data set stored starting at register address ***n*** and recalls ***h*** and ***w*** for it. See gDIM for more.

[47] GCD translates to "ggT" in German.

Name	Keys to press	in modes	Remarks (see *above* for general information)
Geom	h PROB Geom	DECM	(1) 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 ***p₀*** 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 ***p₀*** in **J**.
Geom$_P$	h PROB Geom$_P$		
Geom$_u$	h PROB Geom$_u$		
Geom^{-1}	h PROB Geom^{-1}		
gFLP	h P.FCN gFLP ***n***	¬α	(0) Flips the pixel at position ***x***, ***y*** in the graphic data set starting at register address ***n***. See gCLR for more.
gPIX?	h TEST gPIX? ***n***	¬α	(0) Tests the pixel at position ***x***, ***y*** in the graphic data set starting at register address ***n***. See gCLR for more.
gPLOT	h P.FCN gPLOT ***n***	¬α	(0) Displays the top left sector of the graphic data set (starting at register address ***n***) in the dot matrix section of the LCD. See gDIM for more.
GRAD	g GRAD	DECM	(0) Sets angular mode to *gon* (*grad*).
GRAD→	h X.FCN GRAD→	DECM	(1) Takes ***x*** as given in *gon* (*grad*) and converts them to the angular mode currently set.
gSET	h P.FCN gSET ***n***	¬α	(0) Sets the pixel at position ***x***, ***y*** in the graphic data set starting at register address ***n***. See gCLR for more.
GTO	h GTO ***label***	PR	(0) Inserts an unconditional branch to ***label***.
		¬PR, ¬α	(0) Positions the program pointer to ***label***.
	h GTO . A , B , C , or D	¬α	(0) Positions the program pointer … … to one of these labels, if defined.
	h GTO . ***nnn***		… to step ***nnn*** .
	h GTO . ▲		… directly after previous END, going to the top of current program.
	h GTO . ▼		… directly after next END, going to the top of next program.
	h GTO . .		… to step 000, i.e. top of *RAM*.
GTOα	h P.FCN GTOα	¬α	(0) Takes the first three characters of ***alpha*** (or all if there are fewer than three) as a label and positions the program pointer to it.

Name	Keys to press	in modes	Remarks (see *above* for general information)
H_n	h X.FCN H_n	DECM	(2) Hermite's polynomials for probability (H_n) and for physics (H_{np}). See *below* for details.
H_{np}	h X.FCN H_{np}		
H.MS	f H.MS	DECM	(1) Assumes **X** containing decimal *hours* or *degrees*, and displays them converted in the format `hhhh°mm′ss.dd″` temporarily as shown *above*.
H.MS+	h X.FCN H.MS+	DECM	(2) Assumes **X** and **Y** containing times or *degrees* in the format `hhhh.mmssdd`, and adds or subtracts them, respectively.
H.MS–	h X.FCN H.MS–		
IMPFRC	g d/c	¬α	(1) Sets fraction mode allowing improper fractions in display (e.g. $^5/_3$ instead of $1\,^2/_3$). Converts ***x*** according to the settings by DEN… into an improper fraction if possible. Absolute decimal equivalents of ***x*** must not exceed 100,000. Compare PROFRC.
INC	h P.FCN INC ***r***	¬α	(0) Increments the register ***r*** by 1.
INTM?	h TEST INTM?	¬α	(0) Tests if your *WP 34S* is in an integer mode.
INT?	h TEST INT?	¬α	(0) Tests ***x*** for being an integer, i.e. having a fractional part equal to zero. Compare FP?.
IP	f IP	¬α	(1) Returns the integer part of ***x*** .
iRCL	h X.FCN iRCL ***s***	¬α	(-1) Assumes the source ***s*** contains integer data and recalls them as such. See *below*.
ISE	h P.FCN ISE ***r***	¬α	(0) Works like ISG but skips if `ccccccc` ≥ `fff` .
ISG	g ISG ***r***	¬α	(0) Given `ccccc.fffii` in ***r*** , this function increments ***r*** by `ii`, skipping next program line if then `ccccccc` > `fff` . If ***r*** has no fractional part then `fff` = 0 , of course, and `ii` is set to 1. Note that neither `fff` nor `ii` can be negative, but `ccccc` can.
ISZ	h P.FCN ISZ ***r***	¬α	(0) Increments ***r*** by 1, skipping next program line if then $\lvert r\rvert < 1$. Known from the *HP-16C*.
I β	h X.FCN Iβ	DECM	Returns the regularized (incomplete) beta or gamma function (one of two kinds). See *below* for details.
$I\Gamma_p$	h X.FCN $I\Gamma_p$		
$I\Gamma_q$	h X.FCN $I\Gamma_q$		

Name	Keys to press	in modes	Remarks (see *above* for general information)
JG1582	h MODE JG1582	All	(0) 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. See *above*.
JG1752	h MODE JG1752		
J→D	h X.FCN J→D	DECM	(1) 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***	¬α	(0) Tests if a key was pressed while a program was running or paused. If no key was pressed in that interval, the next program step after KEY? will be executed, else it will be skipped and the code of said key will be stored in ***a*** . Key codes reflect the rows and columns on the keyboard starting top left (see *above*).
KTP?	h P.FCN KTP? ***a***	¬α	(-1) Assumes a key code in address ***a*** (see KEY?). 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	(-1) See *above* for details. In fact, this command will be recorded as RCL L in programs.
LBL	f LBL ***label***	PR	(0) Identifies programs and routines for execution and branching. See the opportunities for specifying a ***label*** in the table *above*.
LBL?	h TEST LBL? ***label***	¬α	(0) Tests for the existence of the label specified, anywhere in program memory. See LBL for more.
LCM	h X.FCN LCM	¬α	(2) Returns the Least Common Multiple of ***x*** and ***y*** [48]. This will always be positive.
LEAP?	h TEST LEAP?	DECM	(0) Takes ***x*** as a date in the format selected, extracts the year, and tests for a leap year.
LgNrm	h PROB LgNrm	DECM	(1) Lognormal distribution with $\mu = \ln \bar{x}_g$ specified in **J** and $\sigma = \ln \varepsilon$ in **K**. See x̄g and ε below. See *below* for details. LgNrm^{-1} returns ***x*** for a given probability F_{Ln} in **X**, ***μ*** in **J**, and ***σ*** in **K**.
LgNrm$_P$	h PROB LgNrmP		
LgNrm$_u$	h PROB LgNrmu		
LgNrm^{-1}	h PROB LgNrm-1		

[48] LCM translates to “kgV” in German.

Name	Keys to press	in modes	Remarks (see *above* for general information)
LINEQS	h X.FCN LINEQS	¬α	(3) Takes a base register number 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 LinF	All	(0) Selects the linear curve fit model $y=a_0+a_1x$.
LJ	h X.FCN LJ	Integer	(-1) Left justifies a bit pattern within its word size as in the *HP-16C*: The stack will lift, placing the left-justified word in **Y** and the count (number of bit-shifts necessary to left justify the word) in **X**. **Example** for word size 8: 10110_2 LJ results in $x = 3$ and $y = 10110000_2$.
LN	g LN	¬α	(1) Returns the natural logarithm of ***x***.
L_n	h X.FCN Ln	DECM	(2) Laguerre's polynomials. See *below* for details.
LN1+x	h X.FCN LN1+x	DECM	(1) For $x\approx 0$, this returns a more accurate result for the fractional part than $\ln(x)$ does.
$L_n\alpha$	h X.FCN Lnα	DECM	(3) Laguerre's generalized polynomials. See *below* for details.
LN β	h X.FCN LNβ	DECM	(2) Returns the natural logarithm of Euler's Beta function. See *β*.
LN Γ	h X.FCN LNΓ	DECM	(1) Returns the natural logarithm of $\Gamma(x)$.
LOAD	h P.FCN LOAD	¬PR, ¬α	Restores the entire backup from flash, i.e. executes LOADP, LOADR, LOADSS, LOADΣ, and returns Restored. Compare SAVE. See the commands mentioned and *Appendix A* for more.
LOADP	h P.FCN LOADP	¬PR, ¬α	(0) 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 – else an error will be thrown.
LOADR	h P.FCN LOADR	¬α	(0) Recovers numbered general purpose registers from the backup (see SAVE and *above*). Lettered registers will not be recalled. The number of registers copied is the minimum of the registers held in the backup and *RAM* at execution time.
LOADSS	h P.FCN LOADSS	¬α	(0) Recovers the system state from the backup. See *Appendix B* for more.

Name	Keys to press	in modes	Remarks (see *above* for general information)
LOADΣ	h P.FCN LOADΣ	¬α	(0) 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***	¬α	(0) Allocates ***n*** local registers (≤ 144) and 16 local flags for the current program. See *above*.
LocR?	h P.FCN LocR?	¬α	(-1) Returns the number of local registers allocated.
LOG$_{10}$	g LG	¬α	(1) Returns the logarithm of ***x*** for base 10.
LOG$_2$	g LB	¬α	(1) Returns the logarithm of ***x*** for base 2.
LogF	h MODE LogF	All	(0) Selects the logarithmic curve fit model $y = a_0 + a_1 \ln x$.
Logis	h PROB Logis	DECM	(1) Logistic distribution with ***μ*** given in **J** and ***s*** in **K**. See *below* for details. 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**.
Logis$_P$	h PROB Logis.		
Logis$_u$	h PROB Logis.		
Logis $^{-1}$	h PROB Logis^{-1}		
LOG$_x$	g LOG$_x$	¬α	(2) Returns the logarithm of ***y*** for the base ***x*** .
^{C}LOG$_x$	CPX g LOG$_x$	DECM	(2) Returns the complex logarithm of $z + i\,t$ for the complex base $x + i\,y$.
LZOFF	h MODE LZOFF	All	(0) Toggles leading zeros like flag 3 does in the *HP-16C*. Relevant in bases 2, 4, 8, and 16 only.
LZON	h MODE LZON		
L.R.	h STAT L.R.	DECM	(-2) Returns the parameters ***a$_1$*** and ***a$_0$*** of the fit curve through the data points accumulated in the summation registers, according to the curve fit model selected (see LINF, EXPF, POWERF, and LOGF). For a straight line (LINF), ***a$_0$*** is the y-intercept and ***a$_1$*** the slope.

Name	Keys to press	in modes	Remarks (see *above* for general information)
MANT	h X.FCN MANT	DECM	(1) Returns the mantissa ***m*** of the number displayed $x = m \cdot 10^h$. Compare EXPT.
MASKL	h X.FCN MASKL ***n***	Integer	(-1) Work like MASKL and MASKR on the *HP-16C*, but with the mask length following the command instead of taken from **X**. Thus, the mask is pushed on the stack. **Example**: For WSIZE 8, MASKL 3 returns a mask word 11100000_2. Use it e.g. for extracting the three most significant bits from an arbitrary byte by AND.
MASKR	h X.FCN MASKR ***n***		
	h MATRIX	DECM	Catalog. See *below*.
MAX	h X.FCN MAX	¬α	(2) Returns the maximum of ***x*** and ***y*** .
MEM?	h P.FCN MEM?	¬α	(-1) Returns the number of free words in program memory, taking into account the local registers allocated.
MIN	h X.FCN MIN	¬α	(2) Returns the minimum of ***x*** and ***y*** .
MIRROR	h X.FCN MIRROR	Integer	(1) Reflects the bit pattern in ***x*** (e.g. 00010111_2 would become 11101000_2 for word size 8).
MOD	h X.FCN MOD	¬α	(2) Returns $y \bmod x$ (see *above* for examples). Compare RMDR.
	h MODE	All	Catalog. See *below*.
MONTH	h X.FCN MONTH	DECM	(1) Assumes ***x*** containing a date in the format selected and extracts the month.
MROW+×	h MATRIX MROW+×	DECM	(0) Takes a matrix *descriptor* ***x***, a destination row number ***y***, a source row number ***z***, and a real number ***t***. It multiples each element m_{zi} of (*X*) by ***t*** and adds it to m_{yi} . The stack remains unchanged. M.ROW+× is similar to *PPC M3*.
MROW×	h MATRIX MROW×	DECM	(0) Takes a matrix *descriptor* ***x***, a row number ***y***, and a real number ***z***. It multiples each element m_{yi} of (*X*) by ***z***. The stack remains unchanged. M.ROW× is similar to *PPC M2*.
MROW⇆	h MATRIX MROW⇆	DECM	(0) Takes a matrix *descriptor* ***x*** and two row numbers ***y*** and ***z***. It swaps the contents of rows ***y*** and ***z*** in (*X*). The stack remains unchanged. M.ROW⇆ is similar to *PPC M1*.
MSG	h P.FCN MSG ***n***	¬α	(0) Throws the error message specified. This will be a temporary message. See *below* for the respective error codes. Compare ERR.

Name	Keys to press	in modes	Remarks (see *above* for general information)
M+×	h MATRIX M+×	DECM	(3) Takes two matrix *descriptors* 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 M-¹	DECM	(0) Takes a *descriptor* of a square matrix in **X** and inverts the matrix in-situ. Doesn't alter the stack.
M-ALL	h MATRIX M–ALL	DECM	(1) Takes a matrix *descriptor* x, saves it in **L**, and returns a value suitable for ISG or DSL looping in **X**. The loop processes all elements in (X). The loop index is DSL if the descriptor is negative and ISG otherwise.
M-COL	h MATRIX M–COL	DECM	(2) Takes a matrix *descriptor* x and a column number y. Returns a loop counter in **X**, dropping the stack. The matrix descriptor is saved in **L**. The loop processes all elements m_{iy} in (X) only. The loop index is DSL if the descriptor is negative and ISG otherwise.
M-DIAG	h MATRIX M–DIAG	DECM	(1) Takes a matrix *descriptor* 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} in (X). The loop index is DSL if the descriptor is negative and ISG otherwise.
M-ROW	h MATRIX M–ROW	DECM	(2) Takes a matrix *descriptor* x and a row number 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} in (X) only. The loop index is DSL if the descriptor is negative and ISG otherwise.
M×	h MATRIX M×	DECM	(3) Takes two matrix *descriptors* 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 (39 digits). The fractional part of x is updated to match the resulting matrix – no overlap checking is performed.
M.COPY	h MATRIX M.COPY	DECM	(2) Takes a matrix *descriptor* y and a base register number x. Copies the matrix (Y) into registers starting at Rx. Returns a properly formatted matrix descriptor in **X**.
M.DY	h MODE M.DY	All	(0) Sets the format for date display.

Name	Keys to press	in modes	Remarks (see *above* for general information)
M.IJ	h MATRIX M.IJ	DECM	Takes a matrix *descriptor* x and a register number y. Returns the column that register represents in $\mathbf{Y}$ and the row in $\mathbf{X}$. The descriptor is saved in $\mathbf{L}$. M.IJ is similar to *PPC M4*.
M.LU	h MATRIX M.LU	DECM	(1) Takes a *descriptor* of a square matrix in $\mathbf{X}$. Transforms (X) into its LU decomposition. in-situ. The value in $\mathbf{X}$ is replaced by a 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 significant for the second and so forth.
M.REG	h MATRIX M.REG	DECM	(3) Takes a matrix *descriptor* x, a row number y, and a column number z. The descriptor is saved in $\mathbf{L}$. M.REG returns the register number in $\mathbf{X}$. It is similar to *PPC M5*.
M.SQR?	h TEST M.SQR?	DECM	(0) Tests a matrix *descriptor* x and returns true if the matrix is square.
NAND	h X.FCN NAND	¬α	(2) Works in analogy to AND. See *there*.
NaN?	h TEST NaN?	¬α	(0) Tests x for being 'Not a Number'.
nBITS	h X.FCN nBITS	Integer	(1) Counts bits set in x like #B does on the *HP-16C*.
nCOL	h MATRIX nCOL	DECM	(1) Takes a matrix *descriptor* x, saves it in $\mathbf{L}$, and returns the number of columns in (X).
NEIGHB	h X.FCN NEIGHB	DECM	(2) Returns the nearest machine-representable number to x in the direction toward y in the mode set [49]. For $x < y$ (or $x > y$), this is the machine successor (or predecessor) of x; for $x = y$ it is y.
		Integer	(2) Returns $x + 1$ for $x < y$, y for $x = y$, or $x - 1$ for $x > y$.
NEXTP	h X.FCN NEXTP	¬α	(1) Returns the next prime number greater than x.
NOP	h P.FCN NOP	¬α	(0) 'Empty' step FWIW.
NOR	h X.FCN NOR	¬α	(2) Works in analogy to AND. See *there*.

[49] You may find NEIGHB useful investigating numeric stability. See NEIGHBOR in the HP-71 Math Pac.

Name	Keys to press	in modes	Remarks (see *above* for general information)
Norml	h PROB Norml	DECM	(1) Normal distribution with an arbitrary mean ***μ*** given in **J** and a standard deviation ***σ*** in **K**. See *below* for details. Norml^{-1} returns ***x*** for a given probability F_N in **X**, ***μ*** in **J**, and ***σ*** in **K**.
Norml$_P$	h PROB Norml▸		
Norml$_u$	h PROB Norml▾		
Norml^{-1}	h PROB Norml⁻¹		
NOT	h NOT	Integer	(1) Inverts ***x*** bit-wise as on the *HP-16C*.
		DECM	(1) Returns 1 for $x = 0$, and 0 for $x \neq 0$.
nROW	h MATRIX nROW	DECM	(1) Takes a matrix *descriptor* ***x***, saves it in **L**, and returns the number of rows in (*X*).
nΣ	h SUMS nΣ	DECM	(-1) Recalls the number of accumulated data points. Necessary for basic statistics.
ODD?	h TEST ODD?	¬α	(0) Checks if ***x*** is integer and odd.
OFF	h OFF	PR	(0) Inserts a step to turn your *WP 34S* off under program control.
		¬PR	(0) Turns your *WP 34S* off.
OR	h OR	¬α	(2) Works in analogy to AND. See *there*.
PERM	g Py,x	¬α	(2) 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: $P_{y,x} = \frac{y!}{(y-x)!} = x! \cdot C_{y,x}$. Compare COMB.
P_n	h X.FCN P▪	DECM	(1) Legendre's polynomials. See *below* for details.
Poiss	h PROB Poiss	DECM	(1) 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**. The Poisson parameter is then $\lambda = n \cdot p_0$ and is calculated automatically. See Poisλ below. Poiss^{-1} returns the maximum number of successes ***m*** for a given probability F_P in **X**, p_0 in **J**, ***n*** in **K**.
Poiss$_P$	h PROB Poiss▸		
Poiss$_u$	h PROB Poiss▾		
Poiss^{-1}	h PROB Poiss⁻¹		

Name	Keys to press	in modes	Remarks (see *above* for general information)
Poisλ	h PROB Poisλ	DECM	(1) Poisson distribution with ***g*** in **X** (as in Poiss) but with the Poisson parameter ***λ*** in **J**. Poisλ$_P$ [50] computes $P_P(g;\lambda)=\frac{\lambda^g}{g!}e^{-\lambda}$ and Poisλ returns $F_P(m;\lambda)=\sum_{g=0}^{m}P_P(g;\lambda)$ with the maximum number of successes ***m*** in **X**. Poisλ^{-1} returns ***m*** for a given probability ***F$_P$*** in **X** and ***λ*** in **J**.
Poisλ$_P$	h PROB PoisλP		
Poisλ$_u$	h PROB Poisλu		
Poisλ $^{-1}$	h PROB Poisλ-1		
PopLR	h P.FCN PopLR	¬α	(0) Pops the local registers allocated to the current routine without returning. See LOCR and RTN.
PowerF	h MODE PowerF	All	(0) Selects the power curve fit model $y=a_0x^{a_1}$.
PRCL	h P.FCN PRCL	¬α	(0) Copies the current program (from flash or *RAM*) and appends it to *RAM*, where it can then be edited (see *above*). Allows duplicating programs in *RAM*. Will only work with enough space at destination.
	RCL	CAT open	
PRIME?	h TEST PRIME?	¬α	(0) Checks if the absolute value of the integer part of ***x*** is a prime. The method is believed to work for integers up to $9\cdot10^{18}$.
	h PROB	DECM	Catalog. See *below*.
PROFRC	f a b/c	¬α	(1) Sets fraction mode like in the *HP-35S*, allowing only proper fractions or mixed numbers in display. Converts ***x*** according to the settings by DEN… into a proper fraction, e.g. 1.25 or $^5/_4$ into $1\,^1/_4$. Absolute decimal equivalents of ***x*** must not exceed 100,000. Compare IMPFRC.
PROMPT	h P.FCN PROMPT	¬α	(0) Displays ***alpha*** and stops program execution. See *above* for more.
PSE	h PSE ***n***	¬α	(0) With a program running, refreshes the display and pauses program execution for ***n*** ticks (see TICKS), with $0 \le n \le 99$. The pause will terminate as soon as you press a key.

[50] Poiss$_P$ corresponds to POISSON(***g; λ;* 0**) and Poiss to POISSON(***g; λ;* 1**) in MS Excel.

Name	Keys to press	in modes	Remarks (see *above* for general information)
PSTO	h P.FCN PSTO	¬PR, ¬α	(0) Copies the current program from *RAM* and appends it to the flash library. The program must have at least one LBL statement with an alphanumeric global label (preferably at its beginning). If a program with the same label already exists in the library it will be deleted first. Alphanumeric labels contained in flash may be browsed by CAT (see *below*) and called by XEQ.
PUTK	h P.FCN PUTK ***a***	¬α	(0) 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.
	h P.FCN	¬α	Catalog. See *below*.
RAD	g RAD	DECM	(0) Sets angular mode to *radians*.
RAD→	h X.FCN RAD→	DECM	(1) Takes ***x*** as *radians* and converts it to the angular mode currently set.
RAN#	f RAN#	DECM	(-1) Returns a random number between 0 and 1 like RAN in the *HP-42S*.
		Integer	(-1) Returns a random bit pattern for the word size set.
RCL	RCL ***s***	¬α	(-1) See the *addressing table above* for ^{C}RCL.
RCLM	RCL MODE ***s***	¬α	(0) Recalls mode settings stored by STOM as described *above*.
RCLS	h P.FCN RCLS ***s***	¬α	Recalls 4 or 8 values from a set of registers starting at address ***s***, and pushes them on the stack. This is the converse command of STOS.
RCL+	RCL + ***s***	¬α	(1) Recalls the content of the source ***s***, executes the specified operation and pushes the result on the stack. E.g. RCL–12 subtracts ***r12*** from ***x*** and displays the result (acting like RCL 1 2 −, but without losing a stack level). In analogy, ^{C}RCL–12 subtracts ***r12*** from ***x*** and ***r13*** from ***y***.
RCL–	RCL − ***s***		
RCL×	RCL × ***s***		
RCL/	RCL / ***s***		
RCL↑	RCL ▲ ***s***	¬α	(1) RCL↑ (↓) replaces ***x*** with the maximum (minimum) of the values in ***s*** and **X**.
RCL↓	RCL ▼ ***s***		
RDP	h X.FCN RDP ***d***	DECM	(1) Rounds ***x*** to ***d*** decimal places ($0 \le d \le 99$), taking the RM setting into account. See RM and compare RSD.

Name	Keys to press	in modes	Remarks (see *above* for general information)
RDX,	h MODE RDX,	All	(0) Sets the decimal mark to a comma.
	h ./,	DECM	(0) Toggles the radix mark.
RDX.	h MODE RDX.	All	(0) Sets the decimal mark to a point.
REALM?	h TEST REALM?	¬α	(0) Tests if your *WP 34S* is in real mode (DECM).
RECV	h P.FCN RECV	¬α	(0) Prepares your *WP 34S* for receiving data via serial I/O. See SEND… and *Appendix A* for more.
REGS	h MODE REGS ***n***	All	(0) Specifies the number of global general purpose registers wanted. With REGS 100 you get the default state (**R00 – R99**), REGS 0 leaves not even a single such register for use.
REGS?	h P.FCN REGS?	¬α	(-1) Returns the number of global general purpose registers allocated (0 … 100).
RESET	h P.FCN RESET	¬PR, ¬α	After confirmation, executes CLALL and resets all modes to start-up default, i.e. 24h, 2COMPL, ALL 00, DBLOFF, DEG, DENANY, DENMAX 9999, D.MY, E3ON, LinF, LocR 0, LZOFF, PROFRC, RDX**.**, REGS 100, SCIOVR, SEPON, SSIZE4, WSIZE 64, and finally DECM. See these commands for more information.
RJ	h X.FCN RJ	Integer	(-1) Right justifies, in analogy to LJ on the *HP-16C*. **Example**: 101100_2 RJ results in $y = 1011_2$ and $x = 2$. See LJ.
RL	h X.FCN RL ***n***	Integer	(1) Works like ***n*** consecutive RLs / RLCs on the *HP-16C*, similar to RLn / RLCn there. For RL, $0 \le n \le 63$. For RLC, $0 \le n \le 64$. See *above* for details of rotating. RL 0 / RLC 0 execute as NOP.
RLC	h X.FCN RLC ***n***		

Name	Keys to press	in modes	Remarks (see *above* for general information)
RM	h MODE RM ***m***	All	(0) 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' [51]). 2: round half down: 0.5 rounds down. 3: round up: rounds away from 0. 4: round down: rounds towards 0 (truncates). 5: ceiling: rounds towards +∞. 6: floor: rounds towards –∞.
RMDR	h RMDR	¬α	(2) Returns the remainder of a division. Equals RMD on the *HP-16C* but works for real numbers as well. See *above* for examples. Compare MOD.
RM?	h P.FCN RM?	¬α	(-1) Returns the floating point rounding mode set. See RM for more.
ROUND	g RND	¬α	(1) Rounds ***x*** using the current … … display format like RND in the *HP-42S*.
		FRC	… denominator like RND in the *HP-35S* fraction mode.
ROUNDI	h X.FCN ROUNDI	¬α	(1) Rounds ***x*** to next integer. ½ rounds to 1.
RR	h X.FCN RR ***n***	Integer	(1) Works like ***n*** consecutive RRs / RRCs on the *HP-16C*, similar to RRn / RRCn there. For RR, $0 \le n \le 63$. For RRC, $0 \le n \le 64$. See *above* for details of rotating. RR 0 / RRC 0 execute as NOP.
RRC	h X.FCN RRC ***n***		
RSD	h X.FCN RSD ***d***	DECM	(1) Rounds ***x*** to ***d*** significant digits, taking the RM setting into account. See RM and compare RDP.
RTN	g RTN	PR	(0) Last command in a typical 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 XEQ instruction that called said routine. If there is none, program execution halts and the program pointer is set to step 000.
		¬PR	(0) Resets the program pointer to the beginning of current program. If the current program is in *FM*, the program pointer will be set to step 000 in *RAM*.

[51] Translates to "kaufmännische Rundung" in German.

Name	Keys to press	in modes	Remarks (see *above* for general information)
RTN+1	h P.FCN RTN+1	¬α	(0) Works like RTN, but moves the program pointer two steps behind the XEQ instruction that called said routine. Halts if there is none.
R-CLR	h P.FCN R–CLR	DECM	(0) Interprets ***x*** in the form `sss.nn` . Clears ***nn*** registers starting with number ***sss*** . If ***nn*** = 0 , it will clear the maximum available. **Example:** For ***x*** = 34.567 , R-CLR will clear **R34** through **R89**. **ATTENTION**: For ***sss*** ∈ [0; 99] and ***nn*** = 0 , clearing will stop at the highest allocated global numbered register. For ***sss*** ∈ [100; 111] and ***nn*** = 0 , clearing will stop at **K**. For ***sss*** ≥ 112 and ***nn*** = 0 , clearing will stop at the highest allocated local register.
R-COPY	h P.FCN R–COPY	DECM	(0) Interprets ***x*** in the form `sss.nnddd` . Takes ***nn*** registers starting with number ***sss*** and copies their contents to ***ddd*** etc. If ***nn*** = 00 , it will take the maximum available. **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***| . Destination will be in *RAM* always. **ATTENTION**: The advice at R-CLR applies, but with 'clearing' replaced by 'copying'.
R-SORT	h P.FCN R–SORT	DECM	(0) Interprets ***x*** in the form `sss.nn` . Sorts the contents of ***nn*** registers starting with number ***sss*** . If ***nn*** = 0 , it will sort the maximum available. **Example:** Assume ***x*** = 49.0369 , ***r49*** = 1.2 , ***r50*** = –3.4 , and ***r51*** = 0 ; then R-SORT will return ***r49*** = -3.4 , ***r50*** = 0 , and ***r51*** = 1.2. **ATTENTION**: The advice at R-CLR applies, but with 'clearing' replaced by 'sorting'.
R-SWAP	h P.FCN R–SWAP	DECM	(0) Works like R-COPY but swaps the contents of source and destination registers.
R→D		DECM	(1) 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 (see *above* for general information)
s	g s	DECM	(-2) Takes the statistical sums accumulated, calculates the sample standard deviations s_y and s_x and pushes them on the stack. See *below* for the formula.
SAVE	h P.FCN SAVE	¬PR, ¬α	(0) Saves user program space, registers and system state to flash memory, and returns Saved. Recall your backup by LOAD. See *Appendix A* for more.
SB	h X.FCN SB *n*	Integer	(1) Sets the specified bit in *x* .
SCI	h SCI *n*	¬α	(0) Sets scientific display format.
SCIOVR	h SCI ENTER↑	¬α	(0) Defines that 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	(1) Shifts digits left (right) by *n* decimal positions, equivalent to multiplying (dividing) *x* by 10^n .
SDR	h X.FCN SDR *n*		
SEED	h STAT SEED	DECM	(0) Stores a seed for random number generation.
SENDA SENDP SENDR SENDΣ	h P.FCN SENDA etc.	¬α	(0) Commands for serial I/O: SENDA sends all *RAM* data, SENDP the program memory, SENDR the global general purpose registers, and SENDΣ the summation registers, respectively, to the device connected. See RECV and *Appendix A* for more.
SEPOFF	h MODE SEPOFF	All	(0) Toggle the digit group separators for integers. Points or commas will be displayed every … … four digits in bases 2 and 4, … two digits in base 16, … three digits in all other integer bases.
	h ./,	Integer	
SEPON	h MODE SEPON	All	
SERR	h STAT SERR	DECM	(-2) Takes the statistical sums accumulated, calculates and returns the standard errors $s/\sqrt{n}$ (i.e. the standard deviations of $\bar{x}$ and $\bar{y}$).
$SERR_w$	h STAT SERR_w	DECM	(-1) Returns the standard error $s/\sqrt{\sum y_i}$ for weighted data, i.e. the standard deviation of $\bar{x}_w$.
SETCHN	h MODE SETCHN	All	(0) Sets some regional preferences (see *above*).

Name	Keys to press	in modes	Remarks (see *above* for general information)
SETDAT	h MODE SETDAT	All	(0) Sets the date for the real time clock (the emulator takes this information from the PC clock).
SETEUR SETIND SETJPN	h MODE SETEUR etc.	All	(0) Sets some regional preferences (see *above*).
SETTIM	h MODE SETTIM	All	(0) Sets the time for the real time clock (the emulator takes this information from the PC clock).
SETUK SETUSA	h MODE SETUK etc.	All	(0) Sets some regional preferences (see *above*).
SF	f SF ***n***	¬α	(0) Sets the flag specified.
SHOW	g SHOW	¬α	Browser. See *below*.
SIGN	h X.FCN SIGN	¬α	(1) Returns 1 for $x > 0$, –1 for $x < 0$, and 0 for $x = 0$ or non-numeric data. Corresponds to $\text{signum}(x)$ for numeric input.
CSIGN	CPX h X.FCN CSIGN	DECM	(1) Returns the unit vector of $x + iy$ in **X** and **Y**.
SIGNMT	h MODE SIGNMT	All	(0) Sets sign-and-mantissa mode for integers.
SIN	f SIN	DECM	(1) Returns the sine of the angle in **X**.
SINC	h X.FCN SINC	DECM	(1) Returns $\frac{\sin(x)}{x}$.
SINH	f HYP SIN	DECM	(1) Returns the hyperbolic sine of x.
SKIP	h P.FCN SKIP ***n***	¬α	(0) Skips ***n*** program steps forwards ($0 \le n \le 255$). So e.g. SKIP 2 skips over the next two steps, going e.g. from step 123 to step 126. If SKIP attempts to cross an END, an error is thrown. **ATTENTION**: If you edit a section of your routine that is crossed by one or more BACK, SKIP, or CASE jumps, this may well result in a need to manually maintain all those statements individually.
SL	h X.FCN SL ***n***	Integer	(1) Works like ***n*** (≤ 63) consecutive SLs on the *HP-16C*. See *above* for details. SL 0 executes as NOP.
SLOW	h MODE SLOW	All	(0) Sets the processor speed to 'slow'. This is also automatically entered for low battery voltage (see *above*). Compare FAST.

Name	Keys to press	in modes	Remarks (see *above* for general information)
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 the *HP-15C*. This also means SLV acts as a test, so the next program step will be skipped if SLV fails. Please refer to the *HP-15C Owner's Handbook* (Section 13 and Appendix D) for more information about automatic root finding.
SLVQ	h X.FCN SLVQ	DECM	Solves the quadratic equation $ax^2+bx+c=0$, with its real parameters on the input 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 complex conjugate of the first – see CONJ). If run directly from the keyboard, the complex indicator C is lit then – in a program, the step after SLVQ will be skipped. In either case, SLVQ returns ***r*** in **Z**. Higher stack levels are kept unchanged. **L** will contain equation parameter ***c***.
SMODE?	h P.FCN SMODE?	¬α	(-1) 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 SPEC?	¬α	(0) True if ***x*** is 'special', i.e. infinite or non-numeric.
SR	h X.FCN SR ***n***	Integer	(1) Works like ***n*** (≤ 63) consecutive SRs on the *HP-16C*. See *above* for details. SR 0 executes as NOP.
sRCL	h X.FCN sRCL ***s***	¬α	(-1) Assumes the source ***s*** contains standard single precision data and recalls them as such. See *below*.
SSIZE4	h MODE SSIZE4	All	Set the stack size to 4 or 8 levels, respectively. See *above*. Note register contents will remain unchanged in this operation. The same will happen if stack size is modified by any other operation (e.g. by RCLM).
SSIZE8	h MODE SSIZE8		

Name	Keys to press	in modes	Remarks (see *above* for general information)
SSIZE?	h P.FCN SSIZE?	¬α	(-1) Returns the number of stack levels allocated.
	h STAT	DECM	Catalog. See *below*.
	h STATUS	¬α	Browser. See *below*.
STO	STO ***d***	¬α	(0) See the *addressing table above* for ^{C}STO.
STOM	STO MODE ***s***	¬α	(0) Stores mode settings for later use as described *above*. RCLM recalls them.
STOP	R/S	PR	(0) Stops program execution. May be used to wait for input, for example.
STOPW	CPX R/S h X.FCN STOPW	DECM, ¬PR	Stopwatch application following the timer of the *HP-55*. See *below* for a detailed description.
STOS	h P.FCN STOS ***d***	¬α	(0) Stores all stack levels in a set of 4 or 8 registers, starting at destination ***d***. See RCLS.
STO+	STO + ***d***	¬α	(0) 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 key-strokes RCL 12 x⇄y − STO 12 would do, but the stack remains unchanged.
STO–	STO − ***d***		
STO×	STO × ***d***		
STO/	STO / ***d***		
STO↑	STO ▲ ***d***	¬α	(0) STO↑ (↓) takes the maximum (minimum) of the values in ***d*** and **X** and stores it.
STO↓	STO ▼ ***d***		
SUM	h STAT SUM	DECM	(-2) Recalls the linear sums ***Σy*** and ***Σx***. Useful for elementary vector algebra in 2D.
	h SUMS	DECM	Catalog. See *below*.
s_w	h STAT Sw	DECM	(-1) Calculates the standard deviation for weighted data (where the weight ***y*** of each data point ***x*** was entered via Σ+). See *below* for the formula.
s_{XY}	h STAT Sxy	DECM	(-1) Calculates the sample covariance for the two data sets entered via Σ+. It depends on the fit model selected. See *below* for the formula See COV for the population covariance.
TAN	f TAN	DECM	(1) Returns the tangent of the angle in **X**.
TANH	f HYP TAN	DECM	(1) Returns the hyperbolic tangent of ***x***.
	h TEST	DECM	Catalog. See *below*.

Name	Keys to press	in modes	Remarks (see *above* for general information)
TICKS	h P.FCN TICKS	¬α	(-1) Returns the number of ticks from the real time clock at execution time. With the quartz crystal built-in, 1 tick = 0.1 s. Without, it may be 10% more or less. So the quartz crystal is inevitably required if the clock is to be useful in long range.
TIME	h X.FCN TIME	DECM, α	(-1) Recalls the time from the real time clock at execution, displaying it in the format `hh.mmss` in 24h-mode. Choose FIX 4 for best results.
T_n	h X.FCN T_	DECM	(2) Chebychev's polynomials of first kind. See *below* for details.
TOP?	h TEST TOP?	PR	(0) Returns `false` if TOP? is called in a subroutine, `true` if the program-running flag is set and the subroutine return stack pointer is clear.
TRANSP	h MATRIX TRANSP	DECM	(1) Takes a matrix *descriptor* ***x*** and returns the descriptor of its transpose. The transpose is done in-situ and does not require any additional memory.
$t_p(x)$	h PROB t_(x)	DECM	(1) Student's t distribution. $t_u(x)$ equals Q(t) and $t^{-1}(p)$ equals t_p in the *HP-21S*. The degrees of freedom are stored in **J**. See *above* for an application.
$t_u(x)$	h PROB t_(x)		
t(x)	h PROB t(x)		
$t^{-1}(p)$	h PROB t-1(p)		
t⇆	h P.FCN t⇆ *r*	¬α	Swaps ***t*** and the contents of ***r***, in analogy to ***x*⇆** .
ULP	h X.FCN ULP	¬α	(1) 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 your *WP 34S* in the mode set. Thus 1 is returned in integer mode.
U_n	h X.FCN U_	DECM	(2) Chebychev's polynomials of second kind. See *below* for details.
UNSIGN	h MODE UNSIGN	All	(0) Sets unsigned mode like UNSGN on the *HP-16C*.

Name	Keys to press	in modes	Remarks (see *above* for general information)
VERS	h X.FCN VERS	¬PR	(0) Shows your firmware version and build number.
VIEW	h VIEW **s**	¬α	(0) Shows the content of address ***s*** until the next key is pressed. See *above* for more.
VIEWα	h P.FCN VIEWα	¬α	(0) Displays ***alpha*** in the top row and - - - in the bottom row until next key is pressed (compare to AVIEW in the *HP-42S*). See *above* for more.
	h VIEW (−)	α	
VWα+	h VIEW **s**	α	(0) Displays ***alpha*** in the top row and the content of address ***s*** in the bottom row until the next key is pressed. See *above* for more.
	h P.FCN VWα+	¬α	
WHO	h X.FCN WHO	¬α	(0) Displays credits to the brave men who made this project work.
WDAY	h X.FCN WDAY	DECM	(1) Takes ***x*** as a date in the format selected, returns the name of the day in the dot matrix and a corresponding integer in the numeric display (Monday = 1, Sunday = 7) [52].
W_m	h X.FCN W_m	DECM	(1) W_p returns the principal branch of Lambert's W for given $x \geq -1/e$. W_m returns its negative branch.
W_p	h X.FCN W_p		
W^{-1}	h X.FCN W-1	DECM	(1) Returns ***x*** for given W_p (≥ -1). See there.
Weibl	h PROB Weibl	DECM	(1) Weibull distribution with its shape parameter ***b*** in **J** and its characteristic lifetime ***T*** in **K**. See *below* for details. Weibl $^{-1}$ returns the survival time t_s for given probability F_W , ***b*** in **J** and ***T*** in **K**.
$Weibl_P$	h PROB Weibl_p		
$Weibl_u$	h PROB Weibl_u		
Weibl $^{-1}$	h PROB Weibl-1		
WSIZE	h MODE WSIZE ***n***	All	(0) Works like on the *HP-16C*, but with the parameter $1 \leq n \leq 64$ trailing 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 P.FCN WSIZE?	¬α	(-1) Recalls the word size set.

[52] These numbers correspond to Chinese weekdays 1 to 6 directly. For Portuguese days ('segunda feira' etc.), add 1 to days 1 to 5.

Name	Keys to press	in modes	Remarks (see *above* for general information)
x^2	g x^2	¬α	(1)
x^3	h X.FCN x^3	¬α	(1)
XEQ	XEQ ***label***	PR	(0) Calls the respective subroutine.
		¬PR, ¬α	(0) Executes the respective program.
	A , B , C , or D (you may need f for reaching these hotkeys in integer bases >10.)	PR	(0) Calls the respective subroutine, so e.g. XEQ C will be inserted when C is pressed.
		¬PR, ¬α	(0) Executes the respective program if defined.
XEQα	h P.FCN XEQα	¬α	(0) Takes the first three characters of ***alpha*** (or all if there are fewer than three) as a label and calls or executes the respective routine.
XNOR	h X.FCN XNOR	¬α	(2) Works in analogy to AND. See *there*.
XOR	h XOR	¬α	(2) Works in analogy to AND. See *there*.
XTAL?	h TEST XTAL?	¬α	(0) Tests for presence of the crystal necessary for a precise real time clock (think of Xmas).
$\bar{x}$	f $\bar{x}$	DECM	(-2) Returns the arithmetic means of the x- and y-data accumulated. See also s, SERR, and σ .
$\bar{x}_g$	h STAT $\bar{x}_g$	DECM	(-2) Returns the geometric means of the data accumulated. See *below* for the formula. See also ε , ε_m , and ε_P .
$\bar{x}_w$	h STAT $\bar{x}_w$	DECM	(-1) Returns $\bar{x}_w = \sum xy / \sum y$, i.e. the arithmetic mean for weighted data (where the weight ***y*** of each data point ***x*** was entered via Σ+). See also s_w and $SERR_w$.
$\hat{x}$	h STAT $\hat{x}$	DECM	(1) Returns a forecast ***x*** for a given ***y*** (in **X**) following the fit model chosen. See L.R. for more.
	h X.FCN	All	Catalog. See *below*.
x !	h !	DECM	(1) Returns $\Gamma(x + 1)$.
		Integer	(1) Returns the factorial $n!$.
x → α	h X.FCN x→α	All	(0) Interprets ***x*** as a character code. Appends the respective character to ***alpha***, similar to XTOA in the *HP-42S*.

Name	Keys to press	in modes	Remarks (see *above* for general information)
***x*⇆**	h x≷ ***r***	¬α	Swaps ***x*** and the contents of address ***r*** , in analogy to ***x*⇆*y*** . See *above* for ᶜ***x*⇆** . Listings will look like x⇆ J, x⇆.12, x⇆→12, etc.
x⇆ Y	x≷y	¬α	Swaps ***x*** and ***y***, performing Re⇆Im if a complex operation was executed immediately before.
ᶜ*x*⇆ *Z*	CPX x≷y	¬α	Swaps x_c and y_c . See *above* for details.
x < ?	h TEST x<_? ***a***	¬α	(0) Compare ***x*** with ***a***. E.g. h TEST x< ? K compares ***x*** with ***k*** , and will be inserted as x<? K in a program. See the examples given *above* for more. x ≈ ? will be true if the rounded values of ***x*** and ***a*** are equal (see ROUND). The signed tests x=+0? and x=–0? are meant for integer modes 1COMPL and SIGNMT, and for DECM if flag **D** is set. In all these cases, e.g. 0 divided by –7 will display –0. CPX f x = ? ***a*** and CPX g x ≠ ? ***a*** compare the complex number $x + i\,y$ as explained *above*.
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***		
$\sqrt[x]{y}$	h X.FCN ˣ√y	¬α	(2)
YEAR	h X.FCN YEAR	DECM	(1) Assumes ***x*** containing a date in the format selected and extracts the year.
y*^*x	f y^x	¬α	(2) In integer modes, ***x*** must be ≥ 0.
	C	¬(α, 13…16)	(2) Shortcut working if label C is not defined.
ŷ	f ŷ	DECM	(1) Returns a forecast ***y*** (in **X**) for a given ***x*** following the fit model chosen. See L.R. for more.
Y.MD	h MODE Y.MD	All	(0) Sets the format for date display.
y⇆	h P.FCN y⇆ ***r***	¬α	Swaps ***y*** and the contents of ***r*** , in analogy to ***x*⇆**.
z⇆	h P.FCN z⇆ ***r***	¬α	Swaps ***z*** and the contents of ***r*** , in analogy to ***x*⇆**.

Name	Keys to press	in modes	Remarks (see *above* for general information)
α	f α	PR & ¬α	Turns on alpha mode for keyboard entry of alpha constants. Each subsequent character (e.g. '?') will be stored in one program step (like α? here) and appended to ***alpha*** in program execution.
		PR & α	Turns on alpha group mode for direct entry of up to three characters in one program step taking two words. Your *WP 34S* will display α' in the top line. Now enter the characters you want to append to ***alpha***. **Example:** Entering f α T f ↑ e s f α t h PSE 1 will result in two program steps stored: α'Tes' α't 1' and Test 1 appended to ***alpha*** in program execution [53].
		¬PR, ¬α	Enters alpha mode for appending characters to ***alpha***. To start a new string, use CLα first.
		¬PR & α	Leaves alpha mode.
αDATE	h X.FCN αDATE	¬integer	(0) 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 αDAY	¬integer	(0) Takes ***x*** as a date, recalls the name of the respective day and appends its first three letters to ***alpha***.
αGTO	h P.FCN αGTO ***r***	¬α	(0) Interprets the contents of ***r*** as character code. Takes the first three characters of the converted code (or all if there are fewer than three) as an alpha label and positions the program pointer to it.
αIP	h X.FCN αIP	All	(0) Appends the integer part of ***x*** to ***alpha***, similar to AIP in the *HP-42S*.
αLENG	h X.FCN αLENG	All	(-1) Returns the number of characters found in ***alpha***, like ALENG in the *HP-42S*.
αMONTH	h X.FCN αMONTH	¬integer	(0) Takes ***x*** as a date, recalls the name of the respective month and appends its first 3 letters to ***alpha***.

[53] Note alpha group mode is left automatically after three characters are put in, so it must be called again for continuation. Some characters must not be entered at third position (see Appendix E).

Name	Keys to press	in modes	Remarks (see *above* for general information)
αOFF	h P.FCN αOFF	¬α	(0) Work like AOFF and AON in the *HP-42S*, turning alpha mode off and on.
αON	h P.FCN αON	¬α	
αRCL	f RCL ***s***	α	(0) 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	(0) Interprets the content of the source ***s*** as a number, converts it to a string in the format set, and appends this to ***alpha***. **Example**: If said content is 1234 and ENG 2 and RDX**.** are set, then 1.23e3 will be appended.
αRL	h X.FCN αRL ***n***	All	(0) Rotates ***alpha*** by ***n*** characters like AROT in the *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	(0) Works like αRL but rotates to the right.
αSL	h X.FCN αSL ***n***	All	(0) Shifts the ***n*** leftmost characters out of ***alpha***, like ASHF in the *HP-42S*. αSL 0 equals NOP.
αSR	h X.FCN αSR ***n***	All	(0) Works like αSL but takes the ***n*** rightmost characters instead.
αSTO	f STO ***d***	α	(0) Stores the first (i.e. leftmost) 6 characters of ***alpha*** in destination ***d*** . Stores all if there are fewer than six.
	h X.FCN αSTO ***d***	¬α	
αTIME	h X.FCN αTIME	¬integer	(0) Takes ***x*** as a decimal time and appends it to ***alpha*** in the format `hh:mm:ss` according to the time mode selected. See 12h, 24h, and TIME. To append a time stamp to ***alpha***, call TIME αTIME.
αXEQ	h P.FCN αXEQ ***r***	¬α	(0) Interprets the contents of ***r*** as character code. Takes the first three characters (or all if there are fewer than three) of the converted code as an alpha label and calls or executes the respective routine.
α → x	h X.FCN α→x	All	(-1) Returns the character code of the leftmost character in ***alpha*** and removes this character from ***alpha***, like ATOX in the *HP-42S*.

Name	Keys to press	in modes	Remarks (see *above* for general information)
β	h X.FCN β	DECM	(2) 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 to avoid ambiguity.
Γ	h X.FCN Γ	DECM	(1) Returns $\Gamma(x)$. Additionally, h ! calls $\Gamma(x+1)$.
γ_{XY}	h X.FCN γXY	DECM	(2) Returns the lower or upper incomplete gamma function , respectively. See *below* for details.
Γ_{XY}	h X.FCN ΓXY		
ΔDAYS	h X.FCN ΔDAYS	DECM	(2) Assumes **X** and **Y** containing dates in the format chosen and calculates the number of days between them like in the *HP-12C*.
Δ%	g Δ%	DECM	(2) Returns $100\cdot\frac{x-y}{y}$ like %CH in the *HP-42S*.
ε	h STAT ε	DECM	(-2) Returns the scattering factors ε_y and ε_x for log-normally distributed data. This ε_x works for the geometric mean $\bar{x}_g$ in analogy to the standard deviation s for the arithmetic mean $\bar{x}$ but multiplicative instead of additive. See *below* for the formula.
ε_m	h STAT εm	DECM	(-2) Works like ε but returns the scattering factors of the two geometric means $\varepsilon_m=\varepsilon^{1/\sqrt{n}}$.
ε_P	h STAT εP	DECM	(-2) Works like ε but with a denominator ***n*** instead of ***n – 1*** , returning the scattering factors of the two populations.
ζ	h X.FCN ζ	DECM	(1) Returns Riemann's Zeta. See *below* for details..
π	h π	DECM	(-1) Recalls π.
$^{c}\pi$	CPX π	DECM	(-2) Recalls π into **X** and clears **Y**.
Π	f π ***label***	DECM	(1) Computes a product using the routine specified. Initially, **X** contains the loop control number in the format `ccccc.fffii` , and the product is set to 1. Each run through the routine specified by ***label*** computes a factor. At its end, this factor is multiplied with said product; the operation then decrements `ccccc` by `ii` and runs said routine again if then `ccccc` ≥ `fff` , else returns the resulting product in **X**.

Name	Keys to press	in modes	Remarks (see *above* for general information)
σ	h STAT σ	DECM	(-2) Works like s but returns the standard deviations of the two populations instead.
Σ	g Σ ***label***	DECM	(1) Computes a sum using the routine specified. 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 by ***label*** computes a summand. At its end, this summand is added to said sum; the operation then decrements `cccccc` by `ii` and runs said routine again if then `cccccc` ≥ `fff` , else returns the resulting sum.
$\Sigma\ln^2 x$ $\Sigma\ln^2 y$ $\Sigma\ln x$ $\Sigma\ln xy$ $\Sigma\ln y$ $\Sigma x\ln y$ $\Sigma y\ln x$	h SUMS Σln²x etc.	DECM	(-1) Recall the respective statistical sums. These sums are necessary for curve fitting models beyond pure linear. Calling them by name significantly improves program readability. Note these sums are stored in special registers in your *WP 34S*. **ATTENTION:** Depending on input data, some or all of these sums may become non-numeric.
σ_w	h STAT σw	DECM	(-1) Works like s_w but returns the standard deviation of the population instead. Formula: $\sigma_w = +\sqrt{\frac{\sum y_i (x_i - \bar{x}_w)^2}{\sum y_i}}$
Σx Σx^2 $\Sigma x^2 y$ Σxy Σy Σy^2	h SUMS Σx etc.	DECM	(-1) Recall the respective statistical sums. These sums are necessary for basic statistics and linear curve fitting. Calling them by name significantly improves program readability. Note these sums are stored in special registers of your *WP 34S*.
Σ+	h Σ+	DECM	Adds a data point to the statistical sums.
	A	DECM	Shortcut working if label A is not defined.
Σ−	h Σ−	DECM	Subtracts a data point from the statistical sums.

Both functions disable stack lift as usual. Both may be used for 2D vector adding and subtracting as well.

Name	Keys to press	in modes	Remarks (see *above* for general information)
$\Phi_u(x)$	h PROB $\Phi_u(x)$	DECM	(1) Standard normal *error probability* $\Phi_u(x)=\int_x^\infty \varphi(\tau)d\tau$, equaling Q in the *HP-32E* and Q(z) in the *HP-21S*.
φ(x)	h PROB $\Phi(x)$	DECM	(1) Standard normal *pdf* $\varphi(x)=\frac{1}{\sqrt{2\pi}}e^{-x^2/2}$.
Φ(x)	f [Φ]	DECM	(1) Standard normal *cdf* $\Phi(x)=\int_{-\infty}^{x}\varphi(\tau)d\tau$.
Φ^{-1}(p)	g [Φ^{-1}]	DECM	(1) Equals Q^{-1} in the *HP-32E* and z_p in the *HP-21S*.
χ^2	h PROB x^2	DECM	(1) Chi square distribution. The *cdf* χ^2 (with its degrees of freedom given in **J**) equals $1 - Q(\chi^2)$, χ^2_u equals $Q(\chi^2)$ and χ^2INV equals χ^2_p in the *HP-21S*.
χ^2INV	h PROB x^2INV		
χ^2_P	h PROB x^2_P		
χ^2_u	h PROB x^2_u		
(-1)^X	h X.FCN $(-1)^x$	¬α	(1) If ***x*** is not a natural number, returns $\cos(\pi \cdot x)$.
+	+	¬α	(2) Returns $y + x$.
	CPX +	DECM	(2) Returns $[x+z,\ y+t,\ \ldots]$. May be used for adding 2D vectors as well.
+/–	+/–	¬α	(1) 'Unary minus', corresponding to $x \cdot (-1)$ or $x_c \cdot (-1)$, respectively.
–	–	¬α	(2) Returns $y - x$.
	CPX –	DECM	(2) Returns $[x-z,\ y-t,\ \ldots]$. May be used for subtracting 2D vectors as well.
×	×	¬α	(2) Returns $y \cdot x$.
	CPX ×	DECM	(2) Returns $[x \cdot z - y \cdot t,\ x \cdot t + z \cdot y,\ \ldots]$. Look at CROSS or DOT for multiplying 2D vectors.
×MOD	h X.FCN ×MOD	Integer	(3) Returns $(z \cdot y) \bmod x$ for $x > 1$, $y > 0$, $z > 0$.

Name	Keys to press	in modes	Remarks (see *above* for general information)
/	/	DECM	(2) Returns y / x .
		Integer	(2) Returns $y \operatorname{div} x$. If $y \bmod x \neq 0$, carry will be set.
	CPX /	DECM	(2) Returns $\left[\frac{x \cdot z + y \cdot t}{z^2 + t^2}, \frac{z \cdot y - x \cdot t}{z^2 + t^2}, \ldots\right]$.
→DATE	h X.FCN →DATE	DECM	(3) Assumes the three components of a date (year, month, and day) supplied 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	(1) Takes x as an angle in the angular mode currently set and converts it to *degrees*. Prefix g may be omitted.
→GRAD	→ GRAD	DECM	(1) Like →DEG, but converts to *gon* (*grad*).
→H	→ f H.d	DECM	(1) Takes x as *hours* or *degrees* in the format `hhhh.mmssdd` as in vintage *HP* calculators and converts it to a decimal time or angle, allowing for using standard arithmetic operations then.
→H.MS	→ f H.MS	DECM	(1) Takes x as decimal *hours* or *degrees* and converts it to the format `hhhh.mmssdd` as in vintage *HP* calculators. For calculations, use H.MS+ or H.MS– then.
→POL	g →P	DECM	Assumes **X** and **Y** containing 2D Cartesian coordinates (x, y) of a point or components of a vector and converts them to the polar coordinates / components (r, ϑ) with the radius $r = \sqrt{x^2 + y^2}$
→RAD	→ RAD	DECM	(1) Works like →DEG, but converts to *radians*.
→REC	f R←	DECM	Assumes **X** and **Y** containing 2D polar coordinates (r, ϑ) of a point or components of a vector and converts them to the Cartesian coordinates or components (x, y) .

Name	Keys to press	in modes	Remarks (see *above* for general information)
⇆	h P.FCN ⇆____	¬α	Shuffles the contents of the bottom four stack levels at execution time. **Examples**: ⇆XXYZ works like ENTER↑, ⇆YZTX works like R↓, ⇆ZTXY works like ^{C}x⇆y, but ⇆ZZZX is possible as well. **ATTENTION / WARNING**: This is a very powerful command though it does not look it. Remember it will not affect the higher levels in an 8-level stack. If you play with this command, you may lose stack contents and can easily make a mess of the stack.
%	f %	DECM	(1) Returns $\frac{x \cdot y}{100}$, leaving **Y** unchanged.
%MG	h X.FCN %MG	DECM	(2) Returns the margin [54] $100 \cdot \frac{x-y}{x}$ in % for a price ***x*** and cost ***y*** , like %MU-Price in the *HP-17B*.
%MRR	h X.FCN %MRR	DECM	(3) 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 %T	DECM	(2) Returns $100 \cdot \frac{x}{y}$, interpreted as % of total.
%Σ	h X.FCN %Σ h STAT %Σ	DECM	(1) Returns $100 \cdot \frac{x}{\sum x}$.
%+MG	h X.FCN %+MG	DECM	(2) Calculates a sales price by adding a margin of ***x*** % to the cost ***y***, as %MU-Price does in the *HP-17B*. Formula: $p_{sale} = \frac{y}{1 - \frac{x}{100}}$ You may use %+MG for calculating net amounts as well. Just enter a negative percentage in ***x***. **Example**: Total billed = 221,82 €, VAT = 19%. What is the net? 221,82 ENTER↑ 19 +/- X.FCN ▲ returns 186,40.

[54] Margin translates to „Handelsspanne“ in German.

Name	Keys to press	in modes	Remarks (see *above* for general information)
√‾	f √x	¬α	(1) If the input is not a perfect square, carry will be set in integer modes.
	D	¬(α, 14, 15, 16)	(1) Shortcut working if label D is not defined.
∫	g ∫y ***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 the *HP-15C*. Please turn to the *HP-15C Owner's Handbook* (Section 14 and Appendix E) for more information about automatic integration and some caveats.
∞?	h TEST ∞?	¬α	(0) Tests x for infinity.
^MOD	h X.FCN ^MOD	Integer	(3) Returns $(z^y) \bmod x$ for $x > 1$, $y > 0$, $z > 0$. **Example**: 10 73 ENTER↑ 55 ENTER↑ 31 X.FCN ▲ XEQ returns 26.
| |	g | |	DECM	(2) Returns $\left(\frac{1}{x}+\frac{1}{y}\right)^{-1}$, being very useful in electrical engineering especially.
🖷ADV	h P.FCN 🖷ADV	¬α	(0) Prints the current contents of the print buffer plus a linefeed. **ATTENTION**: Any printing will only work with an hardware modification (see *below*) or using the calculator emulator in combination with a printer emulator (see *below*). The printer will actually print only when a line feed is sent to it.
🖷CHR	h P.FCN 🖷CHR ***n***	¬α	(0) Sends a single character with the code specified to the printer. Character codes $n > 127$ can only be specified indirectly. 🖷MODE setting will be honored. See 🖷ADV.
🖷PLOT	h P.FCN 🖷PLOT ***n***	¬α	(-1) Prints the graphic data set starting at **R*n***. If its width is 166, the data will be trailed by a line feed. See 🖷ADV and gDIM.

Name	Keys to press	in modes	Remarks (see *above* for general information)
🖶$^{C}r_{XY}$	h P.FCN 🖶rxy *r*	¬α	(0) Prints the register specified and the next one, i.e. prints an entire complex number. A semicolon will separate both components. Works like 🖶r otherwise. **Example**: Assume 🖶MODE 1, SCI 1, ***x*** = –1.2 and ***y*** = 0.34 . Then the output of 🖶rxy X would look like –1.2e0 ; 3.4e–1
🖶DLAY	h MODE 🖶DLAY *n*	All	(0) Takes a delay of ***n*** ticks (see TICKS) to be used with each line feed on the printer. See 🖶ADV.
🖶MODE	h MODE 🖶MODE *n*	All	(0) Sets print mode: 0: Use the printer font and character set wherever possible (default). All characters feature the same width (5 columns + 2 columns spacing). 1: Use the variable pitch display font, resulting in some jitter on the printout but packing more characters in a line. 2: Use the small display font, which allows for packing even more info in a line. 3: Send the output to the serial channel. Works for plain ASCII only – no characters will be translated. Line setup is the same as for serial communication: 9600 *baud*, 8 *bits*, no parity.
🖶PROG	h P.FCN 🖶PROG	¬α	(0) Prints the listing of the current program, one line per step. The current program is the one the program pointer is in at execution time. See 🖶ADV.
🖶r	h P.FCN 🖶r *r*	¬α	(0) Prints the register specified, right adjusted, without labeling the output. If you want a heading label, call 🖶α+ first or use 🖶REGS. See 🖶ADV.
🖶REGS	h P.FCN 🖶REGS	¬α	(1) Interprets ***x*** in the form `sss.nn` . Prints the contents of ***nn*** registers starting with number ***sss*** . Each register takes one line starting with a label. **ATTENTION**: For ***sss*** ∈ [0; 99] and ***nn*** = 0 , printing will stop at the highest allocated global numbered register. For ***sss*** ∈ [100; 111] and ***nn*** = 0 , printing will stop at **K**. For ***sss*** ≥ 112 and ***nn*** = 0 , printing will stop at the highest allocated local register. See also 🖶ADV.

Name	Keys to press	in modes	Remarks (see *above* for general information)
🖶STK	h P.FCN 🖶STK	¬α	(0) Prints the stack contents. Each level prints in one line starting with a label. See 🖶ADV.
🖶TAB	h P.FCN 🖶TAB ***n***	¬α	(0) Positions the print head to print column ***n*** (0 to 165, where ***n*** > 127 can only be specified indirectly). Useful for formatting (in 🖶MODE 1 or 2 in particular). Allows also for printer plots. If ***n*** is less than current position, a linefeed will be entered to reach the new position. See 🖶ADV.
🖶WIDTH	h P.FCN 🖶WIDTH	¬α	(-1) Returns the number of print columns ***alpha*** would take in the print mode set. See 🖶ADV and 🖶MODE. Second use: in 🖶MODE 1 or 2, returns the width of ***alpha*** in pixels (including the last column being always blank) in the specified font.
🖶α	h P.FCN 🖶α	¬α	(0) Appends ***alpha*** to the print line, trailed by a line feed. Compare 🖶α+ and 🖶+α. See 🖶ADV.
🖶α+	h P.FCN 🖶α+	¬α	(0) Sends ***alpha*** to the printer without a trailing line feed, allowing further information to be appended to this line. May be repeated. See also 🖶ADV, 🖶r and 🖶+α.
🖶Σ	h P.FCN 🖶Σ	¬α	(0) Prints the summation registers. Each register prints in one line starting with a label. See 🖶ADV.

🖶+α	h P.FCN 🖶+α	¬α	(0) Appends ***alpha*** to the print line, adjusted to the right and trailed by a line feed. Compare 🖶α and 🖶α+. See 🖶ADV.
			Example: The following program section
			`CLα`
			`α'Left'`
			`α t`
			`🖶α+`
			`CLα`
			`α'Right'`
			`🖶+α`
			will print, if 🖶MODE 1 is set:
			`Left Right`

🖶?	h TEST 🖶?	¬α	(0) Tests if the quartz crystal and the necessary firmware are installed for printing.
🖶#	h P.FCN 🖶# ***n***	¬α	(0) Sends a single byte, without translation, to the printer (e.g. a control code). ***n*** > 127 can only be specified indirectly. 🖶MODE setting will not be honored. See 🖶ADV.

Name	Keys to press	in modes	Remarks (see *above* for general information)
#	h CONST # *n*	PR	Inserts an integer $0 \le n \le 255$ in a single step, thus saving up to two steps and an ENTER.
^{C}#	CPX h CONST # *n*	DECM	Works like # but also clears *y*. The shortcut works only for $1 \le n \le 9$.
	CPX *n*		

Non-programmable Control, Clearing and Information Keys

Keys to press	in modes	Remarks
f 1/x	In CONV	Inverts the current conversion (see *below*).
8 [55]	Asking for confirmation	Answers the question Sure? with **N** for 'no'. Any other input except R/S = **Y** , EXIT or ← for 'no' will be ignored.
	α	Appends an N or n to ***alpha***.
	Else	Enters the digit 8.
ENTER↑	Catalog open	Selects the current item like XEQ below.
	CAT open	Goes to the first routine carrying the label displayed (see *below*).
	α	Leaves alpha mode.
	Else	Acts like the command ENTER described above.
EXIT	Asking for confirmation	Answers the question Sure? with **N** for 'no'. Any other input except R/S = **Y** , 8 or ← for 'no' will be ignored.
	Catalog or browser open	Leaves the catalog or browser without executing anything.
	Command 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.
	PR	Leaves programming mode like P/R . See below.
	α	Leaves alpha mode like ENTER↑. See above.
	Else	Does nothing.

[55] The mode conditions specified here will be checked top down for this command at execution time:

If there is an open question for confirmation, the input will be checked and taken for it;
else if alpha mode is set, an 'N' will be appended to ***alpha***;
else the digit 8 will be inserted.

This method holds for all commands listed here using this triangular symbolic.

Keys to press	in modes	Remarks
ON	Calculator off	Turns your *WP 34S* on.
	Else	ON-key combinations are found in the appendices. Of them, most important are ON + + or − for adjusting display contrast.
h P/R	¬α	Toggles programming mode (PR).
R/S	Asking for confirmation	Answers the question Sure? with **Y** for 'yes'. Any other input except 8 or EXIT or ← for 'no' will be ignored..
	CAT open, ¬PR	Runs the program whose label is displayed (compare XEQ in this table and see further *below*).
	Program running	Stops program execution immediately. Stopped will be shown until the next keystroke. Press R/S again to resume execution.
	¬PR, ¬α	Runs the current program or resumes its execution starting with the current step.
	α	Appends a Y or y to ***alpha***.
	PR	Enters the command STOP described above.
XEQ	Catalog open	Selects the item currently displayed and exits, executing the respective command. See *below*.
	CAT open ¬PR	Runs the first routine found carrying … … the label displayed (see *below*).
	CAT open PR	Inserts a step XEQ ***label*** referring to ...
	Else	Acts like the command XEQ described above.
←	Asking for confirmation	Answers the question Sure? with **N** for 'no'. Any other input except R/S = **Y** , 8 or EXIT for 'no' will be ignored.
	Catalog or browser open	Leaves the catalog or browser like EXIT above.
	Command input pending	Deletes the last digit or character put in. If there is none yet, cancels the pending command like EXIT above.
	α	Deletes the rightmost character in ***alpha***.
	PR	Deletes current program step.
	Else	Acts like the command CLx described above.
f ◀ / g ▶	Integer	Shifts the display window to the left / right like in the *HP-16C*. Helpful while working with small bases. See *above*.
	DECM	Shows the full mantissa until the next key is pressed. See *above*. Compare the command SHOW in previous calculators.

Keys to press	in modes	Remarks
→ f 2 → 8 → 16	DECM	Shows ***x*** as an integer to base 2, 8, or 16, respectively. Returns to the base set with the next keystroke. Prefix g may be omitted here.
f ↑	Catalog open	Calls the character 🖷.
	α	Toggles upper and lower case (the latter is indicated by ⬇).
	Else	Calls 🖷r X (see there).
▲ / ▼	Status display open	Goes to previous / next status window. See *below*.
	Catalog or browser open	Goes to previous / next item therein.
	α	Scrolls the display window six characters to the left/right in ***alpha*** if possible. If fewer than six characters are beyond the limits of the display window on this side, the window will be positioned to the beginning/end of string. Useful for longer strings.
	Else	Acts like the command BST / SST in the *HP-42S*. I.e. browses programs in PR, where ▲ / ▼ will repeat with 5Hz when held down for longer than 0.5s. – Out of PR, SST will also execute the current program step, but the keys will not repeat.

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. Note you cannot enter more than 12 digits in the mantissa.
		in address-ing	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, 16	Numeric input for digits >10. See *above* for more information.

Character	Keys to press	in modes	Remarks
A … Z	A … Z (grey print)	in address-ing	Register input. See the *virtual keyboard* above for the letters applicable.
		α	Appends the respective Latin letter to ***alpha***. Use f ⇧ to toggle between cases.
E	EEX	DECM & ¬FRC	Works like E in the Pioneers.
i	CPX .	DECM & ¬FRC	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***. Use f ⇧ to toggle between cases. See *above* for more.
(	f ◀(	α	Appends the respective symbol to ***alpha***.
)	g)▶		
+	f +		
–	f −		
x	f ×		
/	Second .	DECM	A second . in input turns to fraction mode and is interpreted as explained below. Note you cannot enter EEX after you entered . twice – but you may delete the second dot while editing the input line.
		FRC	First . is interpreted as a space, second as a fraction mark. See *above* for some examples.
	f /	α	Appends a slash to ***alpha***.
+/-	+/−	¬α	Works like +/− in the Pioneers.
±	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 /		

Character	Keys to press	in modes	Remarks
|	h OR	α	Appends a vertical separator to ***alpha***.
🖷	f ↑	Catalog open	Enters the print character for fast access to the respective commands (see *below*).

CATALOGS, BROWSERS AND APPLICATIONS

Due to the large set of operations your *WP 34S* features, most of them are stored in catalogs. Opening a **catalog** will set alpha mode to allow for typing the first character(s) of the item wanted for rapidly accessing it. A subset of the full alpha keyboard shown *above* is sufficient for catalog browsing as pictured here. But there are three differences:

f B (= 1/x) is for reverting conversions easily (see CONV *below*).

f → just calls the character '→', and f EXIT calls the print character in catalog browsing (since case switching is not needed here).

▲ and ▼ will browse the open catalog.

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

EXIT or ← leave the catalog without executing anything, i.e. they cancel the catalog call.

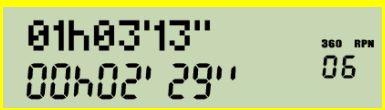
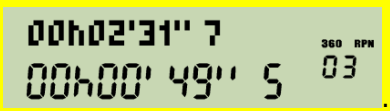
See *below* for some examples.

You may switch catalogs easily by just calling a new one accessible in current mode directly from the catalog you are browsing – no need to EXIT first.

When the last catalog called is reopened, the last item viewed therein is displayed for easy repetitive use. A single function may be contained in more than one catalog.

There are also three ***browsers*** for checking memory, flags, program labels and registers (i.e. CAT, SHOW, and STATUS). Therein, ▲, ▼, EXIT, and ← work as in catalogs. SHOW and STATUS operate in *α_T mode*, however. And some special keys and special rules may apply in browsers as explained in the following.

Furthermore, a **stopwatch application** is provided:

Name	Keys to press	in modes	Remarks
STOPW	h X.FCN STOPW	DECM, ¬PR	Stopwatch like the *HP-55* timer. This works only with a quartz crystal and associated firmware installed (or on the emulator). {The content of **X** will be taken as start time.} [56] When STOPW is started, the display will look like this:  unless the timer was already running.
	CPX R/S		
	Within STOPW, the following keys will work:		
	R/S		starts or stops the timer without changing its value.
	CLx or ←		resets the timer to zero without changing its status (running or stopped).
	EEX		hides or displays tenths {hundredths} of seconds. Startup default is 'display'.
	n n		sets the *current register address* (*CRA*, startup default is 00). Your numeric input will already be displayed in the exponent section like shown here [57]: 
	ENTER↑		stores the present timer value in the current register at execution time in format `hhh.mmssd` without changing the timer status or value. It then increments the *CRA* and displays it like shown above.
	→		hides or displays the *CRA*. Startup default is 'hide'.
	▲ or ▼		increments or decrements the *CRA*, respectively.
	.		combines ENTER↑ and CLx in one keystroke, but the total time since the last explicit press of CLx or ← is shown and updated in the top row like:  or  Note this total time is volatile, however – it will disappear without a trace when CLx or ← is pressed.

[56] Parts printed in { } apply to the *HP-55*, but not to your *WP 34S*. Start times are supported by RCL here.

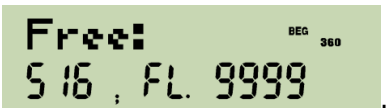
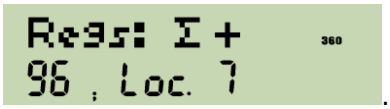
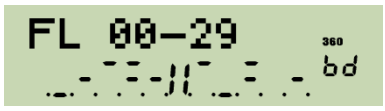
[57] Attempts to specify a *CRA* beyond the allocated address range will be blocked and may cause '- -' or the like being displayed in the exponent section.
On the HP-55, input of a single digit was sufficient for storing, since only 10 registers were featured for this purpose there. Furthermore, there was no automatic address increment.

A	adds the present timer value to the statistics registers like Σ+ would do. This allows for computing its arithmetic mean and standard deviation after leaving STOPW.
+	combines A and . in one keystroke.
RCL ***nn***	recalls ***rnn*** without changing the status of the timer. The value recalled may be used e.g. as start time for further incrementing.
EXIT	leaves the application. Unless already stopped, however, the timer continues incrementing in the background (indicated by the small '=' annunciator flashing) until a) stopped explicitly by R/S within STOPW or b) your *WP 34S* is turned off. While the stopwatch display is limited to 99h59'59''9, internal counting will continue with the display showing the time modulo 100.
Note no other keys will work in STOPW – so e.g. for adding or subtracting split times you have to leave this application.	

The **browsers** featured by your *WP 34S* are listed below:

Keys to press	in modes	Contents and special remarks
h CAT	¬α	Defined alphanumeric (i.e. global) labels. The first item shown is the top global label of the current program [58] – if there is no such label, the end of this program is shown. ▲ and ▼ browse global labels, while the location of the label displayed is indicated in the lower line (RAM, Lib for the library in flash memory, or bUP for the backup region). Duplicate labels also show the primary address, e.g. CALLS 012 when found a second time in RAM, or e.g. CALLS Lib when found a second time elsewhere. f ▲ and f ▼ browse programs, i.e. show the first label in previous or next program (with programs separated by END statements). 0, 1, and 2 allow quick jumps to the top of RAM, Lib, or bUP, respectively. ENTER↑ goes to the alpha label displayed, while XEQ executes it. Both keystrokes will perform a label search as described *above*. RCL executes PRCL for the program displayed. R/S starts the current program, i.e. the one whose label is just displayed, without performing a label search first. CLP deletes the current program, be it in RAM or Lib.

[58] The current program is the one the program pointer is in, as mentioned above.

Keys to press	in modes	Contents and special remarks
g SHOW	¬α	Browses all allocated stack and general purpose registers as well as their contents, starting with **X**. ▲ goes up the stack, continues with the other lettered registers, then with **R00**, **R01**, etc. ▼ browses the registers going down from the highest allocated numbered register (**R99** in startup default) to **R00** if applicable, then continues with **K, J**, etc. . turns to local registers if applicable, starting with **R.00**. Then, ▲ and ▼ browse local registers up and down until another . returns to **X**. Local register addresses may exceed **.15** here! Input of any legal letter or any legal two-digit number jumps to the corresponding register (see *above*). ENTER↑ or RCL recall the register displayed. In programming mode, they enter a corresponding step RCL … (In your *WP 34S*, ◀ and ▶ do what SHOW did in vintage calculators – see *above*.)
h STATUS	¬α	Displays the memory status and browses all user flags, similar to STATUS on the *HP-16C*. It shows the amount of free memory words in *RAM* and flash first, e.g.:  Press ▼ and read if there are summation registers used, plus the number of global numbered registers and local registers allocated:  or  Another ▼ presents the status of the first 30 user flags in one very concise display as explained below. **Example**: If flags 2, 3, 5, 7, 11, 13, 14, 17, 19, 20, 26, and X are set, and labels B and D are defined in program memory, STATUS ▼▼ will display this:  Where the mantissa is usually displayed, there are now three rows of horizontal bars. Each row shows the status of 10 flags. If a particular flag is set, the corresponding bar is turned black. So here the top row of bars indicates flags 0 and 1 are clear, 2 and 3 set, and 4 clear. Then a ‖ separates the first five flags from the next. The following top-row bars indicate flag 5 set, 6 clear, 7 set, 8 and 9 clear. The next two rows show the status up to flag 29.

Keys to press	in modes	Contents and special remarks
h STATUS (continued)		Pressing ▼ once will increment the start address by ten; so the display will look like:  Another ▼ will show flags 20 - 49 etc. until 70 - 99, 80 - 99, 90 - 99. A final ▼ will display the last 12 global flags in rows of four – note flag X is shown being set as we expect:  The exponent section indicates the status of the four hotkeys – if all four labels are defined in programs then ALL will be shown there. ▲ browses backwards. Alternatively, pressing a digit (e.g. 5) will display up to 30 flags starting with 10 times this digit (flags 50 – 79 here). Pressing a legal letter like D will display the top 12 flags.

Finally, see the following **catalogs** featured:

Keys to press	in modes	Contents and special remarks
h CONST	DECM & ¬PR	Constants like in the *HP-35s*, but more. See them listed *below*. While browsing this catalog, the values of the constants are displayed. Picking a constant will recall it.
	Integer	Calls the command # (see *above*).
	DECM & PR	Picking a constant will insert a program step containing the name of the constant selected, preceded by # . This step will then recall the value of said constant in program execution.
CPX h CONST	DECM & ¬PR	Opens the same catalog of constants as h CONST, but picking a constant will execute a complex recall. So, a stack looking like [*x*, *y*, …] before will contain [***constant***, **0**, *x*, *y*, …] after picking.
	DECM & PR	Picking a constant will insert a program step containing the name of the constant selected, preceded by ⌜# . This step will then perform a complex recall of the value of said constant in program execution.
h CONV	DECM	Conversions as listed in a *table below*. While browsing this catalog, the converted content of **X** is displayed. Picking a conversion will return this value.

Keys to press	in modes	Contents and special remarks
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 the standard normal and its inverse.
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	All summation registers. While browsing this catalog, register contents are displayed. Picking a register will recall its contents.
h TEST	¬α	All tests except the two on the keyboard (see next page).
	α	Comparison symbols and brackets, except (,) and ≠ (see *below*).
h X.FCN	DECM	Extra real functions.
	Integer	Extra integer functions.
	α	Extra alpha functions.
		These three catalogs are merged in mode PR to ease programming.
CPX h X.FCN	DECM	Extra complex functions.
h ./,	α	Punctuation marks and text symbols (see *below*).
f →	α	Arrows and mathematical symbols (see *below*).

See the next pages for detailed item lists of the various catalogs. Items are sorted alphabetically within the catalogs (see *above* for the sorting order). You may access a particular item quickly by typing the first characters of its name – see *below* for examples and constraints.

Note neither catalog nor browser nor application calls can be programmed.

Catalog Contents in Detail

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

MODE
12h
1COMPL
24h
2COMPL
BASE
BestF
DBLOFF
DBLON
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
RMF
SEPOFF
SEPON
SETCHN
SETDAT
SETEUR
SETIND
SETJPN
SETTIM

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

P.FCN
BACK
BASE?
CASE
CFALL
CLALL
CLPALL
CLREGS
CLSTK
CLα
CNST
DEC
DROP
DSL
DSZ
END
ERR
FF
FLASH?
f '(x)
f "(x)
gCLR
gDIM
gDIM?
gFLP
gPLOT
gSET
GTOα
INC
ISE
ISZ
LOADP
LOADR
LOADSS
LOADΣ
LocR
LocR?
MEM?
MSG
NOP
PopLR
PRCL
PROMPT
PSTO
PUTK
RCLS
RECV
REGS?
RESET
RM?
RTN+1
R-CLR
R-COPY
R-SORT
R-SWAP
SAVE
SENDA
SENDP
SENDR
SENDΣ
SKIP
SMODE?
SSIZE?
STOS
TICKS
t⇆
VIEWα
VWα+
WSIZE?
XEQα
y⇆
z⇆
αGTO
αOFF
αON
αXEQ
⇆
🖶ADV
🖶CHR
🖶$^C r_{XY}$
🖶PLOT
🖶PROG
🖶r
🖶REGS
🖶STK
🖶TAB
🖶WIDTH
🖶α
🖶α+
🖶Σ
🖶+α
🖶#

STAT
COV
L.R.
SEED
SERR
$SERR_W$
SUM
s_W
s_{XY}
$\bar{x}g$
$\bar{x}_W$
$\hat{x}$
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
FP?
FS?
FS?C
FS?F
FS?S
gPIX?
INTM?
INT?
KEY?
LBL?
LEAP?
M.SQR?
NaN?
ODD?
PRIME?
REALM?
SPEC?
TOP?
XTAL?
x < ?
x ≤ ?
x = +0?
x = -0?
x ≈ ?
x ≥ ?
x > ?
∞?
🖶?

SETUK
SETUSA
SIGNMT
SLOW
SSIZE4
SSIZE8
STOM
UNSIGN
WSIZE
Y.MD
🖶DLAY
🖶MODE

X.FCN varies with the mode set, except in programming [59]. It contains in …

… alpha mode:	… decimal mode:			… integer modes:		**CPX X.FCN**
VERS	$\sqrt[3]{x}$	LCM	W_m	$\sqrt[3]{x}$	RLC	$^{C}\sqrt[3]{x}$
x→α	AGM	L_n	W_p	ASR	ROUNDI	^{C}AGM
αDATE	ANGLE	LN1+x	W^{-1}	BATT	RR	^{C}CNST
αDAY	BATT	$L_n\alpha$	XNOR	CB	RRC	^{C}CONJ
αIP	B_n	LNβ	x^3	CEIL	SB	^{C}CROSS
αLENG	B_n*	LNΓ	x→α	DBLR	SEED	^{C}DOT
αMONTH	CEIL	MANT	$\sqrt[x]{y}$	DBL×	SIGN	^{C}DROP
αRC#	DATE	MAX	YEAR	DBL /	SL	$^{C}e^{x}$ -1
αRL	DATE→	MIN	αDATE	dRCL	SR	^{C}FIB
αRR	DAY	MOD	αDAY	DROP	sRCL	$^{C}g_d$
αSL	DAYS+	MONTH	αIP	FB	ULP	$^{C}g_d^{-1}$
αSR	DECOMP	NAND	αLENG	FIB	VERS	^{C}LN1+x
αTIME	DEG→	NEIGHB	αMONTH	FLOOR	WHO	^{C}LNβ
α→x	dRCL	NEXTP	αRCL	GCD	x^3	^{C}LNΓ
	DROP	NOR	αRC#	LCM	XNOR	^{C}SIGN
	D→J	P_n	αRL	LJ	x→α	^{C}SINC
	erf	RAD→	αRR	MASKL	$\sqrt[x]{y}$	$^{C}W_p$
	erfc	RDP	αSL	MASKR	αIP	$^{C}W^{-1}$
	EXPT	RESET	αSR	MAX	αLENG	$^{C}x^3$
	e^{x} -1	ROUNDI	αSTO	MIN	αRCL	$^{C}\sqrt[x]{y}$
	FIB	RSD	αTIME	MIRROR	αRC#	^{C}β
	FLOOR	SDL	α→x	MOD	αRL	^{C}Γ
	GCD	SDR	β	NAND	αRR	$^{C}(-1)^{x}$
	g_d	SIGN	Γ	nBITS	αSL	
	g_d^{-1}	SINC	γ_{XY}	NEIGHB	αSR	
	GRAD→	SLVQ	Γ_{XY}	NEXTP	αSTO	
	H_n	sRCL	ΔDAYS	NOR	α→x	
	H_{np}	STOPW	ζ	RESET	$(-1)^{X}$	
	H.MS+	TIME	$(-1)^{X}$	RJ	×MOD	
	H.MS–	T_n	→DATE	RL	^MOD	
	iRCL	ULP	%MG			
	Iβ	U_n	%MRR			
	IΓ_p	VERS	%T			
	IΓ_q	WDAY	%Σ			
	J→D	WHO	%+MG			

[59] In programming mode, these three contents are 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. Small font is printed on light grey background on this page. The catalog CPX is listed left. Use ↑ to toggle between cases. 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 q u w x ^{x} γ μ $^{-1}$ * ∞

The letters provided in your *WP 34S* allow for correct writing the languages of more than $3 \cdot 10^9$ people using Greek or simple variants of Latin alphabets, 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, Slovensky, Slovenščina, Srpski, Basa Sunda, Suomi, Svenska, Tagalog, Winaray, and Zhōngwén (with a little trick explained below). This makes the *WP 34S* the most versatile global calculator known. If you know further living languages covered, please tell us. Find the full character set provided in *Appendix E*.

Mandarin Chinese (Zhōngwén) features four tones, usually transcribed like e.g. mā, má, mă, 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 Hànyŭ pīnyīn) representing the third tone here. Pinyin writers, we ask for your understanding.

Accessing Catalog Items the Fast Way

You can browse each and every catalog just using the cursors ▼ and ▲ 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 the first item in this catalog** (e.g. BC? in TEST)	(e.g. ä in CPX)	(e.g. ; in ./,)
2	User input	1st character of command desired (e.g. F)	Desired basic letter (e.g. U)	
	Dot matrix display	**Shows the first item starting with this character *)** (e.g. FC?)	(e.g. ü)	
3	User input	2nd character (e.g. S)		
	Dot matrix display	**Shows the first item starting with this sequence *)** (e.g. FS?)		
…		Continue browsing with ▼ until you reach 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 à 0,25€)	

*) Like in browsing a dictionary, it may be faster to search the letter following alphabetically instead and then browse backwards. In our example, TEST F S ▼ finds FS?C but TEST G ▲▲▲ would do as well.
Remember you can enter Greek letters for a search using prefix g, e.g. g + A for α (compare *above*).
If a character or sequence specified is not found in the catalog chosen then the first item following alphabetically will be shown – see the sorting order *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 catalog of constants. Navigation therein works as explained above. Names of astronomical and mathematical constants are printed on colored background below. Values of physical constants (*including their **relative** standard deviations given in parentheses below*) are from CODATA 2010, copied in July 2011, unless stated otherwise. Green background denotes exact or almost exact values. The redder the background, the less precisely the particular constant is known, even by the national standards institutes and the international scientific community [60].

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 $c/f = \lambda$, or whatever else crosses your mind.

	Numeric value	Remarks
a	365.242 5 d *(per definition)*	Gregorian year
a_0	5.291 772 109 2**E**-11 *(3.2**E**-10)* m	Bohr radius $a_0 = \frac{\alpha}{4\pi \cdot R_\infty}$
a_m	384.4**E**6 *(1**E**-3)* m	Semi-major axis of the Moon's orbit
$a_\oplus$	1.495 979**E**11 *(1**E**-6)* m	Semi-major axis of the Earth's orbit. Within the uncertainty stated here, it equals 1 AU.
c	2.997 924 58**E**8 m/s *(per definition)*	Speed of light in vacuum ≈ 300 000 km/s
c_1	3.741 771 53**E**-16 *(4.4**E**-8)* $m^2 \cdot W$	First radiation constant $c_1 = 2\pi \cdot h \cdot c^2$
c_2	0.014 387 770 *(9.1**E**-7)* $m \cdot K$	Second radiation constant $c_2 = hc/k$
e	1.602 176 565**E**-19 *(2.2**E**-8)* C	Electron charge $e = \frac{2}{K_J R_K} = \Phi_0 G_0$
eE	2.718 281 828 459 045…	Euler's e. Note the letter e represents the electron charge elsewhere in this table.

[60] The numbers printed in parentheses allow for determining the precision of results you may obtain using these constants, through the process of 'error propagation' going back to C. F. Gauß (1777 – 1855). This procedure is essential if your results are to be trustworthy – not only in science. Consult a suitable reference. Yardstick measurements cannot yield results precise to four decimals.

	Numeric value	Remarks
F	96 485.336 5 *(2.2**E**-8)* $\frac{C}{mol}$	Faraday's constant $F = e \cdot N_A$
Fα	2.502 907 875 095 892 8...	Feigenbaum's α and δ
Fδ	4.669 201 609 102 990 6...	
g	9.806 65 m/s^2 *(per definition)*	Standard earth acceleration
G	6.673 84**E**-11 *(1.2**E**-4)* $\frac{m^3}{kg \cdot s^2}$	Newtonian constant of gravitation. See **GM** below for a more precise value.
G_0	7.748 091 734 6**E**-5 *(3.2**E**-10)* / Ω	Conductance quantum $G_0 = 2e^2/h = 2/R_K$
G_C	0.915 965 594 177…	Catalan's constant
g_e	-2.002 319 304 361 53 *(2.6**E**-13)*	Landé's electron g-factor
GM	3.986 004 418**E**14 *(2.0**E**-9)* m^3/s^2	Newtonian constant of gravitation times the Earth's mass with its atmosphere included (according to *WGS84*)
h	6.626 069 57**E**-34 *(4.4**E**-8) J s*	Planck constant
ħ	1.054 571 726**E**-34 *(4.4**E**-8) J s*	$= h/2\pi$
k	1.380 648 8**E**-23 *(9.1**E**-7)* J/K	Boltzmann constant $k = R/N_A$
K_J	4.835 978 70**E**14 *(2.2**E**-8)* Hz/V	Josephson constant $K_j = 2e/h$
l_P	1.616 199**E**-35 *(6.0**E**-5) m*	Planck length $l_P = \sqrt{\hbar G/c^3} = t_p c$
m_e	9.109 382 91**E**-31 *(4.4**E**-8) kg*	Electron mass
M_m	7.349**E**22 *(5**E**-4) kg*	Mass of the Moon
m_n	1.674 927 351**E**-27 *(4.4**E**-8) kg*	Neutron mass
m_p	1.672 621 777**E**-27 *(4.4**E**-8) kg*	Proton mass
M_P	2.176 51**E**-8 *(6.0**E**-5) kg*	Planck mass $M_P = \sqrt{\hbar c/G} \approx 22\mu g$
m_u	1.660 538 921**E**-27 *(4.4**E**-8) kg*	Atomic mass unit = 10^{-3} kg / N_A

	Numeric value	Remarks
$m_u c^2$	1.492 417 954**E**-10 *(4.4**E**-8) J*	Energy equivalent of atomic mass unit
m_μ	1.883 531 475**E**-28 *(5.1**E**-8) kg*	Muon mass
$M_\odot$	1.989 1**E**30 *(5**E**-5) kg*	Mass of the Sun
$M_\oplus$	5.973 6**E**24 *(5**E**-5) kg*	Mass of the Earth
N_A	6.022 141 29**E**23 *(4.4**E**-8) / mol*	Avogadro's number
NaN	not numeric	'Not a Number', e.g. $\ln(x)$ for $x \le 0$ unless in complex domain
p_0	101 325 *Pa* *(per definition)*	Standard atmospheric pressure
q_P	1,875 545 9**E**-18 *(6.0**E**-5) A s*	Planck charge $q_p = \sqrt{4\pi\varepsilon_0\hbar c} \approx 11.7e$. This was in CODATA 2006, but in 2010 no more.
R	8.314 462 1 *(9.1**E**-7)* $\frac{J}{mol \cdot K}$	Molar gas constant
r_e	2.817 940 326 7**E**-15 *(9.7**E**-10) m*	Classical electron radius $r_e = \alpha^2 \cdot a_0$
R_K	25 812.807 443 4 *(3.2**E**-10)* Ω	von Klitzing constant $R_K = h/e^2$
R_m	1.737 530**E**6 *(5**E**-7) m*	Mean radius of the Moon
R_∞	1.097 373 156 853 9**E**7 *(5.0**E**-12) / m*	Rydberg constant $R_\infty = \alpha^2 m_e c/2h$
$R_\odot$	6.96**E**8 *(5**E**-3) m*	Mean radius of the sun
$R_\oplus$	6.371 010**E**6 *(5**E**-7) m*	Mean radius of the Earth
Sa	6.378 137 0**E**6 *m* *(per definition)*	Semi-major axis of the model *WGS84* used to define the Earth's surface for GPS and other surveying purposes
Sb	6.356 752 314 2**E**6 *(1.6**E**-11) m*	Semi-minor axis of *WGS84*
Se^2	6.694 379 990 14**E**-3 *(1.5**E**-12)*	First eccentricity squared of *WGS84*
Se'^2	6.739 496 742 28**E**-3 *(1.5**E**-12)*	Second eccentricity squared of *WGS84* (it is really called e'^2 in this article, sorry)
Sf^{-1}	298.257 223 563 *(per definition)*	Flattening parameter of *WGS84*
T_0	273.15 *K* *(per definition)*	= 0°C, standard temperature

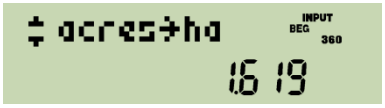
	Numeric value	Remarks
t_P	5.391 06**E**-44 *(6.0**E**-5) s*	Planck time $t_P = \sqrt{\frac{\hbar G}{c^5}} = \frac{l_p}{c}$
T_P	1.416 833**E**32 *(6.0**E**-5) K*	Planck temperature $T_P = \frac{c^2}{k}\sqrt{\frac{\hbar c}{G}} = \frac{M_p c^2}{k} = \frac{E_p}{k}$
V_m	0.022 413 968 *(9.1**E**-7)* m^3/mol	Molar volume of an ideal gas at standard conditions $V_m = RT_0/p_0 \approx 22.4\frac{l}{mol}$
Z_0	376.730 313 461… Ω	Characteristic impedance of vacuum $Z_0 = \mu_0 c$
α	7.297 352 569 8**E**-3 *(3.2**E**-10)*	Fine-structure constant $\alpha = \frac{e^2}{4\pi\varepsilon_0\hbar c} \approx \frac{1}{137}$
γEM	0.577 215 664 901 532 86…	Euler-Mascheroni constant γ_{EM}
γ_p	2.675 222 005**E**8 *(2.4**E**-8)* $\frac{1}{s \cdot T}$	Proton gyromagnetic ratio $\gamma_p = 2\mu_P/\hbar$
ε_0	8.854 187 817…**E**-12 $\frac{A\,s}{V\,m}$	Electric constant or vacuum permittivity $\varepsilon_0 = \frac{1}{\mu_0 c^2}$
λ_C	2.426 310 238 9**E**-12 *(6.5**E**-10) m*	Compton wavelengths of the electron $\lambda_C = \frac{h}{m_e c}$, neutron, and proton, respectively
λ_{Cn}	1.319 590 906 8**E**-15 *(8.2**E**-10) m*	
λ_{Cp}	1.321 409 856 23**E**-15 *(7.1**E**-10) m*	
μ_0	1.256 637 061 4…**E**-6 $\frac{V\,s}{A\,m}$	Magnetic constant or vacuum permeability $\mu_0 := 4\pi \cdot 10^{-7}\frac{V \cdot s}{A \cdot m}$
μ_B	9.274 009 68**E**-24 *(2.2**E**-8)* J/T	Bohr's magneton $\mu_B = e\hbar/2m_e$
μ_e	-9.284 764 30**E**-24 *(2.2**E**-8)* J/T	Electron magnetic moment
μ_n	-9.662 364 7**E**-27 *(2.4**E**-7)* J/T	Neutron magnetic moment
μ_p	1.410 606 743**E**-26 *(2.4**E**-8)* J/T	Proton magnetic moment
μ_u	5.050 783 53**E**-27 *(2.2**E**-8)* J/T	Nuclear magneton $\mu_u = e\hbar/2m_p$

	Numeric value	Remarks
μ_μ	-4.490 448 07**E**-26 *(3.4**E**-8)* J/T	Muon magnetic moment
σ_B	5.670 373**E**-8 *(3.6**E**-6)* $\frac{W}{m^2K^4}$	Stefan-Boltzmann constant $\sigma_B = \frac{2\pi^5 k^4}{15h^3c^2}$
Φ	1.618 033 988 749 894…	Golden ratio $\Phi = \frac{1+\sqrt{5}}{2}$
Φ_0	2.067 833 758**E**-15 *(2.2**E**-8)* $V\,s$	Magnetic flux quantum $\Phi_0 = h/2e = 1/K_J$
ω	7.292 115**E**-5 *(2**E**-8)* rad/s	Angular velocity of the Earth according to *WGS84*
$-\infty$	-InFinitY	May the Lord of Mathematics forgive us calling these 'constants'. Note both are counted as numeric values in your *WP 34S*.
∞	InFinitY	
#		See the very last command in the *IOP* *above*.

Unit Conversions (CONV)

CONV mainly provides the means to convert local to common units [61]. 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.

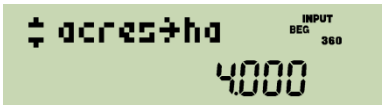
Example: Assume the display set to FIX 3. Then keying in

4 h CONV A will display 

telling you that 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 CONV via EXIT and the display will return to 9.884.

The calculations listed below for your information are user transparent in executing a conversion – those printed on green background in this table apply exactly.

Conversion	Calculation	Remarks	Class
°C→°F	* 1.8 + 32		Temperature
°F→°C	- 32) / 1.8		Temperature
°→G	/ 0.9	Converts to *grads* , also known as *gon*	Angle
°→rad	* π / 180	Equals D→R	Angle
acres→ha	* 0.404 687 3	1 ha = $10^4\ m^2$	Area
ar.→dB	$20lg(a_1/a_2)$	Amplitude ratio	Ratio
atm→Pa	* 1.013 25E5		Pressure
AU→km	* 1.495 979**E**8	Astronomic units	Length
bar→Pa	* 1E5		Pressure

[61] The *SI* system of units is agreed on internationally. Meanwhile, it is adopted by all countries on this planet except two. Thus 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 (English is spoken in all of them). For symmetry reasons, we think about adding some traditional Indian and Chinese units to CONV.

Conversion	Calculation	Remarks	Class
Btu→J	* 1 055.056	British thermal units	Energy
cal→J	* 4.186 8		Energy
cft→l	* 28.316 85	Cubic feet	Volume
cm→inches	/ 2.54		Length
cwt→kg	* 50.802 35	(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	* 1.828 8		Length
feet→m	* 0.304 8		Length
flozUK→ml	* 28.413 06	1 $l = {}^{1}/_{1000}\ m^3$	Volume
flozUS→ml	* 29.573 53		
galUK→l	* 4.546 09		
galUS→l	* 3.785418		
G→°	* 0.9	*Grads* or *gon*	Angle
g→oz	/ 28.349 52		Mass
G→rad	* π / 200		Angle
g→tr.oz	/ 31.103 48		Mass
ha→acres	/ 0.404 687 3	1 ha = 10000 m^2	Area
HP_e→W	* 746	Electric horsepower	Power
hpUK→W	* 745.699 9	British horsepower	Power
inches→cm	* 2.54		Length
inHg→Pa	* 3 386.389		Pressure
J→Btu	/ 1 055.056		Energy
J→cal	/ 4.186 8		Energy
J→kWh	/ 3.6E6		Energy
kg→cwt	/ 50.802 35	(Long) hundredweight = 112 *lbs*	Mass

Conversion	Calculation	Remarks	Class
kg→lb	/ 0.453 592 4		Mass
kg→stones	/ 6.350 293 18		Mass
kg→s.cwt	/ 45.359 24	Short hundredweight = 100 *lbs*	Mass
km→AU	/ 1.495 979**E**8	Astronomical units	Length
km→l.y.	/ 9.460 730**E**12	Light years	Length
km→miles	/ 1.609 344		Length
km→nmi	/ 1.852	Nautical miles	Length
km→pc	/ 3.085 678**E**16	Parsec	Length
kWh→J	* 3.6E6		Energy
lbf→N	* 4.448 222		Force
lb→kg	* 0.453 592 4		Mass
l.y.→km	* 9.460 730**E**12	Light years	Length
l→cft	/ 28.316 85	1 $l = {}^{1}/_{1000}\ m^3$	Volume
l→galUK	/ 4.546 09		
l→galUS	/ 3.785 418		
miles→km	* 1.609 344		Length
ml→flozUK	/ 28.413 06	1 ml = 1 cm^3	Volume
ml→flozUS	/ 29.573 53		
mmHg→Pa	* 133.322 4		Pressure
m→fathom	/ 1.828 8		Length
m→feet	/ 0.304 8		Length
m→yards	/ 0.914 4		Length
nmi→km	* 1.852	Nautical miles	Length
N→lbf	/ 4.448 222		Force
oz→g	* 28.349 52	Ounces	Mass
Pa→atm	/ 1.013 25E5	1 Pa = 1 N/m^2	Pressure
Pa→bar	/ 1E5		Pressure

Conversion	Calculation	Remarks	Class
Pa→inHg	/ 3 386.389		Pressure
Pa→mmHg	/ 133.322 4		Pressure
Pa→psi	/ 6 894.757		Pressure
Pa→torr	/ 133.322 4		Pressure
pc→km	* 3.085 678**E**16	Parsec	Length
pr.→dB	$10\lg\left(P_1/P_2\right)$		Ratio
psi→Pa	* 6 894.757	Pounds per square inch	Pressure
PS(hp)→W	* 735.498 8	Horsepower	Power
rad→°	* 180 / π	Equals R→D	Angle
rad→G	* 200 / π		Angle
stones→kg	* 6.350 293 18		Mass
s.cwt→kg	* 45.359 24	Short hundredweight = 100 *lbs*	Mass
s.tons→t	* 0.907 184 7	Short tons	Mass
tons→t	* 1.016 047	Imperial tons	Mass
torr→Pa	* 133.322 4	1 *torr* = 1 *mm Hg*	Pressure
tr.oz→g	* 31.103 48	Troy ounces	Mass
t→s.tons	/ 0.907 184 7	1 *t* = 1000 *kg*	Mass
t→tons	/ 1.016 047		
W→HP$_e$	/ 746		Power
W→hpUK	/ 745.699 9		Power
W→PS(hp)	* 735.498 8		Power
yards→m	* 0.914 4		Length

The constant **T$_o$** may be useful for conversions of temperatures, too; it is found in *CONST* and is not repeated here because it is only added or subtracted.

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 gas stations 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, the *rem* (i.e. *roentgen equivalent in man*) measured what the *sievert* does today.

Predefined Global Alpha Labels (CAT)

In addition to the label 'δx' reserved for step size in calculation of derivatives (see f'(x) in the *IOP* above), additional labels may already be provided for particular tasks. You will find them listed in CAT when the respective library routines are loaded in flash memory. Thus they will not take any steps from user program memory in *RAM*.

All library routines presently available are found on the *WP 34S* website in the directory *http://wp34s.svn.sourceforge.net/viewvc/wp34s/library/* . They are text files with extension `.wp34s` by convention. They include a suite of basic 3D vector operations, a TVM (time value of money) application, some matrix routines, and more. You may open these text files using e.g. Notepad, and you should find the necessary user information at the beginning of each file.

The library files are also included in the distribution *ZIP* file in source form (`*.wp34s`) and as a precompiled library (`wp34s-lib.dat`) which is part of the `calc_full.bin` and `calc_xtal_full.bin` firmware files – so you get the full library when you load one of these firmware files into your *WP 34S* (see *Appendix A* about how to do this).

If you copy `wp34s-lib.dat` (some 3*kB*) into the directory your *WP 34S* emulator runs in, you can access all those routines via CAT from your emulator as well.

APPENDIX A: SETUP AND COMMUNICATION

How to Flash Your *HP-20b* or *-30b*

Unless you buy a *WP 34S* preflashed as explained *above*, you must do the flashing yourself. Then you need an unmodified *HP-20b* or *-30b* calculator, a special connecting cable, a file to load on your computer, and software for the transmission to your calculator. It is beneficial if your computer has a serial port.

- You will get the programming cable from Gene Wright (USA) [62].

  ATTENTION: If your computer does not feature a hardware serial interface, you will need a USB-to-serial converter to connect this programming cable to your computer. Following our experience, converters containing FTDI chips work – others may not. Such a converter is offered here: *http://commerce.hpcalc.org/usbserial.php* , for example.

- The specific file you need to transmit to your calculator to make it your *WP 34S* is called `calc.bin` and is included in the zipped release package you can download from *http://sourceforge.net/projects/wp34s/files/* . Alternatively, you may download `calc.bin` or one of its siblings alone [63].

  WARNING: Flashing your *HP-20b* or *-30b* will erase the *HP* firmware: your business calculator will then be gone. The file you downloaded will replace the *HP* firmware! Thereafter you will have a *WP 34S RPN Scientific* – i.e. your calculator will react as documented in this manual.

  This also means your calculator will not do anything useful for you between steps 4 and 12 of the procedure described below. It may even look dead – it is not, be assured. If the procedure is interrupted at any time, don't worry: simply start over at step 1.

  As long as the cable is connected to your calculator, it may draw a considerable current from its batteries. If your calculator happens to hang during the flashing procedure, the processor of the calculator may be left running at full speed, draining your coin cells while you are trying to find out what is going wrong. Thus you may want to disconnect your cable when you will not need it for the next few minutes. For repeated flashing, an external $3V\ DC$ power supply may quickly pay for itself. Take care to connect '+' to the outer and '–' to the inner contact. The following procedure will work only with fresh batteries or an external $3V$ supply.

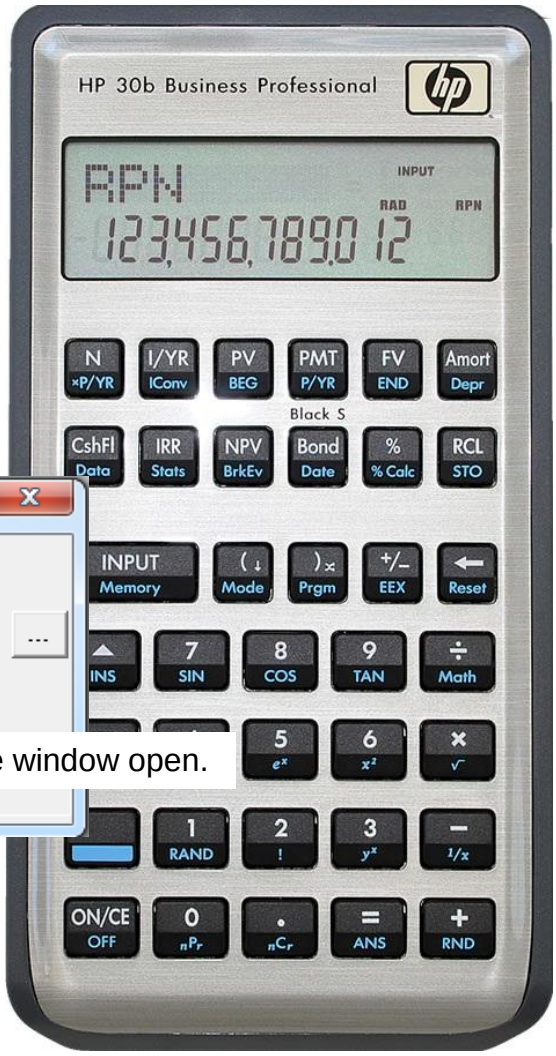
- The necessary computer software is called *MySamBa* and is provided on the project website as well – download and unpack it first.

---

[62] These cables were a limited production run and supply may come to an end soon. Alternatively, there are three small *PCBs* developed by *Harald Pott* (Germany), each hosting a *USB* port and thus allowing to connect to your *PC* via an ordinary micro *USB* cable. See *WP_34S_USB_installation* on our website.

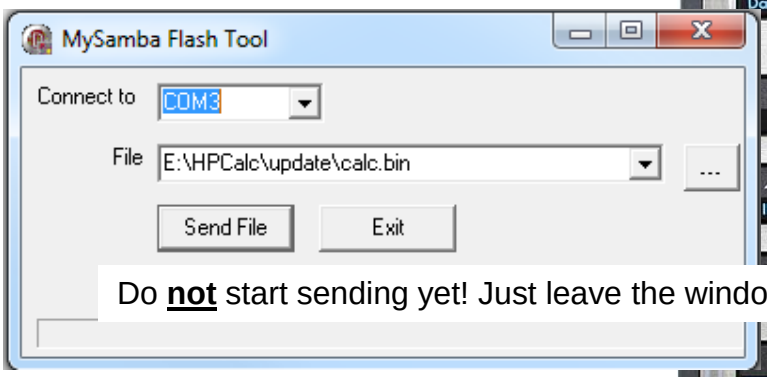
[63] These are the alternatives available at *http://wp34s.svn.sourceforge.net/viewvc/wp34s/trunk/realbuild/* :
`calc.bin` features maximum flash memory, but supports neither STOPW nor print commands,
`calc_xtal.bin` assumes a quartz crystal is installed and includes STOPW but no print commands,
`calc_ir.bin` includes also the print commands requiring the quartz crystal plus an IR-diode built in,
and all files called `..._full.bin` already include the flash memory user code library – others do not.
The amount of flash you have to pay for STOPW is some 1 kB, for printer support some 3 kB more. Make your choice! But check the hardware pre-requisites (see *Appendix H*) or your calculator may hang.

Having prepared all the necessary items, follow the steps below for transforming an *HP-20b* or *HP-30b* into a *WP 34S*:



1. Connect the programming cable to your computer and to the programming port behind the battery door of the calculator.

2. Start *MySamBa*. Pick the port that you use for your cable and select the file that you want to transmit. The window may look like this:



3. Turn the calculator on by pressing and releasing **ON/CE** .

4. Hold down the ERASE button on the cable (do not release it until step 7).

5. Press and release the RESET button on the cable. It will turn the calculator off.

6. Press and release **ON/CE** to turn the calculator on again.

7. Release ERASE now.

8. Press and release RESET again.

9. Press and release **ON/CE** on the calculator. It will look dead. Don't worry – see the note above.

10. Click "Send File" in *MySamBa* and wait for it to finish transmission (it will only take a few seconds). If you have the FTDI USB/serial adapter, you will see the blue TX light blinking.

11. Press and release RESET once more.

12. Press and release **ON/CE** . Your calculator should turn on as a *WP 34S* now.

This procedure was tested both on 32-bit Windows XP and 64-bit Windows 7 x64.

You may install one of the custom *USB*-boards mentioned in footnote 62 even before transforming your calculator into a *WP 34S*. If you did, your calculator will be powered through the *USB* port while connected to your computer using an ordinary micro *USB* cable. This will prevent the battery draining described above, but the standard cable does not feature the buttons ERASE and RESET of the programming cable. Thus, the flashing procedure must be modified as follows:

1. Remove the battery door of your calculator. Connect said *USB* cable to your computer and to the *USB* port of the calculator.

2. Start *MySamBa*. Enter the port and file information as explained above. Do not start sending yet – just leave the window open.

3. Turn the calculator on by pressing and releasing [ON/CE] .

4. Press [ON/CE] + [PMT] simultaneously. Release both keys.

5. Press [ON/CE] + [6] simultaneously. Release [6] and press it a second time. Then release both keys.

6. Press and release the RESET button behind the battery door of the calculator. It will turn your calculator off.

7. Press and release [ON/CE] to turn the calculator on again.

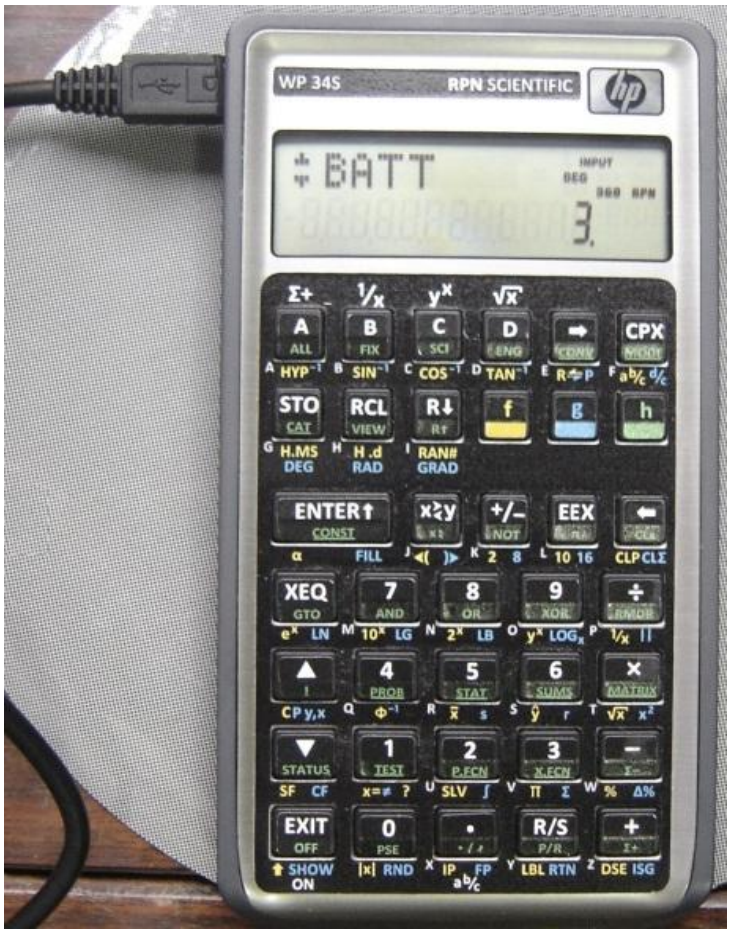
8. Press and release RESET again.

9. Press and release [ON/CE] on the calculator. It will look dead. Don't worry – see the note above.

10. Click "Send File" in *MySamBa* and wait for it to finish transmission (it will only take a few seconds). If you have the FTDI USB/serial adapter, you will see the blue TX light blinking.

11. Press and release RESET once more.

12. Press and release [ON/CE] . Your calculator should turn on as a *WP 34S* now.



Updating Your *WP 34S*

You may want to keep your *WP 34S* up-to-date when a new release is published. We recommend you SAVE your work each time before flashing a new release. After flashing, your backup will then still be available – if you didn't accidently press the ERASE button on the cable but used [ON] + [6] instead.

The updating procedure is basically the same as that for flashing the first time, but stay away from ERASE ! If you want to read the instructions once more, here they are:

> **ATTENTION**: If you have installed one of the custom *USB*-boards mentioned in footnote 62 then read the right side of the following table, else read the left side.

1. Remove the battery door of your *WP 34S*. Connect the programming cable to your computer and to the programming port of your *WP 34S*.

2. Start *MySamBa*. Enter the port and file information as explained above. Do not start sending yet – just leave the window open.
3. Turn your *WP 34S* on by pressing and releasing [ON] .
4. Press [ON] + [D] simultaneously. Release both keys.
5. Press [ON] + [6] simultaneously. Release [6] and press it a second time. Then release both keys.

6. Press and release the RESET button on the cable. It will turn your *WP 34S* off.

7. Press and release [ON] to turn your *WP 34S* on again.
8. Press and release RESET again.
9. Press and release [ON] on your *WP 34S*. It will look dead. Don't worry – it is not.
10. Click "Send File" in *MySamBa* and wait for it to finish transmission (it will only take a few seconds). If you have the FTDI USB/serial adapter, you will see the blue TX light blinking.
11. Press and release RESET once more.
12. Press and release [ON] . Your *WP 34S* should turn on with the new firmware now. Check using VERS.

1. Remove the battery door of your *WP 34S*. Connect the *USB* cable to your computer and to the *USB* port of your *WP 34S*.

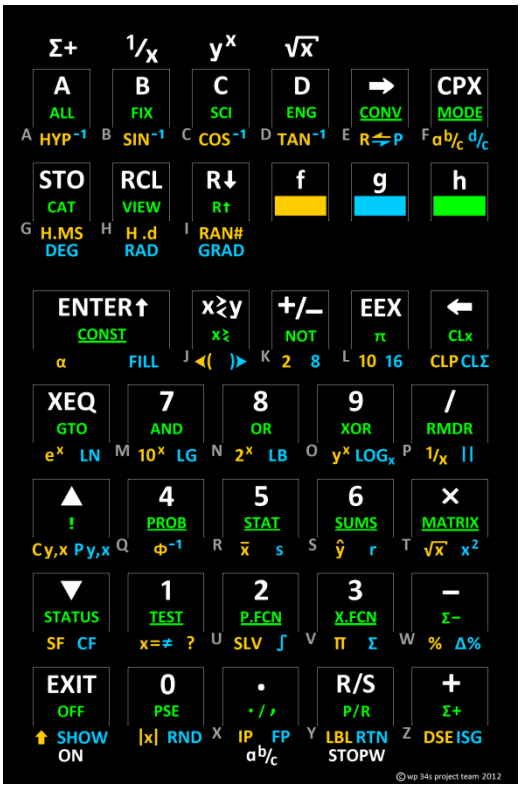
2. Start *MySamBa*. Enter the port and file information as explained above. Do not start sending yet – just leave the window open.
3. Turn your *WP 34S* on by pressing and releasing [ON] .
4. Press [ON] + [D] simultaneously. Release both keys.
5. Press [ON] + [6] simultaneously. Release [6] and press it a second time. Then release both keys.

5. Press and release the RESET button behind the battery door of your *WP 34S*. It will turn your *WP 34S* off.

7. Press and release [ON] to turn your *WP 34S* on again.
8. Press and release RESET again.
9. Press and release [ON] on your *WP 34S*. It will look dead. Don't worry – it is not.
10. Click "Send File" in *MySamBa* and wait for it to finish transmission (it will only take a few seconds). If you have the FTDI USB/serial adapter, you will see the blue TX light blinking.
11. Press and release RESET once more.
12. Press and release [ON] . Your *WP 34S* should turn on with the new firmware now. Check using VERS.

Overlays – Where to Get Them and How to Make Them Yourself if Required

After flashing successfully, a keyboard overlay is very helpful for further work since most labels deviate from those on said business calculators.



You may get fine adhesive vinyl overlays from Eric Rechlin at http://commerce.hpcalc.org/overlay.php You can specify if you want them with white labels on the prefix keys (the original design, see left pictures) or with f, g, and h in their respective colours (shown above). Supporters of the 'Great American Divide' can get them with a ÷ instead of the / as well (see the page before previous page).

If Eric's vinyls are not available for any reason, preliminary paper overlays are most easily made using the picture shown here bottom left. Print this page to scale (this picture should be 68*mm* wide), cut it out, span it over your *WP 34S* using transparent adhesive tape, and you are done – the flexibility of this setup allows actuating the calculator keys though there are no holes in your overlay.

You may – if you know how to handle a sharp pointed knife carefully – also cut along the fine white lines on the left, top, and right side of each key before spanning. Then attach the base paper to the key plate (e.g. using double sided sticky tape) and each key will peek through its own little door. When you stick the end of each flap to the respective key top, your paper cover will come pretty close to a professional overlay.

The picture at right shows the virtual keyboard in full alpha mode (compare the *picture* in the corresponding paragraph above). The labels underlined in red belong to the alpha catalogs.

Printed to scale, this picture is 55*mm* wide and less than 87*mm* high. So you can attach it to the back of your *WP 34S* for ready reference, using transparent adhesive tape. Or you may slide it 'under the hood' where the batteries live, as sort of a Quick Reference Sheet. The first option will be especially advantageous until you know the Greek letters and their relation to their Latin homophonic siblings.

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 *IOP*), 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.

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

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

Mapping of Memory Regions to Emulator State Files

Region		State file	Remarks
Backup	buP	wp34s-backup.dat	Is created by SAVE.
Flash	Lib	wp34s-lib.dat	Is written whenever a flash command is executed.
RAM	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

You will again need the special programming cable mentioned above, or your *WP 34S* includes a *USB* board as described above, or you need a *modified 20b or 30b* as described elsewhere. Said programming cable draws current from the batteries of your *WP 34S*, so disconnect it from your calculator as soon as it is no longer needed.

The emulator and the calculator can talk to each other over the cable used for flashing. 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. With a suitable cable it is even possible to transfer data between two calculators with the commands below.

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

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

On the sender you have three 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, an I/O error is thrown indicating no communication has occurred. If such an error appears in the middle of a transmission, try again.

On a device without the crystal installed, you may also get an I/O error because of the baud rate setting may be a bit too far off. To determine the speed, use the loop

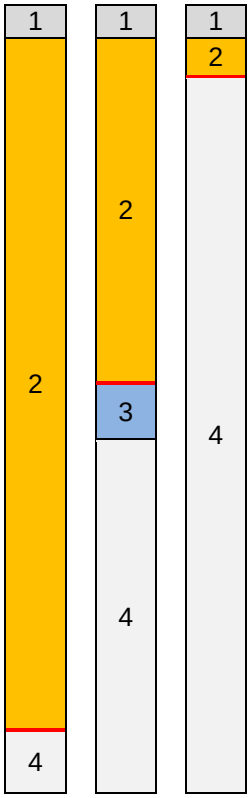
```
CLx
INC X
BACK 001
```

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 a device with the crystal installed, ***x*** is ignored.

The little '=' annunciator is lit while the serial port is in use. Take EXIT to abort the communication if necessary.

APPENDIX B: MEMORY MANAGEMENT

As mentioned above, your *WP 34S* features *RAM* and flash memory (*FM*). The latter may be up to some 8*kB*, depending on the file you loaded in flashing. This appendix, however, covers *RAM* only. It discusses how the available memory is divided in program area, local and global data. The 2*kB* (or 1024 *words*) of non-volatile *RAM* are divided into four 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 (*SRS*) and program memory.

These sectors are ordered top down as pictured at right to scale. The center bar shows the *RAM* configuration in startup default (but with statistics registers allocated). The left bar shows it for maximum register space allowed (but without statistics), the right for maximum program space. This appendix covers the variable boundaries between these sectors.

A complete copy of the non-volatile *RAM* as is can be written to *FM* using SAVE (or ON + STO). Now your *continuous memory* will remain safely preserved not only when your *WP 34S* turns off but also even when its batteries fail. See *Appendix A* for more information about flashing and handling of *FM* in general.

Status and Configuration Data

This sector of 88 *bytes* is fixed at the top of available memory and is completely user transparent. 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 twelve 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 the 12 lettered registers 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 *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: See the global register sector at startup default in the following memory table. The two rightmost columns show what happens after subsequent execution of REGS 96 and REGS 98. The registers are loaded with arbitrary values here so they can be traced easily. *LBG* is indicated by a red horizontal line.

Absolute address	Default startup memory allocation Contents	 Relative register address	After executing REGS 96 Contents	 Relative register address	Then after executing REGS 98 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**	**…**	**…**	***6.4***	**R94**
X-5	***r95 = 23.1***	**R95**	**…**	**…**	***4.8***	**R93**
X-6	***r94 = 6.4***	**R94**	**…**	**…**	**…**	**…**
X-7	***r93 = 4.8***	**R93**	**…**	**…**	**…**	**…**
…	**…**	**…**	**…**	**…**	**…**	**…**
X-94	***r06 = 62.4***	**R06**	***5.7***	**R02**	***29.4***	**R04**
X-95	***r05 = -0.6***	**R05**	***-2.4***	**R01**	***81.3***	**R03**
X-96	***r04 = 29.4***	**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**				
…						

Note the absolute addresses of **R00** up to **R*n-1*** change after REGS ***n*** whenever ***n*** is changed, while their contents are copied. The lettered registers stay at fixed addresses.

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 *SRS*) are tied to *LBG* – their contents will be copied whenever *LBG* moves. This makes it possible 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 general purpose registers. This allows for higher internal precision and prevents destroying these data inadvertently. 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 [64]. They are inserted between *LBG* and *SRS*, 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 overleaf).

After CLΣ, CLALL, or RESET, the memory allocated for the summation registers is released. All pointers are automatically adjusted, so the memory allocation or release will not disrupt a running program. Recall commands like e.g. Σxy or SUM 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) if the registers would no longer fit in *RAM* after that load. You can avoid this by reducing the amount of numbered registers using REGS before the load attempt. This should move the summation data out of the way.

SRS and Program Memory

Both share the remaining space at lowest memory addresses.

The ***SRS*** is used for return addresses and local data. Its upper boundary is given by *LBG* or the lowest summation register if applicable. There is no command to set the size of the *SRS* – it fills all the space down to the top program step currently stored. When new program steps are entered, the *SRS* 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 *SRS*. Thus they cannot overwrite global data; this greatly increases the flexibility of programs. LOCR ***n*** allocates ***n*** local registers and a fixed amount of 16 local flags. It does so by pushing a frame on the *SRS* containing a marker, a flag word, and the registers requested (0 to 144). 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 *SRS* grows in the opposite direction. LOCR? will return the number of local registers currently allocated in the routine you are in.

---

[64] 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.

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

Below the *SRS*, **program memory** holds the stored program steps. A typical program step takes just one word. Multibyte labels and multicharacter alpha strings take two words each. The total size of program memory depends on the number of global and local registers allocated, as explained in the follwing.

Making Room for Your Needs

The 12 special (lettered) registers are always allocated. The *SRS* has a minimum size of six words or levels. Everything else is user distributable within the 982 *words* left for sections 2 to 4, so:

982 = ***r*** + ***s*** + ***p*** with

- ***r*** = number of *words* allocated for global registers. These are 4 per standard register. There are at least 12 and at most 112 of them. So ***r*** varies between 48 and 896 (this maximum is explained *below*); startup default is 448.
- ***s*** = number of *words* allocated for summation registers (70 if they are used; startup default is 0).
- ***p*** = number of *words* available for program steps and *SRS*. One step is already taken by the inevitable final END statement; 6 *words* is the minimum size of the *SRS*. So STATUS will show you a maximum of 933 free *words* in *RAM*, meaning up to 927 free program steps. Startup default is 533 steps. Subroutine nesting and local registers expand the *SRS*, thus reducing the program space available.

If, for instance, you need to do statistics and also use 20 global numbered registers, there will be space for 777 program steps maximum.

You have several options for increasing the free space where you need it (see the picture *above*):

1. Reduce the number of global numbered registers allocated. One register less typically allows for four additional program steps.
2. Move programs to flash memory and clear the respective steps in *RAM*. Four cleared program steps typically allow for one additional register.
3. Release the summation registers when you do not need them anymore. This 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 your application. You may of course combine options. Use STATUS to monitor the free space available and the amount of global and local 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 also be directly addressed using a dot heading the number – the arguments go from **.**00 to **.**15, corresponding to relative addresses from 112 to 127 [65]. 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 *SRS* 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.

RTN or POPLR – executed in a program – check if the current *SRS* pointer points to a local frame (as explained *above*). If true then the pointer is moved above that frame, and the *SRS* is searched from this point upwards for another local frame. If such a frame is found then its pointer is stored; otherwise 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 current local frame is dropped and the next higher (or older) frame is reactivated if one exists.

Manually executing RTN, starting a new program with XEQ, SLV, etc., or program editing will clear the *SRS* and remove all local registers and flags by clearing the pointer. All such data 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. It is an **example** for demonstration only, since this routine will neither set the stack correctly nor will it work for input greater than some hundred:

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

```
LBL 'FAC'
IP
x≥1?
GTO 00
1
RTN
LBL 00
LocR 001
STO .00
DEC X
XEQ 'FAC'
RCL× .00
RTN
```

Assume $x = 4$ when you call FAC. Then it will allocate 1 local register (**R.00**) and store 4 therein. After decrementing x, FAC will call itself.

Then FAC_2 will allocate 1 local register ($\mathbf{R.00_2}$) and store 3 therein. After decrementing x, FAC will call itself again.

Then FAC_3 will allocate 1 local register ($\mathbf{R.00_3}$) and store 2 therein. After decrementing x, FAC will call itself once more.

Then FAC_4 will return to FAC_3 with $x = 1$. This x will be multiplied by $r.00_3$ there, returning to FAC_2 with $x = 2$. This x will be multiplied by $r.00_2$ there, returning to FAC with $x = 6$, where it will be multiplied by $r.00$ and will finally become 24.

Switching between Real and Integer Modes

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 the current stack will be truncated to integers. Going from integer mode to DECM, the current stack contents (being all integers) will be converted to decimal. All other memory contents will stay as they were!

See the fate of some register contents undergoing mode switches in the following **examples**, where ***j***, ***k***, ***r00***, and ***r01*** will be checked by recalling them. :

	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
After 2COMP, WSIZE 32, BASE 10	1 d	20 d	3,075 d	40,964 d	5 12,005 d	629 1462 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	40,964 d	-9 d	629 1462 d
DECM	-9.0	567.0	5.7 -396	4,000.4	3.0 -389	600,000.6
Recall the registers by iRCL			567.0	40,964.0	-9.0	629 1462.0

Note that identical register contents are interpreted quite differently in DECM and integer modes. Even very small integers may lead to very large surprises:

	R00	R01	R02
Contents at start e.g.	0	2	10
Then after WSIZE 64, BASE 16	0 h	2238000000000002 h	223E00000000000 1 h
Recall the registers by sRCL	0 h	2 h	A h
2 STO 01, A STO 02	0 h	2 h	A h
DECM	0	2 -398	1 -397
Recall the registers by iRCL	0	2	10

Thus take care with indirect addressing!

Example: Start with DECM, WSIZE 64, 2COMP, 0 STO 00, 2 STO 01, 10 STO 02 as above. So RCL→01 shall recall ***r02***. Let us check. Key in:

RCL → 01	10.	just for verification: OK
g 16	A h	turn to BASE 16
RCL → 01	Out of range Error	compare the table above
2 STO 01	2 h	
RCL → 01	000000000001 h	now this works in hex mode, too
f ◀	223C h	
f H.d	2'466'846'695'89 10	turn to DECM
RCL → 01	0.	compare the table above – indirection has to ignore all fractional components. Otherwise ISG/DSE loops all simply would not work.

All this is caused by the **internal representation of numbers:** While *integers* are simply stored as such (allowing for $\boldsymbol{n} \le 1.84 \cdot 10^{19}$ in UNSIGN or $|\boldsymbol{n}| \le 9.22 \cdot 10^{18}$ in the other modes using 64 *bits*), *floating point numbers* are stored using a format as follows:

- Real zero is stored as integer zero, i.e. all bits cleared.
- The mantissa of a real number (also known as *significand* in this context) is encoded in five groups of three digits. Each such group is packed into 10 *bits* straight forward, meaning e.g. $555_{10} = 10\ 0010\ 1100_2$ or $999_{10} = 11\ 1110\ 0111_2 = 3E7_{16}$. So the 15 rightmost decimal digits of the *significand* take the least significant 50 *bits*. Trailing zeroes are omitted, so the *significand* will be right adjusted.
- The most significant (64th) bit takes the sign of the mantissa.
- The remaining 13 *bits* are used for the exponent and the leftmost digit of the mantissa. Of those 13, the lowest 8 are reserved for the exponent. For the top 5 *bits* it gets complicated [66]: if they read …
  - `00ttt`, `01ttt`, or `10ttt` then `ttt` takes the leftmost digit of the *significand* ($0 - 7_{10}$), and the top two *bits* will be the most significant *bits* of the exponent;
  - `11uut` then `t` will be added to 1000_2 and the result (8_{10} or 9_{10}) will become the leftmost digit of the *significand*. If `uu` reads 00, 01, or 10 then these two will be the most significant *bits* of the exponent. If `uu` reads 11 instead, there are codes left for encoding special numbers (e.g. infinities).

  In total, we get 16 digits for the mantissa and a bit less than 10 *bits* for the exponent: its maximum is $10\ 1111\ 1111_2$ (i.e. 767_{10}). For reasons becoming

[66] Don't blame us – this part follows the standard IEEE 754.

obvious below, 398 must be subtracted from the value in this field to get the true exponent of the number represented. The 16 digits of the *significand* allow for a range from 1 to almost 10^{16}.

Rewarding your patience so far, we will show you some illustrative examples of the encoding in your *WP 34S* instead of telling you more theory:

Floating point number	Hexadecimal value stored	Bottom *bits* split in groups of 10	Top 14 *bits*	Stored exponent
1.	22 38 00 00 00 00 00 01		0010 0010 0011 10	398
-1.	A2 38 00 00 00 00 00 01		1010 0010 0011 10	398
111.	22 38 00 00 00 00 00 6F		0010 0010 0011 10	398
111.111	22 2C 00 00 00 01 bC 6F	06F 06F	0010 0010 0010 11	395
-123.000123	A2 20 00 00 07 b0 00 7b	07b 000 07b	1010 0010 0010 00	392
$9.99\cdot10^{99}$	23 bC 00 00 00 00 03 E7		0010 0011 1011 11	495
$1\cdot10^{-99}$	20 AC 00 00 00 00 00 01		0010 0000 1010 11	299
$1\cdot10^{-383}$	00 3C 00 00 00 00 00 01		0000 0000 0011 11	15
9.999 999 999 999 99$\cdot10^{384}$	77 FF E7 F9 FE 7F 78 00	9 3E7 3E7 3E7 3dE 000	0111 0111 1111 11	767

The next to last number is the smallest one that can be entered numerically, the last (featuring the digit 9 twelve times) is the greatest. All this follows *Decimal64* floating point format, though not exactly. Dividing 10^{-383} by 10^{15} results in 10^{-398}, being stored as hexadecimal 1. Adding $9.999\cdot10^{372}$ to the greatest number displays $1\cdot10^{385}$ stored as $9.999\,999\,999\,999\,999\cdot10^{384}$:

$1\cdot10^{385}$	77 FF E7 F9 FE 7F 9F E7	9 3E7 3E7 3E7 3E7 3E7	0111 0111 1111 11	767

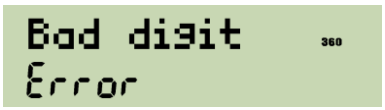
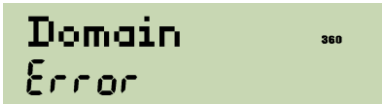
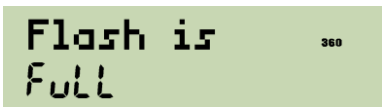
This is the greatest number representable here. Additionally, your WP 34S features three special numbers:

1nFinity	78 00 00 00 00 00 00 00		0111 1000 0000 00	n/a
-1nFinity	F8 00 00 00 00 00 00 00		1111 1000 0000 00	n/a
NaN	7C 00 00 00 00 00 00 00		0111 1100 0000 00	n/a

APPENDIX C: MESSAGES AND ERROR CODES

There are some commands generating messages, be they in the numeric or in the dot matrix section of the display. Of these, DAY, DAYS+, ERR, STATUS, VERS, and WDAY were introduced above in the *section about display*. Others are PROMPT, αVIEW and more alpha commands, and the test commands as mentioned *above*. Also two *constants* will return a special message when called.

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

Message	Error code	Mode(s)	Explanation and examples
	2	DECM	Invalid date format or incorrect date or time in input, e.g. month >12, day >31 .
	9	Integer	Invalid digit in integer input, e.g. 2 in binary or 9 in octal mode. Will be displayed as long as the respective key is pressed.
	13	All	Caused by calling an operation in a mode where it is not defined, e.g. calling a constant in a program written in DECM but executed in an integer mode.
	1	¬α	An argument exceeds the domain of the mathematical function called. May be caused by roots of negative numbers or logs of $x \le 0$ (both if not preceded by CPX), by $0/0$, $\Gamma(0)$, $\tan(90°)$ and equivalents, by $\operatorname{artanh}(x)$ for $\lvert\mathrm{Re}(x)\rvert \ge 1$, by $\operatorname{arcosh}(x)$ for $\mathrm{Re}(x) < 1$, etc.
	23	All	No more space in flash memory. Delete a program from flash to regain space.
	7	All	May appear in an attempt running an old program containing a command which turned non-programmable after said program was written.
	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 not usable.
	16	¬α	Similar to error 1 but a parameter specified in **J** or **K** is out of valid range for the function called. May appear e.g. if LgNrm is called with $j < 0$.

Message	Error code	Mode(s)	Explanation and examples
I/O Error	17	¬α	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. This can be caused by specifying decimals >11, word size >64, negative flag numbers, integers ≥2^{64}, *hours* or *degrees* >9000, invalid times, denominators ≥9999, etc. • A register address exceeds the valid range of currently allocated registers. May also happen in indirect addressing or calling nonexistent local addresses. • An R-operation (e.g. R-COPY) attempts accessing invalid register addresses. • 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*. May be caused by attempts to write too large programs, allocate too many registers, and the like. May happen also in program execution due to too many local data dynamically allocated (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 (if REGS 100 is set).
Too few DATA Points	15	DECM	A statistical calculation was started based on too few data points, e.g. regression or standard deviation for < 2 points.

Message	Error code	Mode(s)	Explanation and examples
	10	All	Keyboard input is too long for the buffer. Will happen e.g. if you try to enter more than 12 digits.
	3	All	An instruction with an undefined operation code occurred. Should never happen – but who knows?
	14	Integer, ¬PR	Register content is too big for the word size set.
	19	All	Attempt to delete or edit program lines in flash memory.
	4	¬α, ¬PR	• Division of a number > 0 (or < 0) by zero. • Divergent sum or product or integral. • Positive (or negative) overflow in DECM (see *above*). • Error 5 is also thrown for a logarithm of +0, while a logarithm of -0 returns NaN.
	5		

Each error message is *temporary* (see *above*), so ← or EXIT will erase it and allow continuation. Any other key pressed will erase it as well, but will also execute with the stack contents present. Thus, another easy and safe return to the display shown before the error occurred is pressing an arbitrary prefix twice.

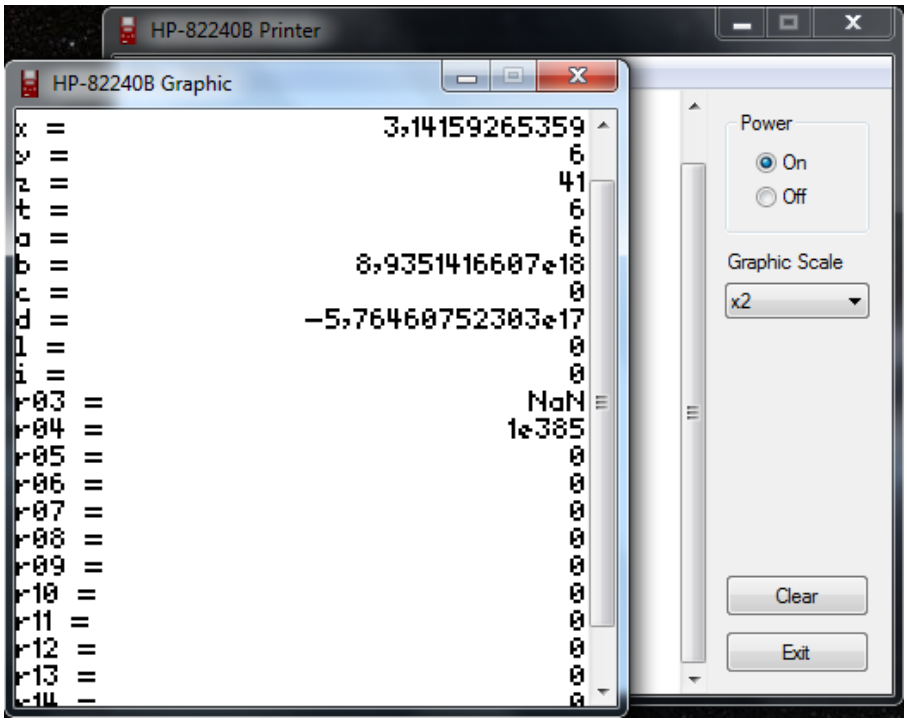
APPENDIX D: THE WP 34S EMULATOR ON YOUR COMPUTER

For the emulator, run `\trunk\windows\bin\wp34sgui.exe` – it features the identical function set as the calculator. While tactile feedback, boot speed, pocketability and battery life suffer, some things become easier, e.g. printing. What else is different?

Typically the emulator is operated by the computer mouse. You click on the respective areas of the keyboard image instead of pressing keys. Right clicking on a key area is a shortcut to its h-shifted label. And there is one important extra: a right click on the WP logo top right opens a menu as seen here.

It contains links to the project website and the manual (under <Help>) as well as to three different skins for the emulator. The 'medium' skin is shown several times in this manual; the other two are smaller and are displayed overleaf.

Furthermore, this menu allows you to import or export numbers, export ***alpha*** as text or the complete 'LCD' screen as a picture to the clipboard for use in other applications.



For simulated printing, use *Christoph Gießelink's HP82240B Printer Simulator*, available here: http://hp.giesselink.com/hp82240b.htm .

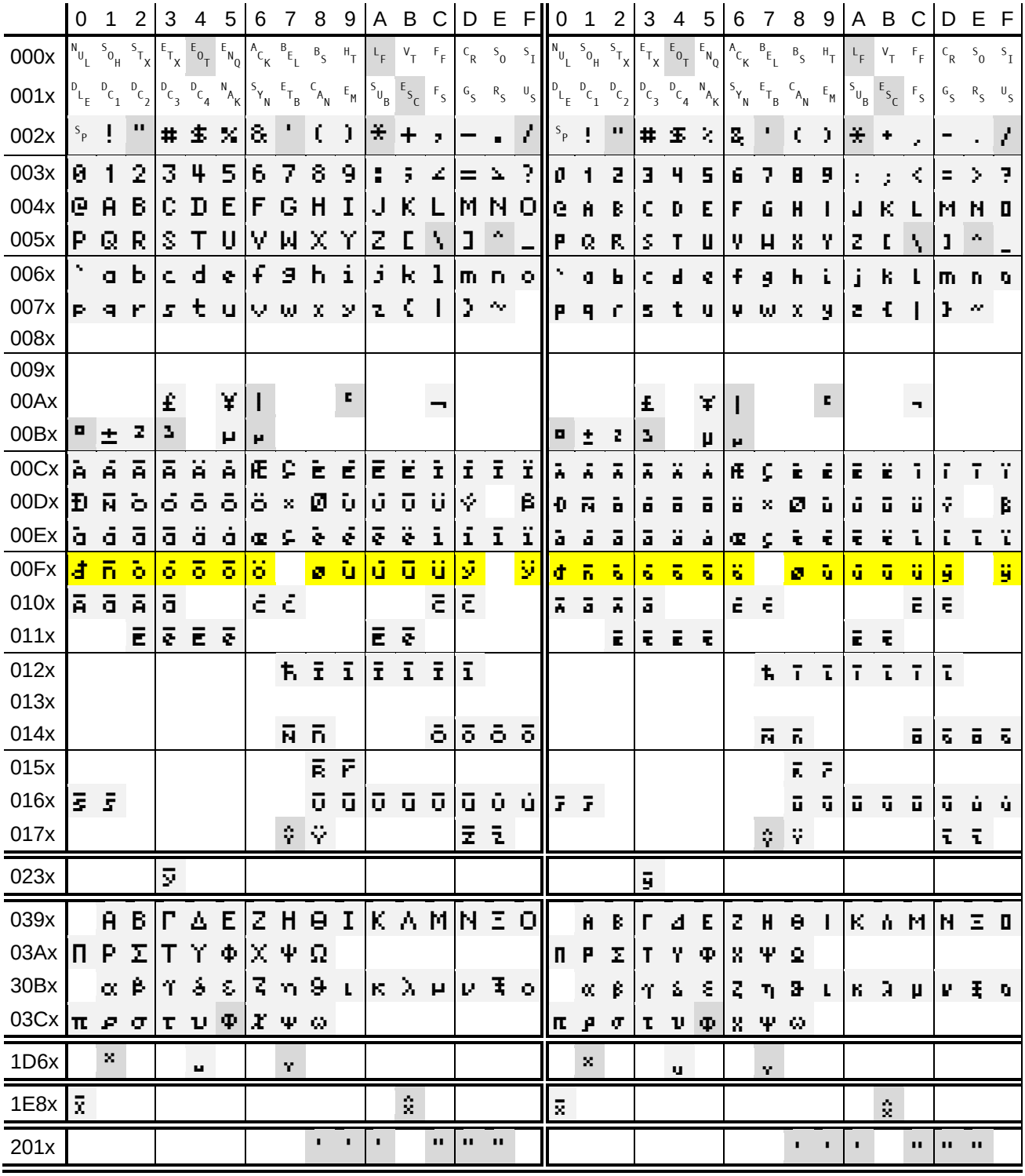
Installation and use are self-explanatory. It is an autonomous application, so you have to start and exit it independently. The 'print' shown here was generated by 🖶STK and 🖶REGS. Guess what stack size was set?

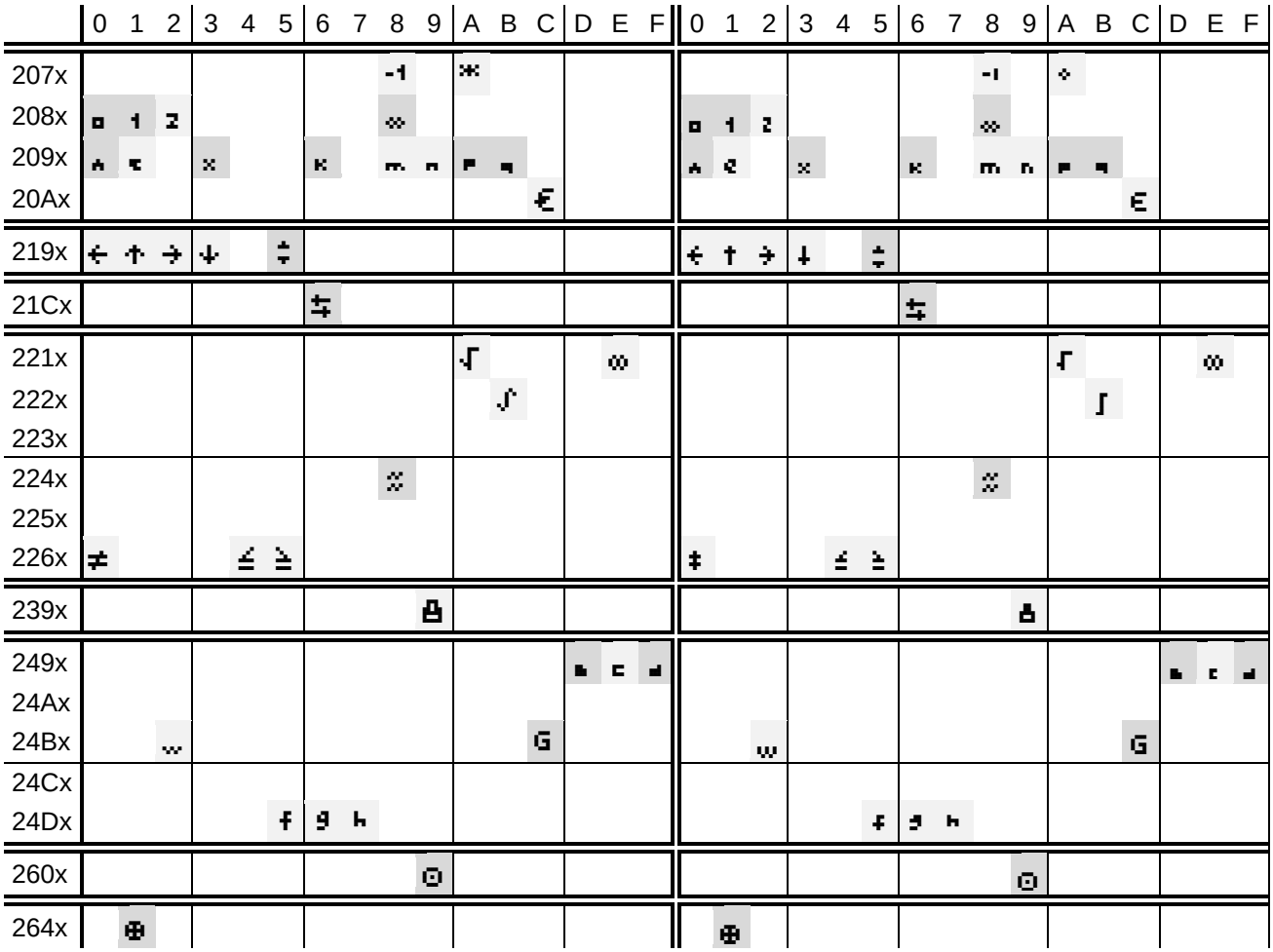

this is π
INPUT
BEG
360 RPN
3,14159265359
Σ+ 1/x y^x √x
A ALL B FIX C SCI D ENG → CONV CPX MODE
A HYP⁻¹ B SIN⁻¹ C COS⁻¹ D TAN⁻¹ E R<>P F a b/c d/c
STO CAT RCL VIEW R↓ R↑ f g h
G H.MS DEG H H.d RAD I RAN# GRAD
ENTER↑ CONST x≷y x<> +/- NOT EEX π ← CLx
α FILL J ◄( )► K 2 8 L 10 16 CLP CLΣ
XEQ GTO 7 AND 8 OR 9 XOR / RMDR
e^x LN M 10^x LG N 2^x LB O y^x LOG_x P 1/x ||
▲ ! 4 PROB 5 STAT 6 SUMS × MATRIX
C P x,y Q Φ⁻¹ R x̄ s S ŷ r T √x x²
▼ STATUS 1 TEST 2 P.FCN 3 X.FCN − Σ−
SF CF x=≠ ? U SLV ∫ V π Σ W % Δ%
EXIT OFF 0 PSE . •/, R/S P/R + Σ+
SHOW |x| RND X IP FP Y LBL RTN Z DSE ISG


this is π
INPUT
BEG
360 RPN
3,14159265359
Σ+ 1/x y^x √x
A ALL B FIX C SCI D ENG → CONV CPX MODE
A HYP⁻¹ B SIN⁻¹ C COS⁻¹ D TAN⁻¹ E R<>P F a b/c d/c
STO CAT RCL VIEW R↓ R↑ f g h
G H.MS DEG H H.d RAD I RAN# GRAD
ENTER↑ CONST x≷y x<> +/- NOT EEX π ← CLx
α FILL J ◄( )► K 2 8 L 10 16 CLP CLΣ
XEQ GTO 7 AND 8 OR 9 XOR / RMDR
e^x LN M 10^x LG N 2^x LB O y^x LOG_x P 1/x ||
▲ ! 4 PROB 5 STAT 6 SUMS × MATRIX
C P x,y Q Φ⁻¹ R x̄ s S ŷ r T √x x²
▼ STATUS 1 TEST 2 P.FCN 3 X.FCN − Σ−
SF CF x=≠ ? U SLV ∫ V π Σ W % Δ%
EXIT OFF 0 PSE . •/, R/S P/R + Σ+
SHOW |x| RND X IP FP Y LBL RTN Z DSE ISG

APPENDIX E: CHARACTER SETS

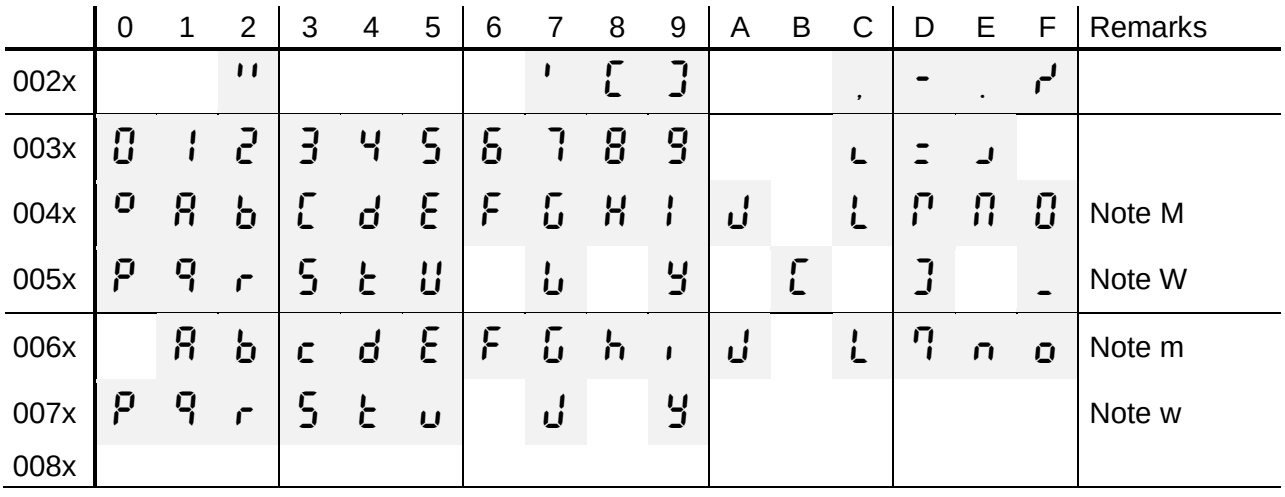
The following table shows the complete dot matrix character set as implemented in big and small font sorted according to the hexadecimal character codes (Unicode). Characters with codes $< 20_{16}$ are for control purposes – some of them (4, 10_{10}, 27_{10}) may be useful for *HP82240B* control.



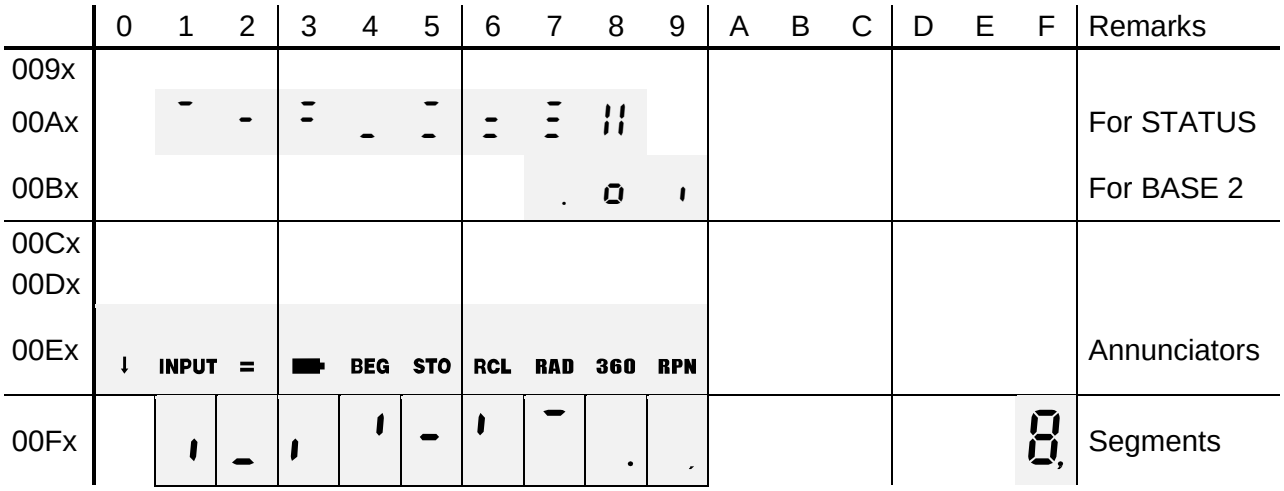


In addition, there are two raster fonts supplied. Characters printed on darker grey show identical patterns in all four fonts. Characters from 00F0 to 00FF (printed on yellow here) cannot be the last ones in a group of three (see LBL etc.). Note the internal character sorting in your *WP 34S* differs from Unicode – the characters of these four dot-matrix fonts are repeated in the internal order at codes greater than E000. Do not use these high codes in texts, however, since some programs (e.g. Preview) may not read them correctly.

For the numeric seven-segment display and the annunciators top right, there is another character set and font provided:



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Remarks
002x			"					'	(	)			,	-	.	/	
003x	0	1	2	3	4	5	6	7	8	9			;	:	;		
004x	@	A	B	C	D	E	F	G	H	I	J		K	L	M	N	Note M
005x	P	Q	R	S	T	U		V		Y		\		]		_	Note W
006x		a	b	c	d	e	f	g	h	i	j		k	l	m	n	Note m
007x	p	q	r	s	t	u		v		y							Note w
008x																	



All fonts are available at
http://wp34s.svn.sourceforge.net/viewvc/wp34s/trunk/windows/winfont/ .

APPENDIX F: CORRESPONDING OPERATIONS TO THE *HP-42S* FUNCTION SET

In the *IOP* above, the corresponding functions of vintage *HP* calculators were mentioned under the respective entry of your *WP 34S*. The table below reverts this in a way: it shows the functions of the *HP-42S* and the corresponding ones of your *WP 34S* unless they carry identical names and are either both keyboard accessible or both stored in a catalog. Remarks printed on light grey indicate commands being default settings or keyboard accessible on your *WP 34S* while you must use a menu on the *HP-42S*. **Functional differences of homonymous commands are covered in the index *above*.**

HP-42S	***WP 34S***	**Remarks**
ABS		Press [|x|].
ACOSH		Press [HYP⁻¹] [COS].
ADV	ADV	
AGRAPH	n/a	The LCD of the HP-30b has only a very small dot matrix section (see *above*). Striving for full fledged graphics therein would be futile.
AIP	αIP	
ALENG	αLENG	
ALL		Press [ALL].
ALLΣ	Superfluous	Your WP 34S always runs in ALLΣ mode.
[ALPHA]	[α]	See the description of alpha mode *above*.
AND		Press [AND].
AOFF	αOFF	
AON	αON	
ARCL	αRCL or αRC#	
AROT	αRL or αRR	
ASHF	αSL or αSR	
ASINH		Press [HYP⁻¹] [SIN].
[ASSIGN]	n/a	The LCD of the *HP-30b* does not allow soft keys. This prevents us from offering you a CUSTOM menu.
ASTO	αSTO	Press [f] [STO] in alpha mode.
ATANH		Press [HYP⁻¹] [TAN].
ATOX	α→x	
AVIEW	αVIEW	
[BASE]	Superfluous	HEXM, DECM, OCTM, BINM, as well as the LOGIC operations AND, OR, XOR, and NOT are on the keyboard of your *WP 34S* in all modes but alpha. A…F are provided in *integer modes* if applicable. The functions corresponding to BIT? and ROTXY are then in X.FCN.

HP-42S	*WP 34S*	**Remarks**
BASE+ etc.	Superfluous	Your *WP 34S* executes these arithmetic commands in integer modes automatically.
BEEP	n/a	The *HP-30b* does not contain the necessary hardware.
BINM	BASE 2	Press 2.
BIT?	BS?	
CATALOG	Replaced	This is split in *various catalogs* on your *WP 34S.*
CF		Press CF.
CLA	CLα	Press CLx in alpha mode.
CLD	Superfluous	Any keystroke will clear a *temporary message*.
CLEAR	P.FCN	CLΣ, CLP, CLST, CLA, and CLX are on the keyboard.
CLKEYS	n/a	See ASSIGN.
CLLCD	n/a	Use α CLx to clear the dot matrix section. See AGRAPH.
CLMENU	n/a	See ASSIGN.
CLP		Press CLP.
CLRG	CLREGS	
CLST	CLSTK	Press 0 FILL.
CLV	n/a	The hardware of the *HP-30b* does not provide space for variables. Thus, variable-based commands (e.g. many matrix commands) cannot be featured on your *WP 34S.*
CLΣ		Press CLΣ.
COMB		Press Cy,x.
COMPLEX	n/a	The hardware of the *HP-30b* does not provide space for different data types. Thus, complex numbers have to be put in two registers each on your *WP 34S* as described *above*.
CONVERT	CONV	
CORR		Press r.
COSH		Press HYP COS.
CPXRES	n/a	See COMPLEX.
CPX?		
CUSTOM	n/a	See ASSIGN.
DECM		Press H.d.
DEG		Press DEG.
DEL	n/a	Not featured. Too dangerous, in our opinion.
DELAY	PDLAY	
DELR	n/a	Matrix commands: see CLV.
DIM & DIM?		
DISP	Superfluous	FIX, SCI, ENG, ALL, RDX., and RDX, are on the keyboard.

HP-42S	*WP 34S*	**Remarks**
DSE		Press DSE.
E	EEX	
EDIT & EDITN	n/a	Matrix commands: see CLV.
ENG		Press ENG.
EXITALL	n/a	See ASSIGN.
FCSTX	x̂	
FCSTY	ŷ	Press ŷ.
FIX		Press FIX.
f FLAGS	TEST	SF and CF are on the keyboard.
FNRM	n/a	Matrix command: see CLV.
FP		Press FP.
GAMMA	Γ	
GETKEY	KEY?	
GETM	n/a	Matrix command: see CLV.
GRAD		Press GRAD.
GROW	n/a	Matrix command: see CLV.
HEXM	BASE 16	Press 16.
I+ and I-	n/a	Matrix commands: see CLV.
INDEX		
INPUT	PROMPT	
INSR	n/a	Matrix command: see CLV.
INTEG	∫	Press ∫.
INVRT	M^{-1}	
IP		Press IP.
ISG		Press ISG.
J+ and J-	n/a	Matrix commands: see CLV.
KEYASN	n/a	See ASSIGN.
KEYG & KEYX		
f LASTx	RCL L	
LBL		Press LBL.
LCLBL	n/a	See ASSIGN. Nevertheless, your *WP 34S* provides local labels as described *above*.
LINΣ	Superfluous	Your WP 34S always runs in ALLΣ mode.
LIST	n/a	Use PROG instead.
LOG	LOG_{10}	Press LG.

HP-42S	*WP 34S*	**Remarks**
MAN	CF T	This print mode is startup default on your *WP 34S.*
f MATRIX	n/a	Matrix commands: see CLV.
MAT?		
MEAN	x̄	Press x̄.
MENU	n/a	See ASSIGN.
f MODES	MODE	
MVAR	n/a	See ASSIGN.
N!	x!	Press !.
NEWMAT	n/a	Matrix command: see CLV.
NORM	n/a	Not featured.
NOT		Press NOT.
OCTM	BASE 8	Press 8.
OLD	n/a	Matrix command: see CLV.
ON	n/a	Not featured.
OR		Press OR.
PERM		Press Py,x.
f PGM.FCN	P.FCN	LBL, RTN, VIEW, PSE, ISG, and DSE are on the keyboard.
PGMINT	Contained in ∫	
PGMSLV	Contained in SLV	
PI	π	
PIXEL	gSET	See AGRAPH.
POLAR	n/a	See COMPLEX.
POSA	n/a	The alpha register contains 30 characters maximum. Automatic searching would be overkill, in our opinion.
PRA	🖶α	
PRLCD	🖶PLOT	See AGRAPH.
f PRGM	P/R	
f PRINT	P.FCN	PRX is on the keyboard.
f PROB	Replaced	COMB, PERM, N!, and RAN are on the keyboard. SEED is in STAT; GAM(MA) is in X.FCN.
PROFF	CF T	Press CF T.
PRON	SF T	Press SF T.
PRP	🖶PROG	
PRSTK	🖶STK	
PRUSR	n/a	See CLV.
PRV		

HP-42S	*WP 34S*	**Remarks**
PRX	🖶r X	Press ↑ except in alpha mode.
PRΣ	🖶Σ	
PSE		Press PSE.
PUTM	n/a	Matrix command: see CLV.
QUIET	n/a	See BEEP.
RAD		Press RAD.
RAN	RAN#	Press RAN#.
RCLEL & RCLIJ	n/a	Matrix commands: see CLV.
RDX,		Press ./. except in alpha mode.
RDX.		
REALRES	n/a	See COMPLEX.
REAL?		
RECT	Superfluous	This mode is permanent on your *WP 34S.* See COMPLEX.
RND	ROUND	Press RND.
RNRM	n/a	Matrix command: see CLV.
ROTXY	RL, RLC, RR, and RRC	
RSUM	n/a	Matrix commands: see CLV.
R<>R		
SDEV	s	Press s.
SF		Press SF
SHOW	◀	
SINH		Press HYP SIN.
SIZE	REGS	
SLOPE	L.R.	
SOLVE	SLV	Press SLV.
SOLVER	Contained in SLV	
SQRT	√‾	
STAT	STAT	MEAN, SDEV, FCSTY, and CORR are on the keyboard; the MODL setting commands are in MODE.
STOEL & STOIJ	n/a	Matrix commands: see CLV.
STR?	n/a	See CLV.
TANH		Press HYP TAN.
TONE	n/a	See BEEP.
TOP.FCN	n/a	See ASSIGN.
TRACE	SF T	Press SF T.

HP-42S	*WP 34S*	**Remarks**
TRANS	TRANSP	
UVEC	^{C}SIGN	
VARMENU	n/a	See CLV.
VIEW		Press VIEW.
WMEAN	$\bar{x}_w$	
WRAP	n/a	Matrix command: see CLV.
X<>		Press x≷.
X<0? & X<Y?	x< ?	
X≤0? & X≤Y?	x≤ ?	
X=0? & X=Y?	x= ?	Press x= ?.
X≠0? & X≠Y?	x≠ ?	Press x≠ ?.
X≥0? & X≥Y?	x≥ ?	
X>0? & X>Y?	x> ?	
XOR		Press XOR.
XTOA	x→α	
YINT	L.R.	Alternatively, press 0 ŷ.
f ∫f(x)	Contained in ∫	
ΣREG & ΣREG?	Superfluous	See *Appendix B*.
→DEC	Superfluous	Conversions are done automatically in *integer modes*.
→DEG	°→rad	Press CONV A ▲ XEQ.
→H.MS		Press → H.MS.
→HR		Press → H.d.
→OCT	Superfluous	Conversions are done automatically in *integer modes*.
→POL		Press P←.
→RAD	rad→°	Press CONV R XEQ.
→REC		Press →R.
%CH	Δ%	Press Δ%.

APPENDIX G: TROUBLESHOOTING GUIDE

Symptoms	Possible Cause	Recommended Therapy
After entering a decimal number like 32.4567, the display just shows  or similar.	Inadvertently entered integer mode.	Press f H .d to return to floating point mode.
You are working in binary integer mode and the display shows  or alike with such strange quotes top right.	Long integer number displayed	See *above* how to deal with such numbers.
Display looks like  or similar. Calculation commands are not executed.	Inadvertently entered programming mode.	Press EXIT to return to run mode.
Display looks strange, but not like the three patterns shown above.	Entered a browser or a catalog.	Press EXIT to return to previous mode and display.
All input goes to the top line.	Inadvertently entered alpha mode.	Press EXIT to return to previous mode.
After entering 1 . 2 3 or alike, an unwanted decimal comma (or point) shows up. How to get the requested point (or comma)?	Wrong radix mark set.	Press h ./, . Alternatively, choose a suitable regional setting from the table *above*.
After entering a decimal number like 3.4567, the display just shows 3. (there is a radix mark, but decimals are missing).	Wrong output format set.	Set h ALL 00 (or h FIX 4 here) to see all the decimals you entered.
Your WP 34S does not understand the dates or times you enter (anymore).	Wrong date or time format set.	Choose a suitable regional setting from the table *above*.

Symptoms	Possible Cause	Recommended Therapy
After entering *hours*, *minutes*, and *seconds*, only *hours* and *minutes* are displayed.	Wrong output format set.	Set h FIX 4 to see the *seconds* as well.
After entering a date like 11.112012, the year is truncated.	Wrong output format set.	Press h FIX 6 to see the complete year.
In long numbers, there are unwanted points or commas every four, three, or two digits.	E3ON or SEPON set.	If in an integer mode, press h MODE S XEQ for SEPOFF; in a floating point mode, press h MODE E XEQ for E3OFF.
After converting a number like 0.46875 into a fraction, you get  . While being true, this is not as precise as you want it.	DENMAX is too low.	Press h MODE D ▼ ▼ ▼ XEQ 0 ENTER↑ to get the maximum precision in fraction mode.

If you experience a strange situation not mentioned here, we recommend you check your display for annunciators as shown *above*.

APPENDIX H: ADDITIONAL INFORMATION FOR ADVANCED USERS

The information provided here may ease your work when you are going to do advanced computations, especially programming. It may require special care and/or a deeper understanding of the respective 'mechanics' of the *WP 34S* – otherwise you might be surprised by the consequences. **Use the following at your own risk!**

Mode Storing and Recalling

The command STOM stores mode data in a register. The following table shows more details. Types are coded G for general, F for fractions, D for decimal, and I for integer.

Bits	Type	Contents
0 .. 3	G	LCD contrast setting (0 … 7, always 4 for the emulator)
4, 5	F	Fractions denominator mode (DENANY = 0, DENFAC = 1, DENFIX = 2)
6 … 19	F	DENMAX (14 *bits* for 0 … 9999)
20	F	Clear for PROFRC, set for IMPFRC
21	F	Set if fraction mode is on
22, 23	D	Decimal display mode (ALL = 0, FIX = 1, SCI = 2, ENG = 3)
24 … 27	D	Number of decimals (4 *bits* for 0 … 11)
28	D	Clear for SCIOVR, set for ENGOVR
29	D	Clear for RDX**.** – set for RDX**,**
30	D	Set if E3OFF
31	I	Set if SEPOFF
32	I	Set if an integer mode is on
33	I	Set if LZON
34, 35	I	Integer sign mode (1COMPL = 1, 2COMPL = 0, UNSIGN = 2, SIGNMT = 3)
36 … 39	I	Integer base (4 *bits* for 2 … 16 coded as 1 … 15)
40 … 45	I	WSIZE (6 *bits* for 1 … 64 coded as 0 … 63)
46	D	Set if DBLON
47	D	Time format (24h = 0, 12h = 1)
48, 49	G	Print mode (0 … 3, see 🖶MODE)
50		Not used yet
51	G	Stack size (SSIZE4 = 0, SSIZE8 = 1)
52, 53	D	Date format (Y.MD = 1, M.DY = 2, D.MY = 3)
54, 55	D	Angular mode (DEG = 0 ,RAD = 1, GRAD = 2)

Bits	Type	Contents
56 … 58	D	Curve fit model (LINF = 0, EXPF = 1, POWERF = 2, LOGF = 3, BESTF = 4)
59	G	Clear for FAST, set for SLOW (always clear on the emulator)
60 … 62	D	Rounding mode (0 … 7, see RM)
63	D	Clear for JG1782, set for JG1582

Commands for Advanced Users

There are three ON combinations not documented above:

[ON] + [C] : Tells the system a quartz **c**rystal is installed for the real time clock. This **hardware modification** is described in *How to install crystal and IR diode* written by *Alexander Oestert* (Germany). It is an inevitable prerequisite for the clock being useful in medium to long range (see TICKS, STOPW), and it is required for print operations as well.

> **WARNING:** If the crystal is not installed, the system will hang if [ON] + [C] is entered or if a firmware is booted requiring the crystal, and a hard reset (actuating the reset button) or a battery pull will be required!
>
> Lack of the IR diode cannot cause a system hang since it cannot be checked automatically. Note this diode may alternatively be connected using one of the small *USB* boards developed by *Harald Pott* (Germany) and described in *WP_34S_USB_installation* .

[ON] + [D] : Enters **d**ebugging mode. This combination was used *above* (followed by [ON] + [S]) but not explained there.

[ON] + [S] stands for *My**S**amBa*: It clears the GPNVM1 bit and turns the calculator off. This will work in debugging mode only (see above) to prevent accidental access to this potentially dangerous feature.

> **WARNING:** After issuing this combination, you can only boot in *MySamBa* boot mode! Without the *MySamBa* software and a proper cable (as mentioned *above*), you are lost!

Double Precision Calculations and Mode Switching

Your *WP 34S* starts in standard (i.e. single precision or *SP*) real mode per default, wherein 16 digit precision is reached in all calculations. Switching between *SP* and integer modes was discussed *above*. Additionally, you may use your *WP 34S* in double precision (*DP*) mode. Each *DP* register will contain 16 bytes instead of eight, allowing for 34 digits instead of 16 (see below). Note matrix commands will not work in *DP*.

DP allows for more precise calculations. While some computations will reach high accuracy, **we do not warrant 34 digit precision in all calculations in *DP* mode.**

The following figure illustrates what happens in memory in transitions between *SP* and *DP* modes, assuming startup in *SP* mode with REGS 16. *RRA* stands for 'relative register address'.

Absolute address	Startup memory allocation Contents	*RRA*	After executing DBLON Contents	*RRA*	Then after DBLOFF Contents	*RRA*
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						

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

Going from *SP* to *DP* mode via DBLON, 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 *DP RRA* 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 then cut into the former program sector.

Returning from *DP* to *SP* mode, the lettered registers are copied again. Everything else stays where and as it was, if you used ≤ 44 *DP* registers – just each *SP RRA* points to only one half of a former *DP* register; and the memory released by the shrinking special

registers allows for adding (or returning) twelve numbered registers on top, each now containing zero.

With >44 *DP* registers, the correspondence becomes more complicated – the number of global registers will not, however, exceed 112.

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**:

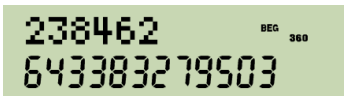
	J	K	R00	R01	R02	R03
Starts with e.g.	3.504 d	14 d	54 d	66 d	543 d	126.441 d
DECM	3.43^{-395}	1.40^{-397}	5.40^{-397}	6.60^{-397}	5.43^{-396}	1.23^{-393}
DBLON [67]	3.43^{-395}	1.40^{-397}	$1.03^{-HI\ 6}$	$1.95^{-HI\ 6}$	n/a	n/a
Recall by sRCL	1.40^{-397}	1.28^{-23}	5.40^{-397}	6.60^{-397}	5.43^{-396}	1.23^{-393}
… and by iRCL	14.00	0.00	54.00	66.00	543.00	126.441.00
DBLOFF	3.43^{-395}	1.40^{-397}	5.40^{-397}	6.60^{-397}	5.43^{-396}	1.23^{-393}
Recall by dRCL	Out of range Error	Out of range Error	0.00	0.00	n/a	n/a
DBLON	3.43^{-395}	1.40^{-397}	$1.03^{-HI\ 6}$	$1.95^{-HI\ 6}$	n/a	n/a
RCL J, STO J, 123E456 STO 00, then recall by sRCL	1.40^{-397}	1.28^{-23}	1.23^{-396}	5.12^{-30}	5.43^{-396}	1.23^{-393}
… and by iRCL	14.00	0.00	123.00	0.00	543.00	126.441.00
DBLOFF	3.43^{-395}	1.40^{-397}	1.23^{-396}	5.12^{-30}	5.43^{-396}	1.23^{-393}
Recall by dRCL	Out of range Error	Out of range Error	+∞ Error	0.00	n/a	n/a

Note iRCL and sRCL keep working as explained *above*.

---

[67] *DP* mode reals are stored coarsely following *decimal128* packed coding, though with some exceptions. The lowest 110 bits take the rightmost 33 digits of the *significand*. Going left, a 12 bit exponent field follows, then 5 bits used and coded exactly as in *SP*, and finally the sign bit. The greatest value of the stored exponent is 10 1111 1111 1111_2 = 12287_{10} . For reasons like the ones explained for *SP*, 6176 must be subtracted from this value to get the true exponent of the floating point number represented. Thus, *DP* supports 34-digit numbers within $10^{-6143} \leq |x| < 10^{+6145}$. Coding works in full analogy to the way described for *SP* above (compare *App. B*).
Even smaller numbers may be entered using a decimal mantissa, but you will lose one digit per factor of 10. The same happens if you divide 10^{-6143} by 10 several times. At 10^{-6176}, only one digit will be left, stored as hexadecimal 1. Divide it by 1.999 999 999 99 and the result will remain 10^{-6176}. Divide it by 2 instead and the result will become zero.
Numbers beyond the interval $10^{-999} \leq |x| < 10^{+1000}$ will be displayed with -HI 6 or HI 6 in their exponent, respectively. Only ◀ will show you the true exponent of them. This way ***r00*** = 1.032 000 000 000 000 054 E-6155 and ***r01*** = 1.951 656 000 000 000 000 000 543 E-6152 are found here.
Returning to *SP* with numbers in lettered registers exceeding the *SP* number range will cause 0 or Infinity being displayed instead.

In *DP* mode, ◀ shows the first (most significant) 16 digits of the 34-digit mantissa of x and its four digit exponent, and ▶ displays the 18 trailing digits, both as *temporary messages*. For **example**, π ◀ returns here

, and ▶ 

.

Remember not every return may be as precise as this one.

Further Commands Used in Library and *XROM* Routines

The operations listed below are used by the programmers of said routines. They ease their work by allowing some more comfort in writing programs than the original function set. These commands and pseudo-commands are explained here to foster understanding of those routines, but they are not accessible through any catalog. The assembler (often already its preprocessor) will translate most of them into proper program steps employing the commands documented above. See the *WP 34s Assembler Tool Suite User Guide* for additional information.

(Pseudo-) Command	*XROM* or *LIB*	Function
A..D→	*XROM*	saves ***a***, ***b***, ***c***, and ***d*** at a volatile temporary storage. Volatile means it will vanish some 500*ms* after last keystroke.
GSB ***label***	Both	This pseudo-command allows for calling long alphabetic labels. The assembler translates GSB into BSRB or BSRF (see *below*) – or into XEQ (via a jump table in *XROM* or via a label in *LIB*) if the distance is too far.
JMP ***label***	Both	This pseudo-command allows for branching to long alphabetic labels. The assembler translates JMP into BACK or SKIP – or GTO (via a jump table in *XROM* or via a label in *LIB*) if the distance is too far.
Num ***sn***	*XROM*	with ***sn*** = 0, π, √2π, … inserts the respective constant like recalling from CONST would do.
POPUSR	*XROM*	See XEQ or XEQUSR.
XEQ ***label***	*XROM*	calls the user routine carrying the global label specified (there are no LBL statements in *XROM*). XEQ'xyz' in *XROM* must be followed by POPUSR immediately to restore the *XROM* execution state (registers, flags, return addresses) correctly. See also xIN.

ATTENTION: Note that STOP and PROMPT will not work in *XROM* on your *WP 34S*. And there are no LBL nor END statements in *XROM* code. – Long alphabetic labels (looking like ***label*****::**) may show up in code written for the assembler but will be resolved.

(Pseudo-) Command	*XROM* or *LIB*	Function
XEQUSR ***label***	*XROM*	calls the user routine containing the function to be solved, integrated, summed, or multiplied, respectively. The label of this function is transmitted as a parameter of the respective user command described above (like e.g. SLV). XEQUSR must be followed by POPUSR immediately to restore the *XROM* execution state (registers, flags, return addresses) correctly..
xIN ***type***	*XROM*	with ***type*** = `NILADIC`, `MONADIC`, `DYADIC`, `TRIADIC`, or `COMPLEX_...` defines how many stack levels are used for parameter input to the function under consideration. Furthermore it does some initialization work (e.g. SSIZE8 and DBLON). xIN is the recommended way to start an *XROM* routine. Thereafter, SSIZE4 is legal but DBLOFF is not. Note xIN cannot nest and *XROM* routines using xIN cannot call user code.
XLBL"***label***"	*XROM*	defines the eXternal LaBeL of this routine. XLBL doesn't generate any code, it provides an entry point that the C function tables can take advantage of.
xOUT ***way***	*XROM*	typically, ***way*** = `xOUT_NORMAL` . xOUT cleans and reverts the settings of xIN, thus taking care of a proper return.
_INT ***n***	*XROM*	1, 2, …, 128, but also calculated constants are possible. _INT inserts the respective constant like # does.
"***text***"	Both	The double quotes allow for convenient entry of text strings. The assembler translates the string into the amount of α-statements necessary.
→A..D	*XROM*	recalls ***a***, ***b***, ***c***, and ***d*** from their temporary storage (very short range, see above).

Furthermore, there are several 'alias' spellings to ease input of some commands or even of individual nonstandard characters on an English computer keyboard. E.g. the Greek letter 'α' is often replaced by a Latin 'a', the print character 🖶 by 'P.' etc. Again, the assembler will take care of translating the aliases.

See the file `8queens_alias.wp34s` (or other files of type `…_alias.wp34s` in 'library') for examples. `Command-Aliases.pdf` in 'doc' contains a multipage list of all aliases used.

Assembler Output

The Assembler generates optimized code for your *WP 34S*. It does not generate optimized code for your further editing nor maintaining your routines. “Optimized” means it will use as little of the limited resources of your *WP 34S* in the programs as possible, less labels in particular. Thus expect heavy use of BACK and SKIP instead of GTO in assembled programs since all required information for this conversion is available at assembly time.

Furthermore, there are two special commands frequently replacing GSB / XEQ in assembled code:

1. BSRB ***n*** calls a subroutine starting ***n*** steps backwards with $0 \le n \le 255$. It pushes the program counter on the subroutine return stack and executes BACK ***n*** then.
2. BSRF ***n*** calls a subroutine starting ***n*** steps forwards. It pushes the program counter on the subroutine return stack and executes SKIP ***n*** then.

The subroutines called this way do not require a starting label. So, BSRB and BSRF are useful if you are short on local labels – like BACK, SKIP, and CASE are. On the other hand, if you edit a section of your routine that is crossed by one or more BSRB, BSRF, BACK, SKIP, or CASE jumps, this may well result in a need to manually maintain all those statements individually. We recommend editing unassembled code and leaving the generation of labelless branches to the assembler for sake of readability and reliability of your routines.

APPENDIX I: ADVANCED MATHEMATICAL FUNCTIONS

Your *WP 34S* contains several operations covering advanced mathematics. They are all implemented for the first time on an RPN calculator; all work in DECM. Find those functions collected here and explained in more detail than in the *IOP*.

Numbers

The following are all one-number functions.

Name	Remarks (see *above* for general information)
B_n	B_n returns the Bernoulli number for an integer $\boldsymbol{n} > 0$ given in **X**: $B_n = (-1)^{n+1} n \cdot \zeta(1-n)$.
B_n*	B_n* works with the old definition instead: $B_n^* = \frac{2 \cdot (2n)!}{(2\pi)^{2n}} \cdot \zeta(2n)$. See *below* for ζ(x).
FIB	Returns the extended Fibonacci number $F_x = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^x - \left(\frac{2}{1+\sqrt{5}}\right)^x \cos(x\pi)\right] = \frac{1}{\sqrt{5}}\left[\Phi^x - \Phi^{-x}\cos(x\pi)\right]$ with Φ denoting the golden ratio.

Statistical Distributions

The following are all one-number functions and are found in PROB. Those distributions not mentioned below are covered sufficiently in the *IOP*.

Basic Name	Remarks (see *above* for general information)
Cauch	Cauchy-Lorentz distribution (also known as Lorentz or Breit-Wigner distribution) with the location $\boldsymbol{x_0}$ specified in **J** and the shape γ in **K** : Cauch$_P$ returns $f_{Ca}(x) = \left\{\pi\gamma \cdot \left[1 + \left(\frac{x - x_0}{\gamma}\right)^2\right]\right\}^{-1}$, Cauch returns $F_{Ca}(x) = \frac{1}{2} + \frac{1}{\pi}\arctan\left(\frac{x - x_0}{\gamma}\right)$.

Basic Name	Remarks (see *above* for general information)
LgNrm	Lognormal distribution with $\mu = \ln \bar{x}_g$ specified in **J** and $\sigma = \ln \varepsilon$ in **K**. LgNrm returns $F_{Ln}(x) = \Phi\left(\frac{\ln(x) - \mu}{\sigma}\right)$ with $\Phi(z)$ denoting the standard Normal *cdf*. LgNrm$_P$ returns $f_{Ln}(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{[\ln(x)-\mu]^2}{2\sigma^2}}$.
Logis	Logistic distribution with ***μ*** given in **J** and ***s*** in **K**. With $\xi = \frac{x-\mu}{s}$, Logis returns $F_{Lg}(x) = \frac{1}{1+e^{-\xi}}$ and Logis$_P$ returns $f_{Lg}(x) = \frac{e^{-\xi}}{s \cdot \left(1+e^{-\xi}\right)^2}$.
Norml	Normal distribution with an arbitrary mean ***μ*** given in **J** and a standard deviation ***σ*** in **K**. Norml returns $F_N(x) = \Phi\left(\frac{x-\mu}{\sigma}\right)$. See *above* for $\Phi(x)$. Norml$_P$ [68] returns $f_N(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$.
Weibl	Weibull distribution with its shape parameter ***b*** in **J** and its characteristic lifetime ***T*** in **K**. Weibl returns $F_W(t) = 1 - e^{-\left(\frac{t}{T}\right)^b}$, and Weibl$_P$ [69] returns $f_W(t) = \frac{b}{T}\left(\frac{t}{T}\right)^{b-1} e^{-\left(\frac{t}{T}\right)^b}$.

More Statistical Formulas

Note that complete results of measured samples must include both an information about the expected value and its uncertainty.

- For samples drawn out of a Gaussian (additive) process, the expected value is the arithmetic mean and its uncertainty is given by its standard error (see $\bar{x}$ and SERR).
- For samples drawn out of a log-normal (multiplicative) process, the expected value is the geometric mean and its uncertainty is given by its scattering factor (see $\bar{x}_g$, ε_m).

[68] Norml$_P$ corresponds to NORMDIST(***x; μ; σ;*** **0**) in MS Excel, Norml to NORMDIST(***x; μ; σ;*** **1**) and Norml^{-1} to NORMINV(***F$_N$; μ; σ***) .

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

- For samples drawn out of other processes other measures apply.

Be assured not everything is Gaussian in real world! Process features can be detected (and should be checked well in advance of calculating e.g. means) using suitable tests – turn to applicable statistical reference literature.

The following functions as named in the left column are all found in STAT (except $\bar{x}$ and s, which are added to this table for comparison only).

Name	Remarks (see *above* for general information)
COV	For a linear fitting model, the population covariance is $COV_{xy} = \frac{1}{n^2}\left(n\sum x_i y_i - \sum x_i \sum y_i\right)$. Compare s_{XY} below.
s	The sample standard deviation is $s = \sqrt{\frac{n\sum x_i^2 - \left(\sum x_i\right)^2}{n\cdot(n-1)}} = \sqrt{\frac{\sum x_i^2 - n\cdot\bar{x}^2}{n-1}}$.
s_W	The sample standard deviation for weighted data (where the weight ***y*** of each data point ***x*** was entered via Σ+) is $s_w = +\sqrt{\frac{\sum y_i \cdot \sum\left(y_i \cdot x_i^2\right) - \left[\sum\left(y_i \cdot x_i\right)\right]^2}{\left(\sum y_i\right)^2 - \sum y_i^2}}$.
s_{XY}	For a linear fitting model, the sample covariance is $s_{xy} = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{n\cdot(n-1)}$. Compare COV above. Note that s_{XX} is just the variance of the *x*-data, i.e. their standard deviation s squared.
$\bar{x}$	The arithmetic mean is calculated as $\bar{x} = \frac{1}{n}\sum x_i$.
$\bar{x}_g$	The geometric mean is calculated as $\bar{x}_g = \sqrt[n]{\prod x} = e^{\frac{1}{n}\sum \ln x}$.
ε	The scattering factor ε_x for log-normally distributed data is calculated via: $\ln(\varepsilon_x) = \sqrt{\frac{\sum \ln^2(x_i) - 2n\cdot\ln\left(\bar{x}_g\right)}{n-1}}$. Compare s.
σ_W	The standard deviation of the population is $\sigma_w = +\sqrt{\frac{\sum y_i\left(x_i - \bar{x}_w\right)^2}{\sum y_i}}$. Compare s.

Orthogonal Polynomials

These polynomials are all found in X.FCN.

Name	Remarks (see *above* for general information)
H_n	(2) 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}	(2) 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**.
L_n	(2) 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 $x\cdot f''(x)+(1-x)f'(x)+n\cdot f(x)=0$.
$L_n\alpha$	(3) Laguerre's generalized polynomials (compare L_n above): $L_n^{(\alpha)}(x)=\frac{x^{-\alpha}e^x}{n!}\cdot\frac{d^n}{dx^n}\left(x^{n+\alpha}e^{-x}\right)$ with ***n*** in **Y** and α in **Z**.
P_n	(1) 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 $\frac{d}{dx}\left[\left(1-x^2\right)\cdot\frac{d}{dx}f(x)\right]+n(n+1)f(x)=0$.
T_n	(2) Chebychev's (a. k. a. Čebyšev, Tschebyschow, Tschebyscheff) polynomials of first kind $T_n(x)$ with ***n*** in **Y**, solving the differential equation $(1-x^2)f''(x)-x\cdot f'(x)+n^2\cdot f(x)=0$.
U_n	(2) Chebychev's polynomials of second kind $U_n(x)$ with ***n*** in **Y**, solving the differential equation $(1-x^2)f''(x)-3x\cdot f'(x)+n(n+2)f(x)=0$.

More Functions

Name	Remarks (see *above* for general information)
g_d	(1) Returns the Gudermann function $g_d(x)=\int_0^x \frac{d\xi}{\cosh\xi}$.
g_d^{-1}	(1) Returns the inverse Gudermann function $g_d^{-1}(x)=\int_0^x \frac{d\xi}{\cos\xi}$.
Iβ	(3) Returns the regularized (incomplete) beta function $\frac{\beta_x(x,y,z)}{\beta(y,z)}$ with $\beta(y,z)$ being *Euler's Beta* (see *there*) and $\beta_x(x,y,z)=\int_0^x t^{y-1}(1-t)^{z-1}dt$ being the incomplete beta function.
IΓp	(2) Returns the regularized (incomplete) gamma function $P(x,y)=\frac{\gamma(x,y)}{\Gamma(x)}$. For $\gamma(x,y)$ see γ_{XY} below, for $\Gamma(x)$ see there.
IΓq	(2) Returns the regularized (incomplete) gamma function $Q(x,y)=\frac{\Gamma_u(x,y)}{\Gamma(x)}$. For $\Gamma_u(x,y)$ see Γ_{XY} below, for $\Gamma(x)$ see there.
γ_{XY}	(2) Returns the lower incomplete gamma function $\gamma(x,y)=\int_0^y t^{x-1}e^{-t}dt$.
Γ_{XY}	(2) Returns the upper incomplete gamma function $\Gamma_u(x,y)=\int_y^\infty t^{x-1}e^{-t}dt$.
ζ	(1) Returns Riemann's Zeta for real arguments, with $\zeta(x)=\sum_{n=1}^{\infty}\frac{1}{n^x}$ for $x>1$, and its analytical continuation for $x<1$: $\zeta(x)=2^x\pi^{x-1}\sin\left(\frac{\pi}{2}x\right)\cdot\Gamma(1-x)\cdot\zeta(1-x)$.

APPENDIX J: 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 already included in CONST); 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 key-stroke; made D.MY default instead of Y.MD; moved degrees to radians conversions to CONV; removed ^{C}CLx, H.MS mode, %+ and %-; corrected errors.
1.18	5.6.11	Expanded program memory; modified label addressing (A ≠ 'A') and fraction mode limits, changed ANGLE to work in real and complex domains, renamed MOD to RMDR, changed the keyboard layout; put BACK, ERR, SKIP, and SPEC? to the main index; added CAT and the I/O commands for flash memory, expanded R-COPY; corrected x→α.
2.0	21.7.11	Entered beta test phase. Added DAY, MONTH, YEAR, FAST, SLOW, S.L, S.R, VWα+, flag A, ON + and –, some constants, and a paragraph about I/O; renamed old DAY to WDAY, RRCL to RCFRG, SRCL to RCFST; added an inverse conversion shortcut, stones⇆kg, and changed Pa⇆mbar to Pa⇆bar; modified the VIEW commands, ALL, DISP, MODE, RCLM, STOM, and X.FCN; repaired hyperlinks; corrected some errors; included flash.txt; updated the first chapters, explained stack mechanics in more detail.
2.1	3.10.11	Added serial I/O commands, DEL_P, DSL, EXPT, IBASE?, INTM?, ISE, KTY?, MANT, NEXTP, PUTK, REALM?, RM, RM?, SMODE?, TOP?, $\sqrt[x]{y}$, signed tests for zero, some constants, and the paragraph about interactive programming; updated the values in CONST to CODATA 2010, also updated SLVQ, SHOW, Σ, Π, and the paragraphs about statistics, predefined alpha labels and memory; corrected some errors; deleted complex ANGLE, →BIN, →DEC, →HEX, and →OCT; redistributed the contents of X.FCN and P.FCN; renamed S.L and S.R to SDL and SDR; put '?' on the alpha keyboard and moved £ to P to make room for π; expanded Appendix A; reorganized the structure of the document; added first aid to the front page; rewrote the keyboard chapter.
2.2	1.11.11	Added MSG, y↔, z↔, and matrix operations, a paragraph about them and two new error messages for them, plus a footnote for DEL_P; updated the introduction to statistics. This version is the last one working with the old overlays. It is maintained to incorporate the latest common bug fixes – last v2.2 build available is 2739 so far.
3.0	21.4.12	Added CLPALL, CROSS, DOT, iRCL, sRCL, END, FLASH?, g_d, g_d^{-1}, GTO.▲ and ▼, LOAD…, LocR, LocR?, MEM?, NEIGHB, PopLR, RDP, REGS, REGS?, RSD, SEPOFF, SEPON, SETJPN, STOPW, t↔, ULP, ↔, #, as well as SUMS and MATRIX catalogs and four conversions; renamed CLFLAG to CFALL, CUBE to x^3 and CUBERT to $\sqrt[3]{x}$, KTY? to KTP?, and αVIEW to VIEWα; split Lambert's W into W_p and W_m; returned →BIN, →HEX, and →OCT; made PSTO and SAVE non-programmable; redefined SHOW, corrected CLREG, deleted DEL_P; 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 and π from CONST; modified the virtual alpha keyboard, some characters and the respective catalogs; redistributed commands in the catalogs; updated and rearranged large parts of the text; added information about complex calculations, bitwise integer operations, browsers, local data, memory management, and the character sets; implemented five new ttf-fonts.
3.1	27.7.12	Added the print commands 🖶…, graphic commands g…, error probabilities, $I\Gamma_q$, MOD, XTAL?, γ_{XY}, Γ_{XY}, ×MOD, ^MOD, an optional back label, and some references to external documents; modified GCD and LCM; expanded STOPW; renamed IBASE? to BASE? and IΓ to $I\Gamma_p$; purged TEST moving non-tests into P.FCN; abandoned the internal expert's catalog; updated and rearranged large parts of the introductory text and the appendices, also explaining the floating point formats, the emulator, troubleshooting, and corresponding operations of *HP-42S*.